UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación

Clasificación de imágenes de melanomas mediante redes profundas y vectores de Fisher

Tesis presentada para optar al título de Magíster en Explotación de Datos y Descubrimiento del Conocimiento

Gastón Elías Liberman
Ingeniero en Sistemas de Información

Director: Dr. Daniel Acevedo
Fecha de defensa: 13/05/2019

Abstract

This work discusses the application of data mining techniques and the deep training of neural networks (deep learning) with the objective of classifying images of pigmented skin lesions into "Melanoma" or "Non-Melanoma." An ensemble of 3 classifiers will be created for this purpose. The first one corresponds to a VGG-16 convolutional network, while the other 2 correspond to 2 Hybrid Models. Each Hybrid Model is composed of a convolutional neural network (VGG-16 without the fully connected layers) and a support vector machine (SVM) as a classifier. These classifiers will be trained with the Fisher vectors (FVs) that are calculated using the descriptors that are output by the aforementioned convolutional neural network. The difference between these 2 Hybrid Models lies in the fact that one has segmented images as an input for the convolutional network, while the other uses non-segmented images. The segmentation is performed by means of a custom network (TernausNet-16) that follows the U-Net philosophy. Finally, we will analyze the performance of the Hybrid Model, the VGG-16 convolutional network and the Ensemble that incorporates the 3 classifiers (Hybrids Models and the VGG-16 convolutional network).

Keywords: Melanoma Classification-Deep Learning-Fisher Vectors.

Resumen

El presente trabajo corresponde a la aplicación de técnicas de minería de datos y al entrenamiento profundo de redes neuronales (deep learning) con el objetivo de clasificar las imágenes de lesiones cutáneas pigmentadas en “Melanoma” o “No Melanoma”. Para tal fin, se creará un Ensamble de 3 clasificadores. El primero corresponde a la red convolucional VGG-16, los otros 2 corresponden a 2 Modelos Híbridos. Cada Modelo Híbrido está compuesto por una red convolucional (la VGG-16 sin las capas totalmente conectadas) y una máquina de vectores de soporte (SVM) como clasificador. Estos clasificadores serán entrenados con los vectores de Fisher (FVs) calculados con los descriptores que son la salida de la red convolucional mencionada anteriormente. La diferencia entre los 2 Modelos Híbridos radica en que uno tiene a las imágenes segmentadas como input de la red convolucional, mientras que el otro utiliza las imágenes sin segmentar. La segmentación se realiza mediante una red custom (TernausNet-16) que sigue la filosofía de la U-Net. Por último, se analizará la performance del Modelo Híbrido, el de la red convolucional VGG-16 y el del Ensamble que incorpora los 3 clasificadores (Modelos Híbridos y la red convolucional VGG-16).

Palabras clave: Clasificación del Melanoma-Entrenamiento Profundo de Redes Neuronales-Vectores de Fisher.

Agradecimientos

Expreso mi agradecimiento a mi director de tesis Dr. Daniel Germán Acevedo por su paciencia infinita y guiarme en el desarrollo de esta investigación. También quiero agradecer a la Dra. Marta Mejail que tuvo una función de codirección y junto a Daniel hicieron posible que tenga una primer experiencia en la participación de un congreso internacional.

Además, agradezco al director de la maestría Dr. Marcelo Soria por su gestión y buena predisposición ante cada consulta de mi parte.

Por último y no menos importante, a la Dra. María Elena Buemi por su coordinación y vincularme con mi director de tesis, a la administración de la maestría y a los profesores por todo el conocimiento brindado.

1 Introducción

1.1 Marco contextual dermatológico

La piel

La piel es el órgano más amplio (en cuanto a su superficie) del ser humano. La misma se encuentra dividida en 3 capas: la epidermis, la dermis y la hipodermis (también llamada tejido subcutáneo), como se indica en la figura 1.1 [1]. En particular, la epidermis se encuentra conformada por células basales, escamosas y melanocitos. Los melanocitos, que producen los pigmentos de la piel, pueden empezar a dividirse sin control y por ende comenzar con el denominado melanoma cutáneo [2].

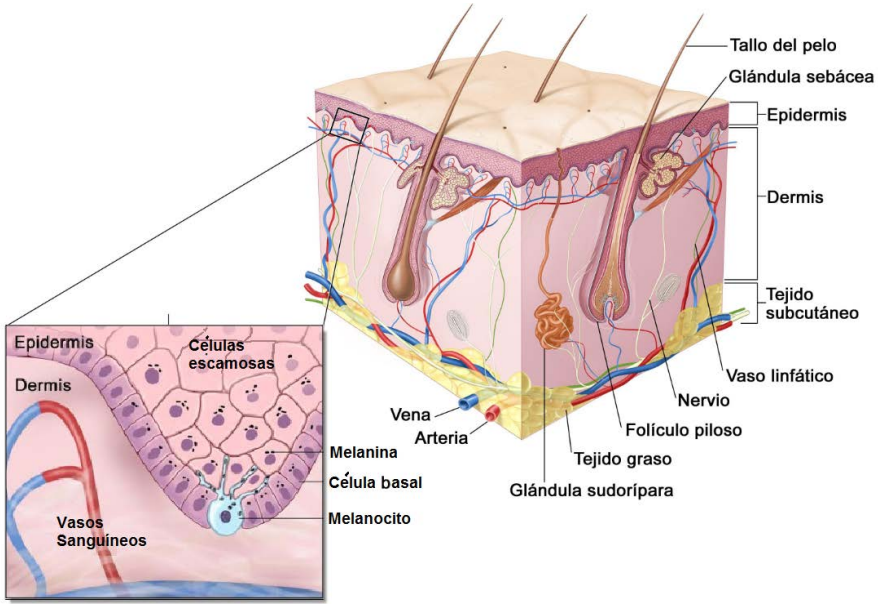


Figura 1.1: Capas de la piel humana.

Cáncer de piel

El cáncer de piel es una afección por la que se forman células malignas (cancerígenas) en los tejidos de la piel. Lo podemos dividir en 2 grandes grupos: cáncer de piel no melanoma y melanoma maligno [2] [3].

Cáncer de piel no melanoma

Se tienen 3 tipos de cáncer de piel principales: el carcinoma basocelular, el carcinoma espinocelular y el carcinoma celular escamoso.

El carcinoma basocelular (ver figura 1.2) se origina en las células del estrato germinativo basal en la epidermis. Es de crecimiento lento, en muy raras ocasiones presenta metástasis y tiene un mínimo riesgo de ser mortal [3].

Figura 1.2: Carcinoma basocelular nodular color piel.

El carcinoma espinocelular (ver figura 1.3) es de crecimiento acelerado y tiene altas probabilidades de hacer metástasis y suele identificarse sobre las lesiones precancerosas.

Figura 1.3: Carcinoma espinocelular. Lesión tumoral queratósica.

El carcinoma celular escamoso o cáncer de células escamosas (ver figura 1.4) se desarrolla en la piel que ha absorbido los rayos del sol durante varios años. Habitualmente aparece en la cara, las orejas, los labios, el dorso de las manos, los brazos y las piernas. Entre un 40% y un 60% de los cánceres de células escamosas comienzan con una queratosis actínica. Siendo esta última pequeñas lesiones eritematosas y escamosas originadas sobre la piel que recibió mucha radiación solar [4].

Figura 1.4: Carcinoma de celulas escamosas.

Melanoma Maligno

Es principalmente cutáneo aunque puede producirse en el ojo o en el intestino. Además es altamente invasivo por su capacidad de generar metástasis. Existen diferentes tipos de melanomas malignos: melanoma extensivo superficial, melanoma lentigo maligno, melanoma nodular y el melanoma lentiginoso acral [3].

El melanoma extensivo superficial (ver figura 1.5) es el tipo de melanoma más frecuente. Representa, aproximadamente, el 70% de los casos de melanoma maligno. Suele manifestarse en el tronco y en las piernas. Clínicamente se presenta como una placa pigmentada, asimétrica y de contornos irregulares. Con respecto al color, coexisten en la misma lesión zonas negras, marrones y rojizas [3].


Figura 1.5: Melanoma de extensión superficial.

El melanoma lentigo maligno (ver figura 1.6) se manifiesta como una mácula hiperpigmentada, siendo el gris, el negro y el marrón los colores predominantes. Suelen aparecer en la cara y más específicamente en la nariz y en las mejillas [3].


Figura 1.6: Melanoma lentigo maligno.

El melanoma nodular (ver figura 1.7) es aquella lesión donde originariamente presenta nódulos. Si bien puede aparecer en cualquier lugar del cuerpo, con más frecuencia suele manifestarse en el dorso o en extremidades en personas de 40 a 60 años. Representa entre el 15% y el 30% de los melanomas malignos [3].


Figura 1.7: Melanoma nodular.

El melanoma lentiginoso acral (ver figura 1.8) se presenta como una mancha marrón oscura, pudiendo aparecer áreas azuladas o despigmentadas. Es de crecimiento lento y aparece con frecuencia en las palmas de las manos, en la planta de los pies y en las zonas subungueales. Representa, solamente, del 2% al 8% de los melanomas malignos en la raza blanca, de un 60% a un 72% en la raza negra y de un 39% a un 46% en los asiáticos [3].


Figura 1.8: Melanoma lentiginoso acral.

Por último existen otros tipos de cáncer de piel menos frecuente y que en conjunto representan menos del 1% de todos los casos de cáncer de piel. Estos son: carcinoma de células Merkel, sarcoma de Kaposi, linfoma cutáneo, tumores de los anexos de la piel y varios tipos de sarcoma [2].

Como vimos anteriormente, existen diferentes tipos de cáncer de piel, no obstante en esta tesis nos enfocaremos a la clasificación de imágenes de melanomas cutáneos. En el presente trabajo disponemos de un conjunto de imágenes que incluyen melanomas cutáneos y lesiones benignas del tipo nevo melanocítico, lentigo solar o queratosis seborreica. A todas estas lesiones cutáneas pigmentadas, a lo largo de la tesis, las englobamos con el término de *lesiones pigmentadas*. Por otro lado, a los melanomas cutáneos los referenciaremos simplemente como *melanomas*.

Dermatoscopía

La dermatoscopía también llamada microscopía de epiluminiscencia, es una técnica no invasiva de diagnóstico que mediante un instrumento llamado dermatoscopio permite examinar las lesiones por debajo de la superficie cutánea amplificando la imagen y eliminando los fenómenos de refracción y reflexión de la luz [5].

El dermatoscopio (ver figura 1.9) es un microscopio o estereomicroscopio ya que dispone de un lente óptico con una amplificación y un juego de leds que constituyen una fuente de luz de polarización cruzada y lineal. La polarización lineal ilumina la superficie de la lesión mientras que la cruzada disminuye el reflejo provocado por la epidermis pudiendo visualizar estructuras anatómicas de la epidermis que a simple vista no podrían ser observadas. Cuando se acopla el dermatoscopio a una cámara digital, se capturan las llamadas imágenes dermatoscópicas.

Figura: 1.9: (Izquierda) Dermatoscopio para mejorar el diagnóstico de las lesiones. (Derecha) Dermatoscopio acoplado a una cámara digital para obtener las imágenes dermatoscópicas.

1.2 Tema de investigación

El tema a investigar corresponde a la clasificación automática de imágenes de melanomas utilizando técnicas de minería de datos (deep learning - machine learning).

El melanoma corresponde al cáncer de piel que causa el 75% de las muertes de todas las enfermedades cancerígenas cutáneas. Se estima que aparecen por año 160000 casos nuevos de melanomas y 48000 muertes según la OMS. Actualmente, el único tratamiento efectivo es la resección quirúrgica del tumor antes que logre un espesor mayor a 1 milímetro. Dicho esto, los dermatólogos para confirmar la presencia o no de un melanoma, proceden a la realización de una biopsia. Este procedimiento incrementa su complejidad cuando la persona tiene un gran número de lesiones pigmentadas sospechosas, ya que resulta ser un proceso invasivo[1]. Además, la detección del melanoma se encuentra totalmente condicionada al entrenamiento del dermatólogo. Con lo cual, resulta importante tener una herramienta que permita clasificar la presencia de un melanoma sin invadir el cuerpo humano y cuya efectividad de clasificación sea superior a la media del dermatólogo, reduciendo la subjetividad de la visión humana.

En el presente trabajo se utilizaron las imágenes recibidas de la Dra. Viviana Kowalckzuc del Hospital Italiano, las imágenes del ISIC Archive[2] y de Dermnet[3]. Se procedió a hacer Web scraping en distintas bases de datos, dado que el melanoma es una enfermedad poco común (en relación a las lesiones pigmentadas benignas) y existe una problemática inherente al desbalanceo de clases. Se recuerda que el Web scraping es la técnica que, mediante programas de software, extrae información de sitios Web.

El sistema consta de un módulo de preprocesamiento de la imagen, uno de segmentación y otro de clasificación. En el módulo de preprocesamiento se rescalan las imágenes y se aplica Max-RGB [6]. Max-RGB o parche blanco es una técnica utilizada para filtrar los efectos de la fuente de luz que pueden producir una distorsión en la imagen (similar al filtro que realiza automáticamente el ser humano). La segmentación fue resuelta por una red custom que sigue la filosofía de la U-Net [7]. Para la clasificación se empleó una red VGG-16 [8] (pre-entrenadas sobre ImageNet[4]) y 2 Modelos Híbridos. Estos Modelos Híbridos utilizan los descriptores locales generados por la red VGG-16 para codificarlos en los vectores de Fisher (FVs) [9] y luego clasificarlos mediante una máquina de vectores de soporte (Support Vector Machine o SVM). Además se visualizaron los mapas de calor sobre las imágenes clasificadas, estos corresponden a las zonas de la imágen que utilizó la red neuronal convolucional (CNN o ConvNet o red convolucional) para poder realizar la mencionada clasificación. El método que se utilizó para obtener el mapa de calor es GRAD-CAM [10]. Este método le servirá al dermatólogo, ya que podrá verificar qué zonas de la lesión toma en cuenta la red para efectuar la clasificación correspondiente.

Por último se creó un Ensamble de los 3 clasificadores y se procedió a comparar dicho clasificador con el Modelo Híbrido de mejor rendimiento y con la CNN convencional (VGG-16) en los distintos conjuntos de datos de prueba.

Los resultados de este trabajo fueron presentados en el artículo “Clasificación de imágenes de melanomas con vectores de Fisher y deep learning” y enviados a la conferencia *23rd Iberoamerican Congress on Pattern Recognition* (**CIARP**) 2018 para su publicación. Dicho artículo fue aceptado el 01/10/2018 y publicado el 22/11/2018.

---

[1] https://es.wikipedia.org/wiki/biopsia#biopsia_por_perforacion
[2] https://isic-archive.com/
[3] https://www.dermnetnz.org/
[4] http://www.image-net.org/

1.3 Estado del arte

Durante los últimos años se incrementó notablemente la utilización de procesos automáticos para la clasificación de imágenes y la clasificación del melanoma no es la excepción.

Uno de los primeros trabajos corresponde a la utilización de la regla ABCD [11], dicha regla es una fórmula que corresponde a una combinación lineal de distintas características de una lesión pigmentada: “asimetría”, “bordes”, “color” y “diámetro”. Dependiendo el score que se obtiene, se clasifica a la lesión pigmentada.

Posteriormente se encuentra el método de Menzies [12], que posee un conjunto de características relacionadas con una lesión pigmentada benigna y otras características que están estrictamente relacionadas con un melanoma. Con lo cual, dependiendo de las características que se encuentren en la lesión, se la considerará como maligna o benigna.

Luego aparece la técnica de los 7 puntos (método Argenziano) [13] en la cual se evalúan 7 características fundamentales de las lesiones pigmentadas, colocando un score en cada una de ellas. A continuación se suman estos scores llegando a un score global, de forma que si supera un límite preestablecido se clasificará a la lesión pigmentada como melanoma.

Por otro lado, en lo que respecta a la segmentación de la lesión pigmentada, empezó a utilizarse principalmente el método Otsu [14]. Este es un procedimiento no paramétrico que segmenta la imagen mediante la selección de un umbral óptimo maximizando la varianza entre las clases con una búsqueda exhaustiva.

Dado estos 4 métodos (Regla ABCD, Menzies, Argenziano y Otsu), empiezan a desarrollarse diferentes sistemas automáticos de detección de melanomas como se puede observar en la tabla 1.1.

Investigación	**Fundamento**	**Segmentación**	**Clasificación**
H Iyatomi et al. 2005 [15]	Regla ABCD	Crecimiento de regiones	Redes Neuronales
G. Capdehourat et al. 2009 [16]	Regla ABCD y método Argenziano	Método de Otsu	Árboles de decisión
J. F. Alcón et al. 2009 [17]	Regla ABCD	Método de Otsu	Árboles de decisión fusionados con redes bayesianas
G. Leo et al. 2010 [18]	Método de Argenziano	Método de Otsu	Árboles de decisión
D. Ruiz et al. 2011[19]	Regla ABCD	Método de Otsu	Perceptrón Multicapa, clasificador bayesiano, KNN
G. Capdehourat et al. 2011 [20]	Soporte clínico	Método de Otsu	Adaboost de Árboles de decisión

Tabla 1.1: Antecedentes previos a la utilización de deep learning.

Como se observa en la tabla 1.1, se utilizaron las redes neuronales en un primer momento, para luego migrar hacia otras técnicas de minería de datos. No obstante, a partir del año 2013, con el incremento de la velocidad de procesamiento del hardware, se empiezan a utilizar nuevamente las redes neuronales pero esta vez con más capas. Luego surgen las redes convolucionales profundas, con una muy buena performance en la clasificación de imágenes, de forma que (en el momento que se escribe el presente trabajo) determinan el state-of-the-art. Además, se empezaron a publicar papers en forma independiente en lo que respecta a la clasificación y a la segmentación de una imagen. En la tabla 1.2 se listan algunos de los trabajos de principal importancia en este campo.

Investigación	**Tipo**	**Método**
Alan C. Bovik et al. 2013 [21]	Clasificación	SGNN (redes neuronales auto generativas) + Algoritmos genéticos (optimizan el número de muestras que observa la red)
O. Ronneberger, P. Fischer, and T. Brox et al. 2015 [22]	Segmentación	Convolución y UpSampling (Redes U-Net)
N. Codella, J. Cai, M. Abedini, R. Garnavi, A. Halpern, and J. R. Smith, et al. 2015 [23]	Clasificación	CNN, Sparse Coding y SVM como ensamblador
Demyanov, et al. 2016 [24]	Clasificación	CNN de 8 capas
J. Kawahara, et al. 2016 [25]	Clasificación	CNN pero convirtiendo las capas totalmente conectadas en capas convolucionales para extraer los descriptores locales
A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun et al. 2017 [26]	Clasificación	CNN - Inception V3
Zhen Yu et al. 2017 [27]	**Clasificación**	**Vectores de Fisher, CNN y SVM como clasificador**

Tabla 1.2: Antecedentes utilizando deep learning.

1.4 Objetivos de la tesis

- Se espera obtener alguna relación entre el mapa de calor y la correcta clasificación de la lesión pigmentada. A modo de ejemplo en la figura 1.10 se muestra la salida de la clasificación de una lesión pigmentada con su correspondiente mapa de calor. Previo a la clasificación, a dicha lesión pigmentada, se le aplicó el método Max-RGB y fue segmentada.

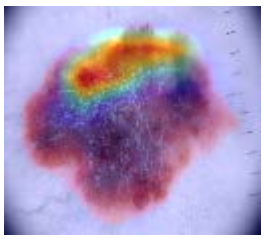


Figura 1.10: Lesión pigmentada segmentada, con escala de grises fuera de la lesión y con un mapa de calor que “explica” la predicción.

- Los modelos quedarán disponibles para que en un futuro puedan implementarse en un dispositivo móvil, lo cual permitirá a los pacientes poder sacarse fotos y obtener la clasificación de las lesiones pigmentadas previa visita al dermatólogo. Esto último incrementará las posibilidades de encontrar una lesión pigmentada maligna en su estadío temprano.

- Con la construcción del Modelo Híbrido, se espera encontrar que tenga una mayor performance en la clasificación de los lesiones pigmentadas en alguno de los conjuntos de datos de prueba, en relación a una CNN convencional (VGG-16).

- Confirmar el uso de los FVs cuando se tiene un conjunto de datos con relativamente pocas imágenes, para aumentar la performance en la clasificación. De esta forma, se podrá acoplar el uso de los FVs en las CNNs para obtener un clasificador con un mayor grado de poder predictivo.

1.5 Organización de la tesis

La tesis consta de 3 fases principales: **El Preprocesamiento, la Segmentación y la Clasificación** de las imágenes (todas ellas explicadas en la sección 3: “Conjunto de datos y métodos”). Además se explicará la teoría que avala la aplicación de las técnicas en cada una de las mencionadas fases (en la sección 2: “Marco teórico”).

Por último se encuentran **los Resultados** (en la sección 4: “Resultados”) de los modelos construidos y **las Conclusiones** (en la sección 5: “Conclusiones y trabajos futuros”) obtenidas en base a los resultados.

A continuación se describe brevemente y a nivel macro las 3 fases nombradas anteriormente (preprocesamiento, segmentación y clasificación) y las secciones respectivas en las cuales se explican los detalles.

Preprocesamiento

En esta fase se obtienen las imágenes de las lesiones pigmentadas de los sitios Web Dermnet e ISIC por medio de Web scraping. Además, en el caso del sitio ISIC, se leyó el código JSON asociado para recuperar solo las imágenes dermatoscópicas y obtener la máxima cantidad de melanomas para equilibrar las clases (“Melanoma” y “No melanoma”). Luego, estas imágenes conjuntamente con las que se recibieron del Hospital Italiano, fueron redimensionadas para que sean utilizadas en los distintos modelos construidos. Además se incorporaron imágenes sintéticas (data augmentation). Por último se aplicó el algoritmo de Max-RGB para minimizar los efectos producidos por el iluminante. Para más detalle ver la sección 2.3 (Preprocesamiento) que explica la teoría del algoritmo Max-RGB y la sección 3.2 (Conjuntos de datos) que explica el Preprocesamiento mencionado e indicado en la figura 1.11.

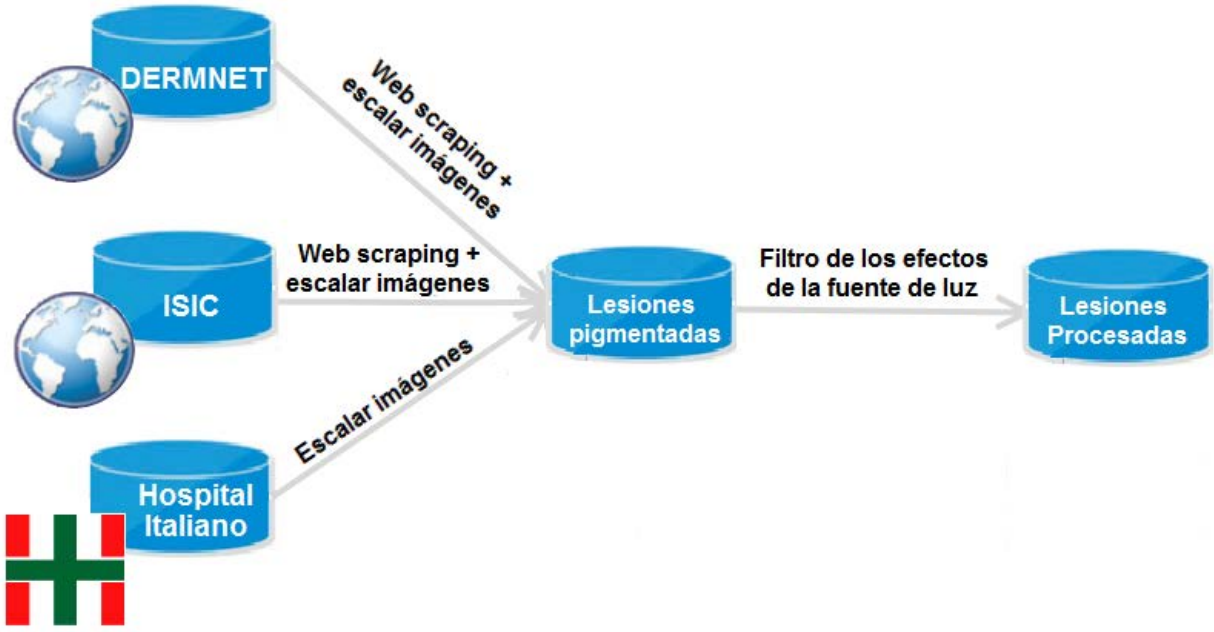


Figura 1.11: Preprocesamiento de las lesiones pigmentadas.

Segmentación

En esta fase, se realizó el entrenamiento de una red custom que sigue la filosofía de la red U-Net con el objetivo de segmentar las imágenes. En la figura 1.12 se observa, a nivel macro, dicha red. Para más detalles ver la sección 2.4 (Segmentación) que explica desde la teoría de los autoencoders hasta la arquitectura que se utilizó en el presente trabajo y la sección 3.3.1 (Segmentador) que explica cómo se realizó el entrenamiento de la red.

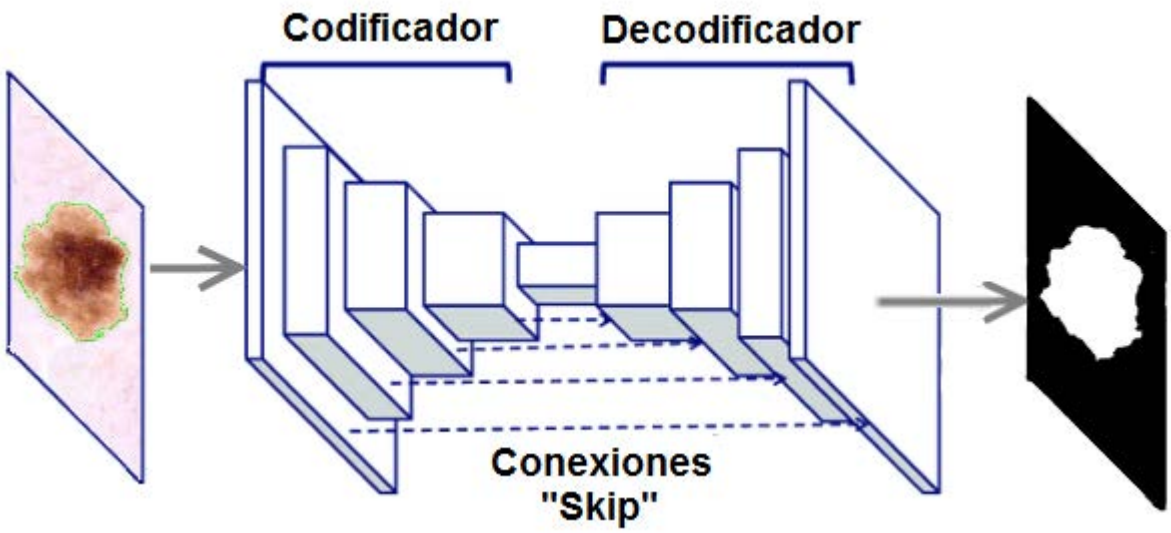


Figura 1.12: Modelo utilizado para segmentar los lesiones pigmentadas.

Las imágenes segmentadas se utilizaron para el entrenamiento de un modelo llamado “Clasificador Auxiliar” explicado a continuación, en el apartado de clasificación. En la figura 1.13 se observa que se aplicó una escala de grises en la piel que se encuentra fuera del área de la lesión pigmentada con el objetivo de no perder el contexto. No obstante, se deja en color la lesión para que exista mayor nivel de información en dicha área. Se decidió realizar este tipo de operación ya que al entrenar el clasificador mencionado con las imágenes segmentadas (sin contexto) se obtuvieron peores resultados.

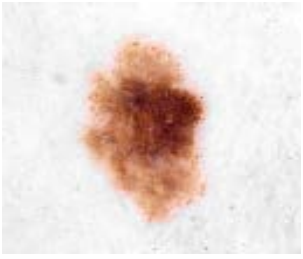


Figura 1.13: Lesión segmentada con escala de grises fuera de la lesión.

Clasificación

En esta fase se entrenan los modelos de clasificación con el objetivo de clasificar una lesión pigmentada en “Melanoma” o “No melanoma”. Dichos modelos de clasificación son:

- **Clasificador 1**: Este clasificador corresponde a la arquitectura VGG-16. Para más detalles ver la sección 2.5.1 (Arquitectura VGG-16) donde explica las diferentes capas que tiene este modelo y cómo fue pre-entrenado con las imágenes de ImageNet. Por otro lado, la sección 3.3.2 (Clasificador 1) explica cómo se realizó el entrenamiento con las imágenes de las lesiones pigmentadas.

- **Clasificador 2**: Este modelo corresponde a un Modelo Híbrido, donde los descriptores locales son extraídos de la última capa de pooling del Clasificador 1 para luego codificarlos en los vectores de Fisher. Se aclara que para realizar dicha codificación, se aplica el análisis de componentes principales (PCA) y un modelo de mezclas de gaussianas (GMM). Luego, los vectores de Fisher son la entrada de una máquina de vectores de soporte (SVM) para realizar la clasificación de la lesión pigmentada. La teoría que explica las técnicas utilizadas para codificar los descriptores locales en los vectores de Fisher y el funcionamiento del SVM se encuentra en la sección 2.5.3 (Modelo Híbrido). Mientras que la explicación del entrenamiento de este modelo con las imágenes de las lesiones pigmentadas se encuentra en la sección 3.3.3 (Clasificador 2 - Modelo Híbrido sin segmentación).

- **Clasificador 3**: Este clasificador codifica los descriptores locales de la misma forma que el Clasificador 2. La diferencia radica que los descriptores locales provienen de la última capa de pooling del Clasificador Auxiliar (en lugar del Clasificador 1). Este Clasificador Auxiliar tiene la misma arquitectura que el Clasificador 1, es decir la VGG-16, no obstante fue entrenado con las imágenes de las lesiones pigmentadas segmentadas. Dado que la arquitectura de este Modelo Híbrido es la misma que la del Modelo Híbrido que involucra el Clasificador 2, la explicación teórica del mismo corresponde a la misma sección que la referenciada en el Clasificador 2, es decir la 2.5.3 (Modelo Híbrido). No obstante, la explicación del entrenamiento de este modelo con las imágenes de las lesiones pigmentadas segmentadas se encuentra en la sección 3.3.4 (Clasificador 3 - Modelo Híbrido con segmentación).

Lo anteriormente explicado se puede observar en la figura 1.14. Además, en dicha figura se visualiza que el esquema básico de clasificación de una imagen corresponde a: extracción de características, representación de la imagen y clasificación. Con lo cual, en el presente trabajo, la extracción de características se obtiene por medio del Clasificador 1 y el Clasificador Auxiliar. La representación de la imagen corresponde al aplanado de los descriptores (que se realiza en el Clasificador 1 y en el clasificador Auxiliar) y a la codificación de dichos descriptores en los vectores de Fisher (llevada a cabo por los Clasificadores 2 y 3). Por último, en lo que respecta a la clasificación, el Clasificador 1 tiene una red totalmente conectada y los Clasificadores 2 y 3 tienen un SVM.

Además, como se mencionó en la sección 1.2, el entrenamiento de los Clasificadores 1 y Auxiliar fue evaluado por medio del método GRAD-CAM. Para más detalles, ver la sección 2.5.2 (Mapa de calor de la red - GRAD-CAM) que explica la teoría de cómo se aplicó el método y la sección 3.3.2 (Clasificador 1) que muestra ejemplos de los mapas de calor en distintas lesiones pigmentadas.

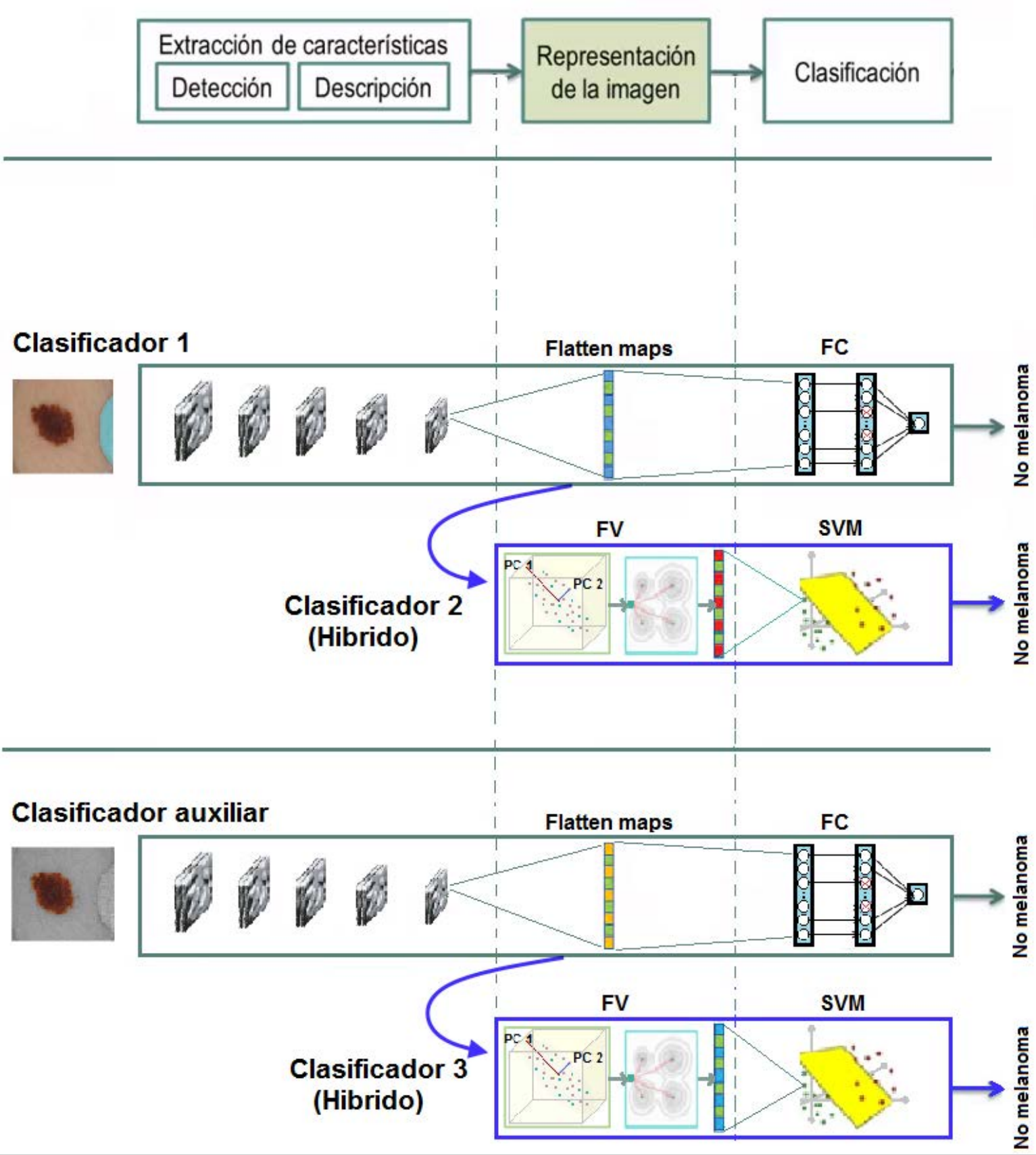


Figura 1.14: Modelos empleados para la clasificación de una lesión pigmentada.

Ensamble

El Ensamble consiste en el promedio de las probabilidades (predicciones) de los 3 clasificadores explicados anteriormente (Clasificador 1, Clasificador 2 y Clasificador 3). En la figura 1.15 se observa este promedio aritmético de probabilidades. Para más detalles ver la sección 3.3.5 (Ensamble).

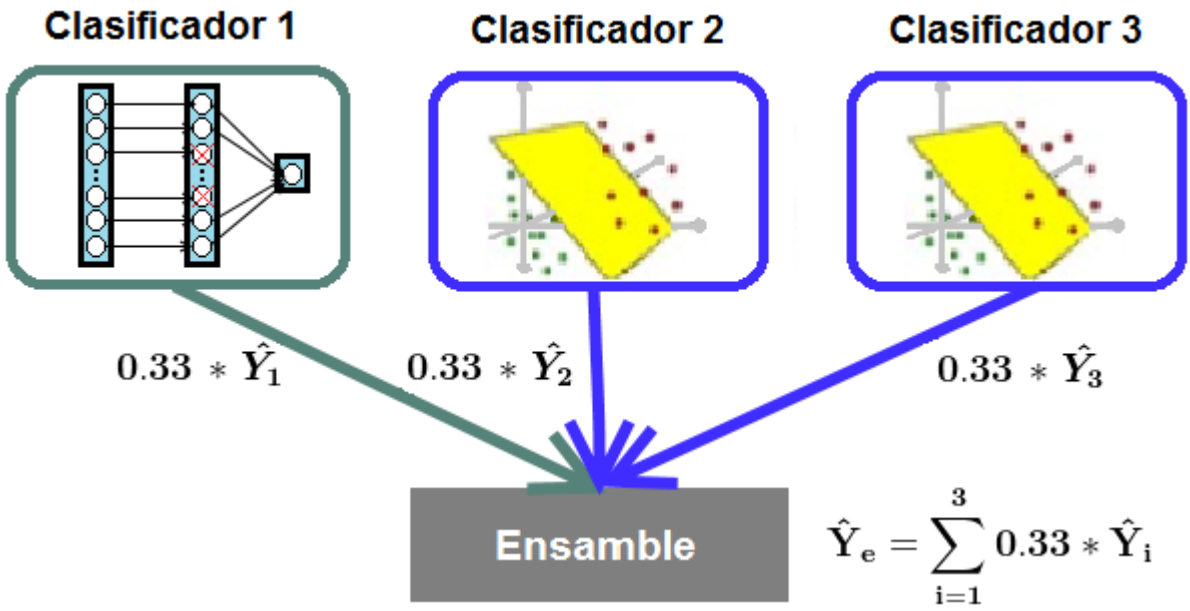


Figura 1.15: Ensamble utilizado para la clasificación final de una lesión pigmentada.

1.6 Entorno

Hardware

RAM: 16GB.
CPU: Intel Core i7-6700 HQ @2.60GHZ.
GPU: Nvidia GeForce 950M - con 4GB de memoria dedicada.
SO: Windows 10, 64 bits.

Software

En lo que respecta al **Preprocesamiento** se utilizaron los lenguajes **Julia** (para el Web scraping) y **python 3.6** (para el algoritmo Max-RGB , manipulación y redimensionamiento de imágenes). Se aclara que para hacer scraping en el sitio ISIC se utilizaron ciertas API's[5].
En la **Segmentación y Clasificación** se utilizó el lenguaje **Python 3.6** con las siguientes librerías usadas frecuentemente en todo el proyecto:

- Numpy (usada para manejar en forma eficiente las operaciones entre tensores).
- Scipy (usada para cálculos matemáticos).
- TensorFlow (usada para la creación de redes convolucionales).
- Tensorboard (usada para la visualización de ciertas métricas de las capas intermedias en la red convolucional).
- Keras (API de alto nivel para manejar el ecosistema de TensorFlow. Utilizada para la generación de datos sintéticos y la creación de redes convolucionales).
- Matplotlib (usada para la impresión de las curvas).
- Sklearn (usada para la aplicación de ciertas métricas con la finalidad de analizar las imágenes, para realizar PCA y para la creación del SVM).
- Python Imaging Library y OpenCV (usadas para la manipulación de las imágenes y la creación del modelo de mezclas de gaussianas (GMM)).
- Itertools, zipfile, csv, random, os, urllib, sys (otras tareas).

Se aclara que el entrenamiento de las redes convolucionales utilizando la CPU era demasiado lento, con lo cual se hizo uso de la GPU (Graphics Processing Unit) para aprovechar el procesamiento en paralelo por medio del toolkit CUDA[6] y la librería cuDNN[7].

[5] https://isic-archive.com/api/v1
[6] https://developer.nvidia.com/cuda-toolkit
[7] https://developer.nvidia.com/cudnn

2 Marco teórico

2.1 Redes neuronales biológicas

Para una computadora el reconocer imágenes es una tarea sumamente difícil, lo que en el caso de los humanos es algo relativamente sencillo. Esto se debe a que los sistemas biológicos poseen una arquitectura distinta a la de una computadora moderna. Por tal motivo, se trata de simular algunas características del cerebro humano para elaborar nuevos modelos de cómputo [28].

Las redes neuronales artificiales pertenecen a una rama de los algoritmos de aprendizaje automático y se basan en el funcionamiento del cerebro humano. Aunque se está lejos de comprender el funcionamiento y la estructura del sistema nervioso, se conoce con cierto detalle la estructura de la neurona como elemento básico del tejido nervioso y la forma de cómo se estructura la corteza cerebral.

La neurona, como toda célula, consta de una membrana exterior (M), que la limita y le sirve de órgano de intercambio con el medio exterior, de un citoplasma (C), que es el cuerpo principal de la célula donde radica el grueso de sus funciones y de un núcleo (N), que contiene el material genético de la célula. El citoplasma presenta unos alargamientos (D), llamados dendritas, que son órganos de recepción. En las dendritas terminan un gran número de fibras (F) que son conductores que llevan la señal o impulso nervioso de los receptores hacia la neurona. Estas fibras terminan en un pequeño corpúsculo llamado sinapsis, que constituye un relevador bioquímico y que sirve para transferir la respuesta de la neurona (señal de salida) hacia otra neurona. Esta señal de salida es conducida por una prolongación cilíndrica alargada de la neurona, que se llama cilindro-eje o axón (A), que en su extremo se divide en varias fibras para comunicarse con otras neuronas o con órganos efectores o motores como pueden ser glándulas o músculos. Lo anteriormente explicado puede visualizarse en la figura 2.1 [29].

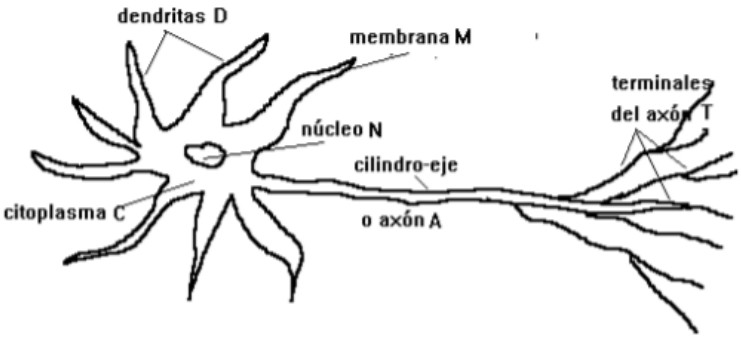


Figura 2.1: Neurona biológica.

2.2 Redes neuronales artificiales

En el caso de las neuronas artificiales, se utiliza un modelo matemático para emular en cierta medida el funcionamiento de las neuronas biológicas. En la figura 2.2 se observa que el impulso nervioso de entrada x_i es multiplicado por una matriz de pesos (aprendible en el entrenamiento) que representa la estructura sináptica de la neurona. El segundo término que el modelo va a aprender durante el entrenamiento es el “sesgo” , el cual se suma a la multiplicación matricial previamente mencionada. Este modelo matemático tendrá como salida una señal de acuerdo a una función de activación f. En el caso particular que la función de activación es del tipo escalón, a esta neurona artificial se la denomina perceptrón. Si se considera que una neurona puede recibir múltiples impulsos N, tiene unos pesos aprendibles BICw_i y un sesgo b, entonces la salida de este modelo estará dada por la siguiente ecuación [30]:

$$y(x,w) = f(\sum_{i=0}^{2} x_i\, w_i \; + b) \tag{2.1}$$

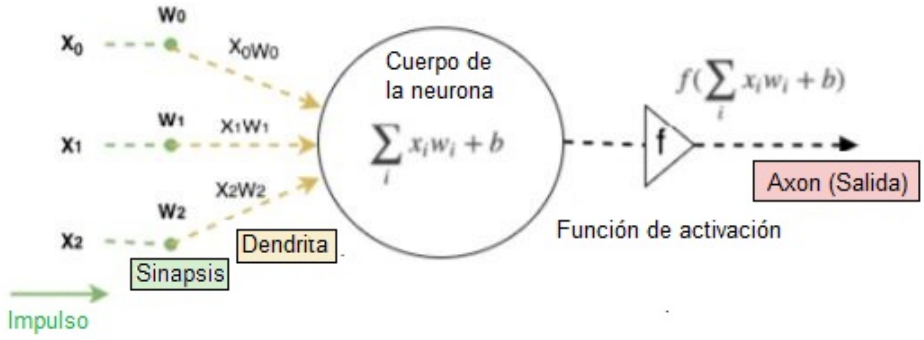


Figura 2.2: Neurona artificial mediante un modelo matemático.

Una red neuronal artificial puede tener millones de neuronas y una de las organizaciones de capas más utilizadas es la siguiente:

- Una capa de entrada: Corresponde a la entrada del modelo.
- Una o más capas ocultas: Conectada a la capa de entrada (o a otra capa oculta en caso de tener más de una capa oculta).
- Una capa de salida: Conectada a la capa oculta.

Este tipo de organizaciones se denomina feedforward. En particular si cualquier neurona ubicada en una capa N tiene como entrada las salidas de las neuronas ubicadas en la capa N-1, y la salida de esta neurona constituye las entradas de las neuronas ubicadas en la capa N+1, diremos que esta arquitectura del tipo feedforward se encuentra totalmente conectada. Si a esta última organización le colocamos la restricción que la función de activación sea del tipo no lineal, entonces la arquitectura resultante se denomina multiperceptrón (MP) y se puede visualizar en la figura 2.3 [30].

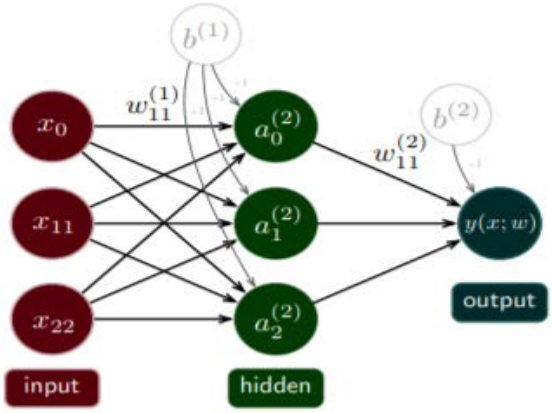


Figura 2.3: Multiperceptrón con 1 capa de entrada, 1 capa oculta y 1 capa de salida.

En la figura 2.3, la activación $a_0^{(2)}$ corresponde a la activación de la sumatoria de las entradas x_i ponderadas por los pesos asociados a dichas entradas. Es decir, se sigue la ecuación 2.1 en la cual, para este caso, $N=3$ ya que existen 3 conexiones a cada neurona de la capa oculta. Se vuelve a aplicar este proceso para las restantes 2 neuronas de la capa oculta, obteniéndose el vector de salidas $a^{(2)}$. Luego, se vuelve a aplicar la ecuación 2.1 en donde el vector de entradas corresponde a $a^{(2)}$, los pesos a $w^{(2)}$ y el bias a $b^{(2)}$. El resultado de esta última operación corresponde a la salida de la red [30].

Con respecto a las funciones de activación que posee cada neurona, resulta evidente que si tenemos una organización feedforward pero con una o más capas ocultas (ver figura 2.4) , entonces la función de activación deberá ser no lineal. Ya que si todas las neuronas de las capas ocultas tienen una activación lineal, esa misma red es equivalente a una red sin capas ocultas [30]. Es decir, la red resultante tendrá una sola capa donde la capa de entrada estará conectada a la de salida, teniendo una matriz de pesos y un sesgo que serán aprendidos durante el entrenamiento. O sea, si suponemos una red de K capas y no tenemos en cuenta el sesgo para que la explicación resulte más sencilla, entonces la salida de la primera capa (un vector) es una transformación lineal de la entrada por los pesos respectivos de esta capa. Esta salida será la entrada de la siguiente capa (que a su vez la multiplicará por sus pesos y le aplicará otra transformación lineal), y así sucesivamente hasta llegar a la K capa. Con lo cual, **después de K transformaciones lineales concatenadas**, la salida de esta capa K puede pensarse como una nueva transformación lineal (la cual llamamos G) de la entrada de dicha red multiplicada por otros pesos aprendibles durante el entrenamiento. **Donde G y los pesos de esta nueva capa resumen las transformaciones efectuadas por las K capas de la red**. Por ende, tener solo funciones de activación lineal le sacaría potencia a la red y la posibilidad de aproximar funciones de clasificación complejas.

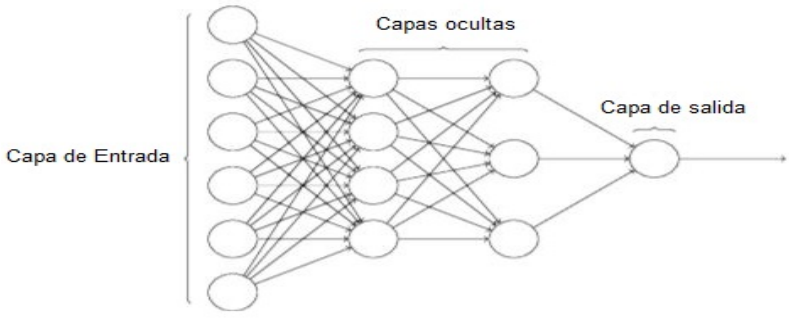
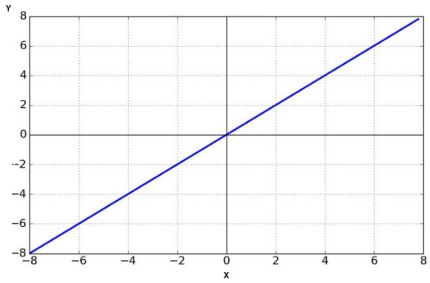


Figura 2.4: Red feedforward con 1 capa de entrada, 2 capas ocultas y una capa de salida.

2.2.1 Funciones de activación

Según lo dicho anteriormente, las funciones de activación deberían ser no lineales para aumentar el rendimiento de la red. No obstante, existen funciones de activación lineales que no son frecuentemente utilizadas, tal cual indica la figura 2.5. En dicha figura observamos la función identidad, con lo cual tener esta función de activación es equivalente a no tener una función de activación, ya que el valor de la entrada se traduce en el mismo valor de la salida [33].



$$y = x$$

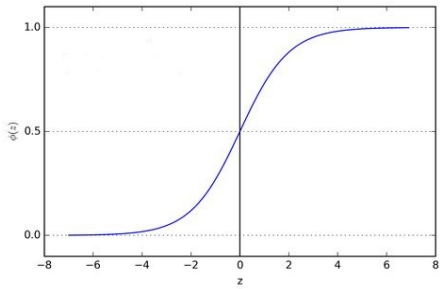
Figura 2.5: Función de activación lineal

Por otro lado, están las funciones no lineales donde resulta deseable que cumpla los siguientes requisitos:

- Función monótona: Esta propiedad garantiza que la superficie del error asociada a una sola capa del modelo sea convexa. Lo que permite una convergencia hacia el mínimo más fácilmente [31].
- Derivable: Esta propiedad se requiere para utilizar métodos de optimización como el del gradiente descendente [32].

Las funciones de activación no lineales más utilizadas son las siguientes:

Función Sigmoide o Logística

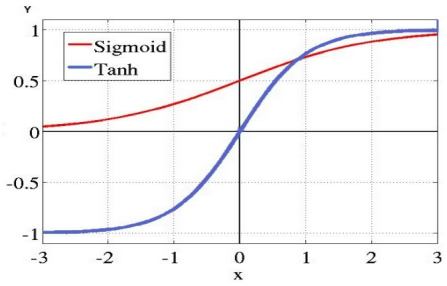


$$\sigma(z) = \frac{1}{1 + e^{-c\,z}} \qquad (2.2)$$

Figura 2.6: Función de activación Sigmoide (No lineal)

La función Sigmoide (ver figura 2.6) dispone del parámetro *C* (ver ecuación 2.2) que permite controlar el crecimiento de la función. La función es monótona y diferenciable. En lo que respecta a su utilización en las capas ocultas, debido a la saturación que produce esta función cuando la entrada tiene un valor absoluto grande (ya sea positivo o negativo) hace que el entrenamiento basado en el gradiente descendente sea dificultoso. Ya que su rango se encuentra en (0,1), se suele utilizar en la capa de salida cuando se requiere calcular la probabilidad asociada a una clasificación, siempre y cuando se utilice una función de pérdida que elimine el efecto de saturación previamente mencionado [33].

Función de activación Hiperbólica

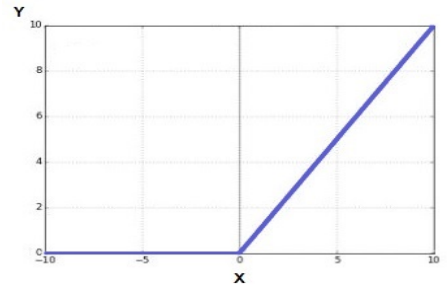


$$y = \frac{e^{zx} - e^{-zx}}{e^{zx} + e^{-zx}} \qquad (2.3)$$

Figura 2.7: Función de activación Tangencial Hiperbólica (No lineal).

Esta función se puede reescribir utilizando la función Sigmoide, con lo cual se encuentra íntimamente relacionada. A diferencia de la función Sigmoide, la función Tangente Hiperbólica (ver figura 2.7) tiene su rango en (-1,1) permitiendo mapear valores de entradas negativos a valores de salida de su mismo signo. Por otro lado, al igual que la función sigmoide dispone del parámetro Z (ver ecuación 2.3), el cual permite calcular la velocidad de crecimiento de la función. Generalmente donde funciona bien la función sigmoide, la función Hiperbólica funciona mejor [33].

Función ReLU (Unidad lineal rectificada)

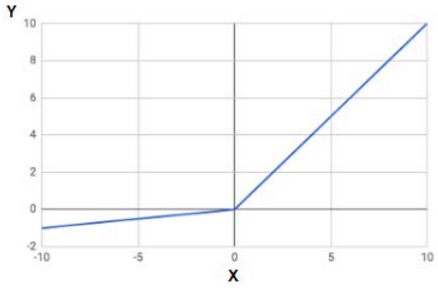


$$y = max(0, x) \qquad (2.4)$$

Figura 2.8: Función de activación ReLU (No lineal)

La función ReLU (Rectified Lineal Unit - ver figura 2.8) es la más empleada (en el momento que escribe el presente trabajo) ya qué se utiliza en casi todos los tipos de redes

convolucionales como así también en otro tipos de redes de aprendizaje profundo. Es derivable y es monótona (su derivada también es monótona). Una desventaja de la función ReLU es que los pesos podrían actualizarse de cierta forma en la que la neurona nunca se active, es decir, que la salida de esa neurona siempre sea cero y cuando se llega a esa condición durante el entrenamiento no se puede salir de ella [29]. No obstante, tiene ventajas importantes comparadas con la función Sigmoide o Tangencial Hiperbólica, como la aceleración en la convergencia de la red (mediante el gradiente descendente estocástico) debido a que no implica cálculos costosos en el entrenamiento [33]. Además no sufre del fenómeno de desaparición del gradiente (explicado en la sección 2.2.5). La desventaja que posee sobre la obtención de “neuronas muertas” (La no activación de la función ReLU) puede contrarrestarse con la utilización de una variante de esta función, la Leaky ReLU (ver figura 2.9). Esta función es una variación de la función ReLU, donde los valores de X menores a 0 se encuentran mapeados a una función $y = aX$ (con ***a*** igual a una pequeña constante y mayor a 0), en lugar de $y=0$ como vimos anteriormente (ver ecuación 2.4) [33].



$$y = max(k * x, x) \quad ; \quad \text{con } 0 < k < 1$$

Figura 2.9: Función de activación Leaky ReLU (No lineal)

2.2.2 Entrenamiento supervisado

Los pesos utilizados para calcular la salida de cada neurona son ajustados durante el entrenamiento de la red. Estos ajustes son “aprendidos” de forma tal que se satisface el mapeo $f : x \in X \mapsto y \in Y$, donde la entrada x (particularmente en este trabajo) corresponde a una imagen y se requiere una salida y. Para realizar dicho entrenamiento en un aprendizaje supervisado, le debemos proveer a la red un conjunto de n ejemplos y el valor verdadero (ground truth) y para cada uno de esos ejemplos, es decir, los n pares $(x_i,y_i)....(x_n,y_n)$. De esta forma, durante el entrenamiento, la función f se ajustará a la salida y. Dicha función, en el caso ideal generalizará para todos los valores de X, y no solo los valores de x_i que pertenecen al conjunto de datos de entrenamiento. Este último proceso involucra una optimización del objetivo según la ecuación 2.5 [34].

$$f^* \approx argmin_{f \in F} \left(\frac{1}{n} * \sum_{i=1}^{n} J(f(x_i), y_i) \right) \tag{2.5}$$

Donde la función J es la función de pérdida (función de loss) y nos indica que tan bueno es el rendimiento de la red, midiendo la distancia entre la predicción $f(x_i)$ y el ground truth y_i para una entrada x_i. Usando esta función de pérdida podemos ir variando los pesos y el bias de las

neuronas de la red en forma iterativa, hasta encontrar una configuración f^* donde la función de pérdida sea mínima.

A modo ilustrativo, podemos introducir a la función de pérdida como el error cuadrático medio (MSE) de la red, según la siguiente ecuación:

$$J(\theta) = \frac{1}{n} * \sum_{x=1}^{n} (f(x_i) - y_i)^2 \quad (2.6)$$

Donde θ representa a todos los pesos y bias de la red, $f(x_i)$ es la predicción e y_i es el ground truth (objetivo o clase) para cada valor de entrada x_i perteneciente al conjunto de datos de entrenamiento de tamaño n. De la ecuación 2.6 podemos observar que la función de pérdida se hace pequeña cuando $f(x_i)$ se aproxima a y_i. Con lo cual, si la función de pérdida se hace realmente pequeña, decimos que la red realiza un buen ajuste a los datos de entrenamiento y concluimos que la red converge.

Además del conjunto de entrenamiento, se debe tener un conjunto de validación y otro de prueba [35]. Los conjuntos de datos de prueba y validación deben ser significativos (a nivel número de ejemplos o instancias) y poseer una distribución de los datos similar al conjunto de datos de entrenamiento. Cada instancia de estos conjuntos de datos tiene asociada el ground thruth, ya que nos permitirá medir el rendimiento de la red. A medida que se va entrenando el modelo, su rendimiento se evalúa en los conjuntos de datos de validación y entrenamiento. Observando los resultados, es posible modificar los hiperparámetros para: mejorar el rendimiento de la red, prevenir el sobreajuste (overfitting) y prevenir el subajuste (underfitting). Los hiperparámetros son variables importantes en la configuración de la red neuronal que afectan aspectos críticos en el proceso de entrenamiento. Luego, una vez realizado el entrenamiento, se procede a evaluar la red neuronal en el conjunto de datos de prueba para verificar su rendimiento sobre datos “nuevos”. Generalmente un modelo bien entrenado tiene un bajo error de generalización, con lo cual se espera que generalice bien los ejemplos incluidos en el conjunto de datos de prueba. Sin embargo, no hay garantías de que una red converja o que la solución encontrada generalice bien. Una mala elección de los hiperparámetros o la elección de una arquitectura equivocada puede provocar problemas como el sobreajuste, el subajuste o simplemente que la red no sea capaz de aprender (ajustar correctamente los pesos y/o bias).

Sobreajuste (Overfitting)

En un conjunto de datos determinado, estará la señal y el ruido. Es decir, habrá una parte de los datos (la señal) que tienen una relación directa con el objetivo o clase del problema en estudio. Por otro lado, estará el ruido que son los datos que no poseen una relación con el objetivo o clase. Por ende, si el modelo aprende el ruido y la señal estará sobreajustando el conjunto de datos. Esto provocará que la función de pérdida sea pequeña en el conjunto de datos de entrenamiento pero inaceptablemente alta en el conjunto de datos de prueba, mostrando la incapacidad del modelo para realizar una buena generalización [36].

Subajuste (Underfitting)

En este caso, la función de mapeo $f : x \epsilon X \mapsto y \epsilon Y$ tiene una complejidad tal que nuestra red no es capaz de aproximarla correctamente (encontrar una función f^*) con el objetivo de reducir la función de pérdida a un valor aceptable. En este caso, contrariamente a lo que sucede con el caso de sobreajuste, el modelo tiene un alto sesgo y poca variabilidad [37].

Transfer learning y fine tuning

La transferencia del aprendizaje (transfer learning) es un método que permite mejorar el aprendizaje de una tarea por medio de la transferencia de conocimiento de otra tarea que ya ha sido aprendida previamente. El transfer learning es popular en el área de deep learning debido a la necesidad de mejorar el entrenamiento de las redes, el cual requiere una gran potencia de cómputo (y de tiempo) para llevarlo a cabo.

En deep learning, para realizar el transfer learning primero se entrena una red base con un conjunto de datos base con el objetivo de realizar una tarea base. Luego, las características aprendidas en esta tarea base son transferidas a una red objetivo que será entrenada con un conjunto de datos objetivo para realizar una tarea objetivo. El proceso anterior tiende a funcionar si las características que se transfieren son generales y no específicas de la tarea base. Este proceso de transferencia es llamado transferencia inductiva donde la cantidad de posibles modelos (espacio de las hipótesis) que se podrían obtener para realizar la tarea objetivo, es reducida mediante la transferencia de conocimiento obtenido mediante otro modelo entrenado para realizar otra tarea (ver figura 2.10) [38].

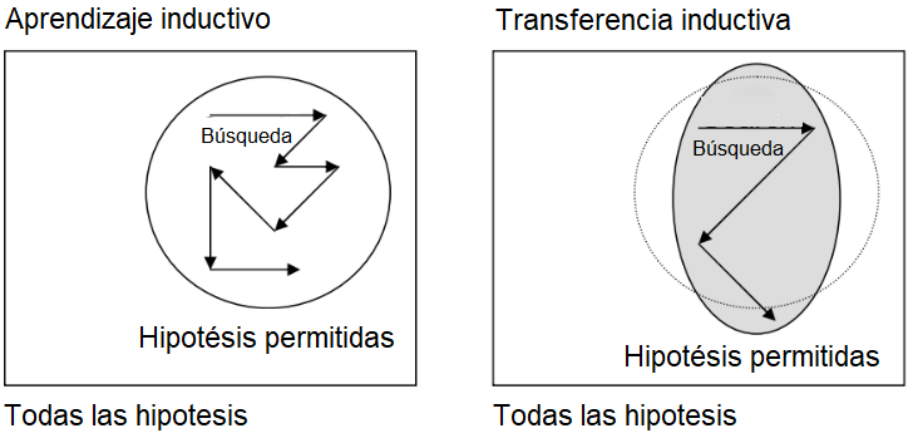


Figura 2.10: Reducción de hipótesis permitidas mediante la transferencia del conocimiento.

No resulta evidente conocer a priori si la aplicación de la transferencia de conocimiento va a producir una mejora en el entrenamiento del modelo hasta el momento que dicha transferencia no es efectuada y el modelo evaluado. No obstante, según Lisa Torrey and Jude Shavlik [39] existen 3 posibles beneficios (ver figura 2.11) que se pueden obtener con la transferencia del conocimiento:

- Alto comienzo: La transferencia de conocimiento permite que el punto inicial del entrenamiento de la red se encuentre más cerca al punto óptimo de performance en relación al punto de origen sin transferencia de conocimiento.

- Alta pendiente: El ratio de mejora del skill (valores altos de rendimiento) durante el entrenamiento es más rápido con la transferencia de conocimiento.

- Alta Asíntota: La convergencia del modelo con transferencia de conocimiento se realiza más rápidamente.

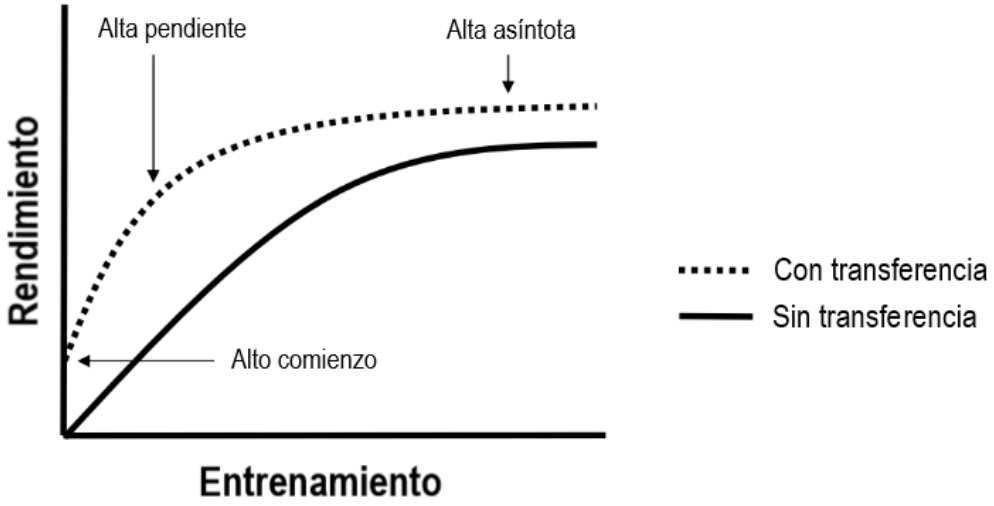


Figura 2.11: Beneficios importantes con respecto al entrenamiento con transferencia de conocimiento.

El transfer learning es ampliamente utilizado en deep learning en lo que respecta a redes que procesan lenguaje natural o imágenes. En la presente tesis se utilizan imágenes con lo cual nos abocaremos a la transferencia de conocimiento en este tipo de redes.

Como se explicará en la sección 2.2.5, las redes convolucionales son ampliamente utilizadas cuando se quieren clasificar imágenes y este tipo de redes son las que se utilizan en el presente trabajo. No obstante, muy poca gente entrena una red convolucional con inicialización de pesos aleatorios ya que se necesita un conjunto de datos de entrenamiento suficientemente grande y que no es sencillo de encontrar. Con lo cual, resulta habitual utilizar una red convolucional pre-entrenada en un conjunto de datos suficientemente grande (por ejemplo ImageNet) para luego utilizarla como inicializadora de pesos o como extractora de características que serán útiles para la tarea objetivo. Es decir, generalmente se emplean los siguientes 2 escenarios posibles para realizar la transferencia de conocimiento [40] :

Red convolucional como extractora de características: Se toma una red convolucional pre-entrenada y se remueven las últimas capas totalmente conectadas. Luego por cada imagen del conjunto de datos de entrenamiento que se pasa por esta red, se obtendrá un vector de características. En el caso particular de la red VGG-16 (explicada en la sección 2.5.1) corresponde a un vector de 25088 dimensiones. Estos vectores son denominados “códigos CNN”, “descriptores” o “deep features”. Luego, se entrenará un clasificador (teniendo como entrada a estos descriptores) como puede ser un SVM (explicado en la sección 2.3.3.6).

Fine tuning (sintonía fina) de la red convolucional: En este caso, además de entrenar el clasificador con los "códigos CNN" (descriptores) se realiza la sintonía fina de los pesos de la red pre-entrenada (por back propagation). Se puede realizar la sintonía fina de todos los pesos de la red o se pueden mantener fijos los de las primeras capas y solo hacer la sintonía fina de los pesos en las capas más profundas. Esto es debido a que los mapas de características de las primeras capas de la red convolucional contienen características generales y que pueden ser útiles para varias tareas (distintos conjuntos de datos). Mientras que los mapas de características de las capas más profundas corresponden a características específicas del conjunto de datos original y que difícilmente se puedan reutilizar en el nuevo conjunto de datos.

Se aclara que los modelos pre-entrenados se pueden obtener de dos diferentes formas. La primera corresponde a la obtención de un conjunto de datos de suficiente tamaño para poder entrenar una red convolucional que servirá como modelo pre-entrenado para poder realizar la transferencia de conocimiento a otra red convolucional. Sin embargo, si se toma como conjunto de datos a ImageNet, las redes convolucionales modernas pueden tardar 2 o 3 semanas de entrenamiento con varias GPUs en paralelo. Con lo cual, surge la segunda alternativa que corresponde a la utilización de modelos pre-entrenados y que se encuentran liberados en determinados sitios web[8] para que se puedan utilizar directamente sin necesidad de incurrir en los "costosos" entrenamientos.

Dependiendo del tamaño del conjunto de datos que se dispone y la similitud con respecto al conjunto de datos de origen (con el cual se obtuvo el modelo pre-entrenado), se plantean 4 reglas heurísticas [40] a tener en cuenta para determinar qué tipo de transferencia de conocimiento conviene utilizar:

1) El nuevo conjunto de datos es pequeño y similar al conjunto de datos original: Debido que son similares , las características específicas en las capas profundas obtenidas con el conjunto de datos origen van a ser relevantes para el nuevo conjunto de datos. Con lo cual, dado que se tiene pocos datos, no es conveniente hacer fine tuning (por problemas de sobreajuste) y se recomienda entrenar directamente un clasificador con los "códigos CNN" (descriptores) obtenidos del modelo pre-entrenado.
2) El nuevo conjunto de datos es grande y similar al conjunto de datos original: Dado que ahora se dispone de más datos con respecto al caso anterior, tendremos menos riesgo de hacer sobreajuste y por ende podemos hacer sintonía fina de todos los pesos de la red mediante backpropagation con el nuevo conjunto de datos.
3) El nuevo conjunto de datos es pequeño y muy distinto al conjunto de datos original: Como el conjunto de datos es pequeño, se recomienda entrenar un clasificador con los "códigos CNN". No obstante, debido que el conjunto de datos es muy distinto al original, es recomendable utilizar los "códigos CNN" (descriptores) que posean características generales (y que se encuentran en las primeras capas de la red convolucional pre-entrenada).
4) El nuevo conjunto de datos es grande y muy distinto al conjunto de datos original: En este caso, no sería necesario la transferencia de conocimiento. Sin embargo, en la práctica resulta habitual utilizar un modelo pre-entrenado y realizar sintonía fina en todos los pesos de la red.

[8] https://github.com/BVLC/caffe/wiki/Model-Zoo

En el presente trabajo y considerando que el tamaño del conjunto de datos que disponemos es mediano [41], nos situamos entre el escenario 3 y 4. Con lo cual, como se explicará en la sección 3.3, se realizó sintonía fina en el último bloque de la arquitectura VGG-16 conjuntamente con las capas totalmente conectadas (el clasificador).

Por último, se menciona que habitualmente se utiliza un ratio de aprendizaje o tasa de aprendizaje (learning rate) pequeña para realizar la sintonía fina de los pesos de la red convolucional. Debe ser pequeña, en comparación a la utilizada en un entrenamiento sin transferencia de conocimiento. Esto es porque se espera que los pesos de la red convolucional ya se encuentren relativamente calibrados y por ende no se desea distorsionarlos demasiado (que podría suceder con una tasa de aprendizaje relativamente alta) [40]. El concepto de tasa de aprendizaje será explicado en la sección 2.2.4.

2.2.3 Regularización

Cuando buscamos la función $f^* \epsilon \mathscr{F}$ en un espacio de hipótesis $\mathfrak{H}$, tal que se aproxime a la función $f : x \epsilon X \mapsto y \epsilon Y$, puede suceder que existan funciones f^* tal que la función de pérdida sea pequeña en el conjunto de entrenamiento pero grande en el conjunto de prueba. O sea, la función f^* no está generalizando en forma correcta y según vimos anteriormente, estamos sobreajustando el modelo (sobreajuste). Para evitar este tipo de situaciones, surge el término de regularización, en el cual se restringe el espacio de hipótesis a determinadas funciones f^* tal que reduzcan o eviten el sobreajuste del modelo. Siguiendo el principio de parsimonia o la navaja de Ockham en los cuales los modelos simples podrán generalizar mejor que los complejos, se trata de restringir el espacio de búsqueda a funciones de baja complejidad. Como se verá a continuación, la regularización involucra distintas estrategias (L^1 y L^2) en la cual se minimiza el sobreajuste, se reduce el error en el conjunto de datos de prueba y, prácticamente, no se altera el error cometido en el conjunto de datos de entrenamiento. Para lograr lo anteriormente mencionado, se colocará un término de regularización a la función de pérdida dando lugar a:

$$f^* \approx argmin_{f \epsilon F} \left(\frac{1}{n} * \sum_{i=1}^{n} J(f(x_i), y_i) \; + \; \alpha \, \Omega(f) \right)$$

La función $\Omega(f)$ se denomina regularización o suavidad funcional, mientras que $\alpha \epsilon [0, \infty)$ es el parámetro de peso que escala el efecto del término de regulación. En el ámbito de las redes neuronales, la función Ω se encuentra en función de los pesos w_i. Típicamente cuando se produce sobreajuste existen valores w_i muy grandes, con lo cual se debe generar una función $\Omega(w)$ en la cual penalice este tipo de casuísticas [42]. Surgen así las siguientes 2 estrategias de regularización:

L^1 Regularización

En este caso se propone una término de regularización indicado en la siguiente ecuación:

$$\Omega(w) = \sum_{i=1}^{n} |w_i|$$

Esta función incluida como término de regularización en la función de pérdida, induce que los pesos sean dispersos (sparsity). Es decir, los pesos de la red serán 0 en su mayoría y los restantes (distintos de cero) van a tener vital importancia en la aproximación de la función f^* a la verdadera función generadora de los datos [43][44].

L^2 Regularización

Se la conoce también con el nombre “weight decay” o “regularización de Tikhonov”. En este caso se propone un término de regularización según indica la siguiente ecuación:

$$\Omega(w) = \sum_{i=1}^{n} w_i^2$$

Con lo cual para grandes pesos de la red, se castigará a la función de pérdida ya que aumentará su valor. Por ende, este tipo de regularización tenderá a encontrar modelos con una gran cantidad de pesos con valores pequeños, en lugar de pocos pesos pero con grandes valores [43][44].

Luego existen otras formas de regularizar el modelo pero que no impactan en forma directa (mediante el agregado de una función $\Omega(w)$) a la función de pérdida. Entre ellos se encuentran el dropout y el aumento de datos, los cuales se explican a continuación.

Aumento de datos (Data augmentation)

Goodfellow [45] menciona “Regularización es cualquier modificación que realicemos al algoritmo de aprendizaje con el objetivo de reducir su error de generalización pero no su error de entrenamiento”. Por esta razón y teniendo en cuenta que el aumento de datos tiende a disminuir la varianza y el sobreajuste, podemos atribuir a esta técnica como un método de regularización. El objetivo es encontrar nuevos patrones (x_i, y_i) en el conjunto de datos de entrenamiento (que generalmente se encuentra con un número limitado de instancias). Cuando se trabaja con imágenes en un entrenamiento supervisado, las operaciones como la traslación, rotación y recortes aleatorios aplicadas al conjunto de datos de entrenamiento constituyen un tipo de data augmentation y ayudan a mejorar la generalización del modelo.

Dropout

Esta técnica permite realizar una regularización muy poderosa y poco costosa (en términos de la potencia de cómputo que requiere). La idea del dropout es excluir ciertos nodos (neuronas) de la red neuronal por cada iteración que se realiza en el entrenamiento (ver figura 2.12). Para realizar dicha exclusión se genera un número aleatorio entre 0 y 1 (una probabilidad) en cada neurona y se lo compara con un hiperparámetro p, el cual es la probabilidad de que un nodo no quede excluido. Si la probabilidad generada es mayor a p,

entonces la neurona es excluida. Al aplicar el algoritmo de backpropagation, esta exclusión se realiza en el “forward”, de modo que cuando se realiza el “backward”, solo se actualizarán los pesos de los nodos que no fueron excluidos. Al excluir un nodo se excluyen todos las conexiones que derivan del mismo [29]. Por ende, por cada iteración se tendrá una subred de nodos, generándose tantas subredes como iteraciones se produzcan en el entrenamiento. Se debe tener en consideración que se comparten parámetros (pesos) entre los distintos modelos generados en las iteraciones, ya que cada subred comparte ciertos parámetros de su modelo padre. Esto último permite crear una gran cantidad de modelos con una menor potencia de cómputo en relación a la que necesitaríamos con un método de ensamble con generación de modelos independientes. Una vez terminadas las iteraciones del entrenamiento, para obtener la predicción de un ejemplo nuevo, se podría realizar un ensamble en el cual se calcula la media geométrica de las salidas de las distintas redes que se generaron en el entrenamiento. No obstante, este enfoque resulta ineficiente dada la gran cantidad de subredes. Por ende, aquí se produce la clave de este método en la cual no es necesario calcular las predicciones de las subredes generadas, sino que se emplea la “regla de inferencia de peso re-escalado”. Con este enfoque, se utiliza la “red base”, es decir, la red con todos los nodos (sin haber aplicado dropout) pero los pesos son re-escalados por el umbral p [46]. Este re-escalado permite capturar el valor esperado de la salida de un nodo en relación al que tenía en la subred. Si bien, lo anteriormente explicado aplica a una red feedforward, esta técnica también se aplica en redes neuronales recurrentes y en las máquinas de Boltzmann.

Debido que el entrenamiento se realiza con distintas subredes, el dropout hace que la red resultante sea menos dependiente o sensible de una neurona o a un bloque de neuronas que tiene relaciones complejas. Esto último conlleva a la generación de sobreajuste por co-adaptación del bloque de neuronas y por ende es lo que el dropout trata de evitar. No obstante cabe aclarar que la coadaptación existe en la red pero solo cuando verdaderamente es necesaria que ocurra para incrementar la performance del modelo [46].

Por último, en lo que respecta a las redes convolucionales (explicadas en la sección 2.2.5), si se incorpora dropout en las capas más profundas, se incrementa la generalización de la red por agregar ruido a los mapas de características. Esto último hace que dichos mapas sean más robustos a las variaciones que pueden existir entre las distintas imágenes [47].

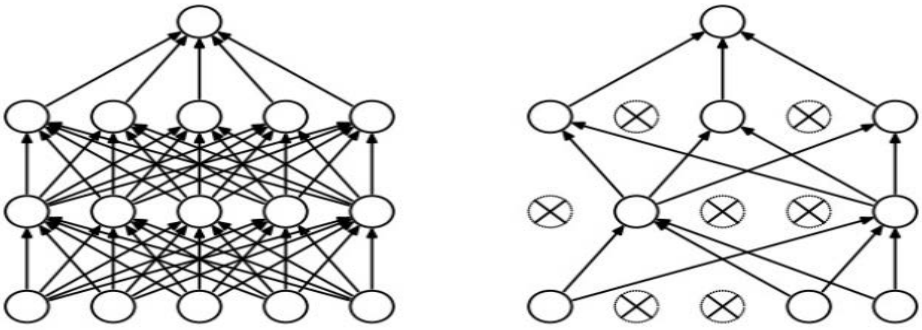


Figura 2.12: (Izquierda) Red neuronal estándar antes del dropout.(Derecha) Red neuronal después del dropout.

2.2.4 Optimización

En términos generales, el proceso de aprendizaje de una red consiste en encontrar el mínimo de una función de pérdida. Con lo cual, el aprendizaje se convierte en un problema de optimización. A continuación se describe el algoritmo de optimización del gradiente descendente.

Gradiente descendente

El objetivo principal es encontrar los pesos en cada nodo de la red en los cuales la función de pérdida se haga mínima. Esto puede ser resuelto variando los pesos de la red en la dirección de la pendiente (negativa) de la función de pérdida. Lo anteriormente mencionado se resuelve con las derivadas de la función de pérdida con respecto a cada peso (derivadas parciales), las cuales constituyen el gradiente de la función de pérdida con respecto a sus pesos. Para resolver este tipo de gradiente, existe una solución que consiste en la "aproximación numérica" del gradiente. Además existe otra solución que se denomina "analítica" la cual aplica directamente el gradiente mediante una fórmula hallada a través del cálculo numérico. Con lo cual se puede emplear:

Gradiente numérico

Para problemas con pocas variables (pesos), es posible encontrar una solución numérica en base a la utilización de "diferencias finitas" para minimizar la función de pérdida. En el caso que la función de pérdida tenga una sola dimensión como entrada (θ), la misma estará definida en J: $\mathbb{R} \mapsto \mathbb{R}$, y su derivada con respecto a su entrada viene dada por la siguiente ecuación:

$$\frac{\partial J(\theta)}{\partial(\theta)} \approx \lim_{h \to 0} \frac{J(\theta+h) - J(\theta-h)}{2h} \tag{2.7}$$

Luego, el "límite" de la ecuación 2.7 se puede resolver (en la implementación) colocando a la constante h un valor muy pequeño (por ejemplos $1e^{-5}$).

En el caso que tengamos un vector de parámetros θ tal que la función de pérdida sea J: $\mathbb{R}^n \mapsto \mathbb{R}$, podemos encontrar una aproximación del gradiente empleando la ecuación 2.7 para cada dimensión de θ. Es decir, se obtendrá otro vector (gradiente) de la misma dimensión que θ en donde en cada posición tendremos una aproximación de la derivada parcial de la función J con respecto a una dimensión. Luego de obtener el gradiente, lo multiplicamos por un hiperparámetro denominado "tamaño del paso" (step-size) y al vector que se obtiene, se lo restamos al vector de pesos θ. Con estos nuevos pesos, al calcular el valor de la función de pérdida se verifica que sea menor al del estadío anterior. Dado que en las redes neuronales existen millones de parámetros, este tipo de cálculo resulta totalmente ineficiente. No obstante, dada su fácil implementación, se lo utiliza como "verificación" del correcto funcionamiento del gradiente análitico y además se lo puede emplear cuando la función de pérdida no es derivable [29].

Gradiente Analítico

Este es el método que se utiliza en la práctica ya que el método anterior, además de ser ineficiente debido que necesita una gran potencia de cómputo, resulta una aproximación del gradiente. En este caso se calcula el gradiente por medio de cálculo numérico y por eso tiene la restricción que la función de pérdida debe ser derivable. El algoritmo mostrado en 2.1 (que emplea este cálculo del gradiente) se lo llama gradiente descendente. En la ecuación 2.8 se observa el gradiente de la función de pérdida con respecto a sus pesos θ y un valor de step-size η (que en el contexto de las redes neuronales es la tasa de aprendizaje). En resumen, dado un vector de parámetros iniciales θ_0 (en el contexto de las redes neuronales son los pesos iniciales W_0), una función de pérdida $J(\theta)$ y un step size η, el algoritmo del gradiente descendente es:

Algoritmo: 2.1 - Gradiente descendente

$$\begin{aligned} &\textit{Inicializar pesos } W = W_0 \\ &\textit{Repetir hasta que } J(\theta) \textit{ converja}: \\ &\qquad W = W - \eta \nabla_\theta J(\theta) \end{aligned} \tag{2.8}$$

Siendo η uno de los hiperparámetros más importantes en las redes neuronales, ya que un valor muy pequeño hará que la función de pérdida tome un valor pequeño después de muchas iteraciones y por ende conllevará un elevado tiempo de ejecución . Sin embargo, si toma un valor muy grande, puede suceder que la red neuronal nunca converja, aún con un tiempo de entrenamiento excesivamente grande [29].

Gradiente descendente estocástico (SGD) y por mini-lotes

En el algoritmo del gradiente descendente, la actualización de los pesos se realiza una vez que el algoritmo “vió” todo el conjunto de entrenamiento. Por ende, si el mismo posee un elevado número de instancias, el tiempo que se tendrá que esperar para que la red converja puede ser excesivamente alto. Con lo cual surge **el gradiente descendente en mini-batch (mini-lotes) y el gradiente descendente estocástico (u online)**. En el primer caso, se estima el gradiente evaluando K ejemplos (lote) del conjunto de entrenamiento y luego se realiza la actualización de los parámetros. Los K ejemplos constituyen el batch size (tamaño del lote) y se configurará durante el entrenamiento. Si bien se tiene una dirección aproximada del verdadero gradiente (el hallado con la totalidad de los ejemplos), se tiene una mayor cantidad de actualización de parámetros (ya que la actualización es por cada lote) y una convergencia más rápida [48]. En el caso del gradiente descendente estocástico, se realiza la actualización de los pesos por cada ejemplo del conjunto de entrenamiento. En este caso, también se está aproximando la dirección del gradiente verdadero, pero es menos eficiente computacionalmente que el gradiente descendente por mini-lotes, ya que en este último se utiliza “código optimizado para uso vectorial” (ya que se dispone de más de un ejemplo del conjunto de entrenamiento por cada lote). Por último, debido a que en el gradiente estocástico se realiza una actualización de los pesos con mayor frecuencia, se produce una mayor fluctuación en la función de pérdida hasta encontrar el mínimo (mayor efecto de “zig-zag”) . Es importante hacer notar que los 2 casos (por mini-lotes o estocástico) son sensibles al sesgo que puede existir en el conjunto de entrenamiento, por ende dicho conjunto de datos debe ser mezclado antes de ser incorporado a los algoritmos previamente mencionados [49].

Momentos

Se pueden incluir mejoras al algoritmo del gradiente descendente utilizando, por ejemplo, el **momento**. Este método, al actualizar un peso de la red, toma en cuenta las direcciones anteriores de los gradientes. Es decir, la actualización de los pesos influenciada por el momento, está dada por la siguientes ecuaciones:

$$v_{t+1} = \mu v_t - \eta \nabla_\theta J(\theta) \qquad (2.9)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

Donde $J(\theta)$ es la función de pérdida, η es la tasa de aprendizaje y v_t es considerado un vector de velocidad que actúa como una memoria de las direcciones de los gradientes de los t pasos anteriores. La influencia de esta memoria es controlada por μ, que es el coeficiente del momento. Con lo cual se deduce que a mayor valor de μ, mayor será la influencia de las direcciones anteriores. Por último, al momento se lo puede pensar como un incremento del step-size proporcional al tamaño y alineación de los gradientes anteriores [50]. La inclusión del momento ayuda al algoritmo del gradiente descendente a converger más rápido, por ejemplo, si se tienen varias direcciones de gradientes anteriores en el mismo sentido. Es decir, en este caso incrementaría la velocidad cuando nos movemos por la superficie de la función de pérdida hacia el mínimo. Además, permite escapar de los puntos mesetas (saddle points) como así también de los mínimos locales. No obstante, existen los casos donde al aplicar el momento, se producen oscilaciones en la trayectoria recorrida (sobre la superficie de la función de pérdida) hasta encontrar el mínimo. Esto último puede producirse ya que el momento imprimió demasiada velocidad en la actualización de los parámetros (ver ecuación 2.9). Un método que se puede aplicar para reducir las oscilaciones (que suceden con más frecuencia cuando la tasa de aprendizaje es alta), es el **momento "Nesterov"** o gradiente acelerado Nesterov [51][52]. El cálculo de la actualización de los pesos con esta nueva variante del momento, se realiza según la siguientes ecuaciones:

$$v_{t+1} = \mu v_t - \eta \nabla_\theta J(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$

En el caso de la aplicación del momento estándar, puede suceder que la función de pérdida aumente luego de la actualización de los parámetros (produciendo las oscilaciones antes mencionadas). Mediante esta versión del cálculo del momento, nos situamos en el punto estimado por la aplicación del momento y recién en ese punto calculamos el gradiente de la función de pérdida con respecto a los parámetros. Con esta modificación, se realiza una "corrección" mediante el nuevo gradiente calculado, al situarnos en el punto estimado por la aplicación del momento $(\theta + \mu v_t)$ antes de la actualización de los parámetros. Por ende, disminuimos o eliminamos las oscilaciones que podríamos obtener con la aplicación del momento estándar [52]. Lo anteriormente explicado puede verse en la figura 2.13.

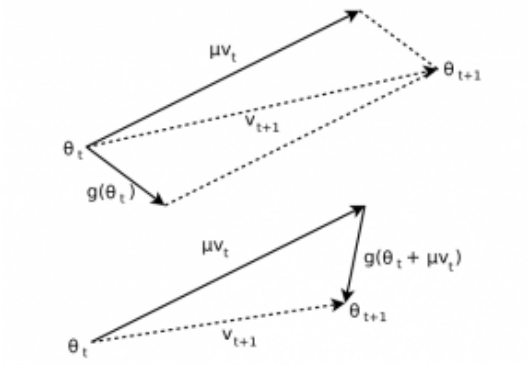


Figura 2.13: (Arriba) Inclusión del momento clásico. (Abajo) Inclusión del gradiente acelerado "Nesterov".

Ajuste de la tasa de aprendizaje

Otra de las formas de impedir el efecto de zig-zag en las zonas profundas y angostas de la superficie de la función de pérdida, es mediante la configuración de la tasa de aprendizaje.

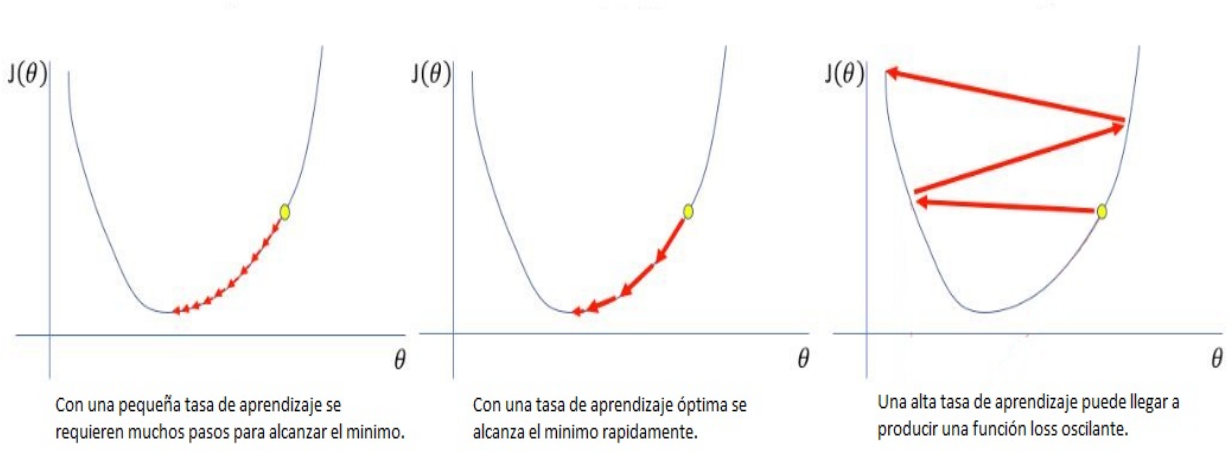


Figura 2.14: Pasos producidos por distintas tasas de aprendizaje.

En la figura 2.14 se observa que lo ideal sería ir disminuyendo la tasa de aprendizaje a medida que nos acercamos al mínimo. Existen 3 tipos de disminución de tasa de aprendizaje (no adaptativas) mayormente utilizados [29]:

- Decaimiento del paso (step decay): Cada cierto número de épocas, se reduce la tasa de aprendizaje por un factor determinado. Un valor típico que se emplea en la práctica es disminuir la tasa de aprendizaje a la mitad cada un determinado número de épocas, aunque se debe evaluar cada problema en forma independiente.

- Decaimiento exponencial (exponential decay): Se emplea la fórmula $\eta = \eta_0 e^{-k t}$. Donde η_0 (tasa de aprendizaje inicial) y k son hiperparámetros, y t es el número de iteración. En esta fórmula, cuanto mayor sea el número de iteración, el factor $e^{-k t}$ será menor y por ende la tasa de aprendizaje calculada η será más chica.

- Decaimiento 1/t (1/t decay): Se emplea la fórmula $\eta = \eta_0 / (1 + kt)$. Donde η_0 (tasa de aprendizaje inicial) y k son hiperparámetros, y t es el número de iteración. Igual que en el caso anterior, cuanto más grande sea el número de iteración, menor será la tasa de aprendizaje final calculada.

Ajuste de la tasa de aprendizaje en forma adaptativa

Una dificultad que se presenta en el ajuste de la tasa de aprendizaje según las 3 formas que vimos anteriormente es que se aplica la misma tasa de aprendizaje para todos los parámetros (pesos). Mediante los optimizadores de tasa de aprendizaje en forma adaptativa, se pueden calcular tasas de aprendizaje por parámetro. Si bien tienen otros hiperparámetros, los que se encuentran por defecto hacen que el optimizador funcione relativamente bien en la mayoría de los casos [29]. Los optimizadores de tasa de aprendizaje adaptativa utilizados frecuentemente son:

Adagrad

Este optimizador [53] adapta la tasa de aprendizaje por parámetro en cada iteración. De forma tal que la tasa de aprendizaje de cada parámetro se encuentra influenciada por los gradientes asociados que tiene en su historial. La tasa de aprendizaje será reducida para aquellos parámetros con grandes gradientes en su historial, inversamente, será aumentada en aquellos parámetros con pequeños gradientes, tal cual se muestra en las siguientes ecuaciones:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i})$$

$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{n}{\sqrt{G_{t,\,ii} + \varepsilon}} \cdot g_{t,i} \qquad (2.10)$$

Donde $g_{t,i}$ es el gradiente de la función de pérdida con respecto al parámetro θ_i en la iteración t y $G_t \in \mathbb{R}^{dxd}$ es una matriz diagonal donde cada elemento en la ubicación i,i es la suma de los gradientes al cuadrado con respecto al parámetro θ_i hasta la iteración t. Por último, ε es un valor muy pequeño con el objetivo que el denominador no sea 0.

Visualizando la ecuación 2.10 y el cálculo de G_t , se observa que este optimizador coloca una tasa de aprendizaje elevada a los parámetros (pesos) asociados a las features poco frecuentes, logrando un paso (step-size) grande en la dimensión de dichos parámetros. Por otro lado, a los parámetros asociados a las features frecuentes le coloca una tasa de aprendizaje pequeña, generando un step-size más chico que en las features poco frecuentes. Existen conjuntos de datos donde las features poco frecuentes tienen una relación directa con la clase que queremos predecir mientras que las frecuentes no. Aplicando este optimizador a este tipo de conjunto de datos, las features “raras” tendrán más importancia que aquellas que aparecen más frecuentemente pero son poco informativas (poca relación con la clase que queremos predecir). Es decir, este tipo de optimizador funciona correctamente en conjuntos de datos dispersos (sparse data).

No obstante, observamos en la ecuación 2.10 que a medida que van transcurriendo la iteraciones, el denominador donde se encuentra la sumatoria de los gradientes al cuadrado tenderá a aumentar, haciendo que la tasa de aprendizaje adaptada sea prácticamente 0. Esto hará que el modelo deje de aprender en forma prematura. Sin embargo, los optimizadores que se verán a continuación solucionan este problema.

Adadelta

Este optimizador [54] implementa 2 mejoras con respecto al Adagrad. La primera es tomar el historial de los cuadrados de los gradientes pero en una ventana W en lugar de todos los gradientes anteriores hasta la iteración t . En segundo lugar, en vez de guardar ineficientemente los W gradientes anteriores, la suma de los gradientes al cuadrado se redefine como el promedio (con decaimiento) de los gradientes anteriores al cuadrado, según la ecuación 2.11. En ella observamos que el promedio de los cuadrados de los gradientes en el instante t se encuentra en función del gradiente al cuadrado en el instante t y el promedio de los gradientes al cuadrado en el instante anterior. Además se incluye el factor de aceleración γ , similar al término de momento, cuyo valor se setea típicamente en 0.9. Por la inclusión de γ decimos que el promedio de los gradientes se calcula con decaimiento.

$$E[g^2]_t = \gamma\, E[g^2]_{t-1} + (1 - \gamma)\, g_t^2 \tag{2.11}$$

Por otro lado, reemplazando $G_{t,\,ii}$ de la ecuación 2.10 por $E[g^2]_t$ la ecuación 2.11, obtenemos el nuevo valor del parámetro en el instante $(t+1)$ dado por la siguiente ecuación:

$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{n}{\sqrt{E[g^2]_t + \varepsilon}} \cdot g_{t,i} \tag{2.12}$$

Debido que el denominador de la ecuación 2.12 corresponde a la media cuadrática (RMS) del gradiente, obtenemos la siguiente ecuación:

$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{n}{RMS(E[g]_t)} \cdot g_{t,i} \tag{2.13}$$

Debido que las unidades para realizar la resta no coinciden, los autores de este optimizador propusieron utilizar las actualizaciones de los parámetros al cuadrado (en lugar de los gradientes al cuadrado) dando lugar a:

$$E[\Delta\theta^2]_t = \gamma\, E[\Delta\theta^2]_{t-1} + (1 - \gamma)\Delta\theta_t^2$$

Además conocemos que:

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \varepsilon}$$

Como $RMS[\Delta\theta]_t$ es desconocido ya que todavía no conocemos el $\Delta\theta$ en el instante *t,* se aproxima con un RMS en el instante anterior $(t\text{-}1)$. Luego, reemplazando este RMS por la tasa de aprendizaje η de la ecuación 2.13, se llega a la ecuación final de la actualización del parámetro, la cual no necesita de una tasa de aprendizaje inicial:

$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{RMS[\Delta\theta]_{t-1}}{RMS(E[g]_t)} \cdot g_{t,i}$$

Rmsprop

Este optimizador [55], fue propuesto por Geoff Hinton y ha sido desarrollado en forma paralela al Adadelta para resolver las debilidades del Adagrad. Está basado en las ecuaciones 2.11 y 2.12, es decir, el estadío previo a la unificación de unidades que propusieron los creadores del Adadelta. Con lo cual, para este optimizador se necesita una tasa de aprendizaje inicial y un momento, en donde Hinton propuso una tasa de aprendizaje de 0.001 y un momento de 0.9, resultando las siguientes ecuaciones finales:

$$E[g^2]_t = 0.9\, E[g^2]_{t-1} + 0.1\, g_t^2$$
$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{0.001}{\sqrt{E[g^2]_t + \varepsilon}} \cdot g_{t,i}$$

Adam

Este optimizador [56] incorpora el promedio de los gradientes al cuadrado v_t del mismo modo que lo hacen Rmsprop o Adadelta (ver ecuación 2.15). La diferencia principal radica que en este optimizador también se calcula el promedio de los gradientes anteriores m_t (ver ecuación 2.14) en lugar de considerar el gradiente en el instante t ($g_{t,i}$ en el caso de Adadelta o Rmsprop).

$$m_t = \beta_1\, m_{t-1} + (1-\beta_1)\, g_t \tag{2.14}$$
$$v_t = \beta_2\, v_{t-1} + (1-\beta_2)\, g_t^2 \tag{2.15}$$

Debido que los vectores v_t y m_t son inicializados con ceros, sucede que durante las primeras iteraciones el cálculo de los mismos, según las ecuaciones 2.14 y 2.15, generalmente tienden a 0. Esto último se acentúa cuando los valores de β_1 y β_2 son cercanos a 1. Con lo cual, los autores propusieron realizar 2 estimadores de estos vectores para contrarrestar este problema. Dichos estimadores están dados por las siguientes ecuaciones:

$$\widehat{m_t} = m_t / (1 - \beta_1^t) \tag{2.16}$$
$$\widehat{v_t} = v_t / (1 - \beta_2^t) \tag{2.17}$$

Luego reemplazando las ecuaciones 2.16 y 2.17 en la ecuación 2.10, se tiene la ecuación final de actualización de los parámetros para este optimizador dada por:

$$\theta_{t+1,\,i} = \theta_{t.i} - \frac{\eta}{\sqrt{\widehat{v_t} + \varepsilon}} \cdot \widehat{m_t}$$

Donde los autores proponen los siguientes valores por defecto: β_1= 0.9, β_2= 0.999 y $\varepsilon = 10^{-8}$.

Backpropagation

Introducción

Rumelhart, Hinton y Williams formalizaron un método para que una red neuronal aprendiera la asociación que existe entre los patrones de entrada y las clases asociadas. Este método, conocido como backpropagation (propagación del error hacia atrás), funciona en más capas de neuronas que las que utilizó Rosemblatt para desarrollar el perceptrón [57].

Este algoritmo se corresponde con los del tipo supervisado, ya que el conjunto de datos entrenamiento está compuesto por los patrones de entrada y el ground truth (objetivo) que corresponde cada uno de ellos. En forma simplificada, este algoritmo contiene dos fases: En la primera fase se aplica un patrón de entrada como estímulo para la primera capa de neuronas de la red. Luego, este se va propagando a través de las capas superiores hasta generar una salida. Esta es comparada con la salida que se desea obtener y se calcula un error por cada neurona de salida. A continuación comienza la segunda fase, en la cual se transmiten estos errores hacia atrás, pasando por todas las neuronas que contribuyeron a la salida. Recibiendo cada una de ellas un porcentaje del error de acuerdo a su participación relativa al error total. En función del error recibido se reajustan los pesos de cada neurona, de forma tal que cuando se le presente a la red otra vez el mismo patrón genere una salida más próxima a la deseada, es decir, el error sea menor que el anteriormente calculado.

Durante el entrenamiento, las neuronas aprenden a reconocer distintas características del espacio total de entrada. Después del entrenamiento, cuando se le presente a la red un patrón arbitrario que no vió antes, generará una salida activa si la nueva entrada tiene un patrón que se asemeja a una característica que las neuronas hayan aprendido a reconocer durante el entrenamiento. Este es un punto importante que se les exige a los sistemas de aprendizaje, que corresponde a la capacidad de generalización. Es por esta razón que no se itera sobre el mismo elemento del conjunto de datos hasta eliminar el error, sino lo que se realiza es un acercamiento a la salida deseada con un elemento, luego con otro y así hasta recorrer todo el conjunto de datos de entrenamiento. A esta recorrida del conjunto de datos se la denomina “Epoch” (“Época”). Lo anteriormente explicado corresponde a la figura 2.15.

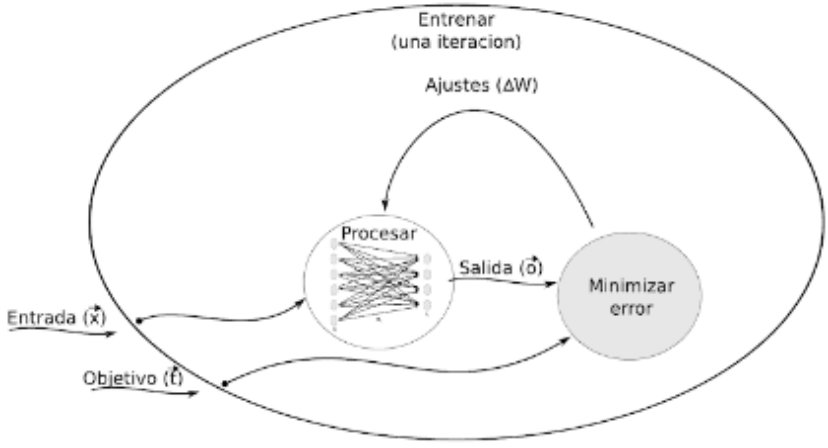


Figura 2.15: Esquema general del algoritmo backpropagation.

Algoritmo

A continuación se explicarán las 2 fases que tiene este algoritmo.

Forward pass

En esta fase se calculan las salidas de cada neurona y la predicción de la red. Si una neurona j tiene i entradas cada una de ellas con un valor x_i y un peso w_i, la salida de dicha neurona denotada con z_j es calculada mediante la aplicación de una función no lineal (y diferenciable) a la suma ponderada (por los pesos) de las entradas. Lo anterior se puede visualizar en las siguientes ecuaciones:

$$z'_j = \sum_i w_{ij}\ x_{ij} + b \tag{2.18}$$

$$z = f(z'_j) \tag{2.19}$$

En la ecuación 2.18, b es el sesgo y en la ecuación 2.19, la función f corresponde a una función escalar en $\mathbb{R}$. No obstante, el cálculo de las salidas de las neuronas, se implementa como un producto matricial de las entradas y los pesos. Para explicar esto último, lo realizaremos con un ejemplo. En la figura 2.16 se observa una capa lineal con 3 neuronas que está incluida en una red de más capas. En la misma se obtiene una salida $Y=f(X,W)$ y para mayor sencillez en la explicación no se consideran las funciones de activación ni el sesgo de cada neurona.

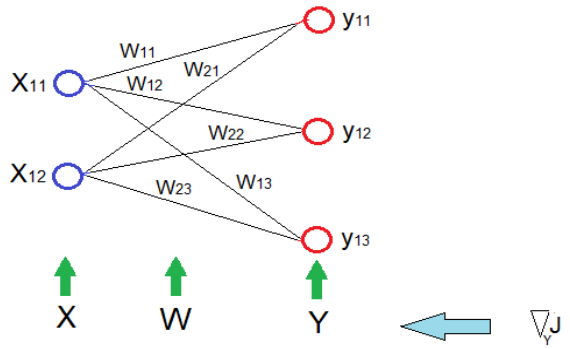


Figura 2.16: Capa lineal dentro de una red con más capas.

Luego, durante el forward pass, se toma una entrada X de dimensiones $N\,x\,D$ y una matriz de pesos W de dimensiones $D\,x\,M$ y se calcula una salida $Y = X\,W$. En este caso: $N=1$, $D=2$ y $M=3$. Con lo cual las entradas y los pesos de esta capa lineal son:

$$X = \begin{bmatrix} x_{11} & x_{12} \end{bmatrix}, \quad W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

Donde X tiene dimensiones $1\ x\ 2$ y W tiene dimensiones $2\ x\ 3$. Luego, se calcula la salida de la capa lineal como producto matricial de X y W dada por la siguiente ecuación:

$$Y = \begin{bmatrix} x_{11}w_{11} + x_{12}w_{21} & x_{11}w_{12} + x_{12}w_{22} & x_{11}w_{13} + x_{12}w_{23} \end{bmatrix} \tag{2.20}$$

Este cálculo se realizará repetidamente entre las distintas capas (donde la salida de una capa es la entrada de la otra) hasta obtener la salida de la red. Por último, una vez obtenida esta salida y teniendo el valor de salida deseado, se calcula la función de pérdida J (que será un escalar) para dar comienzo a la segunda fase.

Backward pass

En esta fase se calcula la influencia que tiene la salida de cada neurona con respecto a la función de pérdida J calculada. En base a esta influencia se actualizan los parámetros de cada neurona de forma tal que la función de pérdida J sea minimizada. Se resalta la importancia de este algoritmo ya que realizar los ajustes de los pesos en forma manual con el objetivo de minimizar la función de pérdida, en redes de millones de neuronas interconectadas, resultaría prácticamente imposible. Esta fase comienza en la función de pérdida J y transmite la influencia (que tiene cada neurona) en el error total cometido hasta llegar a las neuronas de entrada. La transmisión de esta influencia que tiene cada neurona en el error total obtenido (cálculo de la función de pérdida J), se realiza calculando el gradiente de la función de pérdida J con respecto a la salida de cada neurona utilizando la regla de la cadena. Para explicar este concepto nos basamos en la figura 2.17 en la cual se observa, en forma esquemática, una neurona de la red (no necesariamente es una neurona de las mostradas en la figura 2.16, sino que puede pertenecer a una capa superior o inferior). El concepto radica en calcular la función de pérdida con respecto a la salida de una neurona, en este caso $Z;$ luego para circular este gradiente hacia las neuronas de las capas inferiores, lo multiplicamos por el denominado gradiente local. El gradiente local es la derivada parcial de la salida de una neurona con respecto a su entrada. Esta multiplicación se debe a que la conexión de 2 neuronas se la puede ver como una función compuesta $f[g(x)]$ ya que la salida de una neurona corresponde a la entrada de la siguiente y por ende su derivada es $f'[g(x)]\ g'(x)$. Como se dijo anteriormente, por cuestiones de simplicidad, no se tiene en cuenta la función de activación no lineal de la neurona, de lo contrario se tendrá que considerar la derivada de la misma al circular el gradiente hacia las neuronas de las capas inferiores. Se aclara que por una cuestión de performance, en algunas implementaciones, se calculan los gradientes locales en la fase de forward pass (y se guardan en memoria). Esto es posible ya que tanto la entrada como la salida de cada neurona se encuentran disponibles en esta fase y por ende se podrían calcular los gradientes locales [58].

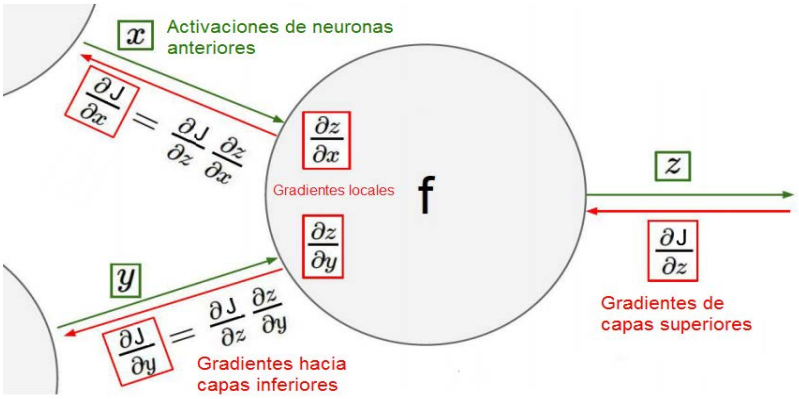


Figura 2.17: Esquema de la circulación del gradiente ($\partial J / \partial z$) hacia las neuronas de las capas inferiores. Se omite la función de activación no lineal de la neurona para que la explicación sea más sencilla.

Volviendo a nuestro ejemplo y basándonos en la figura 2.16, observamos que la derivada parcial $\partial J / \partial Y$ proviene de una capa superior. No obstante, si estuviéramos en la última capa de la red (no hay más capas superiores) y teniendo en cuenta que J es un escalar e Y es un vector, la derivada parcial $\partial J / \partial Y$ será: .

$$\frac{\partial J}{\partial Y} = \begin{bmatrix} \frac{\partial J}{\partial y_{11}} & \frac{\partial J}{\partial y_{12}} & \frac{\partial J}{\partial y_{13}} \end{bmatrix}$$

Luego, el objetivo será calcular $\partial J / \partial X$ para circular el gradiente de la función de pérdida con respecto a la salida de cada neurona en capas inferiores (suponiendo que x_{11} y x_{12} pueden ser las salidas de otras neuronas). Además, se calculará $\partial J / \partial W$ para la actualización de los pesos (mostrados en la figura 2.16). Luego aplicando la regla de la cadena, se llega a las siguientes ecuaciones:

$$\frac{\partial J}{\partial X} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial X} \tag{2.21}$$

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial W}$$

A continuación calcularemos $\partial J / \partial X$, aunque se puede seguir el mismo procedimiento para calcular $\partial J / \partial W$.

$$\frac{\partial J}{\partial X} = \begin{bmatrix} \frac{\partial J}{\partial x_{11}} & \frac{\partial J}{\partial x_{12}} \end{bmatrix} \tag{2.22}$$

De acuerdo a la ecuación 2.22 debemos calcular la derivada parcial de la función de pérdida J por cada elemento de X, obteniendo las siguientes ecuaciones:

$$\frac{\partial J}{\partial x_{11}} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial x_{11}}$$

$$\frac{\partial J}{\partial x_{12}} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial x_{12}}$$

Teniendo en cuenta la ecuación 2.20, procedemos a calcular $\partial Y / \partial x_{11}$ y $\partial Y / \partial x_{12}$, obteniendo las ecuaciones:.

$$\frac{\partial Y}{\partial x_{11}} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x_{11}} & \frac{\partial y_{12}}{\partial x_{11}} & \frac{\partial y_{13}}{\partial x_{11}} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \end{bmatrix}$$

$$\frac{\partial Y}{\partial x_{12}} = \begin{bmatrix} \frac{\partial y_{11}}{\partial x_{12}} & \frac{\partial y_{12}}{\partial x_{12}} & \frac{\partial y_{13}}{\partial x_{12}} \end{bmatrix} = \begin{bmatrix} w_{21} & w_{22} & w_{23} \end{bmatrix}$$

Dado que J , x_{11} y x_{12} son escalares, entonces las derivadas parciales $\partial J / \partial x_{11}$ y $\partial J / \partial x_{12}$ deben ser escalares. Con lo cual, dichas derivadas parciales se resuelven con las siguientes ecuaciones (utilizando el producto interno).

$$\frac{\partial J}{\partial x_{11}} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial x_{11}} = \frac{\partial J}{\partial y_{11}}w_{11} + \frac{\partial J}{\partial y_{12}}w_{12} + \frac{\partial J}{\partial y_{13}}w_{13} \tag{2.23}$$

$$\frac{\partial J}{\partial x_{12}} = \frac{\partial J}{\partial Y}\frac{\partial Y}{\partial x_{12}} = \frac{\partial J}{\partial y_{11}}w_{21} + \frac{\partial J}{\partial y_{12}}w_{22} + \frac{\partial J}{\partial y_{13}}w_{23} \tag{2.24}$$

Finalmente, sustituyendo 2.23 y 2.24 en 2.22 obtenemos la siguiente ecuación:

$$\frac{\partial J}{\partial X} = \begin{bmatrix} \frac{\partial J}{\partial y_{11}}w_{11} + \frac{\partial J}{\partial y_{12}}w_{12} + \frac{\partial J}{\partial y_{13}}w_{13} \\ \frac{\partial J}{\partial y_{11}}w_{21} + \frac{\partial J}{\partial y_{12}}w_{22} + \frac{\partial J}{\partial y_{13}}w_{23} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial J}{\partial y_{11}} & \frac{\partial J}{\partial y_{12}} & \frac{\partial J}{\partial y_{13}} \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix}$$

$$= \frac{\partial J}{\partial Y}W^T \tag{2.25}$$

En la ecuación 2.25 se observa que se calculó $\partial J / \partial X$ sin haber calculado (explicitamente) la $\partial Y / \partial X$ (Ver ecuación 2.21) que resulta ser una matriz Jacobiana. Esta matriz, en los problemas reales puede contener millones de posiciones, debido a la cantidad de dimensiones y número de patrones (si se usa la versión “batch” o “mini-lotes” de este algoritmo) que puede llegar a tener la entrada X. Con lo cual, queda demostrado la eficiencia de este tipo de implementación, calculando $\partial J / \partial X$ con $\partial J / \partial Y$ y los pesos W. Con el mismo enfoque se puede calcular $\partial J / \partial W$ (sin utilizar la matriz Jacobiana $\partial Y / \partial W$) para la actualización de los pesos W (ver figura 2.16), obteniendo la siguiente ecuación [59] :

$$\frac{\partial J}{\partial W} = X^T\frac{\partial J}{\partial Y} \tag{2.26}$$

Con el gradiente calculado en 2.26 y teniendo en cuenta la tasa de aprendizaje η se procede a la actualización de los pesos usando el algoritmo del gradiente descendente (ver sección 2.2.4) mediante la siguiente ecuación:

$$W = W - \eta \ \partial J / \partial W$$

Consideraciones

El ejemplo que se expuso es a modo ilustrativo para explicar el concepto general del algoritmo backpropagation. Cuando se trabaja con conjuntos de datos reales, se debe considerar lo siguiente:

- Como entradas X tendremos tensores de varias dimensiones y varios patrones (en el caso de usar la versión “batch” o “mini-lotes” del algoritmo backpropagation).

- Cuando calculamos la derivada parcial de la salida de cada neurona con respecto a su entrada, se debe tener en cuenta la función de activación y por ende su derivada cuando se realiza la fase de backward.

- El backpropagation en las redes convolucionales requieren un tratamiento más avanzado, aunque la filosofía del funcionamiento del algoritmo sigue siendo la misma. Si se quiere más detalle consultar [60].

En el algoritmo 2.2 visualizamos un resumen, a nivel pseudo código, del entrenamiento de una red utilizando backpropagation del tipo online. Recordemos que es online dado que se produce la actualización de los pesos por cada patrón del conjunto de datos de entrenamiento que ingresa a la red.

Algoritmo: 2.2 - SGD via backpropagation (del tipo online)

Entrada:
Conjunto de datos de entrenamiento $(x_i, y_i).....(x_n, y_n)$
Inicializar:
Pesos de la red
Salida:
Red entrenada
⇢**Por cada época en num_epocas:** //Num_epocas es un parámetro de la red
⇢**Por cada ejemplo (x_i, y_i) del conjunto de entrenamiento:**

Forward Pass :

-Calcular, por cada neurona, $t_j = \sum_{i=1}^{k} w_i \; x_i \; + \; b$, $a_j = f(t_j)$

-Calcular los gradientes locales δ y generar salida de la red $f(x_i)$

-Calcular la función de pérdida: $J(f(x_i), y_i)$

Backward pass:

-Propagar el error hacia atrás (regla de la cadena) calculando: $\partial J / \partial w_{ij}$

-Usar el gradiente descendente para actualizar los pesos de la red de acuerdo a:

$W^{(i+1)} = W^{(i)} - \eta \; \partial J / \partial W^{(i)}$ // η = Tasa de aprendizaje,
// i = Nro. De iteración.

2.2.5 Redes neuronales convolucionales (CNN)

Principio biológico

Las redes neuronales artificiales están motivadas, como se mencionó anteriormente, en el funcionamiento de la neurona biológica. Particularmente, las redes neuronales convolucionales fueron inspiradas por los trabajos en neurociencia de Hubel y Wiesel [61] en lo que respecta a la corteza visual de los gatos en los años 1960s. Específicamente, estudiaron el comportamiento de la corteza visual primaria (V1) de un gato cuando se le presenta ciertas imágenes en una pantalla. Encontraron 3 características en esta corteza, que fueron incluidas en el funcionamiento de las redes convolucionales.

- **V1 tiene un mapa retinotópico**: Las células cercanas en la corteza V1 poseen campos receptivos cercanos (regiones cercanas en el campo visual). En las redes CNN, 2 neuronas cercanas en el mapa de características se corresponden con dos campos receptivos cercanos de la imagen de entrada.

- **Células simples**: La actividad de una célula simple puede caracterizarse como un filtro lineal de un pequeño campo receptivo (alargado) de la imágen, es decir, se activa con una determinada feature (generalmente líneas o bordes). En las CNNs esta propiedad se logra a través de los denominados kernels (núcleos) seguidos por la función de activación no lineal, lo que conforma una unidad detectora de features.

- **Células complejas**: Estas células detectan features como las células simples pero esa detección es casi invariante a la posición de esa feature dentro del campo receptivo que poseen (que es de mayor dimensión que el de las células simples). Esto lo consigue la CNN con la incorporación de las operaciones de pooling.

Los términos relacionados a las CNN escritos anteriormente se explicaran en la presente sección.

Topología de las redes neuronales convolucionales

Las CNNs son una clase especial de redes neuronales que incorporan una arquitectura de forma tal que reduce drásticamente el número de parámetros (pesos) que debe aprender la red en comparación con una red totalmente conectada. Esto permite que puedan procesar tensores con muchas dimensiones como son las imágenes [29].

Las CNNs demostraron tener un buen rendimiento en áreas tales como la clasificación de imágenes o la detección de objetos. Entre los casos de aplicación que tuvieron éxito se encuentran: identificación de caras, señales de tráfico, aumento del rendimiento en la visión de los robots y mejora en el desarrollo de los vehículos autónomos.

Los elementos principales (capas y transformaciones no lineales) que componen una red convolucional son los siguientes:

- Capa convolucional: Aplica la operación de convolución de los pesos de los filtros (o kernels) en una entrada, generando mapas de características.
- Transformaciones no lineales: Se aplica una función de activación no lineal.
- Capa de sub-muestreo (pooling): Realiza una operación de reducción (downsampling) en la entrada, generando mapas de características de menor resolución.
- Capa totalmente conectada: Produce el puntaje de la clasificación.

Cuando estos elementos se ubican en una secuencia, forman una red convolucional (ver figura 2.18) que puede ser entrenada usando backpropagation. Se aclara que la secuencia de bloques mostrada en la figura 2.18 constituye solo un ejemplo. Existen muchas otras secuencias o arquitecturas utilizadas en redes convolucionales, una de ellas se verá en la sección 2.5.1 (arquitectura VGG-16).

Cada una de estas capas tienen hiperparámetros que deben ser configurados previamente al entrenamiento. A continuación se explica el funcionamiento de cada una de estas capas y la transformación no lineal utilizada.

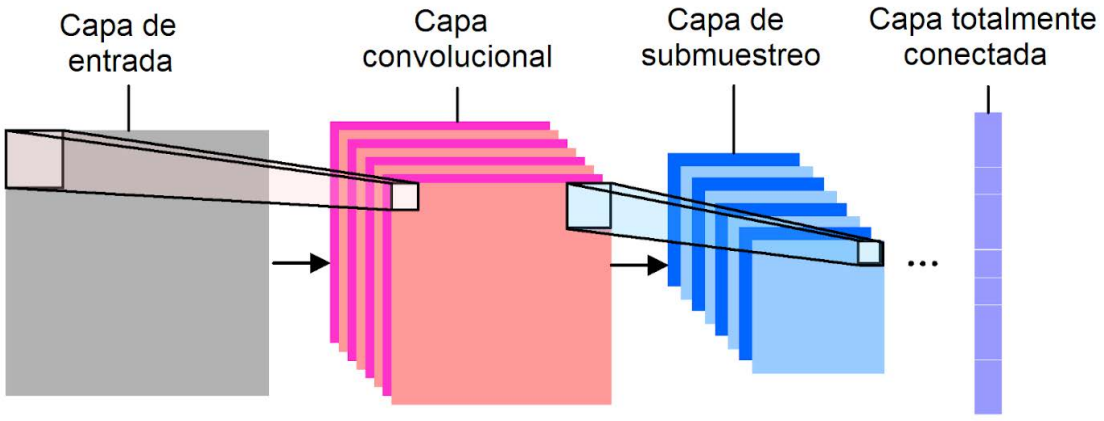


Figura 2.18: Secuencia de bloques básicos que dispone una red convolucional.

Capa convolucional

La convolución es una operación matemática entre 2 funciones para producir una tercera. La primera función x es conocida como la función de entrada, la segunda w como la función de ponderación y la operación de convolución la denotamos con "*". Esto último puede expresarse según la siguiente ecuación [62]:

$$y(t) = (x * w)(t) = \int x(a)\, w(t-a)\, d(a)$$

Donde w: $\mathbb{R}\rightarrow\mathbb{R}$, x: $\mathbb{R}\rightarrow\mathbb{R}$, y: $\mathbb{R}\rightarrow\mathbb{R}$, $a \in \mathbb{R}$ y $t \in \mathbb{R}$. La convolución entre x y w puede ser vista como el producto de las funciones $x(a)$ y $w(a)$ cuando una de una de ellas es invertida y desplazada t unidades.

Cuando implementamos una convolución en un ordenador, la entrada es discreta, es decir se recibe cada cierto tiempo determinado. Con lo que la operación de convolución también debe serlo. Con lo cual, las funciones x y w estarán definidas sólo para valores enteros y a t,

también, se le asignará números enteros. Por ende, podemos definir la convolución discreta dada por la siguiente ecuación:

$$y(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)\, w(t-a)$$

En la terminología de las redes CNN, a la función y se la denomina mapa de características, la función x es considerada la entrada I (que en nuestro caso es una imágen) y la función w es el kernel K. Es decir, la entrada y el kernel no son funciones, sino tensores (un vector multidimensional donde cada dimensión tiene un tamaño discreto definido). Con lo cual, si empleamos una imagen de 3 dimensiones, entonces utilizaremos un kernel de las mismas dimensiones, y la convolución entre el kernel y la imagen estará dado por:

$$Y(i,j,k) = (I * K)(i,j,k) = \sum_m \sum_n \sum_o I(m,\ n,o)\, K(i-m, j-n, k-o) \qquad (2.27)$$

Además se comprueba que la convolución es conmutativa, por ende la ecuación 2.27 es equivalente a la siguiente ecuación:

$$Y(i,j,k) = (K * I)(i,j,k) = \sum_m \sum_n \sum_o I(i-m, j-n, k-o)\, K(m,n,o)$$

La mayoría de las librerías de aprendizaje automático no emplean la operación de convolución, sino la operación de correlación cruzada (cross-correlation). Esta operación es la misma que la convolución solo que no invierte el kernel (ver la inversión en ecuación 2.27). Se implementa la correlación cruzada principalmente por un tema de performance. Si se emplea la operación de convolución, la CNN aprendería los pesos de los kernels pero en forma invertida sin que esto afecte el correcto funcionamiento de la red [63]. La operación de correlación cruzada entre K e I está definida por:

$$Y(i,j,k) = (K * I)(i,j,k) = \sum_m \sum_n \sum_o I(i+m, j+n, k+o)\, K(m,n,o)$$

En la figura 2.19 se observa la diferencia de aplicar una operación de correlación cruzada y convolución a una imagen con un kernel K de 3X3.

Figura 2.19: (Izquierda) Imagen original. (Centro) Resultado de una operación de correlación cruzada del kernel K con la imagen original. (Derecha) Resultado de una operación de convolución del kernel K con la imagen original.

En la figura 2.20 se observa la operación de convolución sin la inversión del kernel. En este caso se tiene una entrada y un kernel, ambos, de 2 dimensiones. Durante la operación de convolución, el kernel se desliza sobre la entrada con un paso (stride) igual a 1. En cada uno de estos pasos se calcula el producto escalar de la región de la entrada que mapea el kernel (llamado campo receptivo) y dicho kernel [64]. A modo de ejemplo, se marcó con doble borde a la celda superior izquierda de la "Salida" con el cálculo respectivo. Dicho resultado es consecuencia de aplicar el kernel a la región local superior izquierda de la entrada, que también se encuentra marcada con doble borde. Una vez completado todos los cálculos para las celdas restantes, obtenemos la "Salida" indicada en la figura 2.20. A esta "Salida" también se la denomina mapa de características (feature map) [63].

En una capa convolucional puede existir más de un kernel, en este caso se va a obtener un mapa de características por cada kernel y luego se deberán apilar los mismos. Por ende, se va a obtener una salida de profundidad n, siendo n el número de kernels presentes en una capa convolucional. Por último, considerando que el padding es el agregado de ceros en los bordes de la entrada con el objetivo de controlar el tamaño espacial de la salida, definimos la siguiente fórmula para calcular el tamaño del mapa de de características. Sea una imagen de entrada I, a la cual se le aplica un kernel K con un padding P y un stride S:

$$Tamaño\ de\ la\ capa\ de\ salida\ de\ convolución : ((I - K + 2P) / S) + 1 \qquad (2.28)$$

La fórmula 2.28 deberá calcularse para el ancho y el alto de la imágen.

El número de kernels, su tamaño, el padding y el stride son hiperparámetros que controlan el volumen de salida de cada capa convolucional de la red y por ende impactan en el entrenamiento de la la misma [40].

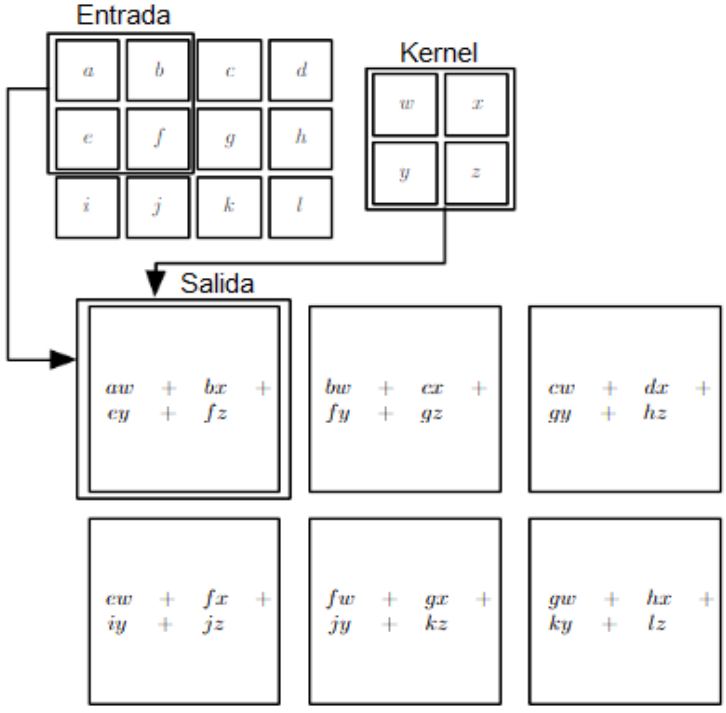


Figura 2.20: Convolución de un tensor de entrada de 2D con un kernel de 2D incorporando un paso (stride) igual a 1.

Para un correcto entendimiento se aclara que el factor padding establece el número de columnas o filas rellenas con ceros que se insertan alrededor de una entrada con el objetivo de controlar el tamaño de la salida de una operación de convolución sobre dicha entrada. Otros de los motivos puede ser cuando se requiere que los píxeles (en caso de que la entrada sea una imágen) de los bordes tomen mayor importancia [40], ya que el kernel de la convolución los tendrá como campo receptivo más cantidad de veces con respecto al estadío anterior (cuando no se aplicó padding). Por ejemplo, con un padding igual a 1, tendremos 2 filas (una arriba y otra abajo de la imágen) y 2 columnas (una a la izquierda y otra a la derecha de la imágen) todas rellenas con ceros. Si bien existe la posibilidad de fijar las cantidades de filas o columnas con ceros en forma independiente [65], en el ámbito de las CNNs se suele aplicar el mismo factor en los 2 ejes. Esto último se observa en la figura 2.21.

Por otro lado, se aclara que el stride es el número de pasos que se mueve el kernel sobre la la entrada. Estos “pasos” pueden ser seteados para cada eje, es decir en la dirección horizontal y vertical del movimiento del kernel sobre la entrada [65]. No obstante, en el contexto de las CNNs, suele setearse el mismo valor en las 2 direcciones. Con el stride se puede reducir el tamaño de cada dimensión del mapa de características, como así también parametrizar con qué grado de superposición queremos que se encuentren los campos receptivos. En la figura 2.22 se observa un kernel (en 2 dimensiones) que se mueve sobre una entrada (también en 2 dimensiones) con un stride igual a 2.

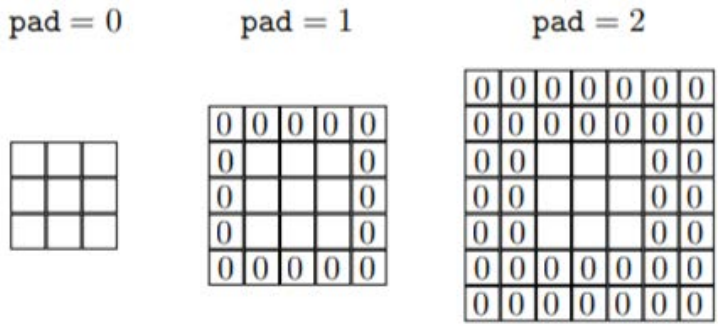


Figura 2.21: Distintos factores de padding aplicados en una imágen.

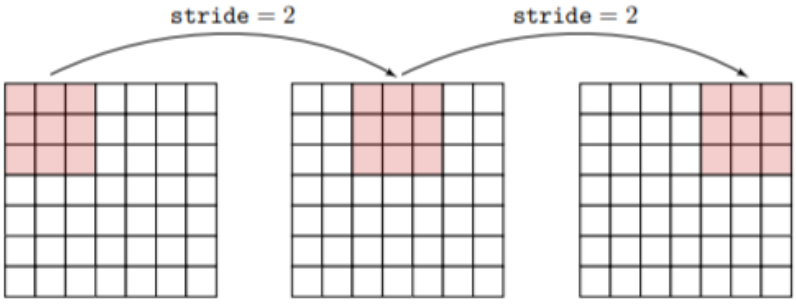


Figura 2.22: Movimiento de un kernel sobre una imagen con un stride igual a 2.

A modo de ejemplo, en la figura 2.23 se muestran 12 mapas de características de los 64 disponibles que posee la primera capa de convolución de la arquitectura VGG-16 (explicada en la sección 2.5.1). Los mapas de características se encuentran en una escala de grises, con lo cual, aquellas zonas más claras indican que el kernel fue excitado por el campo receptivo.

Es decir, en dicho campo receptivo el kernel encontró el feature para el cual fue entrenado para detectarlo. Si se hubieran mostrado kernels muertos (no se excitan con ningún campo receptivo), sus mapas de características se observarían totalmente de color negro y cuando sucede esto en un número elevado de kernels estamos en presencia de un mal entrenamiento de la red.

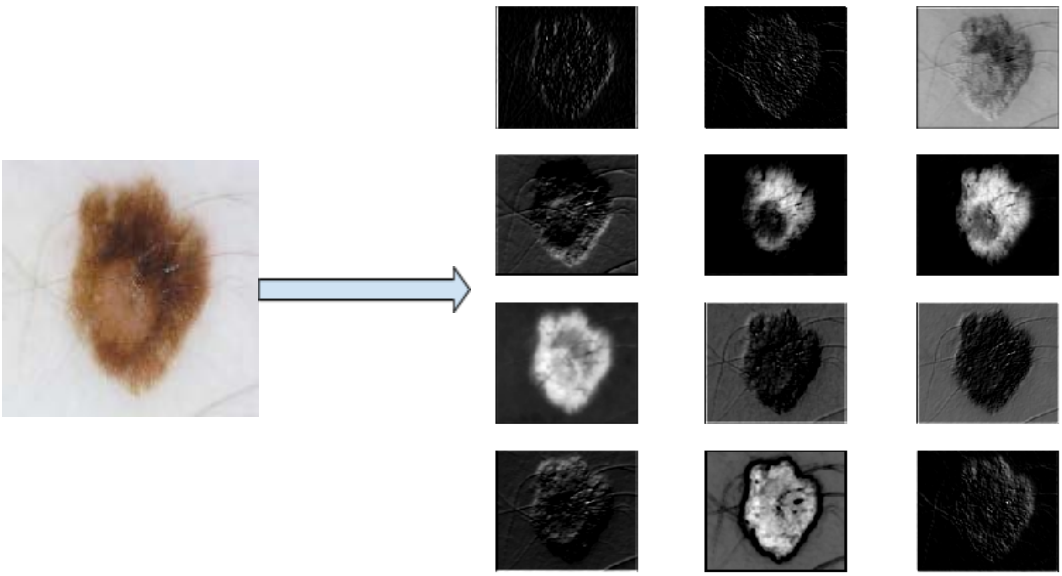


Figura 2.23: 12 mapas de características de la primera capa de convolución de la arquitectura VGG-16.

Cabe aclarar que cuando se trabaja con imágenes, la operación de convolución es en 3 dimensiones. Si bien la profundidad del kernel debe ser igual a la profundidad de la imagen (3 dimensiones, una por cada canal RGB) la salida de la operación de convolución sigue siendo en 2 dimensiones. En este caso la superposición del kernel con la imagen de entrada estará dada en un espacio de 3 dimensiones tal cual muestra la figura 2.24. El alto y ancho de la salida siguen la ecuación 2.28.

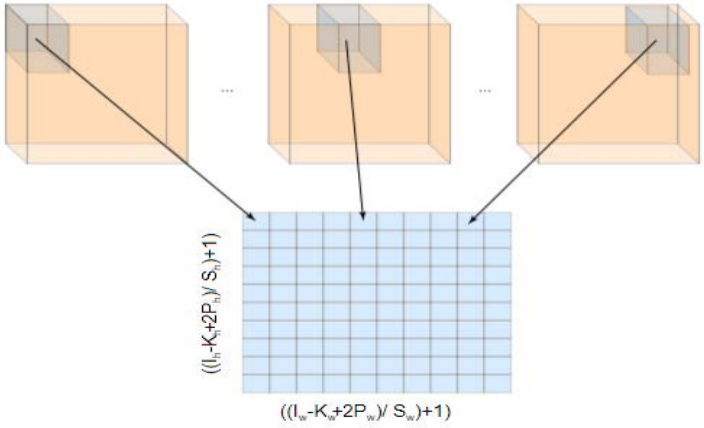


Figura 2.24: Superposición del kernel y la imagen de entrada en un espacio de 3 dimensiones. La salida de la operación de la convolución sigue siendo en 2 dimensiones.

Motivación del uso de la convolución

Existen 3 principales características que tienen este tipo de redes que incrementan la performance de la red en comparación con una red totalmente conectada (vistas en la sección 2.2). Estas características son las interacciones (conexiones) dispersas, compartimiento de parámetros y la representación equivariante [63]. Además pueden procesar entradas de distinto tamaño [66].

Conexiones dispersas

Tomando como ejemplo m neuronas de entrada y n neuronas de salida, si están totalmente conectadas, se tendrán que almacenar $m\ x\ n$ parámetros y se requerirán $O(m\ x\ n)$ operaciones de cómputo (por instancia del conjunto de datos). Si cada salida se limita a tener k conexiones , luego se requerirá aprender $k\ x\ n$ parámetros y se requerirán $O(k\ x\ n)$ operaciones de cómputo. Este enfoque suele tener una buena performance en muchas aplicaciones de aprendizaje automático aún cuando k es menor, en varios órdenes de magnitud, que m [63].

En la figura 2.25 se observa la diferencia en tener una arquitectura con capas totalmente conectadas y otra en la cual se emplean ***conexiones dispersas***. En cada una de ellas se remarca la neurona de salida, S3, y cuales son las neuronas de la entrada que afectan a la misma. A estas unidades de entrada se las denominan **campo receptivo**.

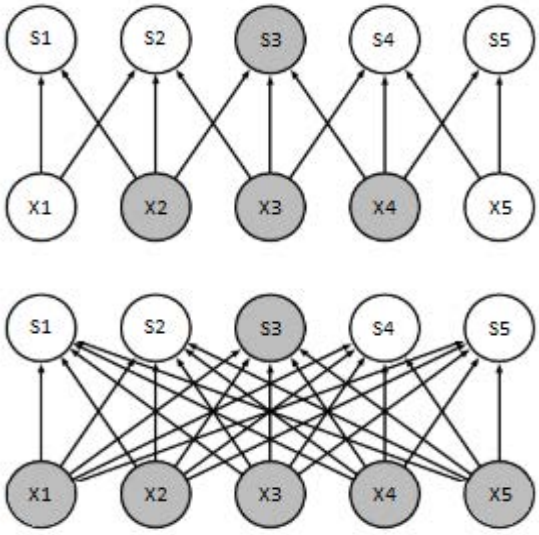


Figura 2.25: (Arriba) Cuando la salida S es formada por una convolución con un kernel de tamaño 3 y un paso de 1, solo 3 neuronas de entrada afectan a S3. (Abajo) Cuando la salida S corresponde a una capa totalmente conectada, todas las neuronas de entrada afectan a S3.

Por otro lado, las neuronas en capas más profundas están conectadas en forma indirecta, a toda o una gran porción de las neuronas de entrada. Esto último puede verse en la figura 2.26. Esto permite que la red pueda describir interacciones complejas entre distintas variables de entrada, donde las neuronas individuales describen cada una de esas interacciones [63]. Cada una de estas neuronas individuales tendrá conectividad dispersa.

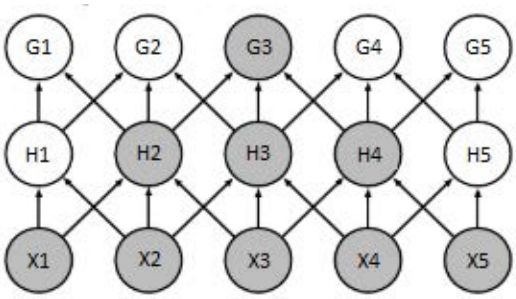


Figura 2.26: La neurona G3 se conecta directamente a las neuronas de la capa H en forma dispersa, pero indirectamente se conecta a todas las neuronas de la entrada X.

Compartimiento de parámetros

En una red totalmente conectada, cada peso que se encuentra en la matriz de pesos es utilizado una sola vez para calcular la salida de una determinada neurona, luego este peso no se vuelve a utilizar para calcular otra salida. En cambio, en una red convolucional, el kernel es usado en cada posición de la imagen (salvo los bordes, dependiendo los parámetros establecidos en la ecuación 2.28). Es decir, en lugar de aprender un conjunto de parámetros por cada posición de la imagen, solo se aprenderán los parámetros involucrados en el kernel ya que este se moverá por toda la imagen realizando la operación de convolución antes explicada. Al realizar esta operación, se generará un mapa de características, donde los pesos del kernel se encontrarán compartidos entre las distintas neuronas de salida que conforman el mapa de características antes mencionado. A los pesos compartidos también se los denomina “pesos atados”. En la figura 2.27 se explica lo anteriormente mencionado, tomando a modo ilustrativo 5 neuronas de entrada y 5 neuronas de salida [63].

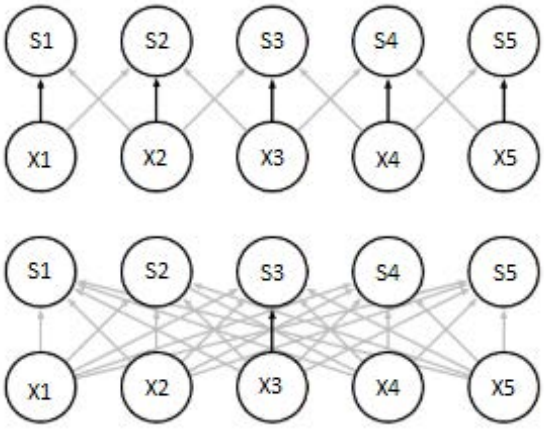


Figura 2.27: (Arriba) Las flechas negras indican el uso del peso ubicado en la posición central de un kernel de 3 posiciones, cuando se mueve por las entradas con un paso igual a 1. Debido a que comparten parámetros, este parámetro ubicado en la posición central es usado en todas las entradas. (Abajo) La única flecha negra indica dónde se usa el parámetro ubicado en la posición central de la matriz de pesos en un modelo con capas totalmente conectadas. Debido que no se comparten parámetros, este peso es utilizado una sola vez.

Como ejemplo, veamos el número de parámetros necesarios en una red totalmente conectada y en una convolucional. Tomemos una imagen de 128 x 128 x 3 y una red totalmente conectada en cuya primer capa oculta se tienen 500 neuronas. Con lo cual para esta capa se necesitan aprender (128 x 128 x 3 + 1) x 500= **24.576.500 parámetros** (El

término “+1” corresponde al bias). Luego, supongamos una red convolucional en la que en una capa convolucional se dispone de 64 kernels de 4 x 4 x 3 cada uno. Según vimos anteriormente, el almacenaje de parámetros es independiente al número de neuronas que posee el mapa de características y esta dado por 64 x (4 x 4 x 3 + 1) = **3136 parámetros** (El término “+1” corresponde al bias). No obstante, calcularemos el número de neuronas existente en el mapa de características, si utilizamos un paso de 2, no aplicamos padding (zero-padding) y empleamos la ecuación 2.28:

Tamaño del mapa de características = $((I - K + 2P) / S) + 1$
Tamaño del mapa de características = $((128 - 4 + 2 * 0) / 2) + 1$
Tamaño del mapa de características = 63

Lo que implica que cada slice tendrá 63 x 63 = 3969 neuronas, y como existen 64 kernels, se tendrán 64 x 3969 = 254.016 neuronas.

En conclusión, se incrementa la cantidad de neuronas de salida en una CNN en relación a una capa totalmente conectada pero el número de parámetros que se deben aprender se reduce en varios órdenes de magnitud.

Se aclara que el ejemplo expuesto es solo a nivel ilustrativo para demostrar la drástica reducción de parámetros que se tienen que aprender. No obstante, para que la comparación sea más realista, se debería comparar el número de parámetros de una arquitectura de capas totalmente conectadas con el de una CNN que tenga un rendimiento equivalente, ambas entrenadas con el mismo conjunto de datos.

Equivariante a la traslación

La propiedad que sea invariante a la traslación es una consecuencia de la propiedad “compartimiento de parámetros” que tienen las CNNs y que el campo receptivo se traslada por toda la imágen. Específicamente una función $f(x)$ es equivariante a una función $g(x)$ si $f(g(x)) = g(f(x))$. Luego, si $g(x)$ es una función de traslación y $f(x)$ es una función de convolución, entonces $f(x)$ es equivariante a $g(x)$ [63]. Como se vió en la figura 2.20, si tenemos una imagen con un objeto y un kernel que detecta ciertas características (features) de dicho objeto, la operación de convolución entre ambos generará un mapa de características 2D. Luego, si trasladamos dicho objeto en la imagen y aplicamos la convolución con el kernel, entonces las características del objeto se trasladarán en igual magnitud en el mapa de características. Con lo cual, sería lo mismo que aplicar la traslación al mapa de características que se obtuvo con la imagen del objeto sin haberlo trasladado, lo que confirma que $f(g(x)) = g(f(x))$.

Entradas de distinto tamaño

En las arquitecturas totalmente conectadas, cada capa tiene parámetros W (pesos), b (bias), X (entradas) tal que la salida Z, es $Z = WX + b$, con entradas y salidas definidas. En las redes convolucionales, los parámetros son los pesos de los kernels, que realizan la convolución con la entrada. Con lo cual, si se varía el tamaño de la entrada, la convolución sigue siendo factible aunque el tamaño de la salida (mapa de características) sea distinto [66]. En el caso que se necesite una salida de tamaño fijo y menor al mapa de características obtenido, es posible incorporar otras técnicas, como el sub-muestreo (Pooling), que se verá en esta sección.

Transformaciones no lineales

Una capa convolucional o un conjunto de capas convolucionales, generalmente son seguidas de una activación no lineal. Si bien las funciones no lineales se han visto en la sección 2.2.1, en el contexto de las redes neuronales convolucionales con aprendizaje mediante backpropagation, la más utilizada es la ReLU. Con el empleo de la ReLU evitamos unos de los problemas más conocidos en el entrenamiento por backpropagation: **la desaparición del gradiente** (Vanishing gradient) [67]. Tal como mencionamos en la sección 2.2.1 la función ReLU está definida como $ReLU(x) = max(0,x)$ y su derivada es la siguiente:

$$\frac{\partial ReLU(x)}{\partial x} = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x < 0 \\ \emptyset & \text{si } x = 0 \end{cases} \tag{2.29}$$

En este caso, los gradientes van a ser 0 o 1, con lo cual no tenemos el problema de la desaparición del gradiente (al menos para los valores positivos de *x*). No obstante, observamos en la ecuación 2.29 que para $x=0$ la derivada no está definida. Esto último no suele ser un problema grave porque los valores de entrada que toma esta derivada corresponden a la pasada de backward del algoritmo backpropagation y rara vez son valores nulos (exactamente cero) y en caso de que sea nula, en la práctica se coloca un 1 o un 0. No obstante tiene la desventaja que durante el entrenamiento puede aparecer el fenómeno de "neuronas muertas" explicado en la sección 2.2.1. Sin embargo, este último fenómeno suele suceder cuando se setea una elevada tasa de aprendizaje. Sin embargo, si a este hiperparámetro se lo ajusta correctamente, difícilmente nos encontremos con el problema de las "neuronas muertas".

En el caso de la función sigmoide, como se mencionó anteriormente, es $\sigma(x)=1/(1 + e^{-X})$. La derivada de la misma usada para calcular el gradiente es $\sigma'(x)= \sigma(x) * (1- \sigma(x))$. Esta última función tiene un máximo en $y=0.25$ cuando $x=0$ y tiene una asíntota en $y=0$. Es decir el rango de esta función es de (0, 0.25]. Esto último implica que cuando se calculan los gradientes locales de una neurona, estos serán pequeños y al multiplicarlo por el gradiente de la función de pérdida con respecto a la salida de esta neurona, será también un número pequeño con lo que se producirá una pequeña actualización de los pesos que corresponden a esta neurona. Además, si existen muchas capas apiladas, el mencionado gradiente de la función de pérdida con respecto a la salida de la mencionada neurona, estará influenciada por el producto de las derivadas de la función sigmoide (regla de la cadena) de las neuronas de las capas más profundas.Luego, si a este producto de gradientes pequeños se lo multiplica por el gradiente local (de la mencionada neurona) se obtendrá un valor prácticamente igual a 0 lo que produce la desaparición del gradiente y por por ende ralentiza o detiene el aprendizaje en la red [29]. Este último fenómeno también aparece cuando se usa la función de activación tanh, solo que esta función tiene sus salidas centradas en el 0, mientras que la sigmoide no. El hecho de que las salidas no estén centradas en 0 (en la función sigmoide), puede producir un zig-zag en la actualización de los pesos, complicando el entrenamiento de la red [29].

Por lo mencionado anteriormente y por lo explicado en la sección 2.2.1, en el presente trabajo se utilizó a la función sigmoide solo en la capa clasificadora y a la función ReLU en las capas convolucionales de la arquitectura VGG-16.

Sub-muestreo (Pooling)

En una red típica CNN, una capa de convolución es seguida de la correspondiente activación de cada uno de los mapas de características. Luego suele incluirse una capa de pooling (submuestreo), tal cual muestra la figura 2.18. El objetivo de esta capa es reducir la dimensión de cada uno de los mapas de características.

El concepto de pooling hace referencia a que cuando se encuentra una característica en un mapa de características, su ubicación exacta no es tan importante como su ubicación relativa a otras características. Usando este concepto se realiza alguna estadística de resumen en un campo receptivo determinado. Las operaciones más comunes que se suelen emplear son las de max-pooling, average-pooling y L^2-norm-pooling. Por ejemplo, al reducir el ancho y alto de un mapa de características a la mitad, se tiene un volumen con datos 4 veces menor que el volumen inicial. Esta reducción de complejidad (en número de operaciones y memoria) tiene la desventaja que perdemos cierta información espacial de la entrada, no obstante se obtienen ciertas ventajas. Por ejemplo, al reducir la dimensión, también disminuimos el número de parámetros que debemos aprender ya que la operación de pooling en sí misma no agrega nuevos parámetros al modelo. Otra de las ventajas de introducir esta capa, es que la red sea “casi” invariante a la traslación. Decimos que una función es invariante a la traslación si una traslación en la entrada no genera cambios en la salida. Con lo cual, si se realiza una pequeña traslación en los valores de la entrada de esta capa de pooling, la mayoría de los valores de salida no van a cambiar [63].

Como ejemplo, se observa en la figura 2.28 una operación de max-pooling utilizando un “pooling-kernel” de 2 x 2 y un stride de 2. En esta operación se toma el valor máximo del campo receptivo al que esté aplicando el “pooling-kernel”. Notar que este kernel, a diferencia del kernel tradicional de las capas convolucionales, no tiene parámetros que se deben aprender (en este caso se obtiene el máximo del campo receptivo, sin necesidad de almacenar pesos en el kernel).

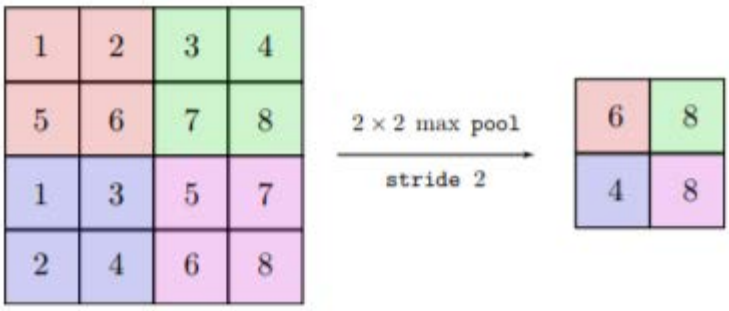


Figura 2.28: Operación de max-pooling.

Dado un kernel de pooling de dimensiones (w_p, h_p) y una entrada de dimensiones (w, h), la salida de la operación de pooling se calcula según la siguiente ecuación:

$$\left(\frac{w - w_p}{stride} + 1, \frac{h - h_p}{stride} + 1\right)$$

La profundidad se mantiene igual que la profundidad de la entrada donde se está aplicando el pooling (es decir, se mantiene el número de mapas de características pero habrá

una reducción del tamaño de las restantes dimensiones de cada uno de estos mapas). Al reducir el volumen de salida, habrá menos parámetros que se requieran aprender (en los siguientes kernels de la red) y por ende se evita el sobreajuste [40].

Por otro lado, como ejemplo de la propiedad de ”casi” invariante a la traslación, se puede observar la figura 2.29.

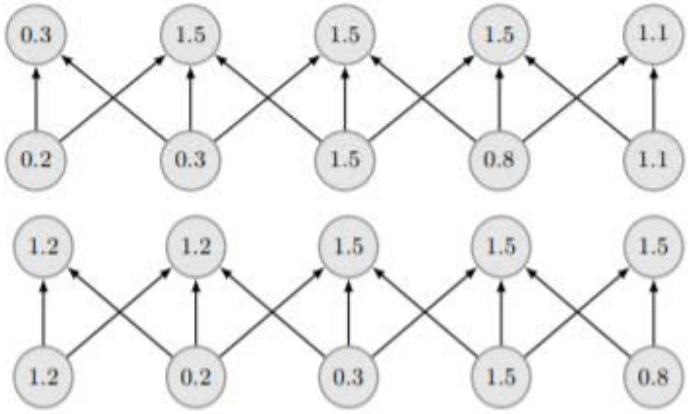


Figura 2.29: Se dispone de neuronas de entrada y salida. (Arriba) Se aplica max-pooling a las neuronas de entrada con un pooling kernel de ancho igual a 3 y un stride de 1. (Abajo) En cada neurona de la entrada se copia el valor de la neurona izquierda (traslación hacia la derecha) y se vuelve a aplicar la misma operación de pooling. Se observa que hubo una variación en 5 neuronas de entrada, producto de la traslación, y provocó una variación de solo 3 neuronas de salida [63].

Otra forma de reducir dimensiones es utilizar una convolución de 1 x 1 que opera sobre la profundidad de los mapas de características. Es decir, si se selecciona un kernel de 1 X 1 con una profundidad igual al volumen (3 dimensiones) de la entrada donde se está aplicando la convolución, se tendrá como salida una matriz de igual dimensión de la entrada pero con profundidad de 1. Este tipo de reducción de tamaño de profundidad (se observa en la arquitectura de Google: “Inception” [68]) se suele aplicar conjuntamente con las operaciones de max-pooling, logrando una reducción de tamaño en las 3 dimensiones (ancho, alto y profundidad).

Capa totalmente conectada

Después de una o varias capas de convolución (y pooling) con sus respectivas activaciones no lineales, es necesario agregar una capa capaz de obtener algún puntaje de clasificación para las clases que se encuentran en el conjunto de entrenamiento con el cual la red fue entrenada. La asignación de este puntaje estará dado por la capa totalmente conectada.

Flatten

La capa totalmente conectada necesita una entrada que corresponde a un vector (de una dimensión). No obstante, en la salida de una capa de convolución (o de pooling) podemos

obtener tensores de n dimensiones. Con lo cual, el objetivo de la operación “flatten” es convertir esta salida en un vector o tensor de 1 dimensión (Flattening) .

Capa totalmente conectada

Para obtener el puntaje de clasificación de las clases se utilizará el modelo de una red neuronal feedforward con capas totalmente conectadas (vistas en la sección 2.2). Con lo cual, la entrada de este modelo será el tensor de 1 dimensión (Flatten) explicado en el párrafo anterior.

Clasificación

Una vez obtenido el puntaje en cada una de las clases, es necesario llevarlo a una distribución probabilística para clasificar casos nuevos o para aplicar una función de pérdida en el entrenamiento de la red. Una de las operaciones más utilizadas para obtener la probabilidad en cada clase es aplicar la función softmax [69] a los puntajes obtenidos (y_i). Siendo K el número de clases existentes, se aplica la función softmax (también llamada función exponencial normalizada) visualizada en la siguiente ecuación:

$$S(y_i) = e^{y_i} / \sum_{j=0}^{k} e^{y_j}$$

Por medio de la fórmula anterior, convertimos el vector de salida (de longitud K) de la red totalmente conectada en un vector, también de longitud K, donde cada valor va a pertenecer al rango [0,1] y que sumados suman 1. Es decir se obtiene una distribución de probabilidad sobre las K diferentes clases.

En el caso de tener 2 clases, otra de las formas de conseguir la distribución de probabilidad es aplicar la función sigmoide mencionada en la sección 2.2.1.

2.3 Preprocesamiento

2.3.1 Espectro electromagnético

Las fuentes luminosas emiten diferentes longitudes de onda, en donde cada longitud de onda visible define un color. De todas las longitudes de onda existentes, el ser humano es capaz de visualizar sólo un subconjunto. Este se encuentra conformado por longitudes de ondas que van desde los 380 nanómetros que corresponde al violeta, hasta los 730 nanómetros que corresponde al rojo [70]. Esto se puede visualizar en la figura 2.30.

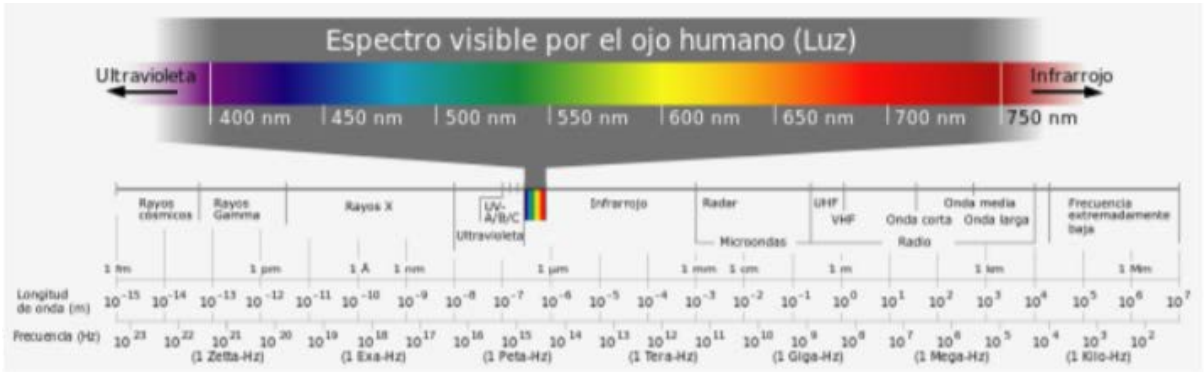


Figura 2.30: Espectro electromagnético visible e invisible por el ojo humano.

2.3.2 Conformación del color

La respuesta de una cámara o del sistema de visión humano a una escena está determinado por 3 factores: la iluminación del ambiente, la propiedad de reflectancia del objeto y la sensibilidad del sensor del sistema de visión. Considerando que la luz es uniforme en la escena, la formación del color en una imagen esta dado por el modelo simplificado [71] que corresponde a la siguiente ecuación:

$$\rho_k = \int_w I(\lambda)\, S(\lambda)\, C_k(\lambda)\, \partial\lambda$$

Donde ρ_k es la respuesta a una superficie de color expuesta a un rango de longitud de onda w (aproximadamente de 400 a 700 nanómetros). Esta respuesta es producida cuando una superficie con función de reflectancia $S(\lambda)$ (donde λ es la longitud de onda) es iluminada con una función de distribución espectral $I(\lambda)$. La luz reflejada de la superficie (llamada señal del color) es recibida por el sensor del sistema de visión cuya respuesta relativa espectral es $C_k(\lambda)$, resultando la percepción del color. Lo anteriormente explicado se puede observar en la figura 2.31.

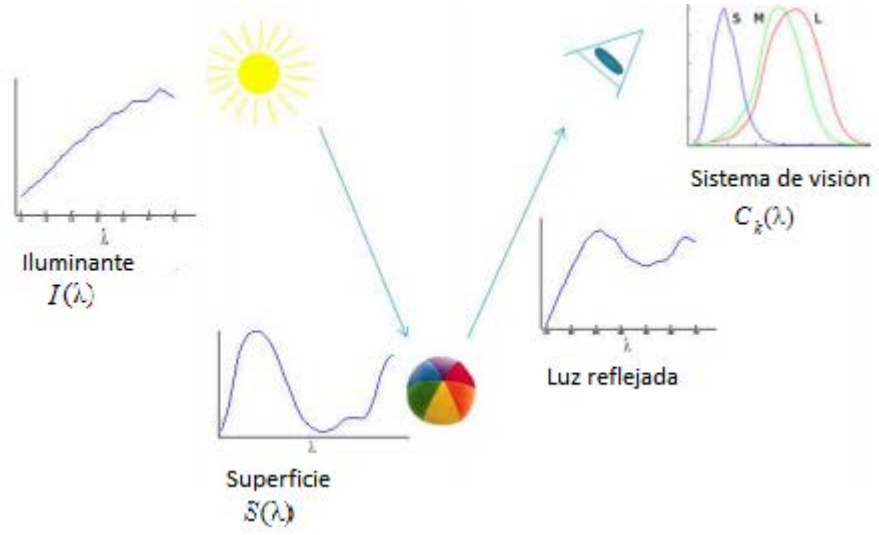


Figura 2.31: Formación del color.

2.3.3 Constancia del color

La constancia del color radica en un proceso inherente al ser humano en el cual consiste que la mayoría de las superficies mantienen la composición cromática ante cambios en la iluminación [71]. Es decir, los colores se siguen percibiendo de la misma manera como si estos son un atributo constante del objeto. Lo que sucede es que el ser humano realiza una corrección, cuando llega la luz refractada por un objeto, para visualizar siempre el mismo color.

Por otro lado, en lo que respecta a la visión por computadora, las imágenes capturadas por una cámara no realizan este proceso de constancia del color (Ver figura 2.32). Con lo cual, la fuente de luz tiene influencia en la conformación del color de la imagen capturada y por ende se trata de emular la corrección realizada por el sistema de visión humano mediante la aplicación de algoritmos. Existen muchas técnicas que abordan esta problemática, pero el algoritmo utilizado en el presente trabajo es el Max-RGB o “parche blanco” (white patch).

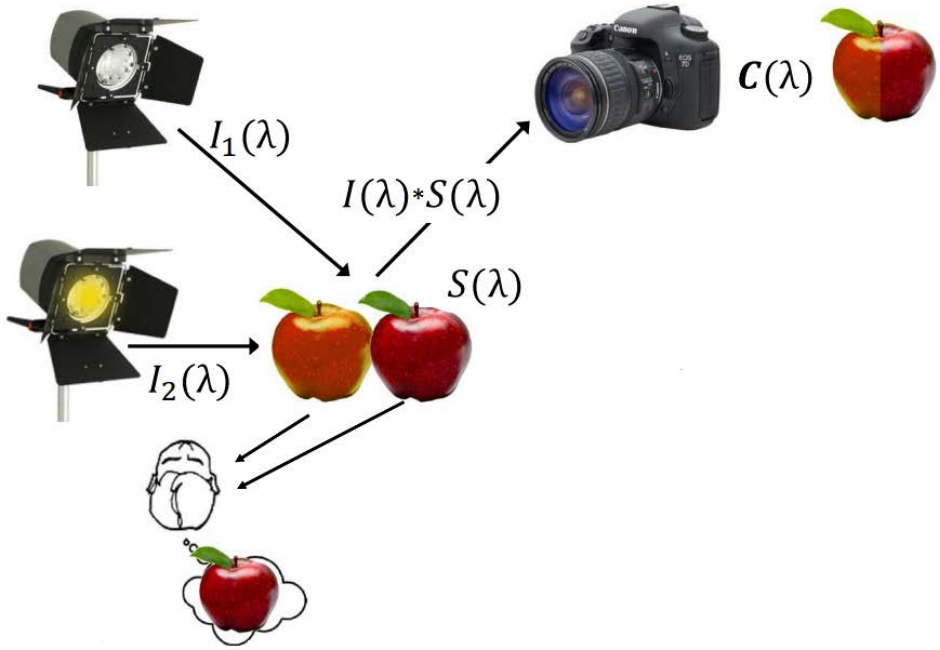


Figura 2.32: Corrección automática de la visión humana para que el color permanezca invariante frente a distintas fuentes de luz.

2.3.4 Algoritmo Max-RGB (White patch)

Según lo mencionado en la sección 2.3.3, se busca una estimación del iluminante para poder emular la corrección que realiza el sistema de visión humano. El algoritmo Max-RGB asume que la iluminación es uniforme a través de la escena. Además, este algoritmo se basa en la siguiente asumpción: “La máxima respuesta en los canales RGB es producto de una reflectancia perfecta. Una superficie con reflectancia perfecta reflejará en forma exacta el rango de luz que captura”. En consecuencia el color de esta reflectancia perfecta es exactamente el color de la fuente de luz. En la práctica, la reflectancia perfecta es calculada tomando cada canal en forma independiente y calculando la máxima intensidad en cada uno de ellos, resultando en el algoritmo Max-RGB dado por la siguiente ecuación:

$$I_{Imax} = max_a \{f_i(x,y)\} \tag{2.30}$$

Donde $f_i(x,y)$ es la intensidad del píxel en la posición (x,y) e i indica el canal del color.
A este vector calculado también se lo denomina blanco hipotético y hace referencia a la teoría de retinex [72], en la cual establece que existe un parche blanco (white patch) en la escena que va a reflejar la máxima cantidad de luz perteneciente al iluminante.

Luego, en la ecuación 2.31 se normalizan todos los píxeles con la estimación del iluminante calculada en la ecuación 2.30.

$$o_i(x,y) = \frac{f_i(x,y)}{I_{Imax}} \tag{2.31}$$

La ecuación 2.31 es la que se usó en las imágenes de las lesiones pigmentadas como método de preprocesamiento para eliminar la distorsión del color producido por la fuente de luz.

2.4 Segmentación

2.4.1 Segmentación semántica y por instancia

Una de las tareas que se realizó en el presente trabajo es segmentar las lesiones pigmentadas (por medio de redes neuronales) que servirán de entrada al Clasificador 3 (explicado en las secciones 2.5.3 y 3.3.4). El tipo de segmentación utilizado es el que se denomina **segmentación semántica** ya que se realiza la misma “clasificando” cada píxel que corresponde a la imagen de entrada (ver figura 2.33). Es decir, la salida de la red debe ser otra imagen, llamada máscara, con las mismas dimensiones de ancho y alto que la imagen de entrada. El valor de cada píxel de la máscara en la posición (x_i, y_i) representa la clase a la que pertenece el píxel en la imagen de entrada que se encuentra en la misma posición (x_i, y_i) [73]. Debido que la segmentación se realiza clasificando los píxeles, se la suele denominar “predicción densa” [74].

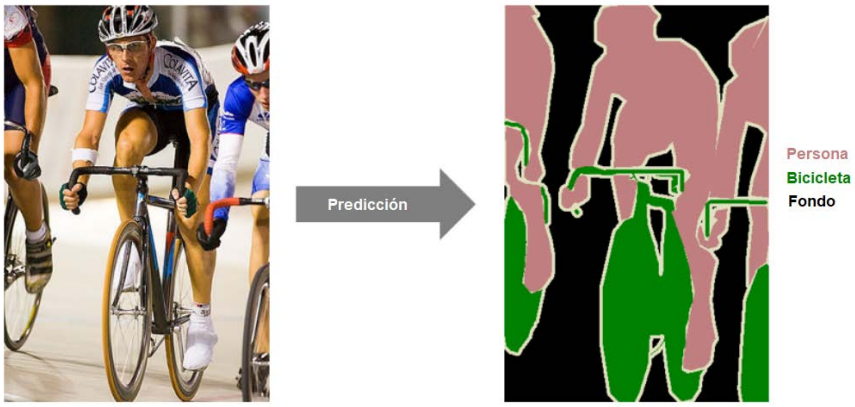


Figura 2.33: Segmentación semántica de una imagen.

Con lo cual, dada una imagen color de dimensiones W x H x 3 o una una imagen en escala de grises de dimensiones W x H x 1, vamos a obtener una máscara cuyas dimensiones es W x H en donde cada píxel tendrá un color asociado a la clase que representa [40]. Por último cabe aclarar que no se realiza una diferencia entre objetos de una determinada categoría (las personas o bicicletas en la figura 2.33). Es decir, este tipo de segmentación semántica no los distingue como objetos separados. Existen otros tipos de modelos, conocidos como **segmentación por instancia** [74] que si hacen esta distinción. No obstante, este tipo de segmentación no se encuentra abordada en la presente tesis ya que en las imágenes que disponemos existe una sola lesión pigmentada por imágen.

2.4.2 Técnicas

Segmentación por patches

En este tipo de técnica se desliza una ventana por toda la imagen y por cada patch (mapeo de la ventana con la imagen) que se extrae, se lo ingresa a la CNN y se predice a qué clase pertenece el píxel central de dicho patch. El principal inconveniente de este método es que requiere una gran potencia de cómputo ya que se tiene que realizar la predicción de todos los píxeles de la imagen de entrada en forma independiente. Por ende, para obtener la máscara, se deberán tomar tantos patches como píxeles tenga la imagen de entrada para realizar la respectiva clasificación del píxel central. No obstante, utilizando este enfoque se han desarrollado algunos trabajos prometedores [75][76]. Lo anteriormente explicado puede visualizarse en la figura 2.34.

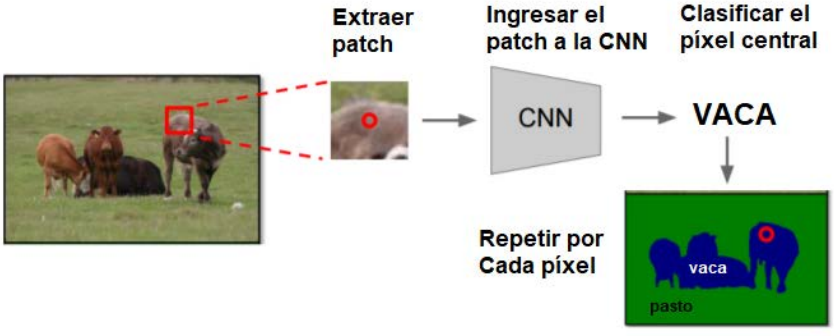


Figura 2.34: Segmentación por patches.

Segmentación directa

En este enfoque, la entrada de la CNN no corresponderá a un patch, sino a la imagen completa. Además, la salida de la red no será la predicción de un píxel (central) como en el caso de la segmentación por patches, sino una máscara. Dicha máscara, como se mencionó anteriormente, tendrá la mismas dimensiones (ancho y alto) que la imagen de entrada.

Para utilizar este enfoque podríamos pensar en una red con una serie de capas convolucionales en donde, por medio del padding, mantendriamos las mismas dimensiones de la imagen de entrada. Con lo cual, estaríamos realizando la predicción de todos los píxeles en forma simultánea. Sin embargo, realizar convoluciones manteniendo la resolución total de la imagen requiere una elevada potencia de cómputo [77]. Esto último se puede visualizar en la figura 2.35 donde H es la altura de la imágen, W es el ancho de la imágen, D es el número de kernels apilados y C el número de clases.

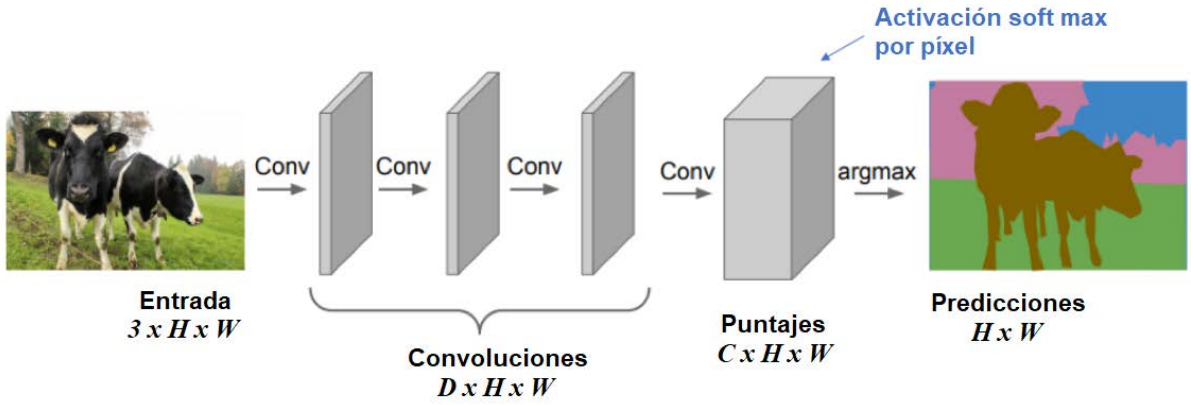


Figura 2.35: Segmentación semántica directa de la imagen por medio de capas convolucionales manteniendo las dimensiones de dicha imágen. Requiere una elevada potencia de cómputo.

A continuación se describe una arquitectura del tipo encoder-decoder que minimiza el problema del costo computacional mencionado en este tipo de segmentación.

2.4.3 Encoder-Decoder

Una solución para evitar los problemas de rendimiento que se producen en la arquitectura de la figura 2.35 es seguir una estructura del tipo encoder-decoder (ver figura 2.36). En donde el encoder será el encargado de reducir la resolución de los mapas características de la misma forma que lo hacen las redes convolucionales que efectúan la tarea de clasificación vistas en la sección 2.2.5. La diferencia radica que en el encoder se elimina el bloque de las capas totalmente conectadas. Con lo cual obtenemos un mapa de características de baja resolución y altamente eficiente para la separación de las clases. De esta forma, la red opera en volúmenes de salida inferiores a los que existen en la arquitectura de la figura 2.35, lo que se traduce en necesitar menos potencia de cómputo [40]. Luego se encuentra el decoder que será el encargado de convertir este mapa de baja resolución a otro de alta resolución, con las mismas dimensiones que la imagen de entrada (obteniendo la predicción por píxel que estamos buscando).

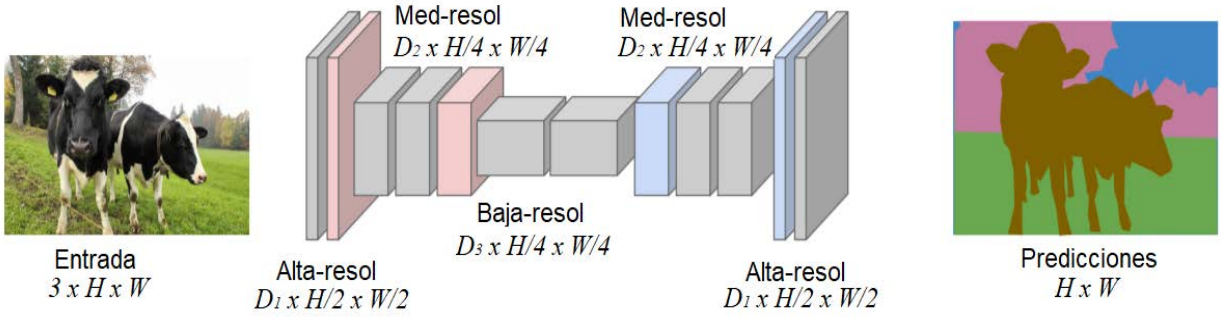


Figura 2.36: Segmentación semántica a través de un encoder-decoder

Sin embargo se plantean ciertas dificultades al pasar de un mapa de baja resolución a otro de alta resolución. A continuación se describen las operaciones que se efectúan en el decoder para solventar esta problemática.

Métodos de upsampling

Existen diferentes métodos para pasar de un mapa de baja resolución a otro de alta resolución. En lo que respecta a las redes neuronales convolucionales, los siguientes 2 métodos son ampliamente utilizados.

Unpooling

Hemos visto que la operación de pooling reduce la resolución de un mapa mediante la aplicación de una estadística (promedio, máximo, etc) a un campo receptivo de entrada. Luego existe la operación unpooling, en donde cada valor de la entrada se distribuye, con alguna lógica o cálculo, en un espacio de resolución más amplio. En la figura 2.37 se observan los 3 métodos. En el caso del “Nearest Neighbor” se repite el valor de la entrada en la salida K veces, siendo K el ratio entre la cantidad de píxeles que tiene la salida y la cantidad de píxeles que tiene la entrada. En el “Bed of Nails” se coloca cada valor de la entrada en la región de K píxeles que le corresponde, y los restantes píxeles en 0. Por último, en max-unpooling se guardan los índices de los máximos encontrados en la operación de max-pooling “simétrica” anterior, para luego colocar los nuevos valores de entrada en los índices de las posiciones guardadas pero en el nuevo espacio de alta resolución.

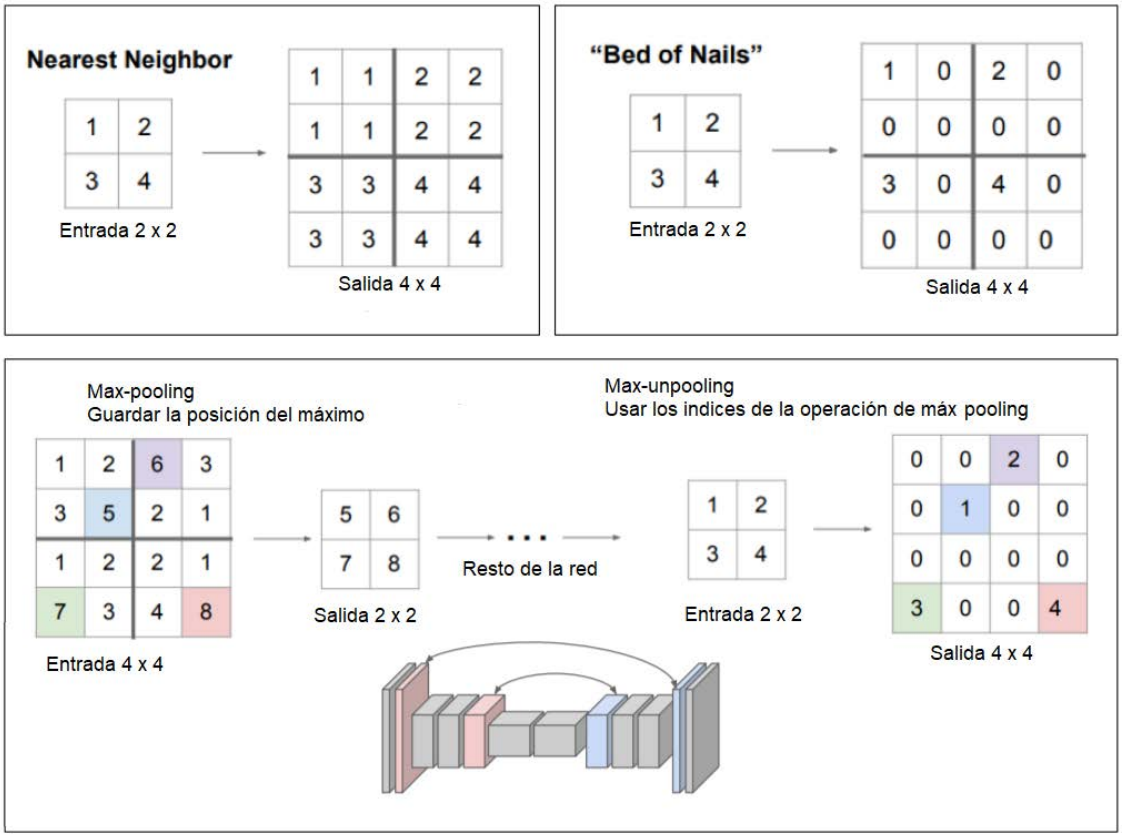


Figura 2.37: Operación de unpooling en el decoder.

Convolución transpuesta (upconvolutional, deconvolutional)

Se explicó que una operación de convolución, en el ámbito de las redes convolucionales, consiste en el producto escalar entre el vector de entrada (campo receptivo) y el kernel respectivo, produciendo un valor en una posición del mapa de características. En una convolución transpuesta [78] se hace justamente lo contrario, se toma un valor de la entrada (mapa de baja resolución) y se lo multiplica por los pesos del kernel, proyectando luego esos resultados en el mapa de características de más alta resolución. Lo anterior puede visualizarse en la figura 2.38 donde los diferentes colores de la salida indican los valores de la entrada a los que hace referencia. Además, se deberán sumar los valores en los cuales el kernel se superpone con otros valores calculados (en pasos anteriores) en la salida. Esto último produce efectos no deseados [79], generando patrones del tipo "mosaico" o "tablero" (ver figura 2.39). Con lo cual se trata de tener determinados tamaños en las dimensiones de la entrada y el kernel, de forma que no produzcan este fenómeno. Además, se observa en la figura 2.38 que por cada paso que damos en la entrada, el kernel efectúa dos pasos en la salida. Por esta razón, otro de los nombres que tiene esta operación es "convolución con stride fraccionario", en este caso por cada ½ paso que damos en la entrada realizamos un paso en la salida.

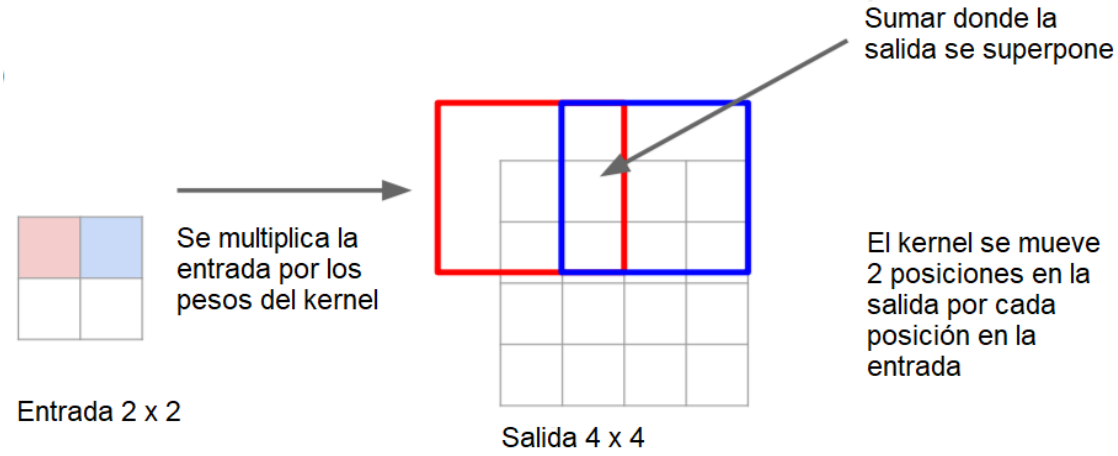


Figura 2.38: Convolución transpuesta con una entrada 2x2, un kernel de 3 x 3, un stride igual a 2 y un padding igual a 1 para generar un mapa de características con una resolución más alta que la entrada.

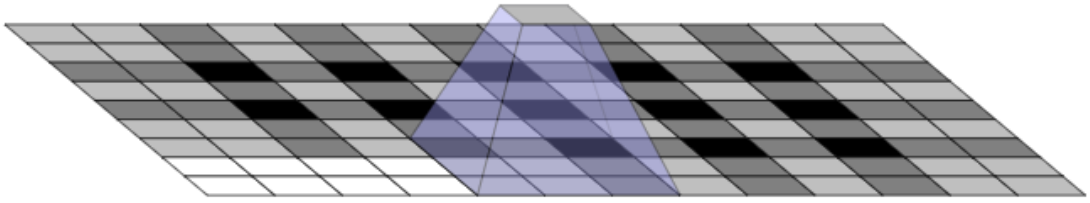


Figura 2.39: Generación de efectos no deseados del tipo “mosaico” o “tablero” en la convolución transpuesta.

Por último, en la figura 2.40 se observa un ejemplo de una convolución transpuesta.

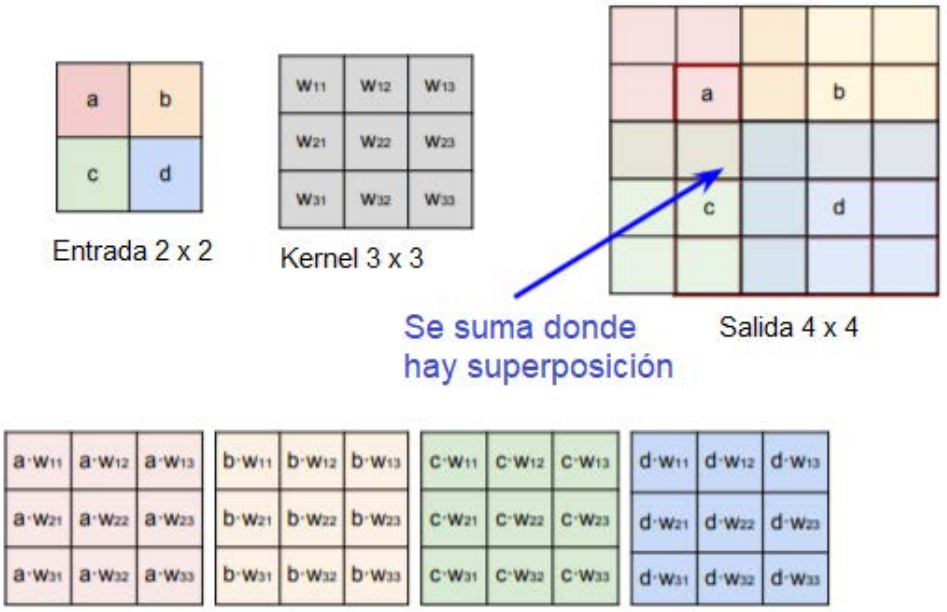


Figura 2.40: Ejemplo de una convolución transpuesta con una entrada de 2x2, un kernel de 3x3, un stride de 2 y un padding igual 1.

Los pesos que componen el kernel se aprenden durante el entrenamiento de la red por medio del backpropagation con el objetivo de minimizar la función de pérdida. Esto último lo convierte en un método muy utilizado de upsampling en las redes convolucionales, ya que existen otros como la interpolación lineal o mediante splines pero poseen parámetros fijos en relación a la minimización de la función de pérdida.

Otra de las ventajas que tiene la operación de convolución transpuesta es que su implementación es trivial. A la hora de realizar una convolución tradicional en una red, esta se implementa como una multiplicación de matrices donde a la imagen de entrada I se la puede codificar en un vector de $1 \text{ x } d$ (siendo d la cantidad de píxeles totales), al kernel se lo codifica como una matriz de convolución C de dimensiones $d \text{ x } m$, siendo m la cantidad de valores resultantes en la matriz de salida (de dimensión $1 \text{ x } m$). Esta matriz de salida será el mapa de características M_{sal}. Cuando se aplica el backpropagation, y estamos en la operación de backward, el gradiente tiene que fluir (por las distintas capas) por lo tanto tendrá que pasar del mapa de características M_{sal} a la imagen de entrada I previamente mencionada. Para realizar esta operación se multiplica la transpuesta de C por M_{sal} obteniendo una matriz de dimensión $1 \text{ x } d$ (mismas dimensiones que I) [77]. Con lo cual, en el caso de la implementación de la operación de convolución transpuesta se puede realizar invirtiendo el orden de aplicación entre el forward y el backward con respecto a la operación de convolución explicada en la sección 2.2.5 [80].

2.4.4 Red neuronal completamente convolucional (FCN)

Este tipo de red [80] (Fully Convolutional Neural network-FCN) es la primera en adaptar la CNN tradicional para que produzca una predicción densa (un vector de probabilidades por cada píxel, donde cada probabilidad corresponde a una clase). Para lograr este mapa de probabilidades, se eliminan las capas totalmente conectadas que cumplian la función de clasificar una imagen (ver figura 2.41). Esta red se basa en los modelos del tipo encoder-decoder, donde para el encoder utilizaron la arquitectura de una red bien conocida como es AlexNet [81] y como decoder emplearon las operaciones de la convolución transpuesta mencionada en la sección 2.4.3. Su nombre se debe a que se basa en capas del tipo convolucional, pooling o de upsampling. Como se mencionó en la sección 2.2.5, al basarse en redes convolucionales, los parámetros se reducen drásticamente (por ende el tiempo de cómputo) y puede procesar imágenes de distintos tamaños.

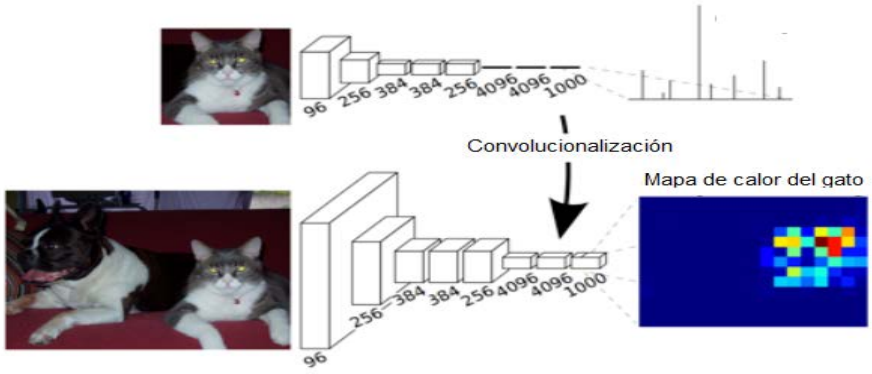


Figura 2.41: (Arriba) Modelo original de la arquitectura AlexNet, en donde la salida es un vector con las probabilidades de que la imagen pertenezca a cada clase. (Abajo) Fully Convolutional Neural network (FCN), en donde su salida es una predicción densa (mapa de probabilidades).

Skip connections

Debido a que el encoder realiza una reducción de la imagen de entrada por un factor de 32, si se realiza un solo upsampling para llegar al mapa de alta resolución (igual dimensión que la imagen de entrada) se produciría un salto muy grande. Esto acarrea una baja precisión en la segmentación que se obtenga (y es lo que sucede en la red llamada FCN-32s). Los autores, al tomar nota de esta casuística implementaron las “skip connections” [30]. Estas conexiones permiten combinar mapas de características obtenidos en distintos niveles de la red. De esta forma, se podrán combinar mapas con información semántica y espacial [80]. La combinación de estos mapas tiene como efecto que el upsampling que se realice produzca segmentaciones más nítidas.

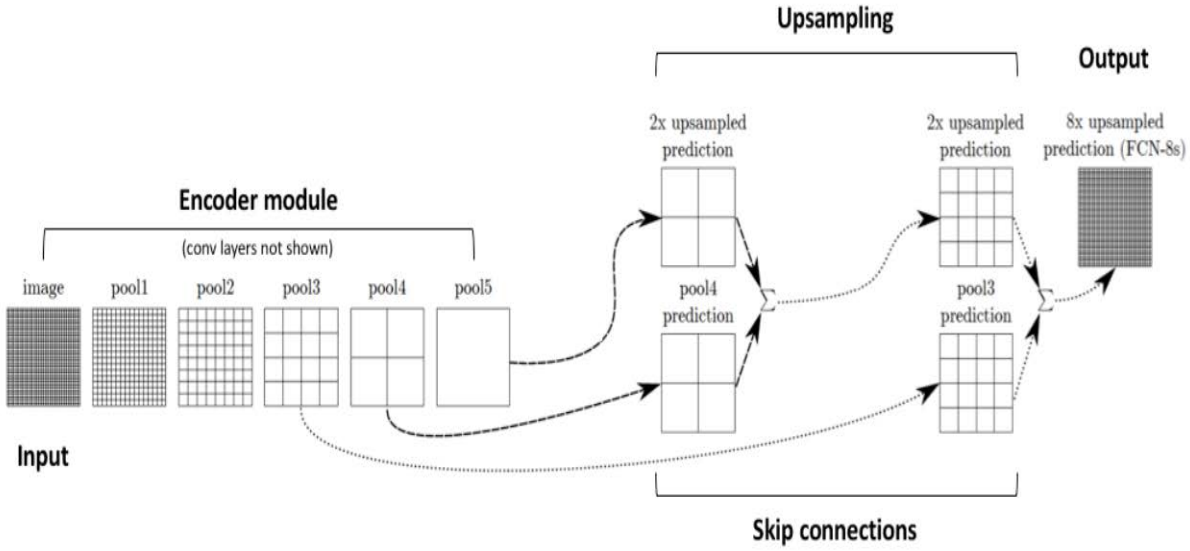


Figura 2.42: Variante de la FCN que emplea skip connections de 2 mapas de características anteriores.

A modo de ejemplo en la figura 2.42 se visualiza una variante de la FCN que emplea skip connections, llamada FCN-8s. Además, se observan las capas de pooling del encoder y se omiten las capas convolucionales. Se recuerda que para que se puedan combinar 2 mapas de características, estos deben tener las mismas dimensiones. Por tal razón, a los mapas de la capa pool5 se le aplica una operación de upsampling con un stride de 2 (obteniéndose mapas de características del doble de tamaño) para luego combinarse con los mapas de características de pool4 y poder realizar el upsampling sobre la combinación de estos 2 conjuntos de mapas. Por ende, en esta combinación de mapas se aplica un upsampling con un stride de 2 y el resultado que se obtiene se combina con los mapas de características de pool3. Luego, a esta última combinación de mapas se le aplica un upsampling con un stride de 8, para lograr las dimensiones de la imagen de entrada. Se aclara que en este proceso se aumentó la resolución de la salida de pool5 en 32 veces (2 x 2 x 8) pero en forma progresiva, logrando una segmentación más nítida que en la FCN-32s [80]. En la figura 2.43 se observa la segmentación de una imagen para distintas variantes de la red FCN, entre ellas la FCN-8s y la FCN-32s.

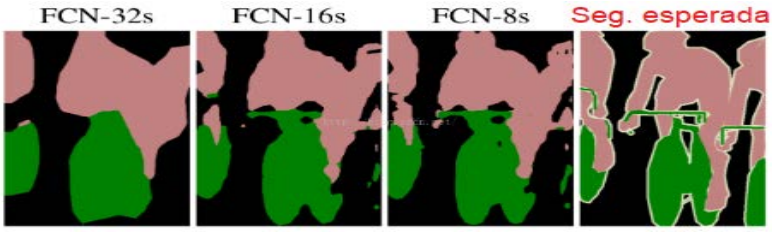


Figura 2.43: Segmentaciones obtenidas utilizando distintas skip connections. FCN-32s no utiliza la información de capas previas. FCN-16s utiliza la información de una capa anterior. FCN-8s utiliza la información de dos capas anteriores.

2.4.5 Arquitectura U-Net

La arquitectura U-Net [82] resulta de una modificación a la red FCN explicada en la sección 2.4.4, realizando una expansión de la capacidad del decoder. Los autores (Olaf Ronneberger, Philipp Fischer y Thomas Brox) llaman “contracting path” al encoder mientras que “expanding path” al decoder. Tanto el contracting path como el expanding path pueden observarse en la figura 2.44. En el paper original, esta arquitectura se utilizó con 3 conjuntos de datos relacionados a imágenes de células y se empleó la cross-entropy como función de pérdida. No obstante, los píxeles fueron ponderados de acuerdo a la distancia que tenían con respecto a los bordes que separan las células (cuanto más cerca, más peso tenían). A continuación se explica la arquitectura de esta red.

Contracting path (Encoder)

El contracting path posee las típicas capas de una red convolucional. Definimos un bloque (ver bloques de líneas punteadas de la figura 2.44) del contracting path a la secuencia compuesta por los siguientes 2 pasos:

1. Dos capas convolucionales (en forma secuencial), donde las salidas de cada una de ellas es el resultado de realizar operaciones de convolución con zero-padding, un kernel 3 x 3 y un stride igual a 1. Todas las operaciones de convolución son seguidas por una función de activación ReLU.
2. Una capa de pooling con kernel 2 x 2, zero-padding y un stride de 2 (con el objetivo de reducir a la mitad la resolución de los mapas de características).

La secuencia anterior comienza con la imagen de entrada (de 572 x 572) y se repite 4 veces, donde el resultado de una secuencia es la entrada de la siguiente. Las capas convolucionales de cada secuencia poseen el doble de los canales que las que tienen las capas de convolución “simétricas” de la secuencia anterior. Con lo cual, la arquitectura del contracting path va a estar formada por 4 bloques (cada uno de ellos con una secuencia de operaciones). En la salida del último bloque se obtiene el mapa de características de más baja resolución de 32 x 32 píxeles, lo que equivale a un vector de 1024 posiciones y que en el contexto de los encoder-decoder se denomina espacio latente (latent space) [83].

Expanding path (Decoder)

Se parte del espacio latente y se aplican (en forma secuencial) 2 capas convolucionales (cada una de ellas con 1024 canales) creando un pequeño mapa de características de 28 x

28. Cada operación de convolución (de estas 2 capas convolucionales) posee un kernel 3x3, zero-padding, un stride igual a 1 y es seguida de una función de activación ReLU. Luego, definimos un bloque (ver bloques de líneas punteadas de la figura 2.44) del expanding path como una secuencia compuesta por los siguientes 3 pasos:

1. Se aplica una capa de upsampling y otra de convolución en forma secuencial. La capa de upsampling (cuya salida es el resultado de aplicar operaciones de upsampling donde cada una posee un kernel de 2 x 2 y un stride igual a 1) lleva los mapas de características al doble de su dimensión. La capa de convolución tiene la mitad de los canales que existen en los mapas donde se aplicó la operación de upsampling, y la salida es el resultado de aplicar convoluciones (que tienen un kernel de 2 X 2, zero-padding y un stride igual a 1) seguidas por una función de activación ReLU.
2. Al resultado anterior se lo concatena con las capas “simétricas” del encoder (vía la skip connection), re-cortándolo previamente para que se pueda realizar dicha concatenación, es decir, ambos mapas concatenados deben tener el mismo tamaño en las dimensiones.
3. Una vez que se concatenaron los mapas, se aplican 2 capas convolucionales (en forma secuencial) que tendrán $N/2$ canales, siendo N igual al número de canales que surge de la concatenación de los mapas del paso 2. Cada operación de convolución de estas 2 capas tiene zero-padding, un kernel 3 x 3, un stride igual a 1 y cada una de ellas es seguida por una función de activación ReLU.

La secuencia anterior comienza en el mencionado mapa de características de 28 x 28 y se repite 4 veces, donde el resultado de una secuencia es la entrada de la siguiente y en cada secuencia se aplica la mitad de los canales que se aplicaron en las capas “simétricas” de la secuencia anterior. Con lo cual, la arquitectura del contracting path va a estar formada por 4 bloques. La salida del último bloque contendrá 64 canales y se aplicará C veces una convolución de 1 x 1 (siendo C el número de clases), mapeando los 64 canales a C canales.

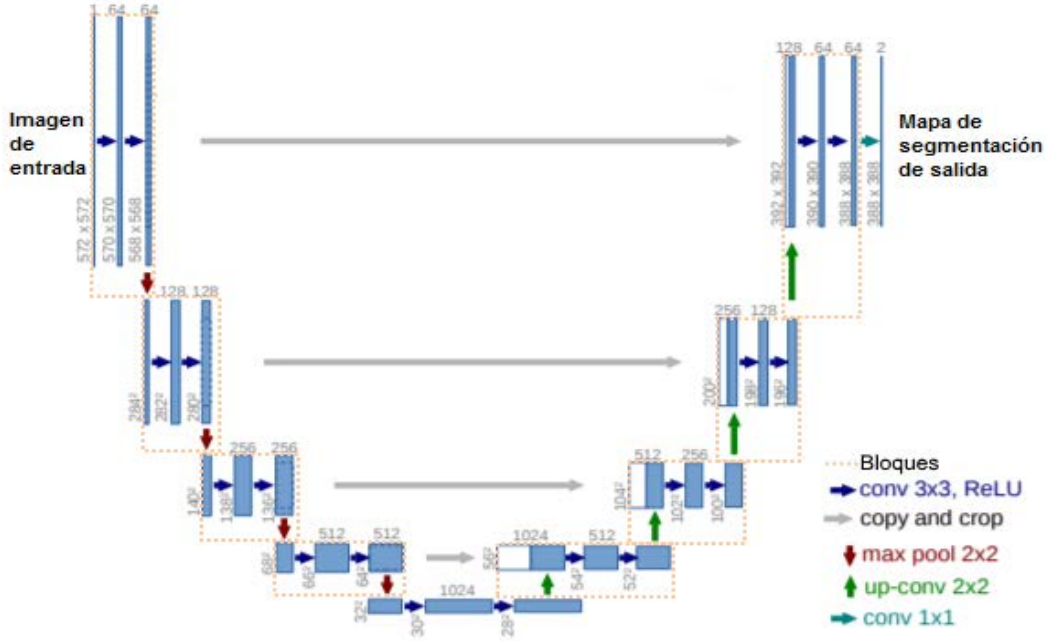


Figura 2.44: Las cajas azules denotan los mapas de características. Arriba de las cajas se encuentra el número de slices o canales. El tamaño del mapa de características ($w\ x\ h$) se encuentra en el extremo inferior izquierdo de cada caja.

Los autores comprobaron que esta arquitectura tiene un mejor rendimiento que la FCN, particularmente cuando se tienen pocos ejemplos en el conjunto de datos de entrenamiento. Esto último es muy valioso ya que, en lo que respecta a la segmentación por medio de las arquitecturas de deep learning, es habitual tener pocos patrones en el conjunto de entrenamiento debido a la dificultad de obtener las máscaras de cada imágen. Por último, los autores verificaron que el “Data augmentation”, al menos para los conjuntos de datos que ellos tenían disponible, realizó una mejora en la segmentación final obtenida.

A medida que recorremos los 4 bloques del contracting path, observamos que el número de canales se multiplica por 2 y los tamaños de las dimensiones de los mapas de características son reducidos (dividiéndolos por 2) hasta llegar al espacio latente. Por otro lado, cuando recorremos los 4 bloques del expanding path, el número de canales es dividido por 2 y las dimensiones de los mapas de características son multiplicadas por 2 hasta llegar a la salida del cuarto bloque. Esto permite realizar la concatenación de los mapas simétricos (por medio de las skip connections) entre el contracting path y el expanding path. Por ende, la red tiene la forma de “U” y por tal motivo recibe el nombre de U-Net.

Existen modificaciones que se realizan a la arquitectura U-Net. Por ejemplo, como encoder o contracting path se puede tomar otra red bien conocida, como es la VGG-16 (para más detalles, ver la explicación de su arquitectura en la sección 2.5.1) sin las capas totalmente conectadas. Luego, una vez obtenido el espacio latente, se sigue la misma forma de “U” para que las dimensiones de los mapas de características situados en el decoder tengan su respectivo mapa simétrico del encoder. Este tipo de variación corresponde a la arquitectura llamada TernausNet-16 [84] visualizada en la figura 2.45 y es la que se utilizó en el presente trabajo.

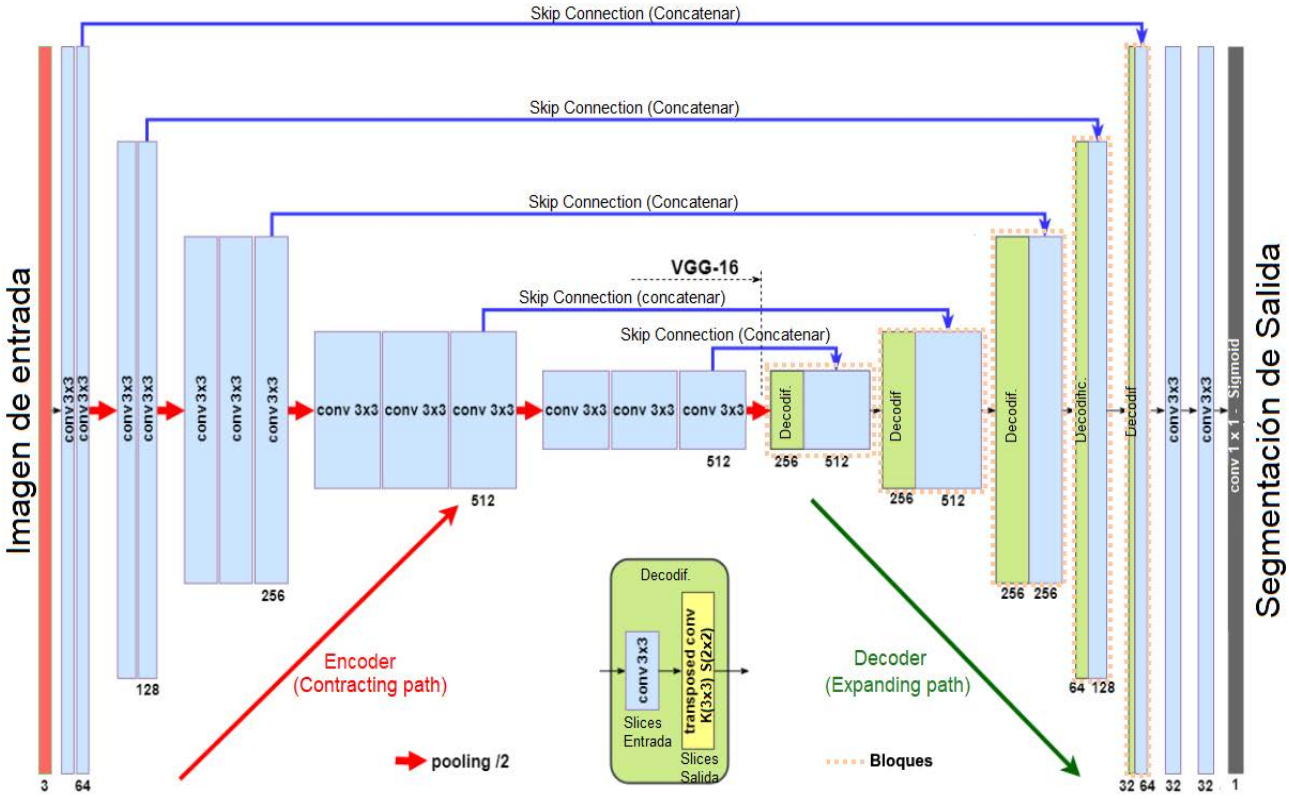


Figura 2.45: TernausNet-16. Utilización de las capas de la red VGG-16 en el encoder y la modificación respectiva en el decoder para que siga el lineamiento de la U-Net.

Contracting path (Encoder)

En el contracting path (ver figura 2.45), los rectángulos azules corresponden a los mapas de características obtenidos en cada convolución. Las dimensiones de estos rectángulos azules hacen referencia al volumen “ancho x alto x profundidad”. Es decir, la base del rectángulo hace referencia a la “profundidad” de la capa y la altura a las dimensiones “ancho y alto” (ya que son iguales). Los números que se encuentran debajo de cada rectángulo corresponden al número de canales (profundidad) que se obtienen en cada capa. Las operaciones de convolución se efectúan con un kernel 3 x 3, un padding igual a 1, un stride igual a 1 y son seguidas de una activación ReLU. Las operaciones de max-pooling se efectúan con un kernel de 2 x 2, zero-padding y un stride igual a 2. Estas operaciones (max-pooling) reducen las dimensiones (ancho y alto) de los mapas de características a la mitad y la última capa de pooling constituye el espacio latente (de 7 x 7). Como se mencionó anteriormente, las capas del contacting path pertenecen a la arquitectura VGG-16 (sin las capas totalmente conectadas) y la misma será explicada con más detalle en la sección 2.5.

Expanding path (Decoder)

En el expanding path (ver figura 2.45) se encuentra el decodificador señalado en color verde y compuesto por las siguientes 2 capas conectadas en forma secuencial:

1. Una capa de convolución cuyos mapas son obtenidos con un kernel 3 x 3, un padding igual a 1 y un stride igual a 1. Además el número de canales que posee es igual al que existe en el volumen de entrada en el cual se aplican las operaciones de convolución.
2. Una capa de upsampling cuyos mapas corresponden a las operaciones de convolución transpuesta con un kernel de 3 x 3, un padding igual a 1 y un stride igual a 2, con lo que duplica el tamaño de las dimensiones de los mapas de características de la entrada.

Los números que se encuentran debajo de cada rectángulo verde corresponden al número de canales (profundidad) que se obtienen a la salida del decodificador. Definimos un bloque al mapa de características de la salida del decodificador concatenado con los mapas que pertenecen a la capa “simétrica” del encoder por medio de las skip connections (ver bloques de líneas punteadas de la figura 2.45). Entonces el expanding path tendrá 5 bloques, donde se parte del espacio latente y la salida de cada bloque corresponde a la entrada del siguiente. Esto último hace que se respete la filosofía de la forma de “U” que tiene que tener la red para seguir el lineamiento de la U-Net. Una vez que se obtiene la salida del quinto bloque, se tienen 3 capas convolucionales. Las primeras 2 poseen kernels de 3 x 3, un padding igual a 1 y un stride igual a 1. La tercera capa de convolución posee kernels de 1 x 1, zero-padding y un stride igual a 1. Se aclara que todas las operaciones de convolución son seguidas de una función de activación ReLU, salvo la última operación de convolución que es seguida por la función sigmoide (ya que es la capa clasificadora y disponemos de solo 2 clases: “Melanoma” y “No melanoma”).

2.5 Clasificación

2.5.1 Arquitectura VGG-16

La red VGGNet [8] (o DCNN, Deep Convolutional Neural Network) es creada por VGG (Visual Geometric Group) de la universidad de Oxford. Su presentación tuvo lugar en ILSVRC (ImageNet Large Scale Visual Recognition Competition) en el año 2014. En la tarea de localización de objetos dentro de una imágen, la red VGGNet fue la ganadora. No obstante, en lo que respecta a la tarea de clasificación, la red Inception V1 [68] de google fue la ganadora superando levemente a la red VGGNet. Sin embargo esta última tuvo grandes avances con respecto a la red ganadora del ILSVRC en el año 2013 (red ZFNet [85]) y a la ganadora del ILSVRC del año 2012 (red AlexNet [81]). Por otro lado, existen trabajos anteriores que tuvieron un muy buen rendimiento utilizando esta arquitectura e imágenes de lesiones pigmentadas similares (de la competencia ISIC 2016). Por ejemplo, citamos el trabajo de Lopez Adria Romero, Xavier Giró-i-Nieto, Jack Burdick y Oge Marques [86]. Por lo mencionado anteriormente, se seleccionó la red VGG-16 para ser utilizada como clasificadora de imágenes y como extractora de características. A continuación se explica su arquitectura y dado que se realizará transfer learning, también se explicará el entrenamiento realizado (por los creadores de esta arquitectura) con el conjunto de datos ImageNet.

Configuraciones

Existen varias configuraciones de la VGGNet, las mismas son mostradas en la figura 2.5.1.1. Por cada configuración, se muestra el número de parámetros (expresado en millones [M]) y el porcentaje de error que se obtiene al utilizar el conjunto de datos de ImageNet. Se observa que al aumentar el número de capas disminuye el porcentaje de error hasta que se llega a la configuración de la VGG-16. Luego al incorporar más capas (VGG-19) el porcentaje de error casi permanece igual, aumenta 2 décimas. Sin embargo, el número de parámetros se incrementa en 6M (144M-138M). Por este motivo cuando se refiere a la VGGNet, generalmente se hace referencia a la VGG-16 o a la VGG-19. En la presente tesis se utilizó la configuración VGG-16 (sombreada en la figura 2.46) con un kernel 3 x 3 en todas las operaciones de convolución.

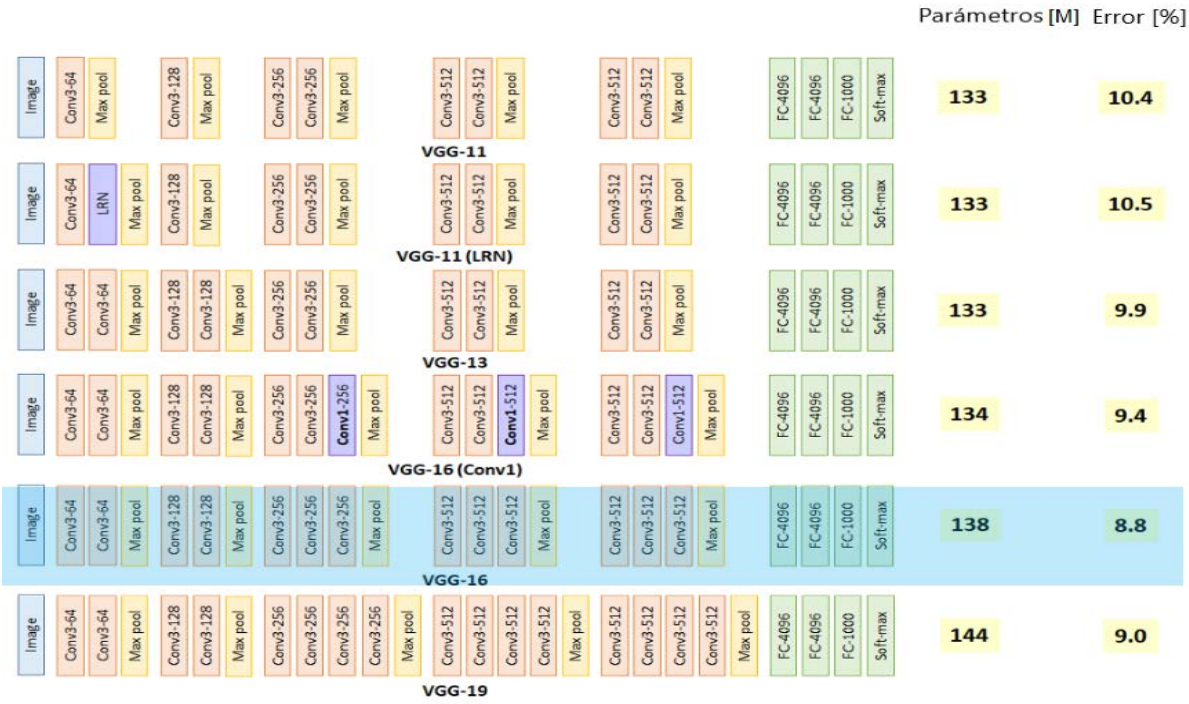


Figura 2.46: Configuraciones de la VGGNet. Se indica el número de canales en cada capa convolucional y en cada capa totalmente conectada.

Arquitectura

El entrenamiento de esta red se realizó con las imágenes de ImageNet fijadas en una resolución de 224 x 224 píxeles. Como se observa en la figura 2.47, existen 6 bloques, donde los primeros 5 están compuestos por 2 o 3 capas convolucionales seguidas de una capa de pooling. Mientras que el último bloque posee 3 capas totalmente conectadas. Los kernels utilizados en todas las operaciones convolucionales son de 3 x 3, es decir, el menor tamaño de campo receptivo para tomar la información alrededor de un píxel central. Además, se utiliza un único stride para todas las operaciones convolucionales fijado en 1. El padding se tomó de tal forma que la resolución espacial se preserva después de una operación de convolución. Con lo cual, para que esto suceda con esta arquitectura se fijó en 1. Todas las operaciones convolucionales son seguidas de la función de activación no lineal ReLU. Además, las salidas de las capas convolucionales de cada bloque poseen el doble de canales con respecto a las salidas existentes de las capas convolucionales del bloque anterior (a excepción de los bloques 4 y 5 que poseen el mismo número de canales - ver figura 2.46). Con respecto a las 5 capas de pooling presentes en esta arquitectura, se fijó un kernel de pooling de 2 x 2 con un stride fijado en 2. En la figura 2.47 se observa que (a medida que se avanza en la arquitectura) en la salida de los primeros 4 bloques se reduce la dimensionalidad de los mapas de características por un factor de 2 y se duplica el número de canales. De esta forma se obtiene un balance entre la reducción de la dimensión y el aumento de la profundidad. Luego de la salida de la última capa de pooling, existen 3 capas totalmente conectadas. Las primeras 2 poseen 4096 neuronas y la última 1000 neuronas, una por cada clase. A la salida de la red se aplica la función de activación Softmax (vista en la sección 2.2.5) para calcular la probabilidad de que las imágenes de entrada pertenezcan a cada una de las 1000 clases. Por último, la red fue entrenada utilizando SGD (Stochastic Gradient Descent) y el algoritmo backpropagation.

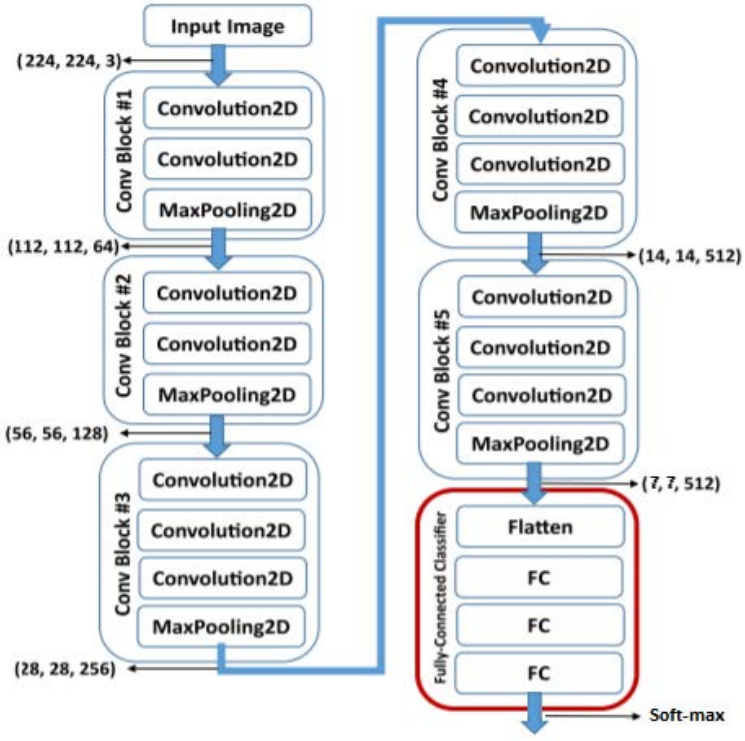


Figura 2.47: Arquitectura de la VGG-16 para la configuración de un kernel 3 x 3. A la salida de cada bloque se indica una 3-tupla donde los primeros 2 valores corresponden al ancho y al largo de cada mapa de características, mientras que el tercer valor es el número de canales.

Operaciones convolucionales en forma secuencial con kernels 3 x 3

Como se vió en la sección 2.2.5, por medio de las conexiones dispersas, una neurona en una capa profunda se conectará indirectamente con una gran cantidad de neuronas que pertenecen a las primeras capas de la arquitectura. Al campo receptivo “indirecto” se lo denomina campo receptivo efectivo. Luego, al tener 2 operaciones convolucionales secuenciales con kernels de 3 x 3, una neurona en el mapa de características resultante de la segunda operación de convolución, tendrá un campo receptivo efectivo de 5 x 5 (ver figura 2.48). Con el mismo razonamiento, si se tienen 3 operaciones convolucionales (aplicadas en forma secuencial) con kernels de 3 x 3, una neurona perteneciente al mapa de características resultante de la tercer operación de convolución, tendrá un campo receptivo efectivo de 7 x 7. Esta última configuración es la que posee la VGG-16 y produjo un gran avance con respecto a las arquitecturas anteriores, ya que los kernels de 7 x 7 de AlexNet y 11 x 11 de ZFNet no serían necesarios. No solo por el equivalente campo receptivo efectivo, sino también porque se requiere un menor número de parámetros para aprender durante el entrenamiento:

- 1 operación de convolución con un kernel de 7 x 7 posee 49 parámetros que deben aprenderse durante el entrenamiento.
- 3 operaciones convolucionales aplicadas en forma secuencial, con kernels de 3 x 3 cada una, poseen 27 parámetros que deben aprenderse durante el entrenamiento.

Con lo cual, en este caso,el número de parámetros se reduce en un 45%

Además, en el caso de tener 3 operaciones convolucionales aplicadas en forma secuencial, en donde en cada una de ellas se emplea una función de activación ReLU, se tendrán 3 activaciones no lineales. Mientras que en el caso de una sola operación convolucional con un kernel de mayor dimensión seguida de una activación ReLU, se tendrá una sola activación no lineal. Por ende, otra de las ventajas de aplicar operaciones convolucionales en forma secuencial es disponer de un mayor número de activaciones no lineales (3 activaciones versus 1 activación), permitiendo que la red tenga una mayor capacidad para modelar distintas funciones (tal cual se explicó en la sección 2.2.1).

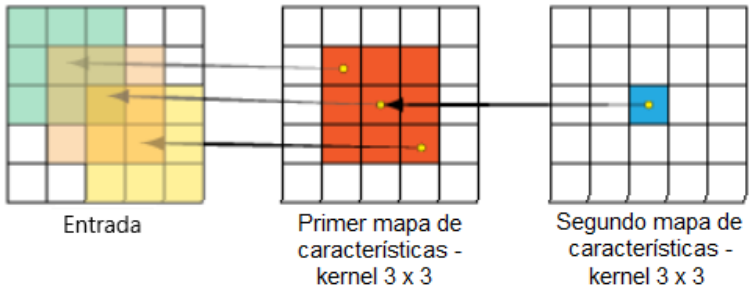


Figura 2.48: Campo receptivo efectivo o indirecto (5 x 5) correspondiente a 2 operaciones convolucionales aplicadas en forma secuencial con un kernel de 3 x 3, un stride igual a 1 y zero-padding.

Parámetros y memoria

La red VGG-16 posee 138 millones de parámetros para ser entrenados y la memoria que se necesita al realizar un forward (por medio del algoritmo backpropagation) es de 96 MB por

imagen (para una imagen color de 224 x 224 píxeles). Como hemos explicado en la sección 2.2.5, las CNNs reducen drásticamente los parámetros que utilizan a través del compartimiento de los mismos por medio de los kernels. Por ende, de los 138 millones de parámetros totales, 121 millones se encuentran en las 3 capas totalmente conectadas. No obstante en estas capas totalmente conectadas se utiliza poca memoria ya que se tienen 4096 neuronas en las primeras 2, y 1000 neuronas en la tercera. Por otro lado, debido a que esta arquitectura utiliza varios canales en sus operaciones de convolución, existe un elevado número de neuronas por cada volumen de mapas de características. Esto hace que las primeras capas que tienen un elevado tamaño en las dimensiones de los mapas y un número de canales considerable, tengan una gran cantidad de neuronas y por ende consuman mucha memoria. Siguiendo el razonamiento anterior, de los 96 MB consumidos por imágen, 25.6 MB (6.4M * 4 Bytes) lo consume el volumen de los mapas de características que son el resultado de aplicar las primeras 2 operaciones convolucionales de esta red (con 64 canales cada operación de convolución) [87].

Entrenamiento multi-escala

Un objeto puede tener distintos tamaños en distintas imágenes. Con lo cual, si entrenamos la red con imágenes que tienen objetos con una misma escala, entonces tendremos una mala clasificación para los mismos objetos pero en otras escalas. Por tal motivo, los autores entrenaron la red de dos formas:

- *Entrenamiento en una escala*: Primero la imagen se escala a un tamaño igual a 256 x 256 o 384 x 384 (dependiendo la configuración de un parámetro que llamaron S) y luego se realiza un recorte de la imagen en 224 x 224 píxeles (Ver figura 2.49).

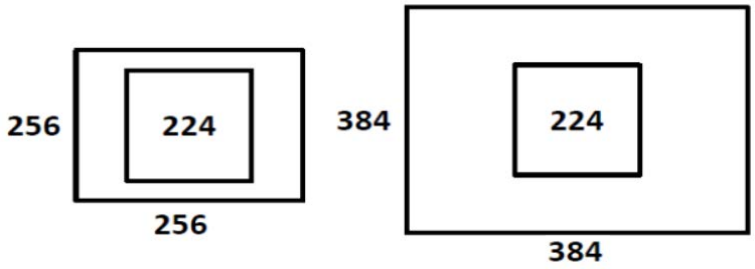


Figura 2.49: Escalado (dependiendo del parámetro S) y recorte de la imágen.

- *Entrenamiento multi-escala*: Una imagen es escalada en distintos tamaños según el valor que toma S, donde S pertenece al intervalo [256, 512]. Luego, como existen varias resoluciones para una misma imagen (256 x 256, 300 x 300, 512 x 512, etc.), se tendrán distintas escalas de objetos en el conjunto de datos de entrenamiento. Luego, una vez realizado el escalamiento, las imágenes son recortadas en 224 x 224 píxeles de forma tal que todas tengan la misma resolución pero con objetos en distintas escalas. Con este tipo de entrenamiento y con las imágenes de ImageNet, en la tarea de clasificación de la competencia ILSVRC 2014 la red VGG-16 tuvo una reducción del error del 8.8% al 8.1%.

Prueba multi-escala

En forma similar al entrenamiento multi-escala, lo que se pretende con la prueba multi-escala, es redimensionar las imágenes del conjunto de datos de prueba a distintas escalas para aumentar la probabilidad de que la clasificación de una imagen sea correcta. Es decir, por cada imágen, se tendrán n imágenes adicionales con distintas escalas de modo que se obtendrá una probabilidad de cada clase por cada una de estas imágenes. Por ende, se realizará el promedio de las probabilidades por cada clase de las $n+1$ imágenes (la imagen original y las n de distintas escalas) y a continuación se seleccionará la clase que tenga la mayor probabilidad. Luego, realizando un entrenamiento multi-escala y prueba multi-escala con las imágenes de ImageNet de ILSVRC 2014, el error se pudo reducir del 8.8% al 7.5%.

Los creadores de esta arquitectura, luego de que finalizaron el entrenamiento de la red VGG-16 anteriormente explicado, disponibilizaron los pesos en la WEB. En este trabajo de tesis, dichos pesos son utilizados para inicializar la arquitectura VGG-16, ya que esta red pre-entrenada se empleó para la construcción del Clasificador 1 y el Clasificador Auxiliar (ver sección 3.3.2). Además en dicha sección se explicará el reemplazo de la función softmax por la sigmoide ya que disponemos de solos 2 clases (“Melanoma” y “No melanoma”). También se reemplazaron las capas totalmente conectadas (entrenadas con el conjunto de datos de ImageNet) por otras nuevas, para ser entrenadas con el nuevo conjunto de datos de lesiones pigmentadas. En la figura 2.50 (a modo conceptual) se encuentra lo anteriormente explicado, para más detalles de cómo se realizó el entrenamiento y el fine tuning, ver la sección 3.3.2.

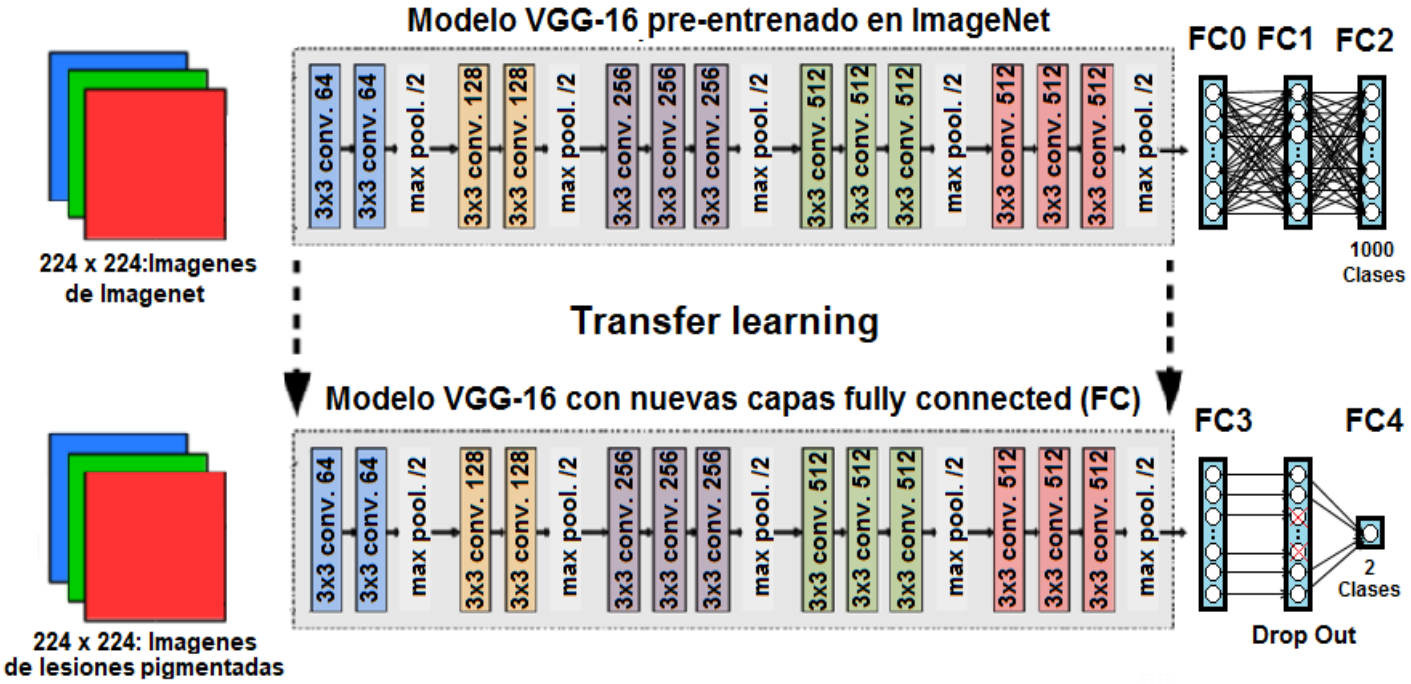


Figura 2.50: (Arriba) Modelo VGG-16 pre-entrenado en ImageNet. (Abajo) Modelo VGG-16 utilizado en el presente trabajo, inicializado con los pesos de ImageNet y entrenado con el conjunto de datos de lesiones pigmentadas.

2.5.2 Mapa de calor de la red (GRAD-CAM)

El método empleado en el presente trabajo para visualizar las zonas de la imagen que “vio” la red convolucional para realizar la clasificación de la lesión es el Grad-CAM [88]. El

objetivo de este método es constatar si la red está “viendo” zonas que comprenden a la lesión pigmentada para verificar que el entrenamiento de la red se esté realizando en forma adecuada. Dicho método corresponde a la extensión del método CAM [89]. Con lo cual primero explicaremos dicho método.

Método CAM

Tal cual se mencionó en la sección 2.2.5, los mapas de características que se encuentran en capas más profundas poseen patrones específicos. Conociendo la importancia que tienen estos mapas en relación a la predicción de una determinada clase, nos indicará en qué regiones de la imagen se está basando la red convolucional para realizar dicha predicción. Por ejemplo, si una red convolucional predice en una imagen la existencia de una persona, entonces se espera que en alguno de los mapas de las últimas capas se encuentren características específicas, por ejemplo la representación del patrón de una mano o de un sector de una nariz. Estos mapas específicos tendrán más importancia (en relación a la clase que se predice) que otros mapas de características más generales. Luego proyectando los mapas de características importantes en la imagen, se tiene un mapa de calor de las zonas que visualizó la red para realizar la predicción. Este es el funcionamiento base de este método.

El método reemplaza las capas totalmente conectadas de una red convolucional por una operación de “global average pooling” (GAP) [90], ubicada luego de la última capa convolucional. La operación GAP transforma cada mapa de características a un solo valor que corresponde al promedio de las neuronas pertenecientes al mapa. Luego, los valores que se obtienen con el GAP serán la entrada a una nueva capa totalmente conectada teniendo a la función softmax como función de activación. Esta capa tendrá pesos que se aprenderán vía backpropagation (durante el entrenamiento de la red). Finalizado el entrenamiento y cuando se le presente una imagen a la red, el score final será la suma de los valores del GAP ponderados por los pesos de la capa totalmente conectada que se aprendieron durante el entrenamiento. Por último, proyectando estos pesos en los mapas de características que hace referencia cada valor del GAP, obtenemos el mapa de activación de la clase. Lo anterior puede visualizarse en la figura 2.51.

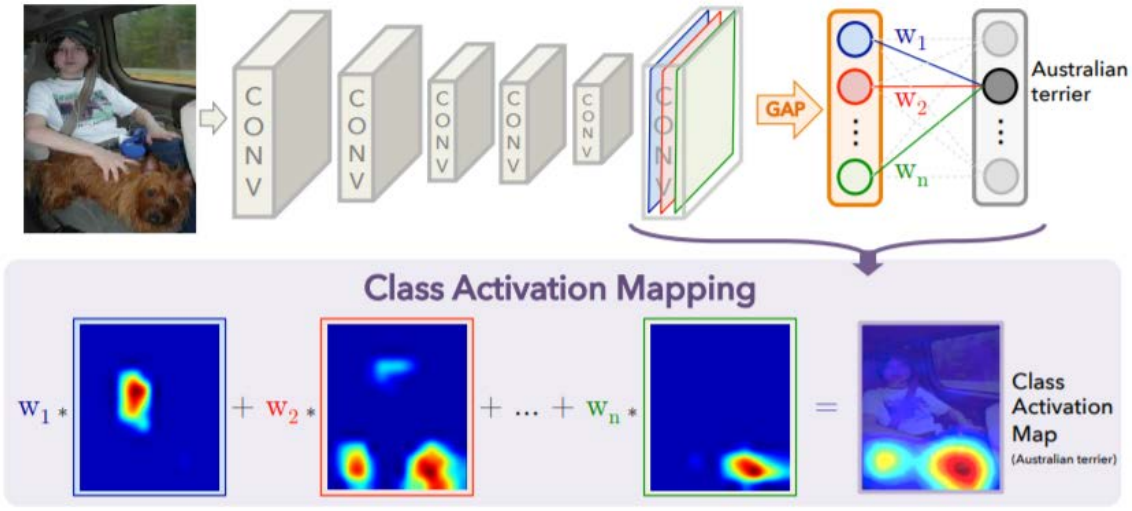


Figura 2.51: Método Class Activation Mapping. (Arriba) Incorporación de la operación GAP (luego de la última capa convolucional) y una nueva capa totalmente conectada. (Abajo) Conformación del mapa de activación de una determinada clase.

Formalmente, para una imagen ingresada en la red obtenemos $f_k(x,y)$, que representa el mapa de características k de la última capa convolucional en la neurona cuyas coordenadas son (x,y). Luego, si se efectúa un GAP para este mapa, obtenemos F^k dado por la siguiente ecuación:

$$F^k = \underbrace{\frac{1}{Z}\sum_{x,y} f_k(x,y)}_{\text{global average pooling}} \tag{2.32}$$

Luego para una clase c, la entrada a la capa softmax, S_c ,estará dada por la ecuación:

$$S_c = \sum_k w_k^c F_k \tag{2.33}$$

Donde w_k^c corresponde a la importancia de F^k para que la red prediga la clase c. En la ecuación 2.33 no se tiene en cuenta el bias de la capa softmax (se setea en 0). Por último, la salida de la capa softmax para una clase c, con n clases disponibles, es $P(c)$ dada por la ecuación:

$$P(c) = e^{s_c} / \sum_{j=0}^{n} e^{s_j}$$

Reemplazando la ecuación 2.32 en 2.33 obtenemos la ecuación:

$$S^c = \sum_k w_k^c \frac{1}{Z}\sum_{x,y} f_k(x,y)$$

$$= \frac{1}{Z}\sum_{x,y}\sum_k w_k^c f_k(x,y)$$

Siendo $M_c = \sum_k w_k^c f_k(x,y)$ el mapa de activación resultante para la clase c (ver figura 2.51).

Se aclara que las dimensiones de este mapa resultante M_c son diferentes al de la imagen de entrada de la red, con lo cual antes de superponer el mapa con la imagen se debe re-escalar el mismo para que ambos posean las mismas dimensiones.

Uno de los inconvenientes que posee este método es que el modelo se debe re-entrenar nuevamente ya que se debe incorporar la operación GAP y obtener los pesos w_k^c . Además, esto conlleva a una baja en la performance en lo que respecta a la clasificación de imágenes de la red. Con lo cual, surge el método Grad-CAM que utiliza la misma arquitectura de la red y no necesita que la red sea reentrenada para poder utilizarlo.

Método Grad-CAM

Este método llamado Gradient-weighted Class Activation Mapping (Grad-CAM) utiliza el gradiente del puntaje (que se obtiene en un red CNN para una clase) con respecto a los mapas de características de la última capa convolucional. El objetivo es establecer las regiones importantes de una imagen que “vió” la red para la discriminación de dicha clase.

En esta técnica, el gradiente del puntaje para un “concepto” (clase) fluye hasta la última capa convolucional para determinar la importancia que tiene cada neurona del mapa de características para la determinación de dicho “concepto” (Ver figura 2.52).

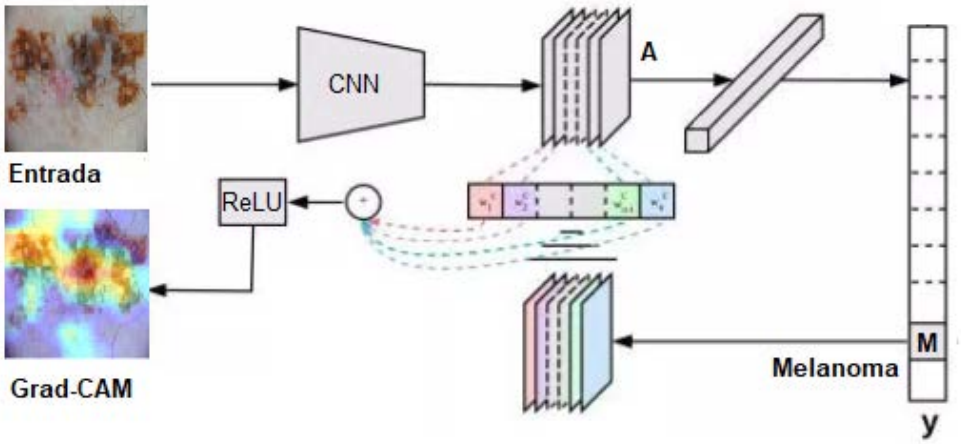


Figura 2.52: Método Grad-CAM utilizado en la clasificación de imágenes.

Luego, a diferencia del método CAM, el cálculo de los pesos (importancia en la predicción del “concepto”) de los mapas de características se realiza por medio del gradiente de y^M (puntaje de la red para la clase M antes de la función sigmoide) con respecto al mapa de característica A^k de la última capa convolucional (ver figura 2.52). Por último se realiza una operación GAP sobre los gradientes calculados (gradiente de y^M con respecto a cada neurona del mapa) y se obtiene el peso relativo del mapa con respecto a la predicción del “concepto”. Lo anterior puede visualizarse en la siguiente ecuación:

$$\alpha_k^M = \underbrace{\frac{1}{Z}\sum_i\sum_j \frac{\partial y^M}{\partial A_{ij}^k}}_{\text{global average pooling}} \tag{2.34}$$

Es decir, los pesos son calculados por medio del algoritmo backpropagation y no se requiere una modificación adicional a la red como es el caso del método CAM. Similar a este método, el método Grad-CAM calcula el mapa de calor como una suma de los mapas de características (de la última capa convolucional) ponderados por los pesos calculados mediante la ecuación 2.34. La diferencia con el método CAM radica que en esta técnica el resultado anterior va seguida de una activación ReLU. La activación ReLU se realiza con el objetivo de tomar los píxeles que tienen una contribución “positiva” con la clase que estamos evaluando (en este caso para la clase M de Melanoma). Una contribución “negativa” significa que esos píxeles contribuyen positivamente a la clase “No Melanoma” y no resulta importante para establecer las zonas que vió la red para predecir el concepto “Melanoma”. Siendo n, la cantidad de canales en la última capa convolucional, el mapa de activación Grad-CAM para una clase M, es el que se indica en la siguiente ecuación:

$$L_{grad-CAM}^M = ReLU\left(\sum_{k=1}^{n} \alpha_k^M A^k\right)$$

Este mapa, al igual que en el método CAM, va a tener las dimensiones que poseen los mapas de características de la última capa convolucional (14 x 14 en el caso de la red VGG-16 con imágenes de entrada de 224 x 224). Por ende, se lo deberá reescalar para que tenga las mismas dimensiones de la imagen de entrada.

Como se mencionó anteriormente, la aplicación de este método se puede realizar sin modificar la arquitectura de la red convolucional. No obstante, si la arquitectura a la cual se aplica el método es compatible con el método CAM (tiene la operación GAP incorporada y se eliminaron las capas totalmente conectadas), entonces los pesos que se obtienen con CAM y Grad-CAM son iguales. Por tal motivo, Grad-CAM es una generalización del método CAM. Para la demostración matemática de esto último ver la sección “A” del apéndice de [88].

2.5.3 Modelo Híbrido

2.5.3.1 Introducción

El Modelo Híbrido toma los descriptores de la última capa de pooling de la arquitectura VGG-16 y realiza la clasificación respectiva. A continuación se explicará el funcionamiento de cada uno de los componentes que conforman el Modelo Híbrido. Por último se expondrá la arquitectura del Modelo Híbrido en forma completa y cómo se relacionan dichos componentes.

2.5.3.2 Modelo de mezclas gaussianas (GMM)

Modelo de mezcla

Sea una población general con subpoblaciones, los modelos de mezcla de distribuciones suelen ser utilizados para inferir propiedades de subpoblaciones partiendo de las observaciones de la mencionada población general.

Sea $Y = [Y_1, Y_2, ..., Y_d]^T$ una variable aleatoria real de D dimensiones. Se dice que Y sigue una distribución de mezcla finita si su función de densidad de probabilidad (fdp) puede escribirse como una combinación lineal de fdp’s elementales. Es decir, sigue la siguiente ecuación:

$$p(y \mid \theta) = \sum_{i=1}^{I} \alpha_i \; p(y \mid C = i, \beta_i) \qquad i \in \{1, ..., I\} \tag{2.35}$$

Donde C representa las componentes de la mezcla, I el número de distribuciones elementales (componentes) de la mezcla y θ representa el conjunto de parámetros del modelo de mezcla dado por:

$$\theta = \{\alpha_1, \alpha_2 .... \alpha_I, \beta_1, \beta_2 ..... \beta_I\}$$

Siendo $\alpha = \{\alpha_1, \alpha_2 .... \alpha_I\}$ y $\beta = \{\beta_1, \beta_2 .... \beta_I\}$. Donde cada β_i constituye el conjunto de parámetros asociados a una distribución de la mezcla y cada α_i corresponde al peso asociado a una distribución de la mezcla [91].

Mezcla de gaussianas

Cuando las distribuciones elementales que componen la mezcla son gaussianas, la función de densidad de probabilidad dada en 2.35 se denomina modelo de mezclas de gaussianas (GMM) y se encuentra dada por la siguiente ecuación [92]:

$$p(y \mid \theta) = \sum_{i=1}^{I} \alpha_i \; N(y \mid C = i, \beta_i) \qquad i \; \epsilon \{1, ..., I\}$$

Siendo:

$$N(y \mid C = i, \beta_i) = \frac{exp\left(-\frac{1}{2}(y - \mu_i)^T \sum_i^{-1} (y - \mu_i)\right)}{2\pi^{D/2} \left|\sum_i\right|^{1/2}}$$

Y los parámetros de cada componente gaussiana son:

$$\beta_i = \{\mu_i, \; \Sigma_i\}$$

Donde $\mu_i \, \epsilon \, \mathbb{R}^D$ y $\Sigma_i \, \epsilon \, \mathbb{R}^{DxD}$ son la media y la matriz de covarianza de la componente i-ésima del modelo de mezcla gaussiano.

Además los pesos de la mezcla deben verificar las siguientes restricciones:

$$\alpha_i \geq 0, \quad i \, \epsilon \, \{1, ..., I\} \qquad \sum_{i=1}^{I} \alpha_i = 1 \tag{2.36}$$

Y las matrices de covarianzas deben cumplir con:

$$\Sigma_i = \Sigma^T \qquad \sigma_{jj}^2 \geq 0, \quad j \, \epsilon \, \{1, ..., D\}$$

Es decir, las matrices de covarianzas deben ser simétricas y las varianzas ubicadas en la diagonal deben ser no negativas.

Por último, debemos encontrar los parámetros de las componentes gaussianas para que modele las observaciones. Esto lo podremos realizar mediante la máxima verosimilitud (explicada en la sección 2.5.3.3), cuya ecuación es la siguiente:

$$\hat{\theta}_{ML}(y) = argmax_\theta \sum_{j=1}^{J} log \left\{ \sum_{i=1}^{I} \alpha_i \; N(y_j \mid C = i, \beta_i) \right\} \tag{2.37}$$

No existe una forma cerrada de $\widehat{\theta}_{ML}$, con lo cual se propone el algoritmo de Expectation-Maximization para resolverlo.

Algoritmo Expectation-Maximization (EM)

Si el número de componentes de la mezcla I es conocido, el algoritmo expectation-maximization es uno de los más utilizados para estimar los parámetros del modelo de mezcla gaussiana [93]. Este algoritmo es iterativo y en cada iteración se procede a calcular los siguientes 2 pasos:

- El primer paso conocido como “E step” (o expectation) consiste en calcular la esperanza de que el punto $x_i \in X$ esté asignado a la componente C_i dado los parámetros del modelo: μ_i, Σ_i, α_i.

- El segundo paso conocido como “M step” (o maximization) consiste en calcular los estimadores (de máxima verosimilitud) de los parámetros μ_i, Σ_i, α_i mediante la maximización de la verosimilitud esperada (calculada en el paso “E step”).

Como el algoritmo es iterativo, la estimación de los parámetros del “M step” sirven como entrada para el “E step”, y así el proceso se repite hasta el criterio de parada especificado.

A continuación, a modo explicativo, se describen los pasos del algoritmo EM para un modelo de mezcla gaussiana univariado de parámetros μ_i, θ_i, α_i (siendo la media, varianza y pesos de las componentes gaussianas). Dado I el número de componentes gaussianas:

1)Expectation

$\forall i, I$ (Para cada uno de los puntos y para cada componente gaussiana):

$$\hat{\gamma}_{iI} = \frac{\hat{\alpha}_I \; N(x_i \mid \hat{\mu}_i \; \hat{\theta}_i)}{\sum_{j=1}^{I} \hat{\alpha}_j N(x_i \mid \hat{\mu}_j \; \hat{\theta}_j)}$$

Es decir, se calcula $\hat{\gamma}_{iI}$ que es la probabilidad que un punto x_i sea generado por la componente C_i. Esto es:

$$\hat{\gamma}_{iI} = p(C_I \mid x_i, \; \hat{\alpha}, \; \hat{\mu}, \; \hat{\theta})$$

2)Maximization

Una vez calculado $\hat{\gamma}_{iI}$ en el “E step”, se calculan las siguientes estimaciones de parámetros (se supone que existen N puntos):

$\forall I$ (Para cada componente gaussiana), se calculan los siguientes parámetros del modelo:

- $\hat{\alpha}_I = \sum_{i=1}^{N} \frac{\hat{\gamma}_{iI}}{N}$

- $\hat{\mu}_I = \frac{\sum_{i=1}^{N} \hat{\gamma_{iI}}\ x_i}{\sum_{i=1}^{N} \hat{\gamma_{iI}}}$

- $\hat{\theta}_I = \frac{\sum_{i=1}^{N} \hat{\gamma_{iI}}\ (x_i - \hat{\mu_I})^2}{\sum_{i=1}^{N} \hat{\gamma_{iI}}}$

3) Evaluación de la función de verosimilitud

Se evalúa la función de verosimilitud (ver ecuación 2.37) con los nuevos parámetros hallados en “Maximization”. Si se llega al criterio de parada (la variación de la verosimilitud de una iteración con respecto a la anterior no supera un umbral predefinido o se alcanza el máximo número de iteraciones) se detiene el algoritmo, de lo contrario se vuelve al paso de “Expectation” con los nuevos parámetros hallados en “Maximization”.

A modo de ejemplo, en la figura 2.53 se observa el ajuste del modelo GMM a la distribución de los datos a medida que transcurren las iteraciones del algoritmo EM.

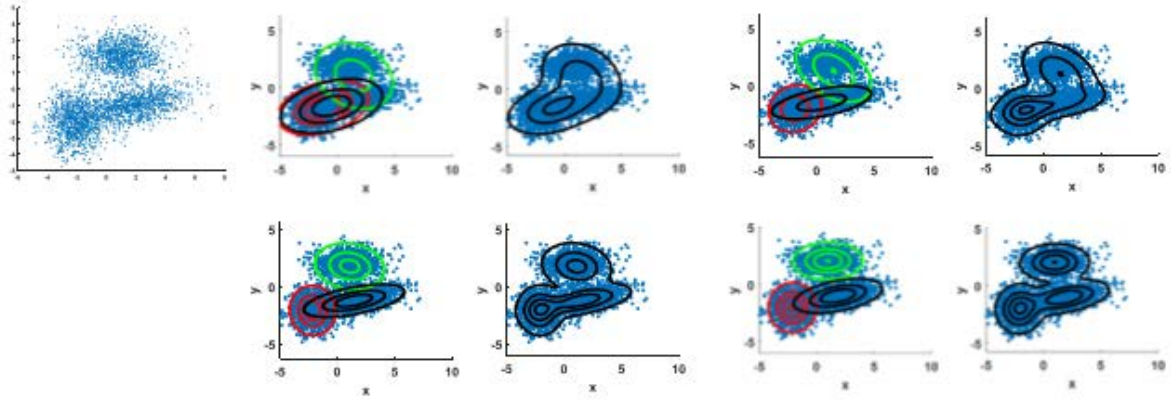


Figura 2.53: Ajuste del modelo GMM a la distribución de los datos en las distintas iteraciones del algoritmo EM.

Cuando el número de componentes de la mezcla no es conocido, se suele hallar el mismo por distintos métodos, para luego aplicar el algoritmo EM. Estos método como las condiciones de iniciación se describen a continuación.

Número de componentes

Uno de los problemas que sucede con la mixtura de gaussianas es que se desconoce el número de componentes gaussianas I a priori. Con lo cual, si colocamos un número grande de I puede que estemos provocando una partición excesiva de los datos y un número pequeño de I puede que no sea suficiente para seguir la distribución del modelo real que genera los datos. Por ende existen diferentes técnicas para estimar el número de componentes de la mezcla como por ejemplo el criterio de información de Akaike y el criterio de información Bayesiano (ambos explicados en la sección 2.5.3.3).

Inicialización

Como hemos mencionado, el algoritmo EM es iterativo, con lo cual los valores de los parámetros iniciales tienen importancia en lo que respecta a la velocidad de convergencia del algoritmo y también en la posibilidad de converger a máximos locales de la función de verosimilitud.

En lo que respecta a los pesos α_i, lo habitual es colocar el mismo peso a todas las componentes (cumpliendo la restricción expuesta en 2.36), y este se calcula en base a la siguiente ecuación [94][95].

$$\alpha_i = \frac{1}{I}$$

Para obtener las medias μ_i, se toman I observaciones elegidas al azar y se las utilizan como medias iniciales para cada una de las componentes gaussianas [94][95][96].

Para obtener los valores de la matriz de covarianzas Σ, lo habitual es utilizar la misma matriz de covarianzas inicial en todas las componentes y la misma será igual a la matriz de covarianzas de los datos [95][96].

La iniciación mencionada se la denomina “inicialización aleatoria” y es la utilizada en el presente trabajo. Cabe destacar que existen otras inicializaciones, como por ejemplo mediante el algoritmo K-means [97].

Debido que el algoritmo es iterativo y en cada iteración la verosimilitud aumenta, es necesario tener un criterio de parada. Un criterio puede ser la fijación de un umbral para verificar que la variación de la verosimilitud de una iteración con respecto a la anterior sea mayor a dicho umbral (de lo contrario el algoritmo termina). Otra de las formas es establecer un número de iteraciones considerable, que es la utilizada en el presente trabajo (se colocaron 100 iteraciones). Se empleó dicha estrategia, principalmente por un problema de limitación de hardware, ya que al establecer un umbral muy pequeño conlleva un número de iteraciones tal que produce tiempos de procesamiento muy elevados.

2.5.3.3 AIC y BIC

Estos indicadores, denominados criterio de información de Akaike (AIC) y criterio de información bayesiano (BIC) abordan el compromiso entre la complejidad y la capacidad predictiva del modelo. Cuanto más complejo sea un modelo, peor será su capacidad de predicción en un abanico amplio de situaciones (sobreajuste). O sea, cuantas más variables incorpore el modelo e interrelaciones complejas entre los componentes, las predicciones del mismo serán menos generalizables. Si un modelo es sencillo e incluye solo los componentes con mayor importancia en el sistema, las predicciones serán más generalizables que en el caso anterior (aceptando un cierto error de precisión en los datos que se dispone).

Para encontrar el modelo más parsimonioso, de entre una batería de ellos necesitamos un indicador capaz de 1) Medir la capacidad explicativa de los datos y 2) Penalizar por su grado

de complejidad. Esto es lo que hacen estos **indicadores de criterio de la información**, y el cálculo va a tener la siguiente forma:

$$xIC = complejidad - bondad\ de\ ajuste \tag{2.38}$$

Tanto en AIC como en BIC la bondad de ajuste se encuentra en función de la verosimilitud, con lo cual primero explicaremos esta función.

Función de verosimilitud (Likelihood)

Muchos procesos estadísticos suponen que los datos siguen algún tipo de modelo matemático que se define mediante una ecuación en la que se desconocen algunos de sus parámetros. Por ejemplo, existen varios métodos para estimar los coeficientes de un modelo de regresión o el de una función de probabilidad (o de densidad). Entre los métodos más utilizados se encuentra el que se lo denomina “método de máxima verosimilitud”.

Se supone un modelo de regresión lineal simple dada por la ecuación $y = ax + b + e$, donde a es la pendiente, b es el intercepto y $e \sim N(0, \sigma)$ es el error o residuo. Luego, se trata de encontrar una combinación de parámetros que minimice la distancia de los datos a la recta, o lo que es lo mismo que maximice la verosimilitud. Es decir, se trata de encontrar un modelo que explique nuestros datos, que sea verosímil. Con lo cual, queremos explicar la variable y a partir de la variable x con un modelo lineal de parámetros $\theta = \{a, b, e\}$. Suponiendo que las observaciones x_i son independientes, definimos a la función de likelihood como la probabilidad conjunta de los datos condicionada al modelo, mediante la siguiente ecuación:

$$p(y_1, y_2, \ldots, y_n \mid \theta) = p(y_1 \mid \theta) \cdot p(y_2 \mid \theta) \ldots p(y_n \mid \theta)$$

Los valores $y_1, y_2, \ldots, y_n$ son conocidos, mientras que los parámetros desconocidos. Con lo cual podemos reformular la ecuación de la siguiente forma:

$$L(\theta) = p(\theta \mid y_1, y_2, \ldots, y_n) = p(\theta \mid y_1) \cdot p(\theta \mid y_2) \ldots p(\theta \mid y_n) \tag{2.39}$$

De acuerdo a los parámetros que tome θ, la función de la ecuación 2.39 tomará distintos valores, y alcanzará un máximo para una combinación de parámetros $\{a, b, e\}$. Con lo cual, diremos que ese conjunto de parámetros maximizan la función de verosimilitud y habremos aplicado el método de **máxima verosimilitud** para ajustar el modelo (y como se mencionó anteriormente se minimiza la distancia de los datos a la recta). En la práctica se utiliza el logaritmo de la verosimilitud (y se denomina **log-likelihood**) con lo que la ecuación 2.39 se convierte en una suma de logaritmos de las probabilidades en lugar de una productoria (lo que equivale a necesitar menos potencia de cómputo). Aplicar esta transformación es posible debido a que el logaritmo es una función monótona creciente.

Criterio de información de Akaike (AIC)

Se basa en la entropía de la información, es decir, es una estimación relativa de la información perdida cuando se utiliza un modelo para representar el “verdadero modelo” que genera los datos. Se define mediante la siguiente ecuación:

$$AIC= = 2\,k - 2\,ln(L)$$

Donde k es el número de parámetros del modelo y L es el valor máximo de la función de verosimilitud. Dada la ecuación 2.38 se observa que la bondad de ajuste viene dada a partir de la máxima verosimilitud del modelo y la complejidad a través del número de parámetros. En este caso, la verosimilitud es la probabilidad conjunta de los datos condicionada a los k parámetros del modelo. Dado un conjunto de modelos candidatos para los datos, el modelo preferido es el que tiene el valor mínimo de AIC. Si todos los modelos candidatos tienen una baja performance en la aproximación de los datos, no dará ningún aviso sobre ello, es decir, es un indicador relativo al resto de los modelos candidatos. Por otro lado, minimizar el AIC en forma asintótica es lo mismo que minimizar el cross validation (validación cruzada) [72].

Criterio de información Bayesiano (BIC)

Este indicador descubierto por Schwarz viene dado por:

$$BIC= = k\,ln(n) - 2\,ln(L)$$

Donde nuevamente, k es el número de parámetros del modelo, L es el valor máximo de la función de verosimilitud y n es el número de datos. Al igual que AIC, la bondad de ajuste viene dada en función de la verosimilitud. No obstante, la medida de complejidad incorpora tanto k como $ln(n)$ con lo que tiene una mayor penalización que el AIC. Dado un conjunto de modelos candidatos, se seleccionará el modelo que tenga el menor valor de BIC.
En forma asintótica, minimizar BIC es equivalente a minimizar el cross validation leave N out (Validación cruzada dejando N afuera) cuando $N = n\,[1-1/(log(n)-1)]$ [98].

El indicador BIC es asintóticamente consistente como criterio de selección, lo que significa que dado una serie de modelos candidatos (incluyendo el modelo verdadero), la probabilidad de que BIC seleccione el modelo correcto será cercana a 1 a medida que $n\to\infty$. Este no es el caso de AIC que tiende a elegir modelos muy complejos a medida que $n\to\infty$. Por otro lado, para muestras finitas, BIC selecciona modelos más simples debido a la alta penalización sobre la complejidad [99].

En el presente trabajo se utilizarán los indicadores AIC y BIC para establecer los conjuntos de componentes gaussianas que se emplearán para la creación de los Clasificadores 2 y 3, en las secciones 3.3.3 y 3.3.4 respectivamente.

2.5.3.4 PCA

El análisis de componentes principales (PCA - principal component analysis) [100] pertenece al tipo de algoritmo “no supervisado”. Fue inicialmente desarrollada por Pearson [101] a finales del siglo XIX y posteriormente fue estudiada por Hotelling [102] en los años 30 del siglo XX. El objetivo del método es resumir la información de un conjunto de variables originales en un número más pequeño de variables, denominadas componentes o factores, con una mínima pérdida de información. Cuando nos referimos a la mínima pérdida de

información, es encontrar un “subespacio principal” en donde la varianza de la proyección de los datos es maximizada (Hotelling); o bien en términos de distancia, minimizar el promedio de la distancia al cuadrado de los puntos al “subespacio principal” (Pearson) [103].

Por medio de este método, se reemplazan las p variables originales por k nuevas variables, donde cada variable k_i es una combinación de las p variables originales. Además k es menor que p. Estas k variables explican una importante varianza del conjunto de datos, de forma que podremos trabajar con el nuevo conjunto de variables que conforman un espacio de menor dimensionalidad.

Con lo cual, el objetivo es encontrar k combinaciones lineales de las variables originales $X_1, X_2, \ldots\ldots, X_p$, que darán como resultado a las nuevas variables $Z_1, Z_2, \ldots\ldots, Z_k$, no correlacionadas entre sí. A cada una de estas componentes Z_i, se las denomina componentes principales y resultan de una “medición” de distintas dimensiones de los datos, ya que son una combinación lineal de las variables observables. Con lo cual, es posible ordenarlas de acuerdo a su varianza para establecer el orden de relevancia de las componentes mediante la siguiente ecuación:

$$var(Z_1) \geq var(Z_2) \geq \ldots\ldots\ldots \geq var(Z_k)$$

Para determinar las componentes principales se pueden seguir los siguientes pasos:

1) Se parte de una matriz de datos D con n observaciones y p variables:

$$D = \begin{bmatrix} Ind & X_1 & X_2 & X_p \\ 1 & x_{11} & x_{12} & x_{1p} \\ 2 & x_{21} & x_{22} & x_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ n & x_{n1} & x_{n2} & x_{np} \end{bmatrix}$$

2) Se calcula la matriz de varianzas y covarianzas C:

$$C = \begin{bmatrix} c_{11} & c_{12} & c_{1p} \\ c_{21} & c_{22} & c_{2p} \\ . & . & . \\ c_{p1} & c_{p2} & c_{pp} \end{bmatrix}$$

Donde C_{ii} es la varianza de X_i y C_{ij} es la covarianza de X_iX_j.

3) Como se mencionó anteriormente, las componentes principales son combinaciones lineales de las variables originales y para poder obtener dichas combinaciones es necesario calcular los valores y vectores propios de la matriz de covarianzas. Siendo

$\vec{v}$ un autovector y λ un autovalor de la matriz de varianzas y covarianzas, se calculan los mismos mediante las siguientes ecuaciones:

$$C\ \vec{v} = \lambda\ \vec{v}$$

$$(C - \lambda\ I)\ \vec{v} = \vec{0} \qquad (2.40)$$

Dado que $\vec{v} \neq \vec{0}$, necesariamente se debe cumplir:

$$|C - \lambda\ I| = 0 \qquad (2.41)$$

A la ecuación 2.41 se la denomina ecuación característica y las raíces del polinomio característico o raíces de la ecuación característica son el conjunto de autovalores asociados a la matriz de varianzas y covarianzas. Una vez obtenidos los autovalores, estos se reemplazan en la ecuación 2.40 y se obtienen los autovectores asociados. Es decir se resuelve la siguiente ecuación:

$$(C - \lambda_i\ I)\ \vec{v} = \vec{0}$$

No obstante lo mencionado, dichos valores y vectores propios (autovalores y autovectores) son calculados a través de algún software estadístico. Luego se calculan las variables Z_i de acuerdo a la siguiente ecuación:

$$Z_i = a_{i1}\, x_1 + a_{i2}\, x_2 + ... + a_{ip}\, x_p$$

Donde a_{i1}, a_{i2}……a_{ip}, representan los elementos de cada vector propio obtenido ($\vec{v_i}$). La varianza de las componentes principales corresponde a los autovalores de la matriz de covarianzas, representada en la siguiente ecuación:

$$var(Z_i) = \lambda_i$$

Una propiedad que tienen los autovalores es que su suma es igual a la suma de los elementos de la diagonal principal de la matriz de covarianzas (la traza de la misma). Es decir, se cumple que:

$$\lambda_1 + \lambda_2 + ... + \lambda_p = c_{11} + c_{22} + ... + c_{pp}$$

Con lo cual, si c_{ii} es la varianza de x_i y λ_i es la varianza de z_i, significa que la suma de las varianzas de las componentes principales es igual a la suma de las varianzas de las variables originales. Por ende, si se seleccionan todas las componentes principales, se explicaría la variabilidad total del conjunto de datos.

Dado que las componentes principales son una combinación lineal de las variables originales, puede suceder que una de ellas tenga influencia excesiva en la determinación de la componente debido, como por ejemplo, a un problema de escala de la variable. Con lo cual, una práctica habitual es normalizar las variables x_1, x_2……x_p de modo que tengan media

0 y varianza 1 al comienzo del análisis. Por ende, los elementos de la matriz C van a representar la correlación entre las distintas variables, con lo que C es la matriz de correlaciones. En este caso, tanto la traza de la matriz como la suma de los autovalores es igual a p, siendo p el número de variables originales.

$$C = \begin{bmatrix} 1 & r_{12} & r_{1p} \\ r_{21} & 1 & r_{2p} \\ . & . & . \\ r_{p1} & r_{p2} & 1 \end{bmatrix}$$

Número de componentes a retener

Existen diferentes criterios [104] para la selección del número de componentes principales. Los que se utilizan habitualmente son:

Criterio de Kaiser: Si se obtienen las componentes principales utilizando la matriz de correlaciones, entonces las varianzas de cada una de las variables originales será 1. Por lo tanto si una componente principal tiene una varianza menor 1, explicaría menos variabilidad que una variable observable. Por ende, este criterio establece eliminar las componentes principales cuya varianza es inferior a 1.

Criterio del bastón roto: Se realiza un gráfico de sedimentación con el número de componentes a retener en el eje de las abscisas y la varianza explicada en el eje de las ordenadas. Observamos como la curva desciende hasta un determinado valor Q y luego se estabiliza. Por ende, tomamos el número de componentes asociado al valor Q como el número de componentes a retener.

Criterio del porcentaje de la varianza explicada: Se toma un número de componentes m, de forma tal que al tomar las primeras m componentes de mayor varianza, expliquen una varianza próxima a un valor especificado por el usuario (por ejemplo un 80%).

Criterio del test de esfericidad de Bartlett: Si la matriz de datos proviene de una distribución normal multivariada con p variables, entonces la hipótesis nula $H_0^{(m)}$ de este test establece que a partir de la componente principal m no hay direcciones de máxima variabilidad, es decir, la distribución es esférica. El test está basado en un estadístico cuya distribución es chi cuadrado y se aplica secuencialmente. Si no rechazamos $H_0^{(0)}$ significa que no existen direcciones principales, pero si rechazamos $H_0^{(0)}$ entonces debemos testear $H_0^{(1)}$ y así sucesivamente para cada componente principal hasta que no rechacemos $H_0^{(m)}$. Por ejemplo, con p=5 tomariamos las primeras 2 componentes principales (m=2) si rechazamos $H_0^{(0)}$ y $H_0^{(1)}$ pero no rechazamos $H_0^{(2)}$.

Interpretación geométrica

En la figura 2.54 se visualizan puntos (observaciones) en un espacio de 3 dimensiones. Las 3 variables fueron normalizadas para evitar los problemas de escala que mencionamos anteriormente. Luego de la normalización se calculó la media de este conjunto de datos, obteniendo el punto violeta. Posteriormente, colocamos los ejes de forma tal que el origen coincida con la media calculada. Se aclara que los puntajes obtenidos denominados “scores” z_i corresponden a la distancia desde la media (proyectada en cada componente principal) a los puntos (proyectados en cada componente principal) [105]. Luego, se busca la primera componente que corresponde a una recta tal que la variabilidad de los scores resulte máxima. Posteriormente, se busca una segunda componente perpendicular a la primera con el mismo razonamiento. Es decir, se rota una segunda recta de forma que sea perpendicular a la primera componente hasta obtener la máxima variabilidad de los scores proyectados. Cuando esto sucede, hemos obtenido la segunda componente principal. Luego, observamos que las 2 componentes principales encontradas forman un plano. La distancia de cada observación al plano encontrado se lo denomina error residual o distancia residual. Se trata de minimizar la suma de los cuadrados de los errores residuales (varianza no explicada por las componentes) ya que es equivalente a maximizar la varianza de las proyecciones (varianza explicada por las componentes).

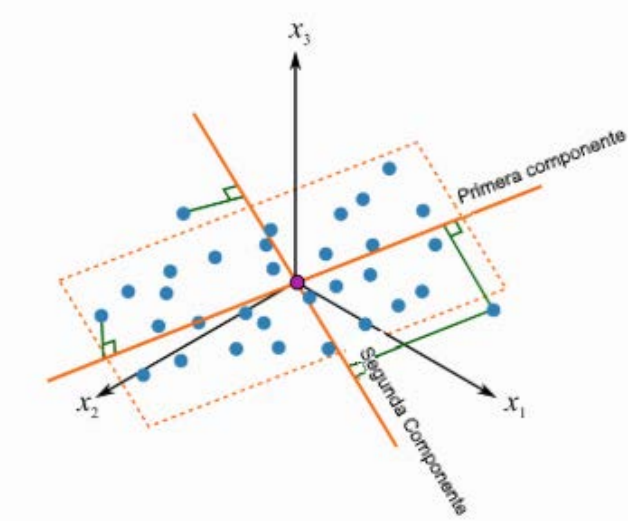


Figura 2.54: Proyección de los puntos en la primer y segunda componente principal generada.

2.5.3.5 Vectores de Fisher

Motivación

Los vectores de Fisher pueden entenderse como una extensión del modelo “bolsa de palabras” (BoW - Bag of Words) para la representación de una imagen. El modelo de “BoW” es un modelo ampliamente conocido, introducido por Csurka para describir una imagen con el objetivo de realizar su posterior clasificación [106]. Este modelo está inspirado en la técnica de análisis de textos, donde cada documento puede ser eficientemente descrito con un vector que almacena la frecuencia de las palabras que aparecen en los mismos. En el contexto de las imágenes, una equivalencia a las palabras son los descriptores locales. Donde el procedimiento es realizar conglomerados (clusters) de estos descriptores (por medio de un algoritmo) para asignarlos al representante de cada uno de esos conglomerados. Estos representantes son llamados palabras visuales (visual words). Una vez obtenidas estas

palabras, podremos realizar un histograma contando la cantidad de descriptores locales que tiene asignado cada una de estas palabras. Luego, estos histogramas se utilizaran para clasificar una imagen. El concepto del modelo de BoW se puede visualizar en la figura 2.55. Se observa que el tamaño del histograma no depende de la cantidad de descriptores o de la resolución de la imágen, sino de la cantidad de palabras visuales. En resumen, un modelo BoW consta de 3 etapas:

- *Extracción de descriptores locales*: Por ejemplo los descriptores SIFT o los mapas de características de una red convolucional.

- *Creación del vocabulario visual*: Hallar los conglomerados de los descriptores.

- Codificación de los descriptores locales: Creación de un histograma en base al conteo de las asignaciones de los descriptores locales a las palabras visuales.

El vector de Fisher establece 2 mejoras con respecto al tradicional modelo BoW:

- Reemplazó la asignación dura (hard assignment) por una asignación blanda (soft assignment), es decir, un descriptor local puede estar asignado (con distinto peso) a más de una palabra visual.

- Además codifica estadísticos de orden más alto (de orden uno y dos), es decir codifica en qué dirección y en qué cantidad, el descriptor local se desvía de su asignación de cluster.

Los vectores de Fisher se basan en los Fisher kernels (núcleos de Fisher), con lo cual primero se explicarán estos últimos y luego los vectores de Fisher. Por último, se explicará la aproximación de la matriz de información de Fisher (FIM) usada para la normalización de los vectores de Fisher.

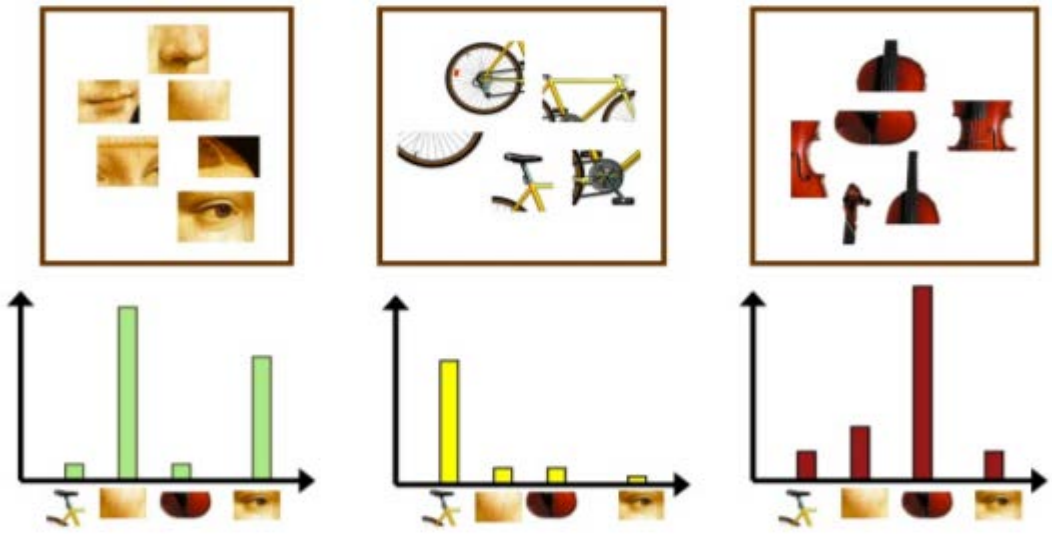


Figura 2.55: Modelo BoW. (Arriba) Conjunto de descriptores locales (de cada imagen) que serán asignados a la palabra visual más cercana. (Abajo) Los histogramas reflejan el conteo de estas asignaciones.

Fisher kernels

Los vectores de Fisher están basados en los Fisher kernels [107]. Estos se utilizan como un nexo entre los modelos probabilísticos generativos y los modelos clasificadores. De esta forma podemos crear un modelo que combina la ventaja de los 2 “mundos”. Por un lado, los clasificadores como el SVM han mostrado un excelente rendimiento. Por otro lado los modelos generativos, debido que describen la distribución de los datos, son capaces de manejar datos faltantes y datos con longitud variable.

Definimos al puntaje de Fisher (Fisher score o simplemente “puntaje” en el contexto de la estadística) como el gradiente del logaritmo de la función de verosimilitud (log-likelihood) de un un modelo generativo $p(\cdot\,;\theta)$ con respecto a sus parámetros θ. Sea $X = \{x_1, x_2, \ldots\ldots x_n\}$ un conjunto de n observaciones, donde cada observación está descrita por el vector $x_i \in \mathbb{R}^D$ donde D es la dimensión del descriptor. Luego, el puntaje de Fisher de las observaciones de X, está dado por:

$$G_\theta^x = \nabla_\theta \log p(X;\theta) \tag{2.42}$$

Se observa que la dimensión del puntaje de Fisher va a estar en función del número de parámetros θ y no del número de observaciones n. Este puntaje puede ser visto como un indicador de cuánto hay que modificar los parámetros del modelo θ para que el modelo ajuste mejor las observaciones X.

Para medir la similitud entre 2 ejemplos X e Y, Jaakkola y Haussler [107] propusieron usar el Fisher kernel utilizando los puntajes de Fisher:

$$K_{FK}(X,Y) = G_\theta^{X\,T} F_\theta^{-1} G_\theta^Y \tag{2.43}$$

Donde F_θ es la matriz de información de Fisher y se obtiene mediante:

$$F_\theta = E_{X\sim\theta}\left[G_\theta^X G_\theta^{X\,T}\right]$$

No obstante, calcular en forma exacta F_ϑ es muy costoso con lo cual se proponen diferentes aproximaciones que se verán en el apartado “Aproximación a la matriz de información de Fisher”. Luego, si existe la inversa de la matriz F_θ, entonces es posible utilizar la descomposición de Cholesky:

$$F_\theta^{-1} = L_\theta^{\,T} L_\theta \tag{2.44}$$

Reemplazando 2.44 en 2.43 se obtiene:

$$K_{FK}(X,Y) = G_\theta^{X\,T} L_\theta^{\,T} L_\theta G_\theta^Y \tag{2.45}$$

Tomando $L_\theta G_\theta^Y$ de 2.45 y reemplazando G_θ^Y por 2.42, se obtiene:

$$g_\theta^Y = L_\theta G_\theta^Y = L_\theta \nabla_\theta \log p(Y;\theta) \tag{2.46}$$

Donde a la ecuación 2.46 se la denomina vector de Fisher normalizado y por ende el Fisher kernel se obtiene como el producto interno de 2 vectores de Fisher normalizados (reescribiendo la ecuación 2.45 en base a la 2.46) como se muestra a continuación:

$$K_{FK}(X,Y) = g_{\theta}^{X\,T}\, g_{\theta}^{Y}$$

El hecho de utilizar los vectores de Fisher normalizados para entrenar un clasificador lineal es equivalente a entrenar un clasificador basado en kernels (por ejemplo el SVM) utilizando el kernel K_{FK}. Esto último tiene grandes ventajas dado que la utilización de clasificadores lineales permite que el entrenamiento de los mismos se realice rápidamente, aún en grandes conjuntos de datos.

Representación de la imagen mediante vectores de Fisher

Para la representación de una imagen mediante vectores de Fisher, se utiliza un modelo GMM para modelar la distribución de los descriptores locales. Se asume que cada observación x_i es independiente del resto. Luego, la función de probabilidad de una imagen X modelada con una GMM está dada por la ecuación 2.47.

$$p(X,\theta) = \prod_{i=1}^{n}\sum_{k=1}^{K} \pi_k \mathcal{N}(x_i,\ \theta_k); \qquad \sum_{k=1}^{K} \pi_k = 1 \tag{2.47}$$

Donde existen n descriptores locales, K componentes gaussianas y $N(.,\ \theta_k)$ es una distribución gaussiana con parámetros θ_k=$\{\pi_k,\ \mu_k, \Sigma_k\}$, es decir, el peso, la media y la matriz de covarianzas (ver ejemplo de un modelo GMM en la Figura 2.56).

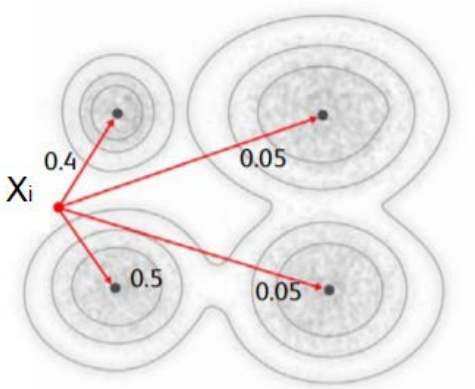


Figura 2.56: Al utilizar el modelo GMM, distintas componentes gaussianas tienen peso sobre una observación x_i.

Se asume una matriz de covarianzas diagonal. Esto reduce el tamaño de la matriz de covarianzas de una matriz D x D a un vector de dimensión D, que es el vector de varianzas. A continuación se observa la distribución gaussiana con la matriz diagonal de covarianzas, parametrizada por su media μ_k y su varianza θ_k:

$$N(x_i, \theta_k) = \frac{1}{(2\pi)^{D/2} \left|\sigma_k^2\right|^{1/2}} exp\left\{ -\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{id} - \mu_{kd})^2}{\sigma_{kd}^2} \right\}$$

Donde el sufijo d hace referencia a la dimensión del vector al cual está aplicado, por ejemplo, al utilizarlo en x_{id} nos referimos a la dimensión d del descriptor x_i .

Para evitar la restricción de los pesos de las componentes gaussianas (la sumatoria de los mismos tiene que ser igual a 1), los re-parametrizamos con el formalismo soft-max [108] de acuerdo a:

$$w_k = \frac{exp(\pi_k)}{\sum_{j=1}^{K} \exp(\pi_j)}$$

En el caso del modelo GMM , todas las observaciones $X=\{x_1,.....,x_n\}$ son independientes, entonces el puntaje de Fisher de estas observaciones para la componente de mezcla gaussiana k, estará dado por:

$$\nabla_{\theta_k} \; log \; p(X;\theta) = \sum_{i=1}^{n} \sum_{l=k}^{k} \gamma(l) \; \nabla_{\theta_k} \; log \; N(x_i;\theta_l)$$

$$\nabla_{\theta_k} \; log \; p(X;\theta) = \sum_{i=1}^{n} \gamma(k) \; \nabla_{\theta_k} \; log \; N(x_i;\theta_k) \qquad (2.48)$$

Donde:

$$\gamma(k) = \frac{w_k N(x_i;\theta_k)}{\sum_{l=1}^{K} w_l N(x_i;\theta_l)} \qquad (2.49)$$

La ecuación 2.49 es la probabilidad posterior (responsabilidad) de la componente k de la mezcla gaussiana para la observación x_i. Esta responsabilidad define una asignación suave (soft assignment) de x_i a la componente k.

Antes de calcular el puntaje final de Fisher para el modelo GMM, recordamos que el logaritmo de una distribución gaussiana teniendo una matriz de covarianzas diagonal de dimensiones D x D, está dado por:

$$logN(x_{i;\theta_k}) = -\frac{1}{2} \sum_{d=1}^{D} \left[log(2\pi) + log(\sigma_{kd}^2) + \frac{(x_{id} - \mu_{kd})^2}{\sigma_{kd}^2}) \right] \qquad (2.50)$$

Donde el sufijo d hace referencia a la dimensión del vector al cual está aplicado, por ejemplo, al utilizarlo en x_{id} nos referimos a la dimensión d del descriptor x_i .

Reemplazando la ecuación 2.50 en 2.48 y derivando la misma con respecto a los pesos, la media y la varianza, obtenemos los puntajes de Fisher de la imagen X con respecto a dichos parámetros de acuerdo a las siguientes ecuaciones:

$$G_{\pi_k}^X = \sum_{i=1}^{n} (\gamma_i(k) - w_k) \tag{2.51}$$

$$G_{\mu_k}^X = \frac{1}{\theta^2} \sum_{i=1}^{n} \gamma_i(k)(x_i - \mu_k) \tag{2.52}$$

$$G_{\sigma_k}^X = \sum_{i=1}^{n} \gamma_i(k) \left[\frac{(x_i - \mu_k)^2}{\sigma_k^3} - \frac{1}{\sigma_k} \right] \tag{2.53}$$

Concatenando los gradientes para todas las componentes gaussianas, obtenemos:

$$G_\theta^X = \left[G_{\pi_1}^X, \left(G_{\mu_1}^X\right)^T, \left(G_{\sigma_1}^X\right)^T, ..., G_{\pi_K}^X, \left(G_{\mu_K}^X\right)^T, \left(G_{\sigma_K}^X\right)^T \right]^T \tag{2.54}$$

La ecuación 2.54 corresponde la representación de la imagen X con un único vector (vector de Fisher). No obstante, se recomienda normalizarlo por medio de la matriz de información de Fisher para luego ser utilizado en el entrenamiento de un clasificador. Con dicha normalización se obtiene un mejor rendimiento en la clasificación y la misma será explicada en la próxima sección.

En resumen, el modelo de vectores de Fisher consta de las siguientes 3 etapas:

- *Extracción de descriptores locales*: Esta etapa no depende de la utilización de los vectores de Fisher. Con lo cual, es la misma etapa que en el modelo BoW. Por ejemplo, pueden usarse descriptores SIFT o mapas de características de una CNN.

- *Creación del vocabulario visual*: En este caso, el vocabulario visual se obtiene a través del modelo GMM. Este es entrenado con los descriptores locales y se utiliza el algoritmo EM visto en la sección 2.5.3.2.
- *Codificación de los descriptores locales*: En esta etapa se calculan las responsabilidades de las distintas componentes gaussianas por descriptor local (asignaciones a los conglomerados) y luego se calcula el vector de Fisher por imágen.

Aproximación de la matriz de información de Fisher (Para GMM)

En este apartado se tratará la aproximación de la FIM, para mayor detalles ver la sección “A” del apéndice de [9].

La aproximación de la matriz de información de Fisher para un modelo GMM está basada en que la distribución de la “asignación suave” (soft assignment) es “casi dura” (hard assignment), es decir, se cumple la ecuación:

$$\forall i \;\; \exists k \;\; \gamma(i) \approx 1$$

Lo que significa que cada descriptor local está asignado a una componente gaussiana k. Bajo estas condiciones, los valores de la matriz FIM están dados por:

$$[F]_{i,j} = E\left[-\frac{\partial^2 log p(x;\theta)}{\partial\theta_i\ \partial\theta_j}\right]$$

Donde θ_i y θ_j son parámetros específicos del modelo GMM (por ejemplo puede ser la media de una componente k y la varianza de una componente k'). Luego, la derivada parcial de la "asignación suave" (soft assignment - ver ecuación 2.49) de una componente k con respecto a la media y a la varianza (de una componente k') está dado por:

$$\frac{\partial\gamma_i(k)}{\partial\theta_{k'}} = \gamma_i(k)\ ([[k = k']] - \gamma_i(k'))\ \frac{\partial log\ p(x;\theta_{k'})}{\partial\theta_{k'}}$$

Donde [[·]] es la notación de Iverson, donde resulta 1 si el argumento es verdadero, de lo contrario es igual a 0. El hecho de asumir que la distribución de "asignación suave" es "casi dura" implica que las derivadas parciales son aproximadamente nulas. Luego, esta asumpción implica las siguientes aproximaciones:

$$\gamma_i(k)\ \gamma_i(k') \approx 0 \qquad si\ k \neq k'$$

$$\gamma_i(k) \approx \gamma_i(k)\ \gamma_i(k') \qquad si\ k = k'$$

Teniendo en cuenta estas aproximaciones y las ecuaciones 2.51, 2.52 y 2.53, las derivadas segunda serán nulas si:

- Involucran los parámetros de media o varianza de distintas componentes gaussianas ($k \neq k'$). Es decir, se cumplirá que la derivada segunda sea nula cuando θ_i y θ_j son 2 parámetros (con $i = j$) pero los mismos pertenecen a distintas componentes gaussianas.

- Involucran una mezcla de parámetros entre el peso, la media o la varianza (esto también es válido cuando dichos parámetros corresponden a una misma componente gaussiana). Es decir, se cumplirá que la derivada segunda sea nula cuando θ_i y θ_j (2 parámetros con $i \neq j$) pertenezcan a la misma o diferente componente gaussiana.

Con lo cual, cuando usamos la asumpción de "asignación dura" para aproximar la segunda derivada con respecto a la media y a la varianza, todos los términos que involucran la varianza y la media (de la misma componente gaussiana k) serán 0. Dando lugar a:

$$[F]_{\sigma_k\mu_k} = -\int_x p(x_i;\theta)\ \frac{\partial^2 log\ p(x_i;\theta)}{\partial\sigma_k\ \partial\mu_k}\ dx \approx 0$$

Esto significa que todos los términos fuera de la diagonal principal de la FIM serán nulos, tomando por ende la forma de una matriz diagonal.

Luego, considerando nuevamente la asumpción de “asignación dura”, tomamos la derivada segunda con respecto a la media resultando la aproximación analítica de la FIM para el parámetro μ_k, dando lugar a:

$$[F]_{\mu_k,\mu_k} = \sigma_k^{-2} w_k \qquad donde\ L_\theta = \sigma_k (w_k)^{-1/2} \tag{2.55}$$

En forma similar si realizamos la derivada segunda con respecto a la varianza, obtenemos la aproximación analítica de la FIM para el parámetro θ_k, dando lugar a:

$$[F]_{\sigma_k,\sigma_k} = 2\,\sigma_k^{-2} w_k \qquad donde\ L_\theta = \sigma_k (2w_k)^{-1/2} \tag{2.56}$$

Por último, para la normalización del gradiente con respecto los pesos, se demuestra en la sección “A” del apéndice de [9] que L_θ es:

$$L_\theta = w_k^{-1/2} \tag{2.57}$$

A continuación, para obtener los gradientes normalizados por medio de la FIM (por medio de la ecuación 2.46), utilizamos las ecuaciones 2.55, 2.56 y 2.57 de L_θ en las ecuaciones 2.51, 2.52 y 2.53, dando lugar a las siguientes ecuaciones:

$$g_{\pi_k}^X = \frac{1}{\sqrt{w_k}} \sum_{i=1}^{n} (\gamma_i(k) - w_k) \tag{2.58}$$

$$g_{\mu_k}^X = \frac{1}{\sqrt{w_k}} \sum_{i=1}^{n} \gamma_i(k) (\frac{x_i - \mu_k}{\sigma_k}) \tag{2.59}$$

$$g_{\sigma_k}^X = \frac{1}{\sqrt{w_k}} \sum_{i=1}^{n} \gamma_i(k)\ \frac{1}{\sqrt{2}} \left[\frac{(x_i - \mu_k)^2}{\sigma_k^2} - 1 \right] \tag{2.60}$$

Por último, concatenamos los gradientes normalizados para todas las componentes gaussianas obteniendo el vector de Fisher normalizado por la FIM (que es el utilizado en el presente trabajo):

$$g_\theta^X = \left[g_{\pi_1}^X, \left(g_{\mu_1}^X\right)^T, \left(g_{\sigma_1}^X\right)^T, ..., g_{\pi_K}^X, \left(g_{\mu_K}^X\right)^T, \left(g_{\sigma_K}^X\right)^T \right]^T$$

Teniendo en cuenta los estadísticos de orden cero, uno y dos de las siguientes ecuaciones:

$$S_k^0 = \sum_{i=1}^{n} \gamma_i(k)$$

$$S_k^1 = \sum_{i=1}^{n} \gamma_i(k) x_i$$

$$S_k^2 = \sum_{i=1}^{n} \gamma_i(k) x_i^2$$

Es posible sustituir dichos estadísticos en las ecuaciones de los gradientes normalizados por medio de la FIM (ver ecuaciones: 2.58, 2.59 y 2.60). Dando lugar al cálculo de los vectores de Fisher según el algoritmo 2.3.

El algoritmo 2.3 es el que se utiliza en el presente trabajo. El mismo se tuvo que programar desde cero, dado que no se encontraron implementaciones existentes que codfiquen los pesos de las componentes gaussianas en los vectores de Fisher. En dicho algoritmo se observan 2 normalizaciones, la normalización de potencia (power normalization - “square rooting”) y la L^2. En [9] los autores verificaron que con estas normalizaciones se incrementa el rendimiento del clasificador que utilice como entrada los mencionados vectores de Fisher.

Algorithm: 2.3 - Genenerar FVs en base a descriptores locales

Entrada:

-Descriptores locales de una imagen $X = \{ x_t \in \Re^D, \ t = 1, ..., T \}$

-Parámetros del modelo GMM $\theta = \{ w_k, \ \mu_k, \ \sigma_k, \ k = 1, ..., K \}$

Salida:

-Vector de Fisher normalizado $g_\theta^X \in \Re^{K(2D+1)}$

1. Calcular estadísticos

- For $k = 1, ..., K$ inicializar acumuladores
  - $S_k^0 \leftarrow 0, \quad S_k^1 \leftarrow 0, \quad S_k^2 \leftarrow 0$
- For $t = 1, ..., T$
  - $\gamma_t(k) = w_k \ N(x_t, \theta_k) \ / \ \sum_{j=1}^{k} w_j \ N(x_t, \theta_j)$ [Asignación suave - Soft assignment]
  - For $k = 1, ..., K$:
    - $S_k^0 \leftarrow S_k^0 + \gamma_t(k)$,
    - $S_k^1 \leftarrow S_k^1 + \gamma_t(k) \ x_t$,
    - $S_k^2 \leftarrow S_k^2 + \gamma_t(k) \ x_t^2$,

2. Calcular los vectores de Fisher

- For $k = 1, ..., K$:

$$g_{\pi_k}^X = (S_k^0 - T w_k) / \sqrt{w_k}$$

$$g_{\mu_k}^X = (S_k^1 - \mu_k S_k^0) / \sqrt{w_k \sigma_k}$$

$$g_{\sigma_k}^X = (S_k^2 - 2 \mu_k S_k^1 + (\mu_k^2 - \sigma_k^2) S_k^0) / \sqrt{w_k \sigma_k}$$

- Concatenar todas los componentes de los FVs en un solo vector.

$$g_\theta^X = \left(g_{\alpha_1}^X, ..., g_{\alpha_k}^X, \left(g_{\mu_1}^X\right)^T, ..., \left(g_{\mu_k}^X\right)^T, \left(g_{\sigma_1}^X\right)^T, ..., \left(g_{\sigma_k}^X\right)^T \right)^T$$

3. Aplicar normalizaciones

- For $k = 1, ..., K(2D+1)$ aplicar la normalización de potencia (power normalization)
  - $[g_\theta^X]_i \leftarrow sign([g_\theta^X]_i) \sqrt{| [g_\theta^X]_i |}$
- Aplicar L^2 - normalización:
  - $g_\theta^X = g_\theta^X / \sqrt{\left(g_\theta^X\right)^T g_\theta^X}$

2.5.3.6 Máquina de vectores de soporte (SVM)

La máquina de vectores de soporte (SVM - Support Vector Machine) fue desarrollada en 1963 por Vapnik y Lerneren en los laboratorios AT&T, pero no fue hasta los años noventa, con la publicación del primer paper por Boser en 1992 y por Cortes y Vapnik en 1995 que se conoció en forma masiva. Corresponde a la familia de algoritmos supervisados (explicado en la sección 2.2) y fue ideada originalmente para la resolución de problemas de clasificación binaria, en los cuales las clases eran linealmente separables [109]. Por tal motivo, originariamente se lo denominó con el nombre “Hiperplano de Separación Óptima” (OSH - Optimal Separating Hiperplane). En la solución que se obtenía se clasificaban en forma correcta todas las muestras, colocando el hiperplano de separación lo más lejano posible de las mismas [110]. Es decir, se trata de encontrar un hiperplano que separe las instancias en 2 partes, en donde en cada parte estarán los puntos del conjunto de datos de entrenamiento que pertenecen a una categoría determinada (“Melanoma” o “No melanoma”) . Esto es mostrado en la figura 2.57.

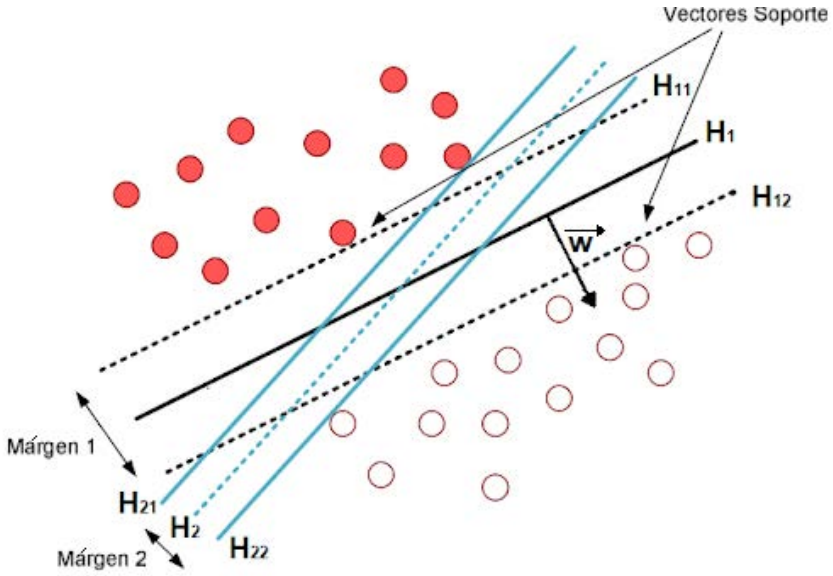


Figura 2.57: Fronteras de decisión y márgenes en un clasificador SVM.

En la figura 2.57 los puntos rellenos de color rojo pertenecen a una clase mientras que los otros puntos pertenecen a otra clase. En esta técnica se trata de encontrar un hiperplano (H_1 o H_2) que separe las 2 clases basado en el concepto de “máximo margen”. A estos planos H_i se los denomina frontera de decisión. Además, se observa que cada hiperplano H_i está asociados a un par de hiperplanos de soporte (H_{i1} y H_{i2}) que son paralelos al mencionado hiperplano H_i y pasan por los puntos más cercanos a H_i . La distancia entre estos hiperplanos de soporte se llama “margen”. En la figura 2.57 se observa que ambos hiperplanos (H_1 y H_2) dividen los puntos de acuerdo a su clase. No obstante, H_1 tiene un margen más amplio que H_2 y por ende H_1 tendrá un mejor rendimiento en clasificar muestras nuevas que H_2. Con lo cual, cuanto más amplio el margen, menor será el error de generalización para la clasificación de muestras desconocidas.

SVM con clases linealmente separables

Dado un conjunto linealmente separable de ejemplos S= {(x_1,y_1),.....,(x_n,y_n) }, donde $x_i \in \mathbb{R}^D$ e $y_i \in$ {+1,-1} se puede definir un hiperplano de separación (ver figura 2.57) como una función lineal que es capaz de separar dicho conjunto sin error:

$$w * x + b = 0$$

Donde w y x son vectores, la dirección de w es normal a la frontera de decisión lineal (hiperplano) y "*" es el producto interno entre 2 vectores. Para cualquier punto x_i que se encuentre por arriba del hiperplano se verifica:

$$w * x_i + b = k \quad , \; donde \; k > 0 \tag{2.58}$$

Y para cualquier punto x_j que se encuentre por debajo del hiperplano, se verifica:

$$w * x_j + b = k' \quad , \; donde \; k' < 0 \tag{2.59}$$

Si dividimos las ecuaciones 2.58 y 2.59 por k y por -k' respectivamente (re-escalando los valores de b y el vector w), obtenemos las ecuaciones que pertenecen a los hiperplanos de soporte H_{11} y H_{22}:

$$H_{11} : w * x_i + b = 1 \tag{2.60}$$
$$H_{12} : w * x_j + b = -1 \tag{2.61}$$

Luego, la distancia entre los 2 hiperplanos que es el margen ("margen 1" en la figura 2.57) se halla dividiendo las restricciones 2.60 y 2.61 por $||w||$ y luego restando las ecuación 2.61 a la 2.60, obteniendo:

$$\frac{w}{||w||} * (x_i - x_j) = \frac{2}{||w||} \tag{2.62}$$

La parte izquierda de la ecuación 2.62 puede ser vista como la distancia entre los hiperplanos H_{11} y H_{22}. Con lo cual, el objetivo principal del SVM es la de maximizar $d = 2/||w||$, que es equivalente a minimizar el funcional $||w||^2 / 2$ sujeta a las restricciones [110]:

$$w * x_i + b \geq 1 \qquad si \; y_i = 1 \tag{2.63}$$
$$w * x_i + b \leq -1 \qquad si \; y_i = -1 \tag{2.64}$$

Unificando las ecuaciones 2.63 y 2.64 se obtiene la ecuación:

$$y_i (w * x_i + b) \geq 1 \tag{2.65}$$

Donde y_i es la clase asociada al punto x_i. Imponiendo esta restricción, el SVM colocará todos los puntos que tienen asociada la clase $y_i = 1$ por arriba del hiperplano H_{11} y a los puntos que tienen asociada la clase y_i = -1 los ubicará por debajo del hiperplano H_{12}.

El problema de optimización de minimizar $f(w) = ||w||^2 / 2$ sujeta a la restricción de la ecuación 2.65 puede ser resuelto por el método de los multiplicadores de Lagrange[9]. La condición impuesta sobre nuestra función será reemplazada por las condiciones propias de los multiplicadores de Lagrange, lo cual será más fácil de manejar. Con lo cual, se deberá encontrar el punto de silla de la siguiente función Lagrangiana [111].

$$max_{\alpha}\, min_{w,b}\; L_p(w, b, \alpha) = ||w||^2/2 + \sum_{i=1}^{N} \alpha_i[(w * x_i + b) - 1] \qquad con\ \alpha_i >= 0 \tag{2.66}$$

Donde α_i son los multiplicadores de Lagrange y N es el número de ejemplos en el conjunto de datos de entrenamiento. Para encontrar el punto de silla debemos minimiza L_p con respecto a $w\, y\, b$, y luego maximizarla con respecto a los multiplicadores de Lagrange (con $\alpha_i >$ 0). Este punto de silla debe satisfacer las condiciones de Karush-Kun-Tucker (KKT) [111]. Luego, calculando las derivadas parciales de L_p con respecto a w y a b e igualando a cero, obtenemos las ecuaciones:

$$\frac{\partial L_p}{\partial w} = 0 \;\Rightarrow w = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{2.67}$$

$$\frac{\partial L_p}{\partial b} = 0 \;\Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.68}$$

Por último sustituyendo 2.67 y 2.68 en 2.66 y simplificando obtenemos la forma dual [111] de la función objetivo del SVM, la cual se encuentra expresada por:

$$max_{\alpha}\; L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j\, y_i\, y_j\, (x_i * x_j) \tag{2.69}$$

Sujeta a las siguientes restricciones:

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Se observa en la ecuación 2.69 que la forma dual depende solo de α y el problema se transforma en maximizar la función L_d en lugar de minimizar la función primal L_p. La resolución de la función dual puede ser resuelta por técnicas de programación cuadrática convencionales. Una vez encontrado el $\alpha^0 = \{\alpha_i^0 .....\alpha_N^0\}$que es solución de 2.69, el hiperplano de separación óptima de variables w y b, tendrá a w_0 (reemplazando α^0 en la ecuación 2.67):

$$w_0 = \sum_{i=1}^{N} \alpha_i^0\; y_i\; x_i$$

Mientras que b_0 puede ser obtenido mediante la aplicación de la segunda condición de KKT [111]:

$$\alpha_i^0\, [\; y_i\; (w * x_i + b) - 1] = 0 \tag{2.70}$$

[9] https://es.wikipedia.org/wiki/Multiplicadores_de_Lagrange

De la ecuación 2.70 se deduce que aquellas instancias del conjunto de entrenamiento en las cuales $\alpha_i > 0$ satisfacen las restricciones de 2.65. Geométricamente son los puntos más cercanos al hiperplano H_1, son llamados vectores de soporte y se encontrarán en el hiperplano H_{11} o H_{12} (ver figura 2.57). Las instancias con $\alpha_i = 0$ no son consideradas vectores de soporte y se consideran irrelevantes para la generación del hiperplano. Con lo cual, conociendo w_0 podemos hallar b_0 de la ecuación 2.70 cuando $\alpha_i > 0$:

$$b_0 = y_i - w_0 * x_i \quad \text{; para cualquier vector de soporte } x_i$$

Luego, el problema de clasificar un nuevo punto x_i se resuelve evaluando el signo de $w_0\, x_i + b_0$.

Por último, en el replanteamiento del problema en la forma dual, los datos de entrenamiento aparecerán en forma de productos escalares, lo cual no solo facilita los cálculos, sino que permite la generalización para los casos no lineales (ver apartado: SVM con clases no separables linealmente).

SVM con clases "cuasi" linealmente separables (margen suavizado)

Existen casos en que es preferible clasificar erróneamente algunos de los ejemplos del conjunto de entrenamiento para obtener un hiperplano que tenga un margen máximo (ver figura 2.58). Es decir, se permiten ciertas instancias mal clasificadas y se encuentra un hiperplano óptimo para el resto de los ejemplos que son linealmente separables. Un hiperplano que separe bien las clases pero que tenga un margen pequeño es propenso a realizar sobreajuste y por ende no podrá clasificar correctamente ejemplos desconocidos. Por otro lado, aquellos hiperplanos que tengan un margen más grande y que tengan algunos puntos mal clasificados, podrán clasificar ejemplos desconocidos con una mejor Precisión. Es decir, se trata de encontrar un compromiso entre el margen y el número de instancias mal clasificadas. Con lo cual, teniendo en cuenta este compromiso, se calculará un margen y se llamará "margen blando".

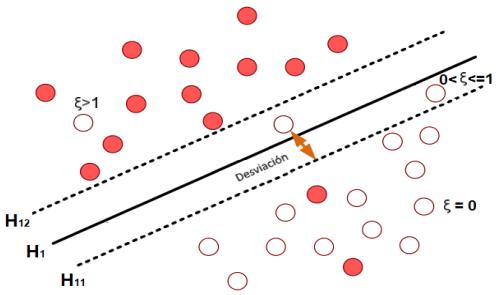


Figura 2.58: Fronteras de decisión con márgenes "blandos".

A las restricciones impuestas en 2.63 y 2.64 se les deberá agregar una variable de holgura ξ que tendrá en consideración el margen blando (ver figura 2.58). Dada una instancia x_i (con una clase y_i) se le asociará una variable de holgura ξ_i, siendo la misma una desviación de x_i con respecto al hiperplano (H_{11} o H_{12}) asociado a su categoría y_i. De esta forma si la desviación es 0 ($\xi_i = 0$), corresponderá a una instancia bien clasificada y separable. Una desviación mayor a 0 pero menor o igual a 1 ($0 < \xi_i <= 1$), corresponderá a una instancia

correctamente clasificada pero no separable, es decir situada dentro del margen calculado. Mientras que para una instancia mal clasificada, la desviación será mayor a 1 ($\xi_i > 1$). Con lo cual, a la sumatoria de estas variables de holgura se las puede ver como una penalización de los ejemplos no separables, ya sea bien clasificados o mal clasificados. Por ende, se agrega a la función objetivo esta penalización y de esta forma se balancea el tamaño del margen y los ejemplos no separables, dando lugar a la minimización de la siguiente función objetivo:

$$min_{w,b} f(w) = ||w||^2/2 + C \left(\sum_{i=1}^{N} \xi_i\right) \tag{2.71}$$

Sujeta a las siguientes restricciones:

$$w * x_i + b \geq 1 - \xi_i \qquad si\, y_i = 1$$
$$w * x_i + b \leq -1 + \xi_i \qquad si\, y_i = -1$$

Donde C (o costo) de la ecuación 2.71 es especificado por el usuario y establece el grado de penalidad para los ejemplos no separables. Grandes valores de C corresponden a grandes penalizaciones mientras que para bajos valores de C el modelo es menos estricto sobre los errores de clasificación. Con lo cual, podemos utilizar el parámetro C para controlar el ancho del margen y por ende optimizar el balance entre el sesgo y la varianza tal como indica la figura 2.59.

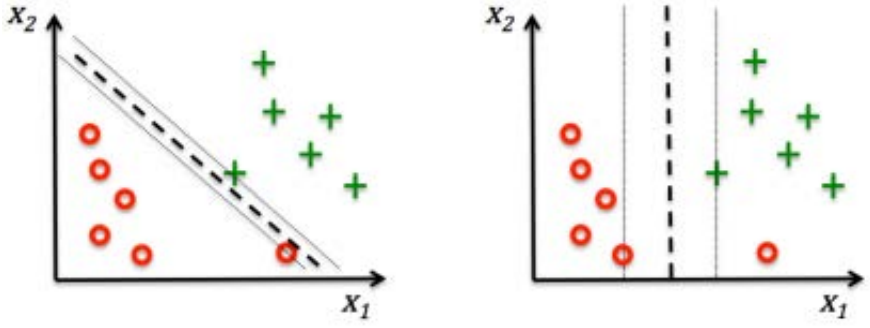


Figura 2.59: (Izquierda) Margen de separación para grandes valores de C. (Derecha) Margen de separación para bajos valores de C.

Utilizando los multiplicadores de Lagrange en la ecuación 2.71 y realizando las derivadas parciales con respecto a w, b y ξ e igualando a 0, llegamos a la misma forma dual expresada en la ecuación 2.69. No obstante, los multiplicadores de Lagrange difieren para los calculados en el caso “linealmente separable”. En este caso, además de cumplir $\alpha_i >= 0$, deberán cumplir con $\alpha_i <= C$. Obteniendo la siguiente forma dual a ser maximizada:

$$max_\alpha L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j (x_i * x_j)$$

Sujeta a las siguientes restricciones:

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$
$$0 \leq \alpha_i \leq C$$

SVM con clases no separables linealmente

Existen conjuntos de datos complejos para los cuales no es posible hallar una frontera de clasificación lineal para separar las clases. Con lo cual, surgen los SVM de clasificación no lineal. El concepto de los clasificadores SVM no lineales radica en transformar el conjunto de datos en otro conjunto de datos de dimensión mayor, donde las clases son separables linealmente (ver figura 2.60). El Teorema de Cover [112] proporciona la justificación de por qué una transformación no lineal a un espacio de mayor dimensión aumenta las posibilidades de disponer de un conjunto de datos separables linealmente, si estos no lo eran en el espacio original. En este caso la función dual presentada en 2.69 para el caso linealmente separable, se mantiene igual, solo hay una diferencia y radica en reemplazar los atributos x por la transformada de lo mismos, $\phi(x)$ al espacio de mayor dimensión, obteniéndose la siguiente ecuación la cual deberá ser maximizada:

$$max_{\alpha} L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j (\Phi(x_i) * \Phi(x_j)) \quad (2.72)$$

El principal problema en transformar este conjunto de datos a una dimensión mayor es que se tiene un aumento de complejidad en el clasificador. Además, la función de mapeo del conjunto de datos original al conjunto de datos de mayor dimensión $\phi(x)$ es desconocida. Recordemos que por definición, una función kernel (núcleo) es una función $K : X \times X \rightarrow \mathbb{R}$ que asigna a cada par de elementos del espacio original, X, un valor real correspondiente al producto escalar de las imágenes de dichos elementos en un nuevo espacio (espacio de características). Con lo cual, los productos escalares de la forma $\phi(x_i)*\phi(x_j)$ que se tienen en 2.72 se pueden calcular haciendo uso del kernel definido como:

$$K(x_i, x_j) = \Phi(x_i) * \Phi(x_j) \quad (2.73)$$

Y reemplazando la ecuación 2.73 en 2.72, obtenemos:

$$max_{\alpha} L_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad (2.74)$$

En la ecuación 2.74 resolvimos el cálculo costoso $\phi(x_i)*\phi(x_j)$ mediante la incorporación de un kernel, el cual opera con los atributos del espacio original y por ende no hay necesidad de conocer la función de mapeo $\phi(x)$. A esto último se lo conoce como el “truco del kernel” y conlleva a una reducción de la complejidad del clasificador y de la potencia de cómputo necesaria para el mismo. Para que lo anterior sea válido, la función kernel debe complir con el teorema de Mercer [111]. La ecuación 2.74 se resuelve de la misma forma que la ecuación 2.69, ya que en el espacio de alta dimensión lo los ejemplos son separados linealmente.

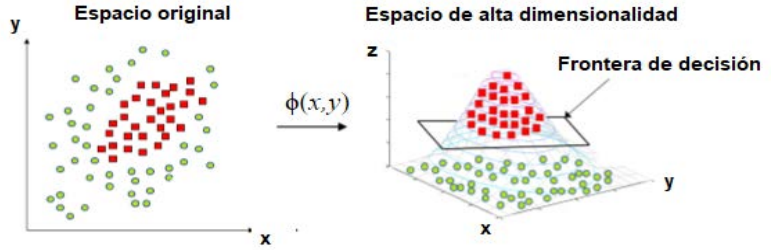


Figura 2.60: Transformación de los datos del espacio original a uno de alta dimensión (espacio de características) mediante la función de mapeo $\phi(x)$.

Las funciones de kernel utilizadas habitualmente son:

Kernel lineal: Se recomienda cuando la representación de los datos es dispersa.

$$K(x_i, x_j) = x_i * x_j$$

Kernel polinomial: Es un kernel muy utilizado para modelar relaciones no lineales. No obstante, a medida que aumenta d (grado del polinomio) la superficie de clasificación se hace más compleja.

$$K(x_i, x_j) = (1 + x_i * x_j)^d$$

Kernel Gaussiano: También denominado radial es uno de los más utilizados. El parámetro θ controla la forma del hiperplano que separa las clases.

$$K(x_i, x_j) = exp\left(-\frac{||x_i - x_j||^2}{2\sigma}\right) = exp\left(-\gamma \left|\left| x_i - x_j \right|\right|^2\right); \quad \text{con } \gamma = 1/2\sigma$$

Donde γ es un parámetro a optimizar por el usuario e indica hasta donde llega la influencia de un ejemplo de entrenamiento en el cálculo de la frontera de decisión. Para valores bajos de γ , ejemplos lejos de la frontera de decisión son tenidos en cuenta para su cálculo, con lo cual se crea un modelo con bajo sesgo, alta varianza y se corre el riesgo de subajuste. Mientras que para altos valores de γ solo se tienen en cuenta los ejemplos cercanos a la frontera para la determinación de la misma, dando lugar a un modelo con alto sesgo, baja varianza y se corre el riesgo de sobreajuste.

2.5.3.7 Arquitectura del Modelo Híbrido

Una vez explicados los elementos y modelos que componen el Modelo Híbrido en forma separada, en la figura 2.61 se visualiza la arquitectura del mismo y cómo se interrelacionan estos elementos. A continuación se explicará el modelo en forma conceptual mientras que en las secciones 3.3.3 y 3.3.4 se explicará con mayor nivel detalle.

En primer lugar se tomó la red VGG-16 pre-entrenada con las imágenes de ImageNet. Luego esta red fue reentrenada con las imágenes de las lesiones pigmentadas (realizando fine tuning). Los detalles de cómo se realizó el entrenamiento y el fine-tuning se encuentra en la sección 3.3.3 y 3.3.4. Finalizado el entrenamiento, se empleó la mencionada red para la extracción de características. La extracción corresponde a los mapas de características de la de la última capa de pooling. Es decir, son 512 mapas con dimensiones 7 X 7. Se aclara que las dimensiones de los mapas de características son de 7 x 7 porque se utilizaron imágenes escaladas con las mismas dimensiones que las que se utilizaron en ImageNet, es decir 224 x 224. Posteriormente, a estas características extraídas se las redimensionan como si estuvieran ingresando a la capa totalmente conectada de la VGG-16 (es decir, se utiliza el método flatten) y luego se aplica PCA. Dependiendo de la cantidad de componentes principales que utilicemos, será la longitud de nuestro descriptor. En el caso genérico aquí

expuesto, la longitud es D. Luego, se aplicará un modelo GMM para modelar la distribución de los descriptores. En este caso genérico presentado, la GMM tendrá N componentes. A continuación se realizará la "asignación suave" del descriptor a cada componente gaussiana para posteriormente realizar la codificación de los vectores de Fisher. En el vector de Fisher estarán los 3 gradientes normalizados por la matriz de información de Fisher ($g^X_{\pi_k}$, $g^X_{\mu_k}$, $g^X_{\theta_k}$), donde las dimensiones son 1, D y D respectivamente. Dichos gradientes son calculados para cada componente gaussiana y dado que existen N componentes gaussianas, la dimensión del vector de Fisher por cada imagen será: $N*1 + N*D + N*D = N*(2D+1)$ (para más detalles ver 2.5.3.5). Por último, estos vectores de Fisher constituirán la entrada para realizar el entrenamiento de un SVM.

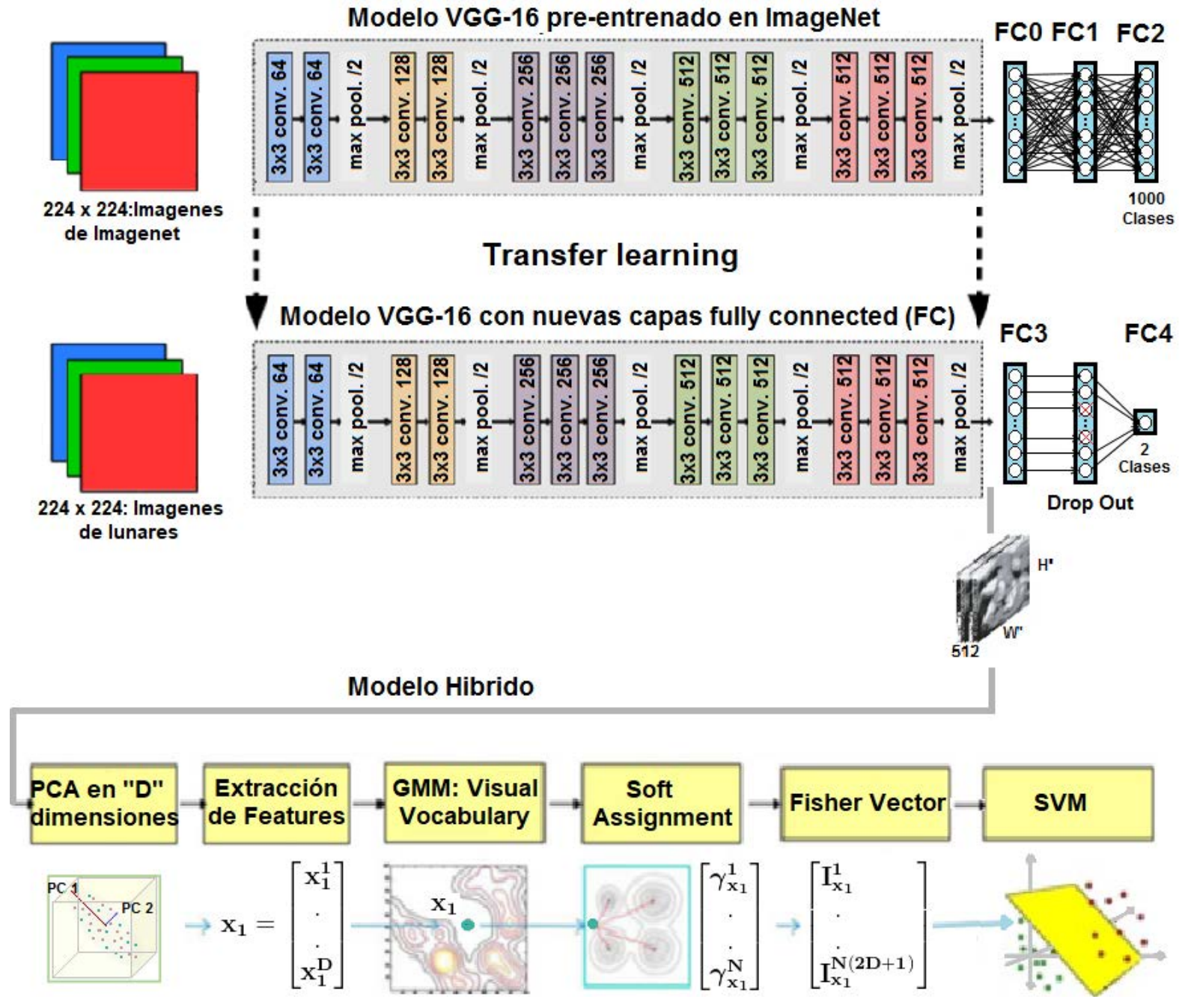


Figura: 2.61: Arquitectura del Modelo Híbrido utilizado en el presente trabajo.

3 Conjuntos de datos y métodos

3.1 Métricas utilizadas

Previamente a especificar cuáles métricas son las utilizadas para evaluar los clasificadores binarios, explicaremos los conceptos que sirven de base para la construcción de dichas métricas.

- **VP o verdaderos positivos**: Son aquellas lesiones pigmentadas clasificadas como melanomas y que efectivamente son melanomas.
- **VN o verdaderos negativos**: Son aquellas lesiones pigmentadas clasificadas como “no melanomas” y que efectivamente no son melanomas.
- **FP o falsos positivos**: Son aquellas lesiones pigmentadas clasificadas como melanomas pero que en realidad son “no melanomas”.
- **FN o falsos negativos**: Son aquellas lesiones pigmentadas clasificadas como “no melanomas” pero que en realidad son melanomas.

Los conceptos anteriores se pueden visualizar en la figura 3.1.

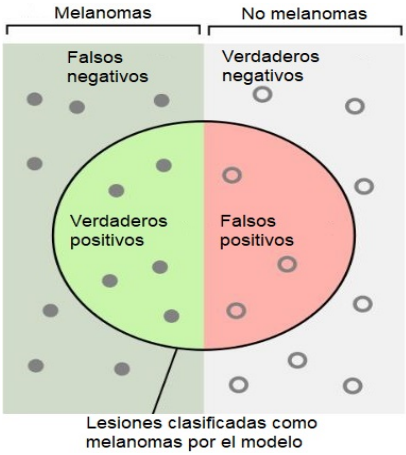


Figura 3.1: Los círculos llenos representan las lesiones del tipo “Melanoma” mientras que los círculos vacíos referencian las lesiones del tipo "No melanoma". Los círculos que se encuentran dentro de la elipse corresponden a las lesiones clasificadas como melanomas por el modelo predictivo.

Las métricas utilizadas en el presente trabajo las dividiremos en **métricas complementarias** y **métricas principales**. Las métricas principales corresponden a las que usaremos para comparar los distintos modelos o evaluar el entrenamiento de los mismos. Las métricas complementarias son aquellas que se tendrán en cuenta solo cuando distintos modelos comparados o evaluados tienen la misma (o muy similar) métrica principal.

Métricas complementarias

- **Especificidad o Razón de Verdaderos Negativos (VNR):** Mide la proporción de pacientes clasificados correctamente que no tienen Melanoma (VN) del total de los pacientes que no tienen melanomas. Lo anterior se puede visualizar en:

$$\text{VNR} = \frac{Número\ de\ verdaderos\ negativos}{Número\ de\ ejemplos\ negativos} = \frac{VN}{VN + FP} \quad (3.1)$$

- **Recall o Sensibilidad o Razón de Verdaderos Positivos (VPR)**: Indica la fracción de los melanomas que son correctamente clasificados (VP) sobre el total de melanomas de la base. Lo anteriormente mencionado se traduce en:

$$\text{VPR} = \frac{Número\ de\ verdaderos\ positivos}{Número\ de\ ejemplos\ positivos} = \frac{VP}{VP + FN} \quad (3.2)$$

  Esta métrica es utilizada en las curvas ROC y PR.

  Se observa que en el denominador de la ecuación 3.2 se encuentran los falsos Negativos y cuando estos tienden a 0, la Sensibilidad tenderá a 1. Por otro lado la Especificidad no tiene en cuenta los falsos negativos. Por ende, en el contexto del diagnóstico de melanomas, la Sensibilidad es más importante que la Especificidad dado que el costo de clasificar incorrectamente a un paciente como sano (sin melanoma) es mucho más serio que clasificar incorrectamente a un paciente como enfermo (con melanoma). En el primer caso, el costo puede llegar a ser la pérdida de una vida mientras que en el segundo caso es una pérdida de dinero y tiempo (se enviará al paciente a realizarse los estudios médicos pertinentes cuando en realidad no era necesario). Al querer aumentar la Sensibilidad y reducir los falsos negativos, se incrementarán los falsos positivos haciendo que la Especificidad disminuya (ver ecuaciones 3.1 y 3.2). Sin embargo tampoco queremos que la Especificidad sea muy baja (muchos falsos positivos) ya que tanto el tiempo y el dinero no son ilimitados. Por ende, lo que se busca es un balance entre la Especificidad y la Sensibilidad, es decir, tener una Sensibilidad relativamente alta sin descuidar la Especificidad.

- **Ratio o Razón de Falsos Positivos *(*FPR)**: A este indicador también se lo conoce como razón de falsas alarmas, computa la fracción de los melanomas clasificados incorrectamente (falsos positivos) en relación al total de lesiones pigmentadas que no son melanomas. Lo anterior se encuentra representado por:

$$\text{FPR} = \frac{Número\ de\ falsos\ positivos}{Número\ de\ ejemplos\ negativos} = \frac{FP}{VN + FP} \quad (3.3)$$

  Esta métrica corresponde a las abscisas de la curvas ROC. De acuerdo la ecuación 3.1 y a la ecuación 3.3, se observa que FPR = 1 - Especificidad.

- **F1-Score:** Este indicador computa el promedio armónico entre la Precisión y el Recall. Su rango de valores se encuentra comprendido entre 1 (perfecta Precisión y Recall) y 0 cuando se obtiene el peor balance entre estos 2 indicadores. Se calcula de la siguiente forma:

$$\text{F1-Score} = 2 * \frac{Precisión * Recall}{Precisión + Recall}$$

- **Precisión:** La Precisión o valor predictivo positivo se define como la proporción de verdaderos positivos (VP) contra todos los resultados positivos (tanto verdaderos positivos (VP), como falsos positivos (FP)). Este indicador es utilizado en la ordenada de la curva PR y se calcula de acuerdo a:

$$\text{Precisión} = \frac{número\ de\ predicciones\ positivas\ correctas}{número\ de\ predicciones\ positivas} = \frac{VP}{VP + FP}$$

- **Curva Precisión-Recall (PR) y PR-AUC:** La curva PR muestra la relación entre la **Precisión** y el **Recall** para todos los posibles umbrales (valor a partir del cual decidimos que un caso es un positivo) de discriminación de clases. Se trata de obtener clasificadores con un balance entre estos 2 indicadores, ya que al aumentar uno, el restante tiende a disminuir. Por tal motivo, esta curva se utiliza para comparar el rendimiento de distintos clasificadores y visualizar el equilibrio existente entre el Recall y la Precisión. Una forma de resumir cada curva PR a un solo número es por medio del **PR-AUC** (Área bajo la curva PR) o también llamado Average Precisión (**AP**). Este último indicador permite comparar los clasificadores de una forma sencilla.

Métricas principales

Segmentación

- **Coeficiente de Dice (índice de Sørensen-Dice):** Este coeficiente se utilizó como función de pérdida en el módulo del Segmentador cuando la red TernausNet-16 fue entrenada. Dada la máscara binaria X que indica el ground truth de los píxeles que se corresponden a una lesión pigmentada, y la máscara binaria Y a la obtenida por la red, entonces el Coeficiente de Dice está definido por:

$$\text{Coeficiente de Dice} = \frac{2 * |X \cap Y|}{|X| + |Y|}$$

Dado que son máscaras binarias, la intersección indica las posiciones donde ambas máscaras son verdaderas (o que indican la existencia de melanoma). Además, se aclara que la función "|·|" cuenta la cantidad de píxeles verdaderos. Con lo cual, este coeficiente tendrá su valor máximo en 1 cuando los píxeles que indican la presencia de melanoma en la máscara predicha por la red coincidan con los de la ground truth. Por otro lado, alcanzará su valor mínimo (en 0) cuando no existan píxeles de melanoma en común. Este coeficiente es utilizado en la función de pérdida, donde el rango de dicha función es [-1,0] ya que está definida cómo:

$$Función\ de\ pérdida = - Coeficiente\ de\ Dice$$

- **Coeficiente de Jaccard:** Este índice se utiliza para evaluar la performance del modelo en función de las máscaras predichas respecto de las verdaderas (ground truth). Se corresponde al cociente entre la cantidad de píxeles verdaderos que se superponen de ambas máscaras, y el total de píxeles verdaderos que surgen de la unión de las dos máscaras. Dada la máscara binaria X que indica el ground truth de los píxeles que se corresponden a una lesión pigmentada, y la máscara binaria Y a la obtenida por la red, entonces el Coeficiente de Jaccard está dado por:

$$\text{Coeficiente de Jaccard} = \frac{|X \cap Y|}{|X \cup Y|}$$

Recordemos que la función "|·|" cuenta la cantidad de píxeles verdaderos de una máscara.

Clasificación

- **Exactitud (Accuracy):** Esta métrica toma en consideración el número de lesiones pigmentadas en las cuales se tuvo una predicción correcta en relación al número total de lesiones pigmentadas de la base de datos. Con lo cual, se calcula de la siguiente forma:

$$\text{Exactitud} = \frac{\textit{número de predicciones correctas}}{\textit{número de ejemplos de la base}} = \frac{VP+VN}{VP+VN+FP+FN}$$

Se la utilizó en el entrenamiento de las redes neuronales convolucionales, obteniendo un valor de Exactitud en cada época.

- **Entropía cruzada binaria (Binary cross entropy):** En la teoría de la información, la Binary cross entropy de dos distribuciones de probabilidad (p y q) es la media de bits necesarios para identificar un evento en un conjunto de posibilidades, si el esquema de codificación está basado en una distribución de probabilidad q, en lugar de la verdadera probabilidad p. En el mundo del aprendizaje automático, se trata de ver que tan cercana es la predicción q de la verdadera distribución p. En el presente trabajo, por cada instancia tendremos una distribución de probabilidad real de que la instancia sea melanoma $p(Melanoma)$ (que será 0 o 1) y una probabilidad estimada (por el clasificador) de que la instancia sea melanoma $q(Melanoma)$. Luego, dada las ecuaciones:

$$H(p,q) = -\sum_{x=(Melanoma,No\ melanoma)} p(x)\ log(q(x))$$
$$p = p(Melanoma) \quad ; \quad \hat{p} = q(Melanoma)$$

La Binary cross entropy se calcula de la siguiente forma:

$$H(p,q) = -p\ log(\hat{p}) - (1-p)\ log(1-\hat{p})$$

El rango se encuentra comprendido entre [0,∞], donde 0 significa que la probabilidad *p(Melanoma)* es igual a la probabilidad *q(Melanoma),* e ∞ significa que la distancia entre ambas probabilidades es máxima. Por ende se puede utilizar a la Binary cross entropy como función de pérdida en el entrenamiento de los modelos de clasificación. En este caso la función de pérdida J está definida como el promedio de la Binary cross entropy en las N instancias, calculada de la siguiente forma:

$$J = \frac{1}{N}\sum_{i=1}^{N} H(p_i, q_i)$$

- **Curva Receiver Operating Characteristic (ROC) y ROC-AUC:** La curva ROC es la representación de la razón o ratio de verdaderos positivos (**VPR**) en función de la razón o ratio de falsos positivos (**FPR**) según se varía el umbral de discriminación (valor a partir del cual decidimos que un caso es un positivo). Este tipo de curva se utiliza en clasificadores binarios. Por otro lado, el indicador ROC-AUC es el área bajo la curva de la curva ROC y puede ser interpretado como:

    - El promedio de la Sensibilidad sobre todos los valores de FPR (razón de falsos positivos).

    - El promedio de la Especificidad sobre todos los valores de Sensibilidad [113].

    - La probabilidad de que el clasificador establezca un ranking más alto en un caso positivo aleatorio que en un caso negativo aleatorio [114]. En el caso de esta tesis, será la probabilidad de que el clasificador establezca un ranking más alto en un paciente aleatorio con melanoma que en un paciente aleatorio sin melanoma.

Justamente la última definición del ROC-AUC es clave para que esta métrica sea seleccionada para comparar todos los modelos de clasificación. Esto radica que los falsos negativos en este trabajo tienen un costo muy alto dado que estaríamos "perdiendo" un paciente con melanoma con el riesgo que ello implica. Por este motivo, los organizadores de la competencia ISIC2017[10] utilizan este indicador para comparar los distintos modelos. Por otro lado, estudios recientes demuestran que la curva PR es más informativa que la ROC en conjuntos de datos desbalanceados [115]. No obstante, nuestro conjunto de datos está balanceado y por lo antes mencionado, se concluye usar la curva ROC y la métrica ROC-AUC como elementos principales para comparar los modelos de clasificación.

El cálculo del ROC-AUC se define mediante la integral indicada en la ecuación 3.6, donde se tiene en cuenta las funciones VPR y FPR definidas en las ecuaciones 3.4 y 3.5 con su dominio e imagen respectiva.

$$VPR(T) : T \rightarrow y(x) \tag{3.4}$$

$$FPR(T) : T \rightarrow x \tag{3.5}$$

$$ROC - AUC = \int_{x=0}^{1} VPR(FPR^{-1}(x))\, dx = P(X_1 > X_0) \tag{3.6}$$

[10] https://challenge.kitware.com/#phase/5840f53ccad3a51cc66c8dab

En donde T son los puntos de corte, VPR(T) es la razón de verdaderos positivos (Sensibilidad), FPR(T) es la razón de falsos positivos (1-Especificidad), X_1 es el puntaje del modelo para una instancia aleatoria positiva (paciente con melanoma) y X_0 es el puntaje del modelo para una instancia aleatoria negativa (paciente sin melanoma). Notar que que dentro de la integral de la ecuación 3.6 se utilizó la función inversa de FPR para obtener en la imagen de dicha función los puntos de corte T. Es decir, dado el dominio de FPR en 3.5, obtenemos FPR^{-1}: $x \rightarrow T$ y estos puntos de corte constituyen la entrada a la función VPR (Ver ecuación 3.4). Para la demostración de la ecuación 3.6 que hace referencia a la equivalencia entre el ROC-AUC y la $P(X_1 > X_0)$ se sugiere leer [114].

Este indicador varía entre 0.5 y 1, siendo 1 cuando el clasificador separa las clases sin cometer ningún error y 0.5 cuando el clasificador funciona en forma aleatoria. Se aclara que obtener valores menores a 0.5 y por ende debajo de la diagonal principal (la diagonal principal indica una clasificación aleatoria) puede deberse a un mal etiquetado de las clases o un algoritmo de entrenamiento erróneo.

Lo anteriormente explicado se refleja en la figura 3.2, donde se observan distintos tipos de curvas ROC.

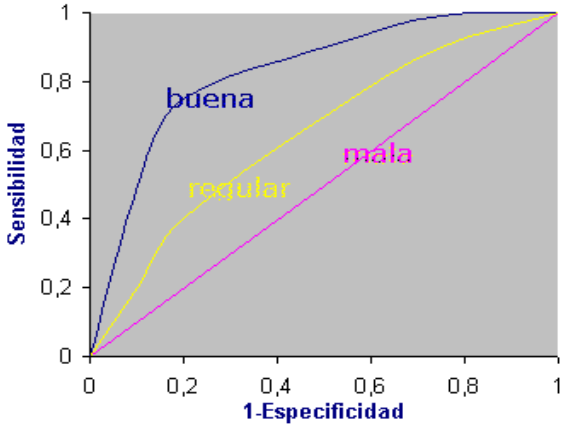


Figura 3.2: Distintos tipos de curvas ROC.

3.2 Conjuntos de datos

En el presente trabajo se utilizarán imágenes del ISIC Archive, Dermnet y de la Dra. Viviana Kowalckzuc del Hospital Italiano.

Para obtener las imágenes del sitio ISIC, se desarrolló un script en el lenguaje *Julia* en el cual se utilizan las APIs[11] proporcionadas por el ISIC Archive para acceder a las imágenes respectivas. Al utilizar las mencionadas APIs, en primer lugar se lee una base de datos JSON la cual almacena información relacionada a las imágenes. Luego, se seleccionaron sólo las imágenes dermastocópicas ya que presentan un mayor grado de detalle; con lo cual se leyó la estructura del código JSON hasta encontrar la etiqueta que especifica tales tipos de imágenes. Además, se filtraron las imágenes en las cuales el valor de la etiqueta "diagnóstico" corresponde a "Melanoma" y por otro lado se almacenaron las imágenes cuya etiqueta corresponda a "Benigno". Es decir, el objetivo es entrenar clasificadores del tipo "Melanoma" versus "No melanoma", donde esta última clase puede incluir lesiones benignas del tipo nevo melanocíticos, lentigos solares o queratosis seborreica. Estas imágenes de lesiones pigmentadas se encuentran en los conjuntos de datos UDA1, UDA2, MSK1 y MSK2. Dichos conjuntos de datos corresponden a los utilizados en la competencia de ISIC realizada en el

[11] https://isic-archive.com/api/v1

año 2017. Además, se utilizó el conjunto de datos de prueba de la competencia ISIC 2016 para la validación de los modelos luego de realizar todas las pruebas.

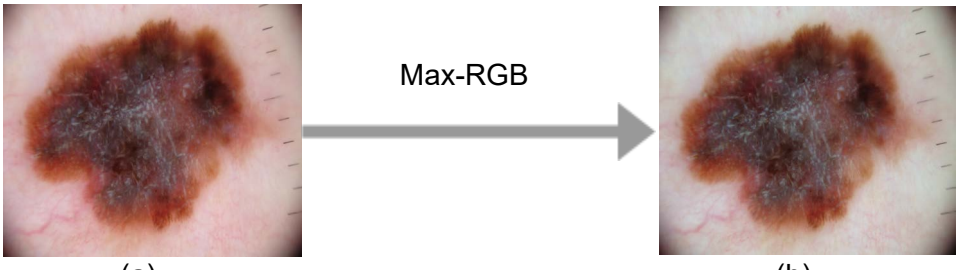
Para obtener las imágenes de Dermnet, se accedió al sitio WEB de dicha compañía donde se almacenan las imágenes[12] de los melanomas. Luego, con la librería *urllib* de python que funciona como interface para trabajar con URLs, se procedió a obtener las imágenes que comienzan con “Malignant-Melanoma” para obtener 50 melanomas adicionales.

Por último disponemos de las imágenes proporcionadas por la médica dermatóloga Viviana Kowalckzuc del Hospital Italiano. Estas corresponden a una menor calidad de resolución que las mencionadas anteriormente (ISIC y Dermnet). Además existen imágenes repetidas, con lo cual se realizó un proceso de depuración y se seleccionaron 100 imágenes de las 193 recibidas en un primer momento. Lo anterior explicado se visualiza en la tabla 3.1.

Fuente	Conjunto de datos	Lesiones benignas	Melanoma
ISIC Archive (Web)	UDA1	398	157
ISIC Archive (Web)	UDA2	37	23
ISIC Archive (Web)	MSK1	773	301
ISIC Archive (Web)	MSK2	-	317
Dra. Viviana Kowalckzuc	Hospital Italiano	50	50
Dermnet (Web)	Melanoma	-	50
	Total:	1258	898
ISIC 2016 (Web) - Validación de modelos		304	75

Tabla 3.1: Conjuntos de datos utilizados para entrenar y validar los modelos de clasificación.

Una vez que se obtuvieron las imágenes, a cada una de ellas se le aplicó el algoritmo Max-RGB (constancia del color) explicado en la sección 2.3. Como se explicó en la mencionada sección, la finalidad es reducir el efecto de la posible distorsión del iluminante en las imágenes para que sean más consistentes, y por ende ser utilizadas en distintos modelos de aprendizaje automático. A modo de ejemplo, se observa en la figura 3.3 una imagen de un melanoma y la resultante de aplicar Max-RGB.



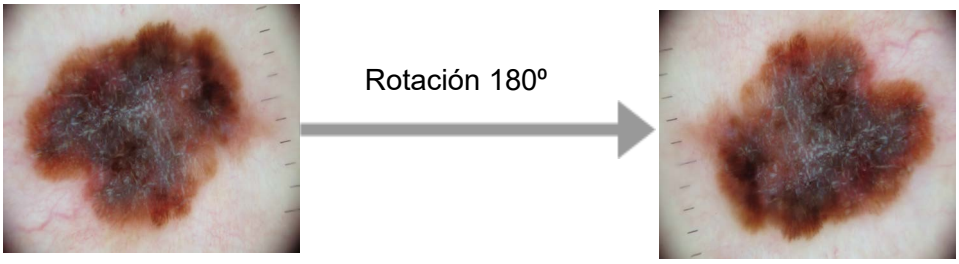
(a) (b)

Figura 3.3: (a) Imagen original de un melanoma. (b) Melanoma con Max-RGB.

Luego, para el entrenamiento de los modelos, los conjuntos de datos se dividieron en:

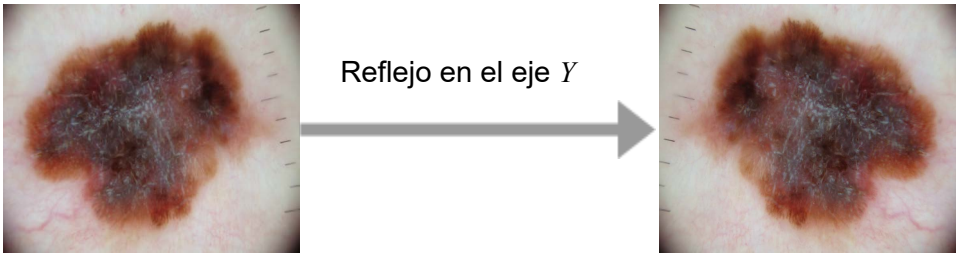
[12] http://www.dermnet.com/dn2/allJPG3/

Entrenamiento: Se encuentra formado por el 70% del total de las imágenes. Debido a que este conjunto de datos será utilizado para entrenar los modelos, se incluyeron sólo las imágenes de mayor calidad, es decir, las dermatoscópicas. Para tal fin, se obtuvieron sólo las imágenes provenientes de los sitios ISIC y Dermnet. En este conjunto de datos se tendrá 900 lesiones pigmentadas benignas y 628 melanomas. Debido que el total de 1528 imágenes no suele ser suficiente para entrenar modelos de aprendizaje profundo (deep learning), se decidió realizar *data augmentation* y se obtuvieron mejores resultados. Es decir, a las imágenes de este conjunto de datos se las rotó en 90, 180 y 270 grados, y en cada una de esas rotaciones se obtuvieron 3 imágenes adiciones producto de realizarles un zoom y reflejarlas en los ejes X e Y. Con lo cual se tendrá 12 "versiones" de cada imagen (3 + 3*3), conformando un conjunto de datos de entrenamiento de 18.336 imágenes (1528 * 12). A modo ilustrativo en la figura 3.4 se observa un melanoma en el cual se le aplicó Max-RGB y luego se lo rotó 180º. Mientras que en la figura 3.5 se observa un melanoma en el cual se le aplicó Max-RGB y luego se le realizó la operación de reflejo en el eje Y.



(a) (b)

Figura 3.4: (a) Melanoma con Max-RGB. (b) Melanoma rotado en 180º.



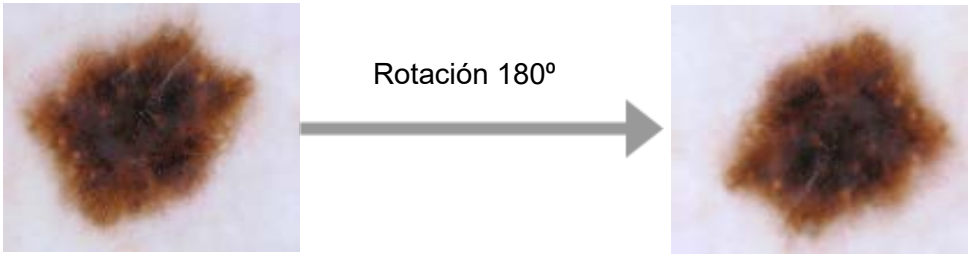
(a) (b)

Figura 3.5: (a) Melanoma con Max-RGB. (b) Melanoma reflejado en el eje Y.

Validación: Formado por el 20% del total de las imágenes, se lo utilizó para optimizar los parámetros del modelo entrenado con el conjunto de datos de entrenamiento. Con la misma lógica que la empleada en el conjunto de datos de entrenamiento, se seleccionaron las imágenes de mayor resolución (imágenes dermatoscópicas). Con lo cual, solo se tomaron aquellas que provienen de los sitios ISIC y Dermnet. Este conjunto de datos está compuesto por 428 imágenes, de las cuales 260 son lesiones pigmentadas benignas y 168 son melanomas.

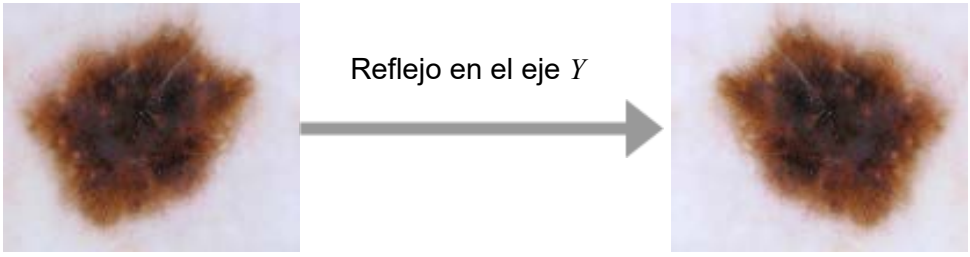
Prueba: El conjunto de datos de Prueba está conformado por las imágenes del sitio ISIC y las enviadas por la médica dermatóloga Viviana Kowalckzuc del Hospital Italiano. Al conjunto de datos de prueba compuesto por las imágenes del Hospital Italiano se lo denominó "Prueba HI", mientras que al conjunto de datos de Prueba con imágenes del sitio ISIC se lo denominó "Prueba ISIC" y "Prueba ISIC 2016". Este último conjunto de datos se lo utilizó para la validación del rendimiento de los modelos (comparándolo con un modelo externo) luego de realizar todas las pruebas. Por otro lado, tanto el conjunto de datos de Prueba HI como el

conjunto de datos de Prueba ISIC tienen 50 melanomas y 50 lesiones pigmentadas benignas. Además, se aplicaron operaciones de rotación y reflejo al conjunto de datos de Prueba ISIC, obteniéndose el conjunto de datos de Prueba sintético llamado “Prueba ROT”. Este último conjunto de datos contiene 250 lesiones pigmentadas benignas y 250 melanomas. Las 500 imágenes surgen de incluir las 100 imágenes que se encuentran en Prueba ISIC y 4 versiones adicionales de cada una de ellas. Dichas versiones son el resultado de rotar (en 90º y 180º) y reflejar (en los ejes X e Y) cada una de las imágenes incluidas en Prueba ISIC. A modo de ejemplo, en las figuras 3.6 y 3.7 se visualizan las operaciones de rotación y de reflejo en el eje Y respectivamente, ambas aplicadas a una lesión benigna.



(a) (b)

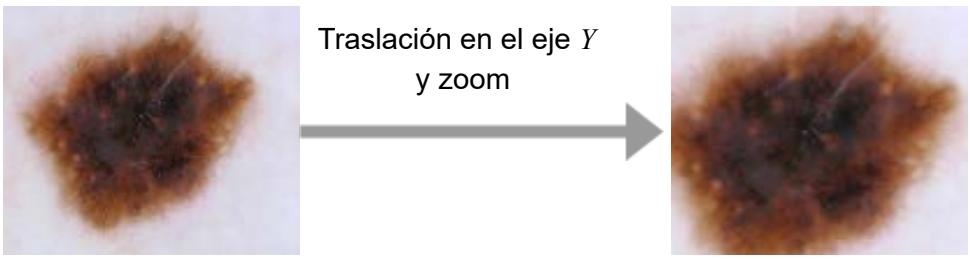
Figura 3.6: (a) Lesión benigna con Max-RGB. (b) Lesión benigna rotada en 180º.



(a) (b)

Figura 3.7: (a) Lesión benigna con Max-RGB. (b) Lesión benigna reflejada en el eje Y.

Por otro lado el conjunto de datos llamado “Prueba XY” se encuentra constituido por las imágenes incluidas en Prueba ISIC y 4 versiones adicionales de cada una de ellas. Estas versiones son el resultado de aplicar 4 operaciones de traslación en los ejes X e Y con el correspondiente zoom en cada una de las imágenes. Debido a que la traslación y el zoom son aleatorios, se podrían obtener imágenes que incluyan solamente piel. Con lo cual, se diseñó un algoritmo tal que la combinación de la traslación y el zoom, recupere un área de la lesión representativa. Esto se logró debido que las lesiones en las imágenes se encuentran centradas, con lo cual se partió de un recorte central de la imagen y pequeñas traslaciones aleatorias. Estas operaciones pueden observarse en la figura 3.8. Con lo cual, este conjunto de datos sintéticos (Prueba XY) está conformado por 250 melanomas y 250 lesiones benignas.



(a) (b)

Figura 3.8: (a) Lesión benigna con Max-RGB. (b) Lesión con zoom y traslación en el eje Y.

En resumen, lo anteriormente explicado se puede visualizar en la figura 3.9.

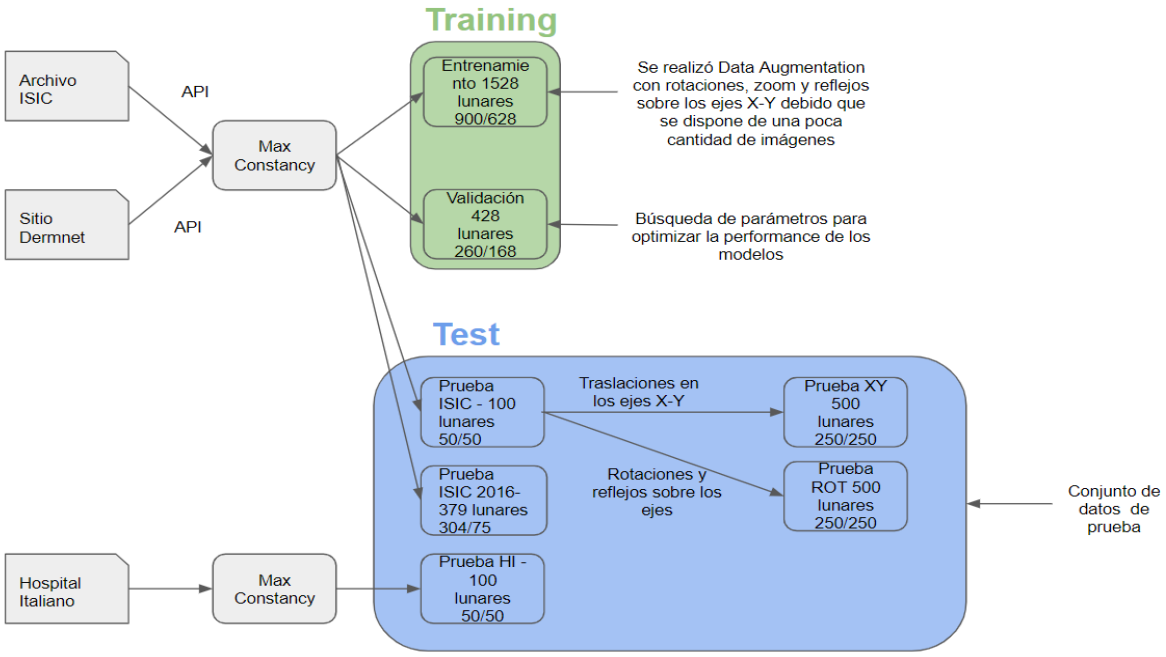
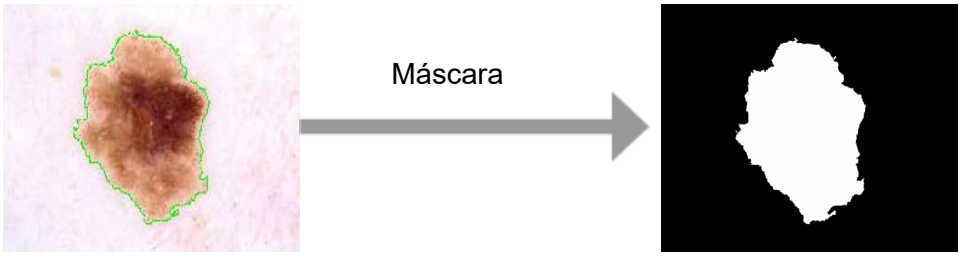


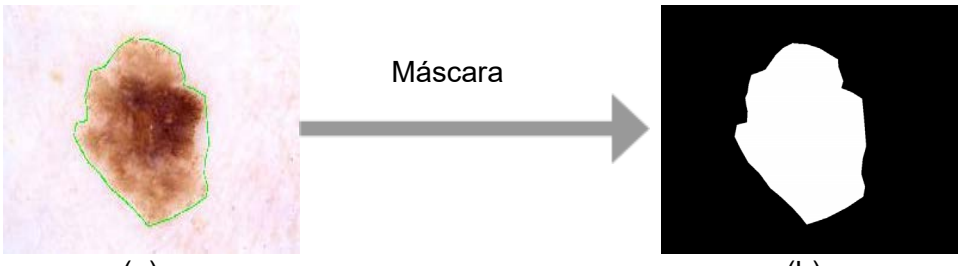
Figura 3.9: Conjuntos de datos de entrenamiento, validación y prueba.

Además, se recuperaron las máscaras de las imágenes que provienen del sitio ISIC y que están incluidas en el conjunto de datos de entrenamiento y validación. Estas máscaras se utilizaron para entrenar la red TernausNet-16 (explicada en la sección 2.4.5). Las máscaras que se obtuvieron corresponden al nivel de “Experto” ya que por cada imagen de una lesión pigmentada se disponen de varias máscaras con distintos grados de precisión. Con lo cual, al recorrer la estructura JSON del sitio ISIC, se seleccionaron la máscaras que han sido revisadas por expertos y que son consideradas las más precisas de las existentes en este sitio. A modo ilustrativo, se observa que en la figura 3.10 se dispone de una imagen de un melanoma, la segmentación novata y su máscara. Mientras que en la figura 3.11 se visualiza la misma imagen del melanoma pero con una segmentación experta y su máscara respectiva.



(a) (b)

Figura 3.10: (a) Melanoma con Max-RGB y segmentación novata. (b) Máscara del melanoma.



(a) (b)

Figura 3.11: (a) Melanoma con Max-RGB y segmentación experta. (b) Máscara del melanoma.

Con el mismo criterio utilizado en las lesiones pigmentadas, al par “lesión-máscara” se le aplicó *data augmentation*. Es decir, si por ejemplo a la lesión pigmentada se la giró 180 grados, entonces se realizó la misma rotación a la máscara respectiva. Esta última operación se realizó al par y no solamente a la lesión pigmentada ya que, como se verá en la próxima sección, el resultado esperado de la red TernausNet-16 en el entrenamiento será la máscara. Con lo cual, si se rota a la lesión pigmentada, se deberá rotar la máscara para que el entrenamiento sea consistente. Por último, los conjuntos de datos de entrenamiento y validación utilizados en el entrenamiento de la red TernausNet-16, son los mostrados en la figura 3.9 pero con las imágenes que provengan sólo del sitio ISIC. Es decir, se eliminaron las imágenes obtenidas del sitio Dermnet ya que este sitio no dispone de las máscaras respectivas y éstas son necesarias para el entrenamiento de la red TernausNet-16.

3.3 Métodos

En esta sección se explicará en detalle los distintos modelos y su entrenamiento, resumidos en los siguientes 6 puntos :

- Segmentador: Está constituido por una red llamada TernausNet-16, y se lo utiliza para segmentar las imágenes. En esta sección se explicará el entrenamiento de la mencionada red segmentadora.

- Clasificador 1: Se utilizó una red VGG-16 (descripta en la sección 2.5.1) y se realizó *Fine-Tuning* (ver sección 2.2.2). Es decir, primero se eliminaron las capas totalmente conectadas (Fully-Connected) y se pasaron las imágenes por las capas convolucionales. En la salida de esta red (última capa de pooling) se encuentran los descriptores llamados “Deep Features”. Luego con estos “Deep Features” se entrenó una mini red (explicada en la sección 3.3.2) de capas totalmente conectadas. Por último, se entrenó la red VGG-16 (sin las capas totalmente conectadas) conjuntamente con esta mini red. En cada época que transcurre durante el entrenamiento, se calculará la Exactitud en el conjunto de datos de validación y entrenamiento (para evitar el sobreajuste), eligiendo luego la combinación de parámetros más conveniente en función de los valores que se obtienen de la Exactitud y de la función de pérdida.

- Clasificador Auxiliar: Está constituido por una red VGG-16 y para su entrenamiento se procede de la misma forma que la explicada para el Clasificador 1, solo que las imágenes se encuentran segmentadas por el Segmentador.

- Clasificador 2 (Modelo Híbrido sin segmentación): Se codificarán los descriptores (de la última capa de pooling) del modelo Clasificador 1 en los vectores de Fisher. Estos serán la entrada a un SVM para realizar su entrenamiento con el objetivo de clasificar las imágenes.

- Clasificador 3 (Modelo Híbrido con segmentación): Se procede de la misma forma que la explicada para el Clasificador 2, solo que las imágenes se encuentran segmentadas por el Segmentador y el modelo clasificador involucrado es el Clasificador Auxiliar.

- Ensamble: Para una determinada imagen, realiza el promedio de las probabilidades obtenidas en los clasificadores: Clasificador 1, Clasificador 2 y Clasificador 3.

3.3.1 Segmentador

El módulo de segmentación consiste en segmentar las imágenes por medio de una red TernausNet-16 (llamada U-Net dinámica). Se denomina dinámica, porque la sección del encoder fue modificada en relación a la U-Net original. Como encoder se utilizaron las capas de las VGG-16 (ver sección 2.5.1) y como decoder se utilizó la misma filosofía de la red U-Net tal cual se explicó en la sección 2.4.5. El input del modelo con el cual se realizó el entrenamiento fueron las imágenes de las lesiones pigmentadas de los conjuntos de datos de entrenamiento y validación con sus respectivas máscaras. Una vez entrenado, se aplica el modelo a una imagen de un lesión pigmentada nueva y se obtiene la máscara respectiva. En la figura 3.12 se visualiza lo anteriormente explicado.

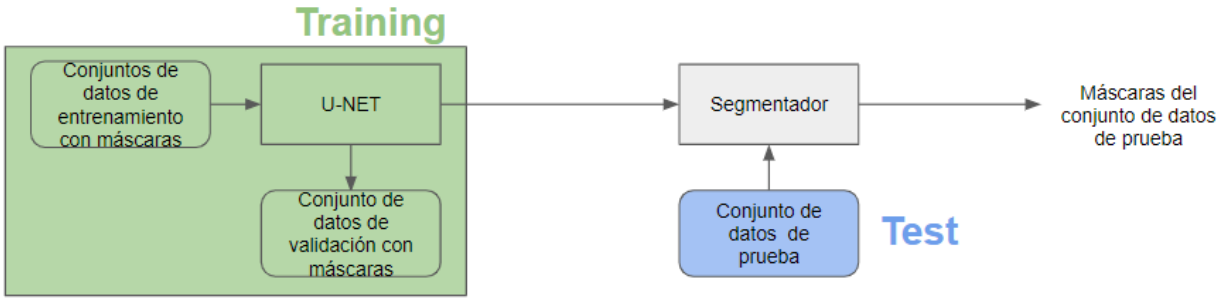


Figura 3.12: Funcionamiento del Segmentador.

La arquitectura de la red TernausNet-16 (explicada en la sección 2.4.5) es la mostrada en la figura 3.13.

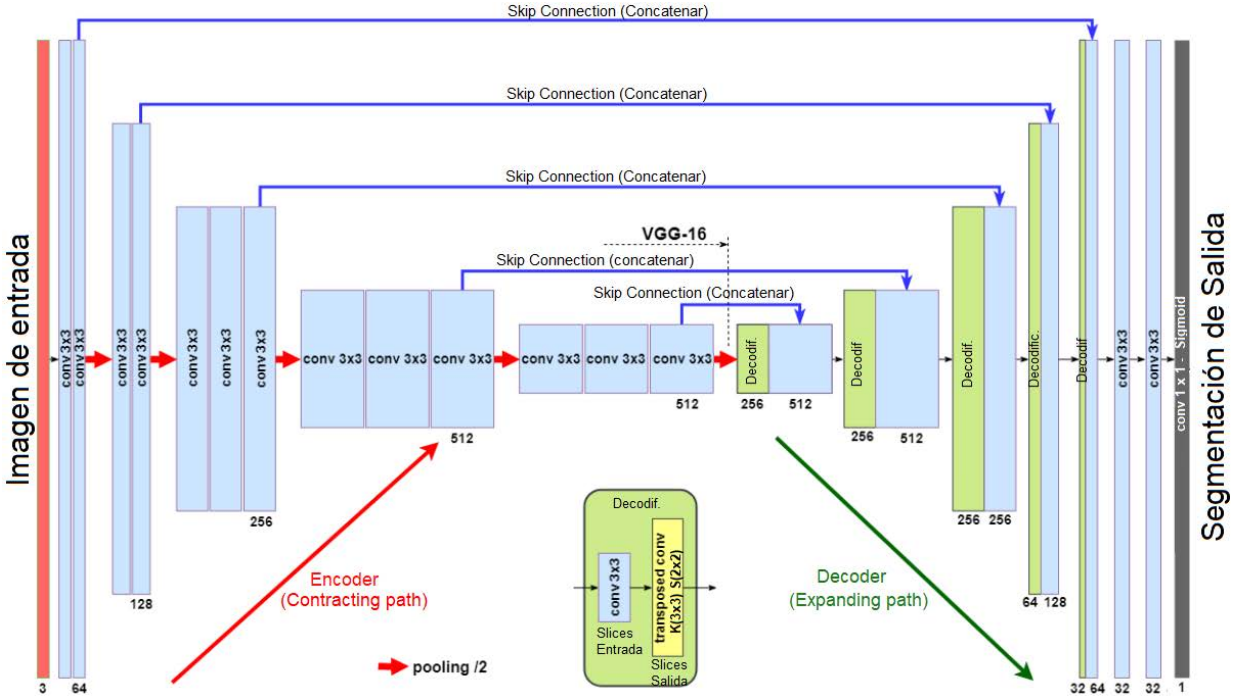


Figura 3.13: TernausNet-16. Utilización de las capas de la VGG-16 en el encoder, y la modificación respectiva en el decoder para que siga el lineamiento de la U-Net.

Los pesos disponibles de la VGG-16 (que se encuentran en la WEB de la universidad de

Oxford)[13] se obtuvieron entrenando dicha red con las imágenes de ImageNet redimensionadas a 224 x 224 (explicado en la sección 2.5.1). Dado que la red de la figura 3.13 se basa en la arquitectura VGG-16 y en el presente trabajo se usaron los mencionados pesos para inicializar las capas del encoder de la red (transfer learning), resulta recomendable utilizar las imágenes de las lesiones pigmentadas con las mismas dimensiones (que las imágenes de ImageNet, o sea, 224 x 224) para poder realizar el entrenamiento. No obstante se probaron otras dimensiones: 256 x 256 o 512 x 512 y no se obtuvieron mejores resultados, solo se experimentó un mayor tiempo de entrenamiento de la red. Luego de redimensionar las imágenes a 224 x 224, se calculó la media por cada canal RGB con la finalidad de sustraer dicha cantidad a cada píxel. El objetivo de realizar esta última operación es centrar los datos alrededor del 0 en el rango de [-255,255]. Esto provoca que las imágenes sean más homogéneas y permite que el gradiente esté en rangos controlados cuando se realiza el backpropagation. Ver implementación de VGG_ILSVRC_16_layer[14].

Como se explicó en la sección 3.3, se utilizó un conjunto de datos de entrenamiento, otro de validación en el cual se evaluaron los parámetros de la red y por último uno de prueba (Prueba ISIC 2017) en el cual se obtuvo el puntaje final cuando se aplicó este modelo. Tanto el conjunto de datos de entrenamiento, el de validación y el de Prueba ISIC 2017, están compuestos por las imágenes de las lesiones pigmentadas y las respectivas máscaras. Como se mencionó en la sección 3.1, las métricas utilizadas en la segmentación fueron el Coeficiente de Jaccard (para medir la similitud entre la máscara verdadera y la máscara predicha) y el Coeficiente de Dice para la función de pérdida.

El entrenamiento de la red fue dividido en 3 partes:

a) Se inicializaron los pesos de las capas del encoder con los provenientes del entrenamiento de la red VGG-16 con las imágenes de ImageNet (transfer learning). Los pesos del decoder fueron inicializados por medio de la inicialización Xavier [116].
b) En 130 épocas se congelaron los pesos de las capas que conforman la sección del encoder de la red, entrenando solo las capas del decoder.
c) En 130 épocas adicionales se descongelaron los pesos de la sección del encoder y se entrenó la red en forma completa.

En la figura 3.14 se visualiza la variación del Coeficiente de Jaccard y el Coeficiente de Dice en función de las épocas, en los conjuntos de entrenamiento y validación. Las curvas mostradas pertenecen al punto “b” antes citado, o sea, es el entrenamiento de la red cuando se congelaron los pesos de las capas del encoder, entrenando sólo las capas del decoder.

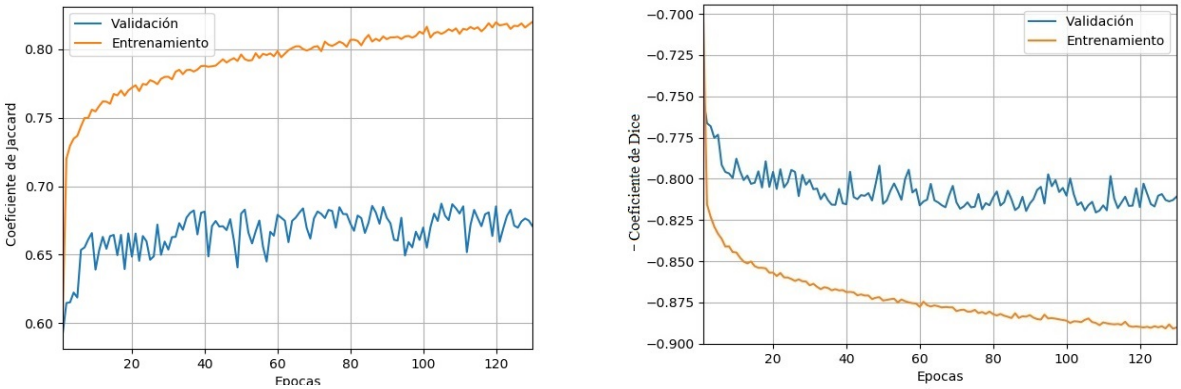


Figura 3.14: Coeficiente de Jaccard (Gráfico de la izquierda) y Coeficiente de Dice (Gráfico de la derecha) ambos en función de las épocas y evaluados en los conjuntos de datos de entrenamiento y validación durante el entrenamiento de las capas del decoder.

[13] http://www.robots.ox.ac.uk/~vgg/research/very_deep/
[14] https://gist.github.com/ksimonyan/211839e770f7b538e2d8

Se observa que en el conjunto de datos de validación, el Coeficiente de Jaccard varía en las proximidades del valor 0.7.

Por otro lado, para verificar si las capas del encoder se encontraban congeladas, en la figura 3.15 se tomó (como ejemplo) la segunda capa convolucional perteneciente al encoder, es decir aquella capa convolucional con 64 kernels donde cada uno de ellos tiene un tamaño de 3 x 3. Durante el entrenamiento se verificó que en esta capa las distribuciones de los pesos entre las distintas épocas eran idénticas, lo que sugiere que los pesos de los distintos kernels permanecieron constantes, lo cual es cierto ya que fueron congelados. Es decir, son los mismos pesos que fueron obtenidos en el punto “a” del entrenamiento (transfer learning).

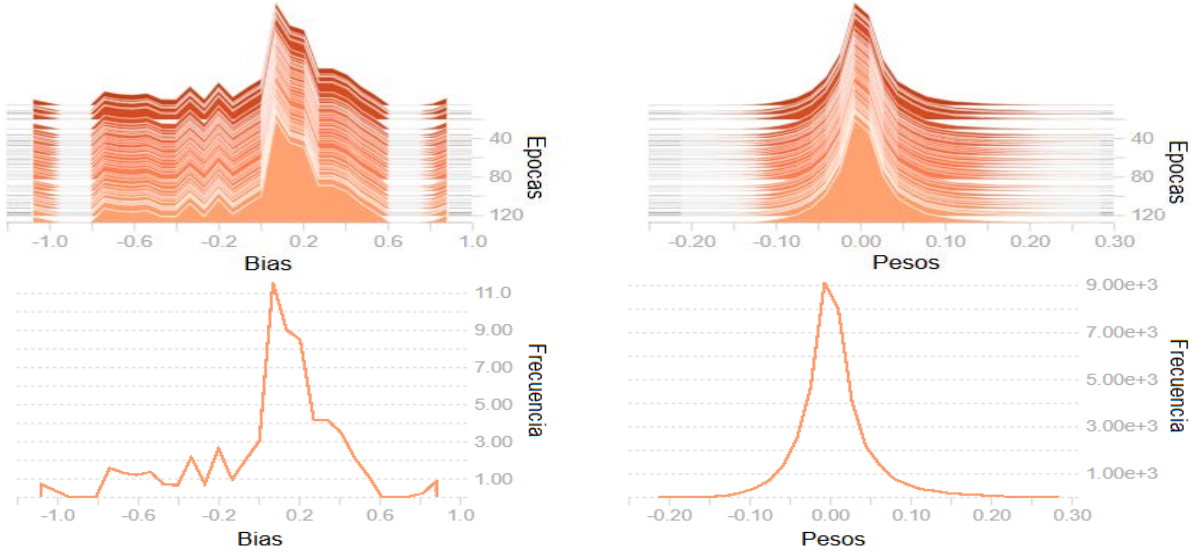


Figura 3.15: (Arriba) Histogramas, en las distintas épocas, de los pesos y el bias de la segunda capa convolucional de la sección del encoder durante el entrenamiento de las capas del decoder. (Abajo) Se visualiza la frecuencia de cada histograma que se genera en cada época. En este caso, estos se superponen ya que poseen la misma distribución y por ende se observa un único histograma.

Finalmente, se pasa al punto “c” del entrenamiento, es decir se descongelaron los pesos de las capas pertenecientes al sector del encoder y se entrenó todo el modelo (todas las capas) con 130 épocas adicionales. Las curvas visualizadas en la figura 3.16 muestran los Coeficientes de Jaccard y Dice en función de las épocas según el entrenamiento mencionado.

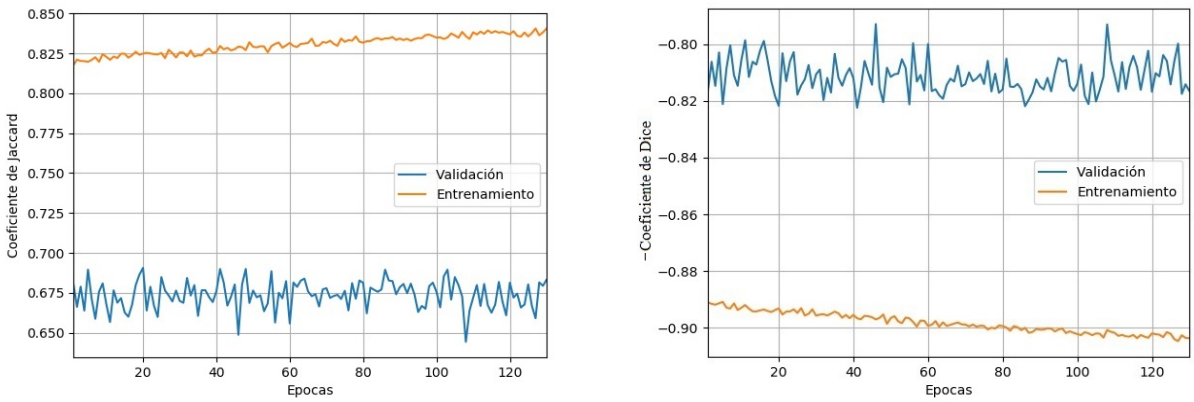


Figura 3.16: Coeficiente de Jaccard (Gráfico de la izquierda) y Coeficiente de Dice (Gráfico de la derecha) ambos en función de las épocas y evaluados en los conjuntos de datos de entrenamiento y validación, durante el entrenamiento de las capas del encoder y del decoder.

Luego, en la figura 3.17 se visualiza la variación de los pesos y los bias tomando como ejemplo la última capa convolucional (perteneciente a la sección del decoder), es decir aquella con 32 kernels donde cada uno de ellos tiene un tamaño de 3 x 3. Dicha variación se encuentra representada en función de las épocas que transcurren en el entrenamiento de la red y debido a que las capas de la sección del decoder siempre fueron “entrenables”, los pesos y el bias no permanecen constantes a medida que varían las épocas. Se observa además que las variaciones de los pesos se encuentran en rangos controlados. Es decir, no se visualizan los 2 fenómenos que puede tener el gradiente: la desaparición o el crecimiento exponencial del mismo, generando grandes variaciones (o ninguna) en los pesos y/o bias.

Esta verificación del comportamiento del gradiente se realizó en todas las capas (si bien aquí solo se muestra la última capa convolucional) y en las 2 etapas del entrenamiento mencionadas anteriormente (punto “b” y punto “c”).

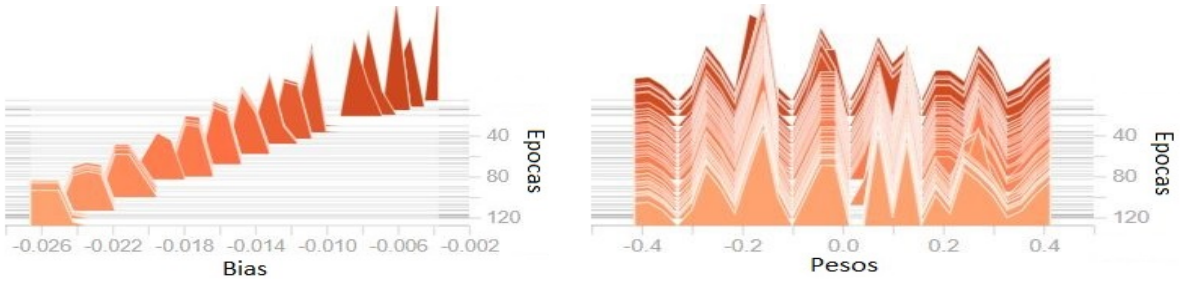


Fig 3.17: Histogramas de la variación del bias y los pesos de los kernels de la última capa convolucional de la red.

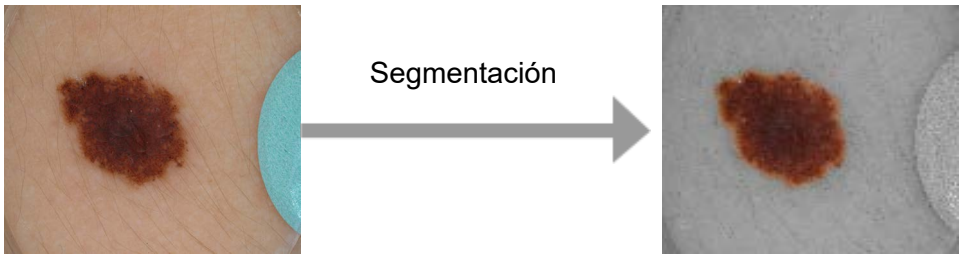
Durante el entrenamiento se guardaron, en distintos archivos, los pesos de la red cada vez que se producía un incremento del Coeficiente de Jaccard en el conjunto de datos de validación (con respecto al máximo Coeficiente de Jaccard obtenido en una época anterior). Es decir, se obtuvieron diferentes archivos de pesos cada uno asociado a un Coeficiente de Jaccard distinto. Finalizado el entrenamiento, se evaluaron las curvas de las figuras 3.14 y 3.16 (para verificar la existencia de sobreajuste o subajuste) conjuntamente con los índices de Jaccard correspondientes a cada archivo de pesos guardado. De dicha evaluación, se eligió tomar el archivo de pesos de la red en el cual se obtuvo un Coeficiente de Jaccard de 0.681 en el conjunto de datos de validación, sucedido en la época 20 del fine tuning (época 150 (130+20) del total del entrenamiento).

Para verificar la calidad del Segmentador que se obtuvo con el entrenamiento explicado, se evaluó dicho modelo en el conjunto de datos de Prueba oficial de ISIC 2017. Este se encuentra conformado por 600 imágenes con sus respectivas máscaras. El objetivo fue calcular el Coeficiente de Jaccard medio (Coeficiente de Jaccard promedio de todas las imágenes) y se lo comparó con los otros grupos de la competencia. Con el modelo propuesto se obtuvo un Coeficiente de Jaccard de 0.75, siendo este un valor que correspondió a la octava posición en el ranking de los participantes. El ganador (grupo Titans) obtuvo un Coeficiente de Jaccard de 0.79. Debido que el objetivo de esta tesis es evaluar el rendimiento de los clasificadores híbridos versus los clasificadores del tipo CNN y no está enfocado en la segmentación, concluimos que el Segmentador obtenido tiene un rendimiento aceptable. Los parámetros seleccionados para esta red son los mostrados en la tabla 3.2.

Parámetro	Valor	Comentario
Optimizador	Adam	Tasa de aprendizaje inicial: 10^{-5}
Función de pérdida	Coeficiente de Dice	
Métrica de performance	Coeficiente de Jaccard	
Batch size	4	
Epocas	260	2 entrenamientos de 130 épocas.
Ejemplos por época	1524 lesiones pigmentadas	

Tabla 3.2: Parámetros del Segmentador.

Una vez finalizado el entrenamiento de la red TernausNet-16, se realizó la predicción sobre las lesiones pigmentadas de los distintos conjuntos de datos de prueba, obteniendo las máscaras predichas. Cada máscara se aplicó a la imagen respectiva que segmenta, de forma que el área de la lesión que se encuentra fuera de la máscara quedó determinado por una escala de grises, y a la lesión dentro de la máscara se le dejó el color de la imagen original. En la figura 3.18 se visualiza un nevo benigno segmentado por el módulo del Segmentador.



(a) (b)

Figura 3.18: (a) Lesión benigna que se le aplicó Max-RGB. (b) Lesión benigna segmentada por el Segmentador.

Estas imágenes segmentadas se utilizaron para el entrenamiento del Clasificador Auxiliar y dicho clasificador es necesario para construir el Clasificador 3 (Modelo Híbrido con segmentación) propuesto en el presente trabajo.

3.3.2 Clasificador 1 (VGG-16)

El objetivo de este módulo es clasificar las imágenes de las lesiones pigmentadas, en las cuales se les aplicó Max-RGB, en “Melanoma” o “No melanoma”. En la figura 3.19 se observa el funcionamiento de este clasificador. La entrada corresponde al conjunto de datos de entrenamiento más las imágenes sintéticas (data augmentation) tal cual se explicó en la sección 3.2. El entrenamiento de la red fue evaluado tanto en el conjunto de datos de entrenamiento como en el de validación, con la finalidad de evitar el sobreajuste. Una vez finalizado el entrenamiento, se aplicó el modelo obtenido a los conjuntos de datos de prueba, con el objetivo de clasificar cada una de las lesiones pigmentadas. Con la finalidad de construir el Ensamble, la salida de este clasificador es la probabilidad que una lesión pigmentada sea un melanoma en lugar de la clase predicha.

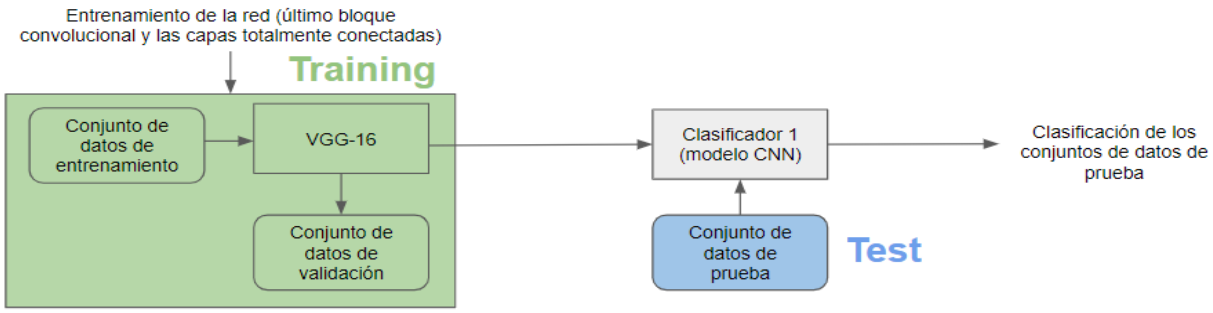


Figura 3.19: Funcionamiento del Clasificador 1.

Tal cual indica la figura 3.19, la red que se utilizó es la VGG-16, explicada en la sección 2.5.1. A continuación se explicará la forma en que se entrenó la misma.

Entrenamiento

Dado que se dispone de una GPU GeForce 950M, no muy potente en el momento que se escribe el presente trabajo, se realiza un entrenamiento optimizado considerando las limitaciones de hardware.

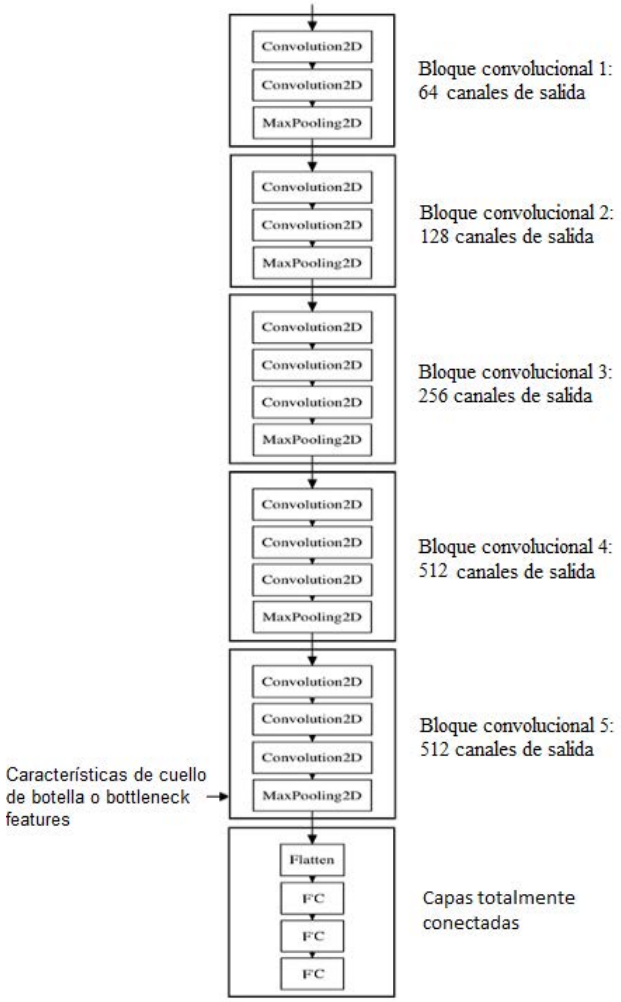


Figura 3.20: Arquitectura del Clasificador 1 (VGG-16), donde se señalan los bottlenecks features utilizados para una fase del entrenamiento.

En la figura 3.20 se visualiza la arquitectura de este clasificador y se entrenó el mismo de acuerdo a los siguientes 3 pasos:

1. Se realizó el transfer learning de los pesos que se obtuvieron con el entrenamiento de la arquitectura VGG-16 con ImageNet (para más detalles ver sección 2.5.1).
2. Se creó una pequeña red totalmente conectada y se inicializaron sus pesos por medio de la inicialización He [116]. Luego, se entrenó dicha red (durante 70 épocas) con los mapas de características que se obtuvieron en la salida de la última capa de pooling de la arquitectura mostrada en la figura 3.20. Estos mapas de características son denominados bottlenecks features (características de cuellos de botella).
3. Se reemplazó el último bloque (capas totalmente conectadas) de la arquitectura mostrada en la figura 3.20 por la pequeña red entrenada en el paso anterior. Luego, con esta arquitectura resultante, se entrenó el último bloque convolucional y la pequeña red acoplada durante 140 épocas. Esto último se denomina fine tuning.

A continuación se describen en detalle los últimos 2 pasos.

Paso 2 del entrenamiento

En este paso se creó una pequeña red del tipo totalmente conectada cuya arquitectura se observa en la figura 3.21, y se utilizó la inicialización He para inicializar sus pesos. El objetivo de esta inicialización es que la varianza de la entrada de cada capa sea igual a la varianza de la salida. Al aplicar el algoritmo backpropagation durante el entrenamiento, este balanceo de la varianza trata de evitar la desaparición o la explosión del gradiente.

Por otro lado, en la figura 3.20 se visualiza la arquitectura de la VGG-16 y que en la salida de la última capa de pooling se encuentran las denominadas características de cuello de botella. Con lo cual, se ingresaron a esta red (VGG-16) las imágenes correspondientes a los conjuntos de datos de entrenamiento y validación, obteniéndose en la salida de la última capa de pooling las características de cuello de botella de entrenamiento y de validación. **Con estos 2 conjuntos de datos nuevos se entrenó (y se validó) la mencionada pequeña red (ver figura 3.21) del tipo totalmente conectada.**.

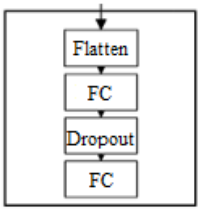


Figura 3.21: Red totalmente conectada, entrenada con las características de cuello de botella de las imágenes del conjunto de datos de entrenamiento.

Como se observa en la figura 3.21 se tiene una operación “dropout” (con un valor de 0.4). Como se explicó en la sección de 2.2.3, el dropout permite desconectar un porcentaje de neuronas por cada iteración del entrenamiento. Esto último, funciona como una técnica de regularización en la cual mejora la generalización de la red, previniendo el sobreajuste. En esta pequeña red, la primer capa “FC” (Fully Connected, totalmente conectada) tiene la función de activación ReLU, mientras que en la última capa “FC” se utiliza la función de activación sigmoide. Como se explicó en la sección 2.2.1, esta función permite calcular la probabilidad que una lesión pigmentada sea considerada como “Melanoma” o “No

melanoma". La primer capa "FC" tiene 256 neuronas, mientras que la segunda tiene 1 neurona. Con la cual, en la salida de dicha neurona, se calcula la probabilidad que una lesión pigmentada sea "Melanoma" y calculando el complemento, obtenemos la probabilidad que dicha lesión sea "No melanoma".

En la figura 3.22 se puede observar el rendimiento conseguido en el entrenamiento de la red que se visualiza en la figura 3.21.

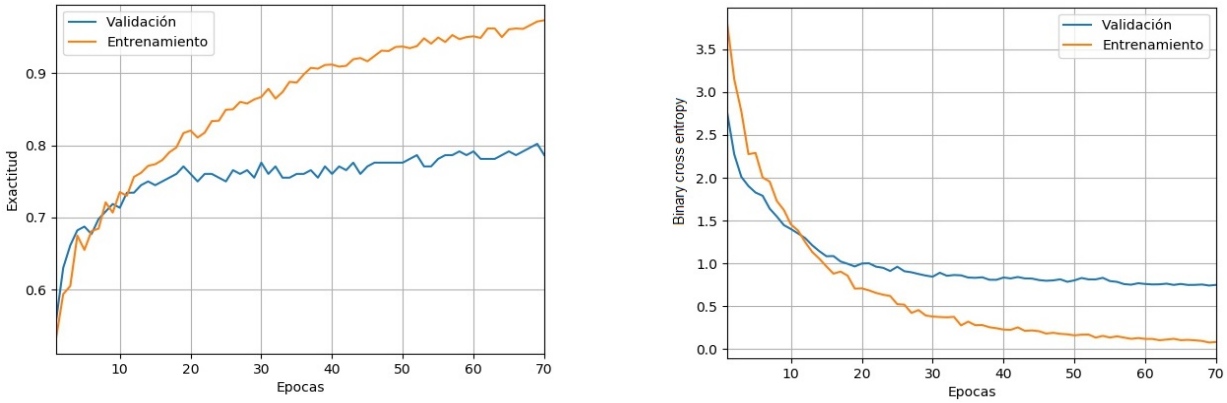


Figura 3.22: Rendimiento del la red visualizada en la figura 3.21, entrenada con las características de cuello de botella obtenidas con las imágenes (sin segmentación) de los conjuntos de datos de entrenamiento y validación. (Izquierda) Exactitud en función de las épocas. (Derecha) Binary cross entropy en función de las épocas.

A medida que entrenamos esta pequeña red, podemos visualizar en la figura 3.23 la distribución de los pesos y bias (sesgo) en la última capa totalmente conectada. Las actualizaciones de los pesos se encuentran controladas y las distribuciones de los pesos entre las épocas son distintas, verificando la **no** aparición de los 2 fenómenos del gradiente (explosión y desaparición) explicados en la sección 2.2.5. Si existiera la desaparición del gradiente todas las distribuciones de los pesos serian iguales en las distintas épocas lo que significa que la red ha dejado de aprender. Por otro lado, la explosión del mismo, generaría grandes actualizaciones de los pesos dando lugar a un modelo inestable y sin la capacidad de aprender correctamente los ejemplos del conjunto de datos de entrenamiento. Esta verificación del comportamiento del gradiente también se realizó en la primer capa totalmente conectada, si bien aquí solo se muestra la estabilidad del gradiente en la última capa totalmente conectada (ver las 2 capas totalmente conectadas o "FC" en la figura 3.21).

Figura 3.23: Histogramas de los pesos y del bias de la última capa totalmente conectada durante el entrenamiento de la red de la figura 3.21, teniendo como entrada a las características de cuello de botella de las imágenes (sin segmentación) del conjunto de datos de entrenamiento.

Los parámetros que se utilizaron para entrenar esta pequeña red (mostrada en la figura 3.21) son los indicados en la tabla 3.3.

Parámetro	Valor	Comentario
Optimizador	Adam	Tasa de aprendizaje inicial: 10^{-5}
Función de pérdida	Binary cross entropy	
Métrica	Exactitud	
Batch size	16	
Epocas	70	

Tabla 3.3: Parámetros del Clasificador 1 en el paso 2 del entrenamiento.

Paso 3 del entrenamiento (Fine tuning)

En este paso del entrenamiento se reemplazó el último bloque de la arquitectura mostrada en la figura 3.20 por la red de la figura 3.21 (entrenada en el paso 2 del entrenamiento). La arquitectura resultante es la que corresponde a este clasificador (Clasificador 1) y es la que se muestra en la figura 3.24. Una vez obtenida la arquitectura final, se realizó fine tuning en el último bloque convolucional y en la red acoplada (bloques de color verde y amarillo), ya que estos 2 elementos tienen los pesos “no congelados”. En la figura 3.25 se observa el rendimiento de este clasificador durante el mencionado fine tuning en los conjuntos de datos de entrenamiento y validación.

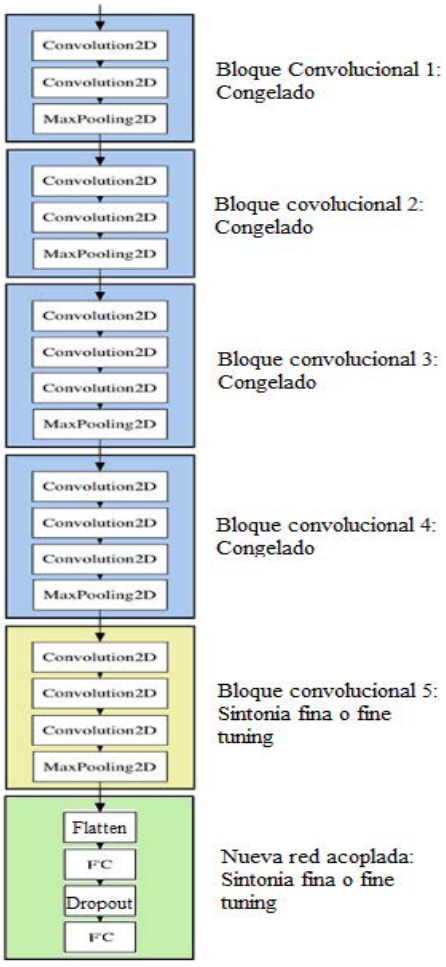


Figura 3.24: Arquitectura del Clasificador 1.

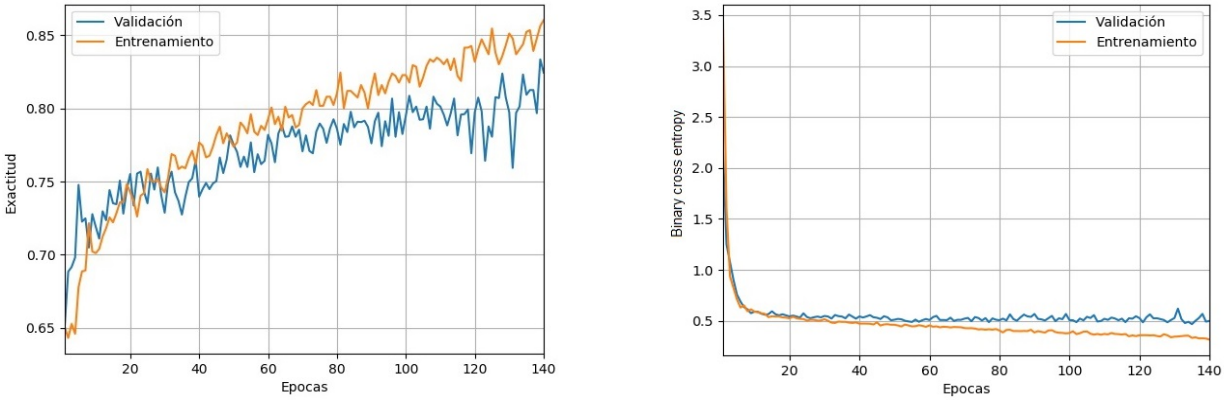


Figura 3.25: Rendimiento del Clasificador 1 durante el fine tunning (en el último bloque convolucional y en la red acoplada totalmente conectada) utilizando las imágenes (sin segmentación) de los conjuntos de datos de entrenamiento y validación. (Izquierda) Exactitud en función de las épocas. (Derecha) Binary cross entropy en función de las épocas.

Se aclara que los bloques de color azul en la figura 3.24 se encuentran con los pesos congelados y corresponden a los pesos obtenidos en el entrenamiento con las imágenes de ImageNet. Cuando se menciona que se realizó fine tuning hacemos referencia al nuevo entrenamiento del modelo, de modo que los pesos de las capas no congeladas se volvieron a ajustar. Para no corromper lo aprendido en el entrenamiento previo (paso 2 del entrenamiento), se utilizó como optimizador al SGD (Stochastic Gradient Descendent). Con este optimizador, las magnitudes de las actualizaciones de los pesos son muy pequeñas y controladas (seteando una tasa de aprendizaje de bajo valor) y por ende tenemos una menor probabilidad de destruir lo aprendido anteriormente. Con lo cual, se tomó lo aprendido como base y se siguieron ajustando los pesos para llegar a un modelo más performante, en este caso un puntaje de Exactitud más elevado. Estos ajustes o variaciones de pesos de la última capa convolucional se pueden visualizar en la figura 3.26. Nuevamente, no se observan los fenómenos de desaparición del gradiente o de crecimiento exponencial. Este análisis también se efectuó en la salida de las demás capas que no se encuentran congeladas de la arquitectura mostrada en la figura 3.24.

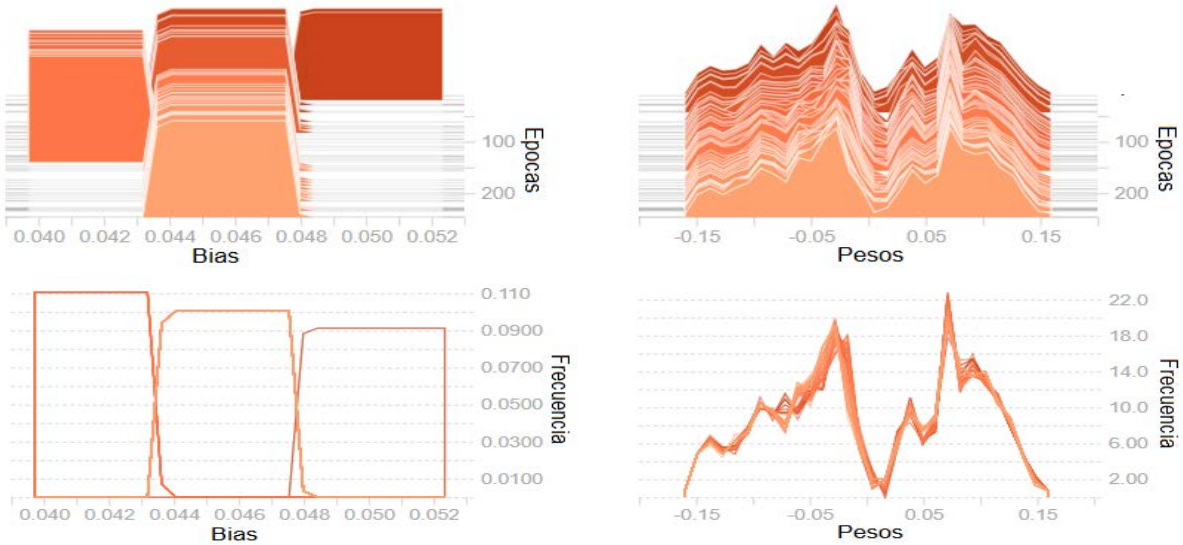


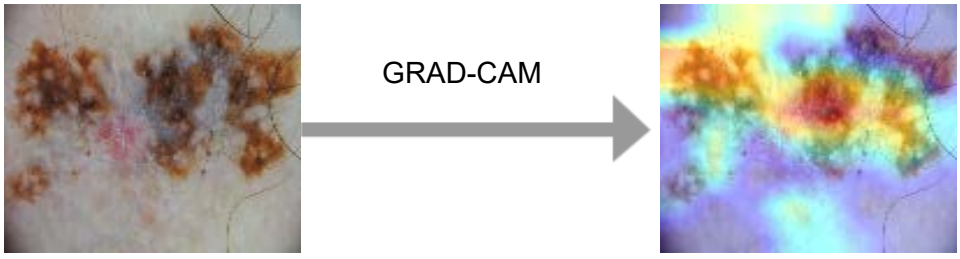
Figura 3.26: Histogramas de los pesos y del bias de la última capa convolucional del Clasificador 1 cuando se lleva a cabo el fine tuning con las imágenes (sin segmentación) del conjunto de datos de entrenamiento.

Este modelo se entrenó con los parámetros indicados en la tabla 3.4.

Parámetro	Valor	Comentario
Optimizador	SGD	Tasa de aprendizaje: 10^{-5} Momento: 0.9
Función de pérdida	Binary cross entropy	
Métrica	Exactitud	
Batch size	16	
Epocas	140	
Número de ejemplos por época	1528	18336 imágenes con data augmentation

Tabla 3.4: Parámetros del Clasificador 1 durante el fine tuning (paso 3 del entrenamiento).

Por último, una vez entrenado el modelo se desea visualizar qué partes de las imágenes tomó en cuenta la red (por medio de los mapas de calor) para realizar la clasificación correspondiente. Para tal fin, se aplicó un método (denominado GRAD-CAM y explicado en la sección 2.5.2) a cada imagen del conjunto de datos de validación. A modo de ejemplo, en la figura 3.27 se visualiza el mapa de calor con las zonas que la red tomó en consideración para clasificar a una lesión pigmentada (del conjunto de datos de validación) como melanoma. Se observa que la parte central de la lesión es donde la red encuentra la mayor cantidad de patrones para clasificar a esta imagen como un melanoma. De esta forma, al entender que partes de la lesión pigmentada toma en cuenta la red para su clasificación, podemos verificar si nuestro modelo está funcionando correctamente. Si observamos un mal funcionamiento, se deberá volver a entrenar el clasificador variando los parámetros o incorporando un mayor número de instancias (imágenes de lesiones pigmentadas). Nos referimos a un mal funcionamiento cuando, por ejemplo, la red clasifica las imágenes utilizando zonas que se encuentran fuera del área de la lesión pigmentada. En el caso mostrado de la figura 3.27, se visualiza que prácticamente todo el mapa de calor se encuentra por arriba de la lesión, con lo cual podemos concluir que para esta imagen el clasificador está funcionando correctamente. Se aclara que cuando se hace referencia a un funcionamiento correcto de la red, no implica que la clasificación de la clase sea correcta, solo que toma lo sectores de la imagen adecuados para realizar la clasificación.



(a) (b)

Figura 3.27: (a) Melanoma con Max-RGB. (b) Melanoma con el correspondiente mapa de calor generado por el método GRAD-CAM.

Con este mismo criterio, se verificó el funcionamiento del modelo entrenado, observando el mapa de calor en el restante número de imágenes que pertenecen al conjunto de datos de validación.

Por otro lado, para entender aún más el modelo entrenado, se visualizan las imágenes de entrada que producen una máxima activación en los kernels de las distintas capas convolucionales de esta arquitectura. Es decir, se trata de encontrar una imagen para un kernel (de una capa determinada) de forma tal que mediante una función de pérdida definida, se maximice el mapa de activación (Activation map) de dicho kernel. Con lo cual, como queremos encontrar un máximo, se definió a la función de pérdida como la media del mapa de activación. Luego, se calculó el gradiente de esta función de pérdida respecto a la imagen de entrada y se aplicó el gradiente ascendente para encontrar la imagen de entrada que maximiza la mencionada función de pérdida (y en consecuencia el mapa de activación del kernel). Por ende, los pesos del kernel permanecieron fijos, mientras se modificó la imagen de entrada en cada iteración del algoritmo, buscando que el mapa de activación (del kernel) sea máximo. Se seteó un número de iteraciones y en cada una de ellas se procedió a calcular el gradiente de la función de pérdida con respecto a la imagen de entrada con el objetivo de volver a modificar dicha imagen de entrada en el sentido del gradiente calculado. Luego de una determinada cantidad de iteraciones (20 en este caso), se visualiza la imagen generada que representa los patrones que excitan al kernel. La visualización de las imágenes de entrada (correspondientes a los bloques 1, 2, 3 , 4 y 5 de la figura 3.24) que más activan a los kernels se observan en la figura 3.28. En esta última figura, los kernels que se encuentran en los primeros niveles (bloques 1 y 2) de la arquitectura de la figura 3.24, codifican estructuras simples o finas. Luego, a medida que se avanza en los niveles, se observa que la activación de los kernels ocurre con estructuras más complejas. De forma tal que en los últimos niveles, las imágenes de entrada (que maximizan la activación de los kernels) contienen patrones específicos.

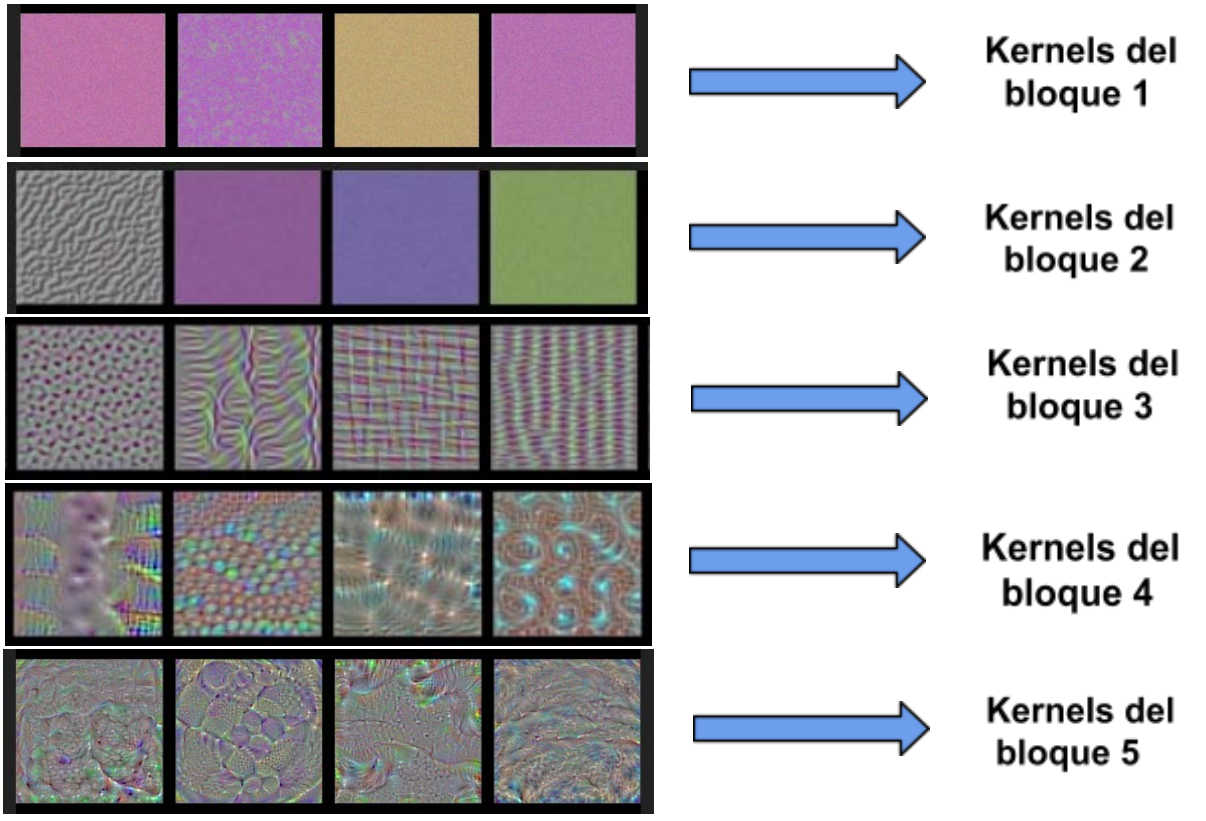


Figura 3.28: Patrones de entrada que excitan a algunos kernels de los 5 diferentes bloques de la arquitectura mostrada en la figura 3.24.

Clasificador Auxiliar

Para construir este clasificador, se siguió el mismo procedimiento que para el Clasificador 1. **Es decir, tiene la misma arquitectura y se siguió el mismo proceso de entrenamiento**. No obstante existen 2 diferencias:

- Las imágenes que conforman los conjuntos de datos de validación y entrenamiento se encuentran **segmentadas**, tal cual se explicó en la sección 3.3.1, por medio del Segmentador. Es decir, la lesión contenida en la segmentación se encuentra en colores mientras que la imagen fuera de la segmentación se encuentra en una escala de grises.
- Finalizado el entrenamiento de este clasificador, **se lo utilizó solamente como extractor de los mapas de características de la última capa de pooling (llamados características de cuello de botella)**. O sea, a este clasificador no se lo empleó para calcular la probabilidad de que una lesión pigmentada sea “Melanoma” (por medio de las capas totalmente conectadas). No obstante, estos mapas de características extraídos son la entrada del Clasificador 3 (Modelo Híbrido con segmentación) y con este último modelo se calculó la probabilidad de que una lesión pigmentada sea “Melanoma”. En la figura 3.29 se observa el funcionamiento del clasificador Auxiliar.

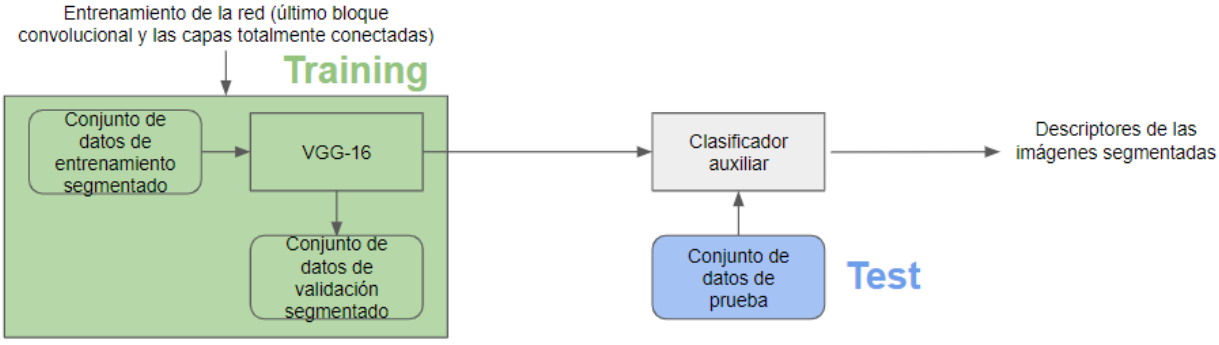


Figura 3.29: Funcionamiento del clasificador Auxiliar.

Al igual que en el Clasificador 1, se procede con los siguientes 3 pasos para realizar el entrenamiento:

1. Se realizó el transfer learning de los pesos que se obtuvieron con el entrenamiento de la arquitectura VGG-16 con ImageNet (para más detalles ver sección 2.5.1).
2. Se creó una pequeña red totalmente conectada y se inicializaron sus pesos por medio de la inicialización He [116]. Luego, se entrenó dicha red con las características de cuellos de botella mostradas en la figura 3.20 durante 70 épocas.
3. Se reemplazó el último bloque (capas totalmente conectadas) de la arquitectura mostrada en la figura 3.20 por la pequeña red entrenada en el paso anterior. Luego, se entrenó el último bloque convolucional y la pequeña red acoplada durante 140 épocas. Esto último se denomina fine tuning.

A continuación se describen en detalle los últimos 2 pasos.

Paso 2 del entrenamiento

Al igual que en el entrenamiento del Clasificador 1, en este paso se creó una pequeña red del tipo totalmente conectada cuya arquitectura se observa en la figura 3.21, y se utilizó la inicialización He para inicializar sus pesos. Luego, se la entrenó con las características de cuello de botella correspondientes a la salida de la última capa de pooling de la arquitectura VGG-16 (ver figura 3.24). A diferencia del Clasificador 1, las características de cuello de botella se obtuvieron haciendo pasar por la red (VGG-16) las **imágenes segmentadas** del conjunto de datos de validación y de entrenamiento. Mientras que en el Clasificador 1, para obtener las características de cuello de botella, se utilizaron las imágenes sin segmentación. Durante el entrenamiento, el rendimiento de esta mini red (de la figura 3.21) se puede observar en la figura 3.30.

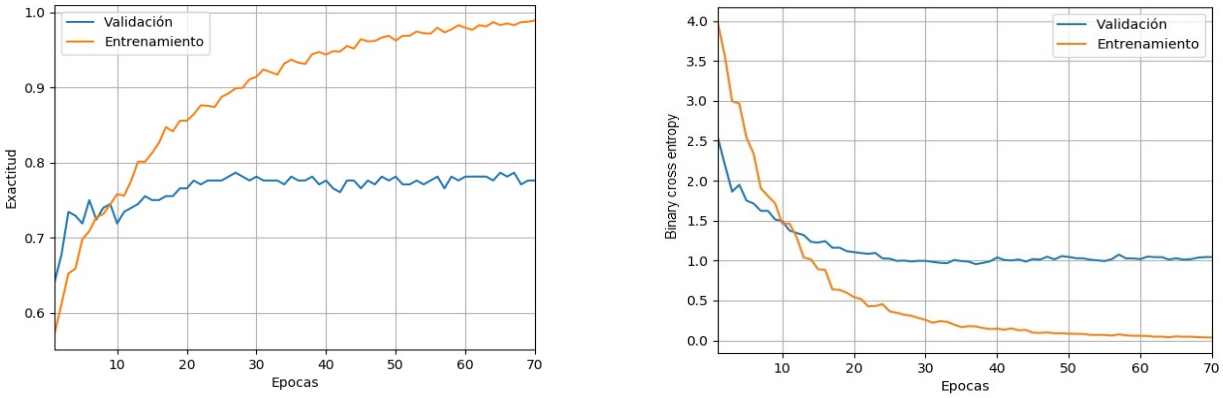


Figura 3.30: Rendimiento del la red visualizada en la figura 3.21, entrenada con las características de cuello de botella obtenidas con las imágenes segmentadas de los conjuntos de datos de entrenamiento y validación. (Izquierda) Exactitud en función de las épocas. (Derecha) Binary cross entropy en función de las épocas.

Durante esta fase de entrenamiento, en la figura 3.31 se visualiza la distribución de los pesos y bias (en función de las épocas) de la última capa totalmente conectada de esta pequeña red (ver figura 3.21). Con lo cual se observa que los pesos y bias se encuentran en rangos controlados, no observándose la desaparición o explosión del gradiente. Si bien se visualiza el análisis de la no existencia de estos 2 fenómenos en la última capa totalmente conectada de la red mostrada en la figura 3.21, también se verificó que el gradiente se encuentre controlado en la primer capa totalmente conectada de la mencionada red.

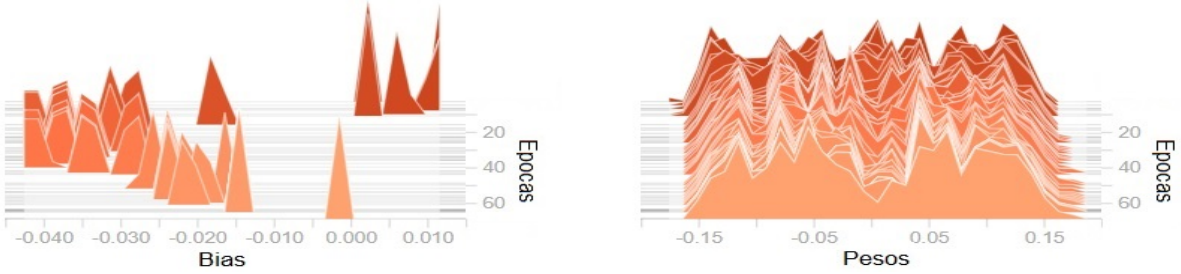


Figura 3.31: Histogramas de los pesos y bias de la última capa totalmente conectada de la red (mostrada en la figura 3.21) teniendo como entrada a las características de cuello de botella de las imágenes segmentadas del conjunto de datos de entrenamiento.

Paso 3 del entrenamiento

Al igual que en el Clasificador 1, se procedió a hacer fine tuning, actualizando los pesos de las capas que se encuentran en el bloque convolucional 5 y el último bloque (capas totalmente conectadas) de la arquitectura mostrada en al figura 3.24. Durante esta fase del entrenamiento, en la figura 3.32 se observa la performance de este clasificador y en la figura 3.33 la distribución de los pesos y bias (en función de las épocas) de la última capa convolucional del bloque convolucional 5. En esta última figura se observa que se realizan pequeñas actualizaciones en los pesos y/o bias. Esto se corresponde a que el ajuste de los mismos durante el fine tuning se realiza con una tasa pequeña de aprendizaje y con un optimizador SGD, lo que permite tener controlado la variación del gradiente a través de las épocas. Con lo cual, no se observan los fenómenos de desaparición del gradiente o de crecimiento exponencial. Como se mencionó en el entrenamiento del Clasificador 1, si existieran estos fenómenos podrían provocar actualizaciones de pesos y/o bias exponenciales o nulas, lo que desembocaría en un entrenamiento defectuoso.

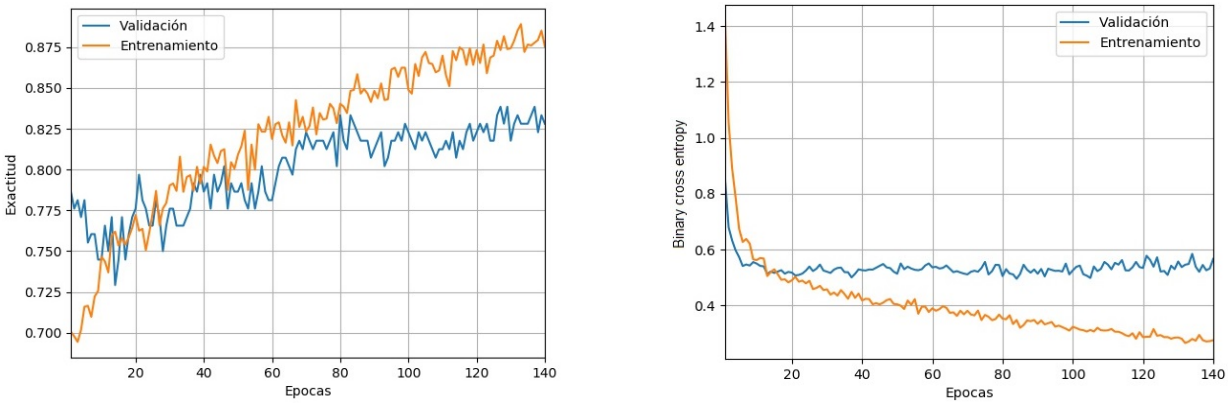


Figura 3.32: Rendimiento del Clasificador Auxiliar durante el fine tunning (en el último bloque convolucional y en la red acoplada totalmente conectada) utilizando las imágenes segmentadas de los conjuntos de datos de entrenamiento y validación. (Izquierda) Exactitud en función de las épocas. (Derecha) Binary cross entropy en función de las épocas.

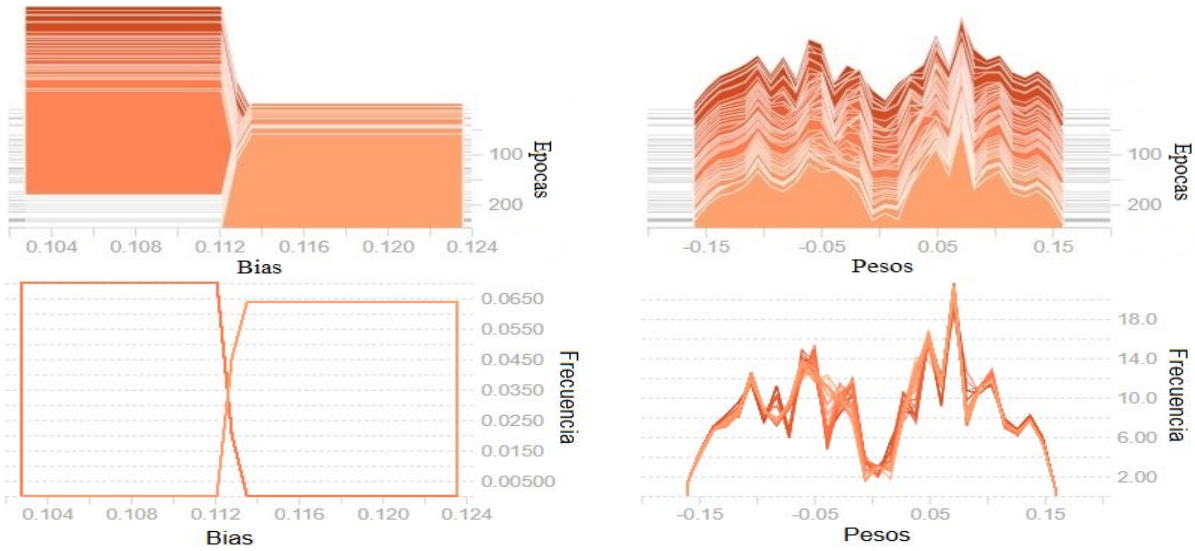
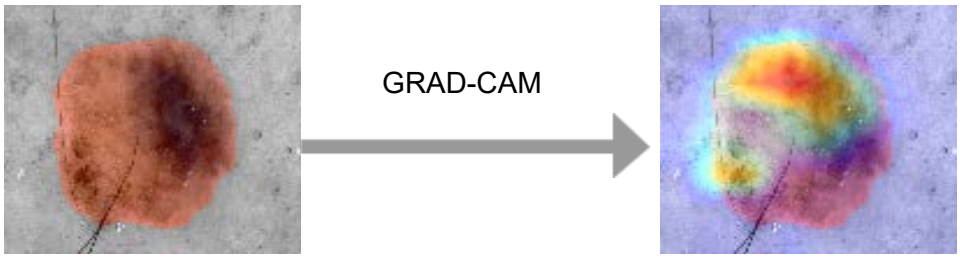


Figura 3.33: Histogramas de los pesos y del bias de la última capa convolucional del Clasificador Auxiliar cuando se lleva a cabo el fine tuning con las imágenes segmentadas del conjunto de datos de entrenamiento.

Se aclara que se analizó el comportamiento del gradiente en cada una de las capas que no se encuentran congeladas (ver figura 3.24), es decir, las que pertenecen al bloque convolucional 5 y al último bloque (capas totalmente conectadas). No obstante, por una cuestión de espacio, se explicita sólo el análisis del comportamiento del gradiente en la última capa convolucional (visualizado en la figura 3.33).

Los parámetros de este Clasificador Auxiliar para realizar el paso 2 del entrenamiento (entrenamiento de la mini red totalmente conectada) y el paso 3 del entrenamiento (fine tuning), son los visualizados en las tablas 3.3 y 3.4 respectivamente. Es decir, son los mismos parámetros utilizados que en el entrenamiento del Clasificador 1.

Al igual que en el Clasificador 1, el funcionamiento de este clasificador se verificó observando el mapa de calor generado por el método GRAD-CAM en las predicciones realizadas en las lesiones pigmentadas (con segmentación) pertenecientes al conjunto de datos de validación. Esto último puede observarse en la figura 3.34.



(a) (b)

Figura 3.34:(a) Lesión pigmentada segmentada con Max-RGB.(b) Lesión pigmentada segmentada con el correspondiente mapa de calor generado con el método GRAD-CAM.

3.3.3 Clasificador 2 (Modelo Híbrido sin segmentación de imágenes)

Tal cual explicamos en la sección 2.5.3, es posible realizar un Modelo Híbrido compuesto por un generador de mapas de características (o descriptores) como puede ser la salida de la última capa de pooling del Clasificador 1, un codificador de estos descriptores por medio de los vectores de Fisher y un clasificador como podría ser un SVM. Este Modelo Híbrido es el que se muestra en la figura 3.35.

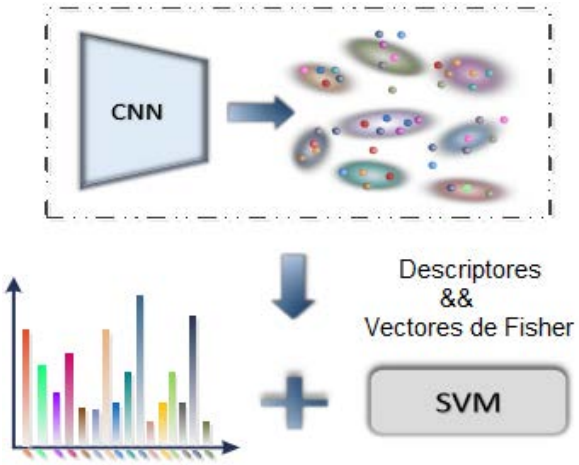


Figura 3.35: Modelo Híbrido. Compuesto por una red CNN (generadora de los descriptores que son codificados por medio de los vectores de Fisher) y un SVM como clasificador.

Para entrenar este modelo, se deben cambiar los parámetros de sus distintas componentes para obtener una mejor clasificación final (ver figura 3.36). Esto último se realizó de la siguiente forma:

1) En la sección 3.3.2 se explicó el entrenamiento del Clasificador 1 con el objetivo de obtener la mejor clasificación. En esta sección usamos dicho clasificador, previamente entrenado, hasta la última capa de pooling. Esto es debido a que se desea obtener los descriptores (y no la clasificación mediante las capas totalmente conectadas) de las imágenes de los conjuntos de entrenamiento y validación.
2) Luego se varió el "número de componentes gaussianas" y el "número de componentes principales", con el objetivo de obtener diferentes aproximaciones de la distribución real de los descriptores (del punto anterior) reducidos a una dimensión determinada. Posteriormente, estas aproximaciones de la distribuciones de los descriptores fueron codificadas en los vectores de Fisher.
3) Los descriptores codificados en los vectores de Fisher constituyeron la entrada de un SVM, en el cual se obtuvieron los parámetros óptimos mediante una validación cruzada y variando los parámetros de "costo(C)", "tipo de kernel" y "gamma" para que la métrica ROC-AUC sea máxima.

Se aclara que en la validación cruzada del entrenamiento del SVM se utilizaron 5 folds y se utilizó la validación cruzada de K iteraciones (folds). En este tipo de validación, los datos de muestra se dividieron en K subconjuntos iguales. Uno de los subconjuntos se utilizó como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada fue repetido durante K iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Luego se realiza la media de los resultados (ROC-AUC) hallados en cada subconjunto de prueba de cada iteración para obtener un único resultado.

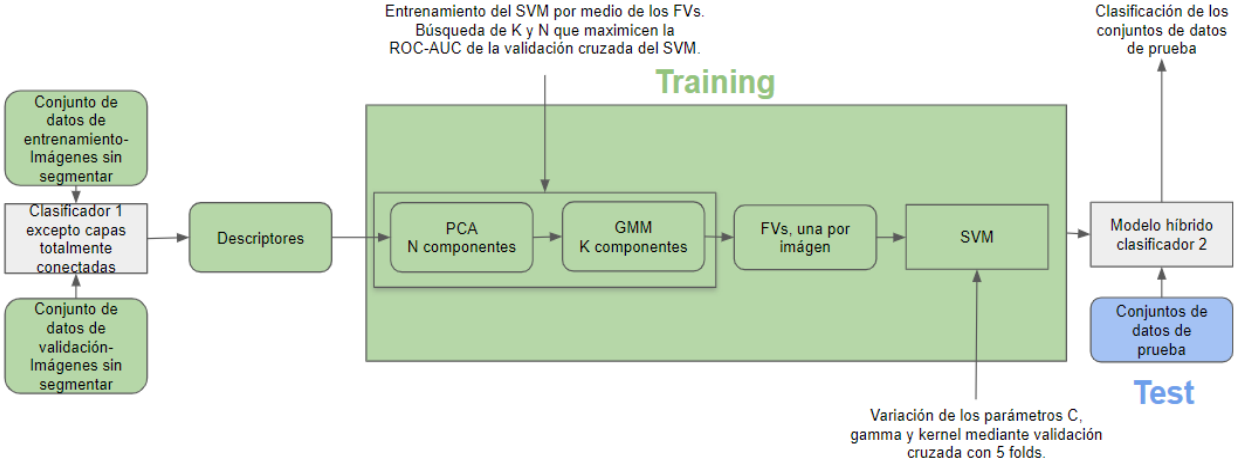


Figura 3.36: Entrenamiento del Modelo Híbrido con los descriptores de las imágenes sin segmentar.

Se ingresaron al Clasificador 1 (previamente entrenado) las imágenes (sin segmentar) de los conjuntos de datos de entrenamiento y validación. Posteriormente se obtuvieron los mapas de características de la última capa de pooling de dichas imágenes. Estos descriptores constituyeron el conjunto de datos de entrenamiento del Modelo Híbrido. Se aclara que no se separó un conjunto de descriptores de validación dado que, como se mencionó anteriormente, se empleó la validación cruzada con 5 folds sobre el conjunto de descriptores de entrenamiento generado (ver figura 3.36).

Una vez que se obtuvieron los descriptores, en la figura 3.36 se observa que el primer parámetro a variar del Modelo Híbrido es el número de componentes principales. Para verificar la cantidad de componentes principales que necesitamos tomar para que expliquen a los descriptores en forma precisa (poca pérdida de información), se realizó un gráfico de sedimentación y se buscó cuál es el punto de quiebre. Es decir, en donde la curva dejó de disminuir con rapidez o presentar pendientes negativas pronunciadas. Observamos en la figura 3.37 que con 128 componentes sucede este fenómeno, no obstante, a este número de componentes se lo puede considerar como “de mínima”.

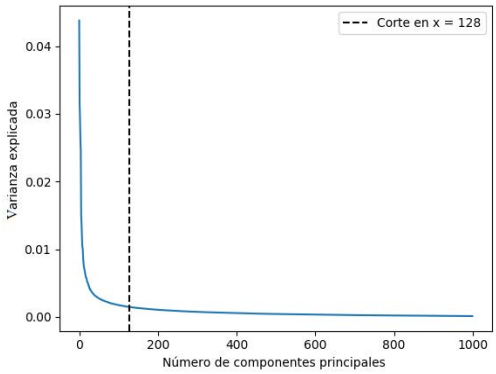


Figura 3.37: Varianza explicada (de los descriptores de las imágenes sin segmentar) en función del número de componentes principales (gráfico de sedimentación). Se observa que para 128 componentes principales se produce un punto de quiebre.

Se comprobó que al aplicar PCA en los descriptores (que poseen 25088 dimensiones) con una retención de más de 600 componentes principales, provocó que el tiempo del entrenamiento del modelo aumente significativamente sin obtener una mejora en el rendimiento que lo justifique. Por ende, se estableció como cota superior las 600 componentes principales. En la figura 3.38 se observa la varianza explicada de las componentes y en color verde se indica el conjunto de componentes que se tomaron para el entrenamiento del modelo. Dichas componentes (entre 128 y 600) explican entre un 55% y un 85% de la varianza de los descriptores.

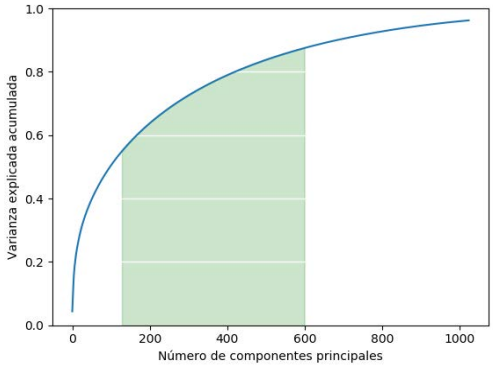


Figura 3.38: Varianza explicada acumulada (de los descriptores de las imágenes sin segmentar) según el número de componentes principales que se retengan.

Con lo cual, se entrenó el Modelo Híbrido con un conjunto de componentes principales incluidas en el intervalo [128,600] (ver figura 3.38), ya que el objetivo final fue lograr la

máxima ROC-AUC en la validación cruzada del SVM y no obtener el número de componentes principales que mejor expliquen los descriptores. De hecho, el experimento realizado en [9] indica que se obtienen mejores resultados con la utilización de PCA que sin la utilización de PCA. Además, el número óptimo de componentes principales que hallaron en dicho experimento no fue precisamente el mayor número de componentes (del conjunto de componentes) que analizaron. Tomando en consideración esto último, se entrenó el Modelo Híbrido con las componentes principales incluidas en el siguiente conjunto: N={128,256,400,500,600}.

Luego, se pretende obtener el conjunto K que incluye distintos números de componentes gaussianas (k_i) que representan distintas aproximaciones a la distribución de los descriptores. Para tal fin, se aplicó PCA a los mencionados descriptores reteniendo 1000 componentes principales para obtener un conjunto de datos de menor dimensión y que explica el 95% de la varianza. Prácticamente este nuevo conjunto de datos explica la totalidad de la varianza de los descriptores originales. Con lo cual, se utilizó este nuevo conjunto de datos para la determinación del conjunto K de distintos números de componentes gaussianas, empleando los criterios de información de Akaike y Bayesiano (explicados en la sección 2.5.3.3). En la figura 3.39 se observa el puntaje de AIC y BIC que se obtuvo al variar el número de componentes gaussianas en el intervalo [1, 29]. Según lo explicado en la sección 2.5.3.3, los modelos que mejor se aproximan a la distribución verdadera de los datos son los que obtienen el menor puntaje. Observamos que para k=*5* se obtiene un mínimo en AIC, y en BIC se obtiene un bajo puntaje dentro de su rango de valores ascendentes. No obstante, dado que el modelo de clasificación involucra tanto el modelo GMM como el SVM, se varió el número de componentes gaussianas en las inmediaciones del mínimo encontrado con el objetivo de hallar la mejor clasificación en la validación cruzada del SVM. Si bien en la figura 3.39 se observa que las componentes se encuentran comprendidas en el rango [1, 29], se desea probar un número mayor de componentes gaussianas. Dichas componentes no se graficaron en la curva por temas de performance, ya que a medida que el número de componentes gaussianas aumenta se requiere mayor potencia de cómputo. Es decir, para visualizar la curva en forma correcta se calcularon los puntajes AIC y BIC para cada una de las componentes gaussianas comprendidas en el intervalo [1,29]. Con lo cual, cuando se quiso agrandar el intervalo a [1,100] se debió calcular el AIC y BIC en 100 modelos GMM, causando la cancelación del proceso por limitaciones del hardware. No obstante, es posible probar valores puntuales de componentes, por ende, se tomaron las componentes gaussianas incluidas en el siguiente conjunto: K={5,15,20,40,45,50,60,100}.

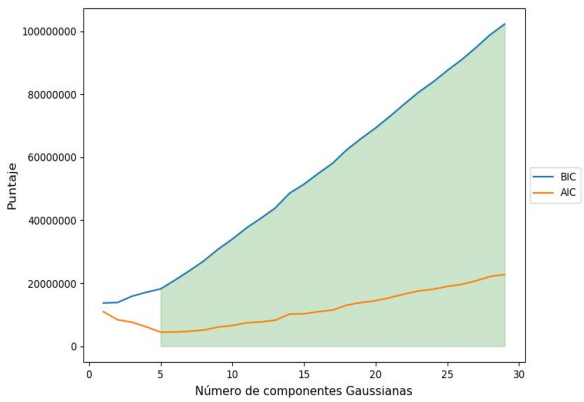


Figura 3.39: Puntaje del AIC y BIC según el número de componentes gaussianas que aproximan la distribución de los descriptores de las imágenes sin segmentar.

Por cada par (n_i, k_i) que surge de la combinación de valores entre los conjuntos N (cada elemento corresponde a un número de componentes principales) y K (cada elemento corresponde a un número de componentes gaussianas), se entrenó el SVM con validación cruzada (con 5 folds) combinando los valores de los hiperparámetros mostrados en la tabla 3.5.

Parámetro	Valor	Comentario
Costo (C)	{1, 10, 100, 1000}	Se encontraron los parámetros óptimos del SVM (para obtener una ROC-AUC máxima) por cada una de los 40 combinaciones entre K y N. Donde: K={5,15,20,40,45,50,60,100} N={128,256,400,500,600}
Gamma	$\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$	
Kernel	{Lineal, RBF}	

Tabla 3.5: Hiperparámetros del SVM para el Clasificador 2.

Luego, por cada par de valores (n_i, k_i), se obtuvieron los hiperparámetros óptimos del SVM (ver tabla 3.5) que maximizan el ROC-AUC en la validación cruzada. Según lo observado en la figura 3.40, la máxima ROC-AUC se obtiene con 15 componentes gaussianas, 256 componentes principales, kernel del SVM= Lineal o RBF, C = 1 y gamma = 0.1.

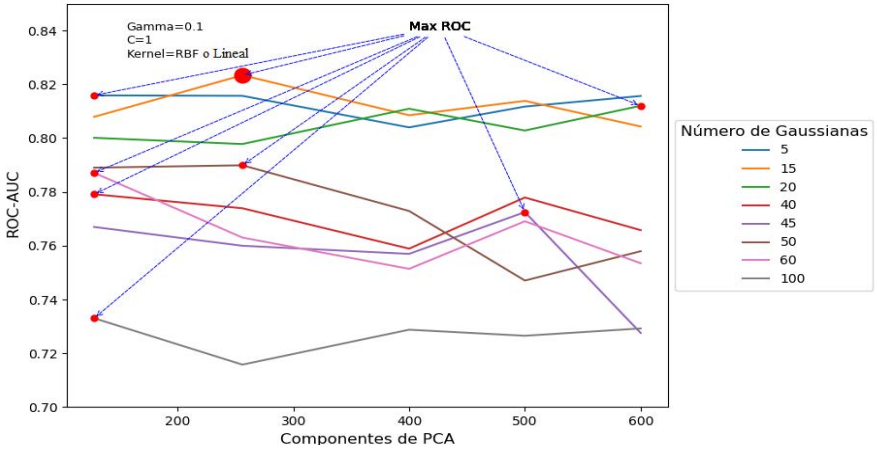


Figura 3.40: ROC-AUC máxima del SVM (del Clasificador 2) por cada combinación del número de componentes gaussianas (k_i) y número de componentes principales (n_i).

Para justificar los hiperparámetros óptimos del SVM que se muestran en la figura 3.40 (para la máxima ROC-AUC hallada), se exponen las figuras 3.41 y 3.42 donde se observa el ROC-AUC que se obtuvo para las combinaciones de los parámetros de gamma y costo, tanto para un kernel lineal como para uno RBF. Estas combinaciones son el producto de setear previamente a k=15 y n=256 que es donde se se encuentra la máxima ROC-AUC visualizada en la figura 3.40. En estas combinaciones, efectivamente, se observa que para C=1, kernel RBF o Lineal y gamma=0.1 se obtiene una máxima ROC-AUC de 0.823. Los demás valores que se observan como 0.823 en la figura 3.41, en realidad se encuentran comprendidos entre 0.8226 y 0.8229. Con lo cual son redondeados a 0.823 y por ende no son considerados máximos. En la figura 3.42 se observa que los valores de ROC-AUC no se encuentran en función del parámetro gamma. Esto se debe a que el kernel lineal no lo tiene como parámetro

ya que gamma es un parámetro para el kernel RBF. La explicación de los parámetros gamma y costo se encuentra en la sección 2.5.3.6.

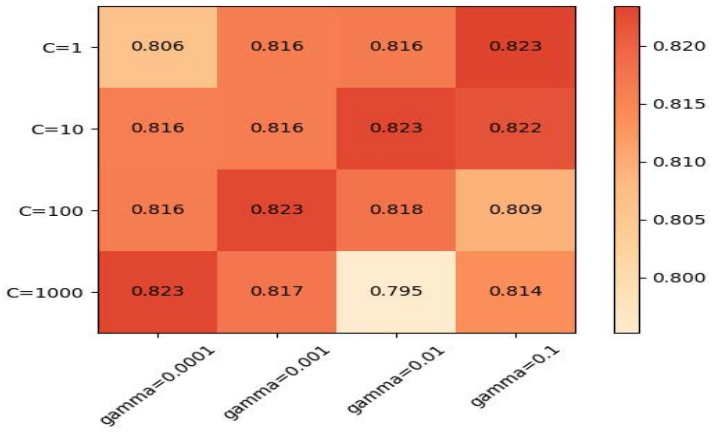


Figura 3.41: ROC-AUC del SVM (del Clasificador 2) para la combinación de los parámetros gamma y costo, utilizando un kernel RBF para 15 componentes gaussianas y 256 componentes principales.

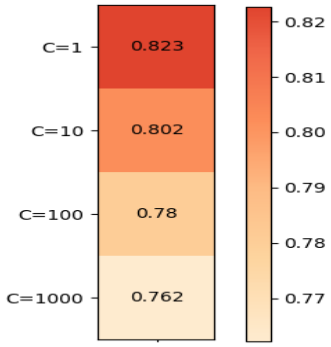


Figura 3.42: ROC-AUC del SVM (del Clasificador 2) para los distintos valores de costo, utilizando un kernel lineal para 15 componentes gaussianas y 256 componentes principales.

En la figura 3.43 se visualiza el ROC-AUC que se obtuvo en los distintos folds de la validación cruzada para la máxima ROC-AUC obtenida (0.823) que corresponde a los siguientes parámetros del Modelo Híbrido: k=15, n=256, C=1, gamma=0.1 y kernel= Lineal o RBF.

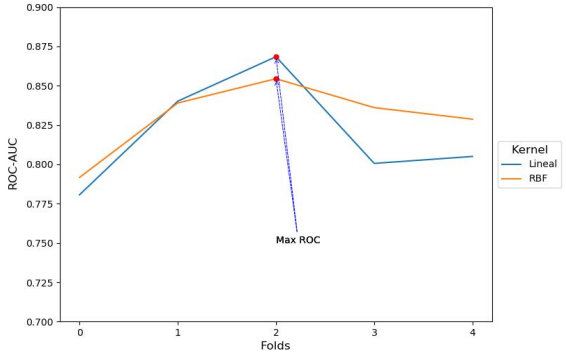


Figura 3.43: Valores del ROC-AUC (del Clasificador 2), en las distintos folds de la validación cruzada, para la máxima ROC-AUC obtenida (0.823).

Por último, se aclara que utilizando el criterio de la navaja Ockham, se seleccionó el tipo de kernel lineal ya que con este se alcanza el mismo resultado (ROC-AUC = 0.823) y tiene una menor complejidad que si se utiliza el kernel RBF.

3.3.4 Clasificador 3 (Modelo Híbrido con segmentación de imágenes)

El Clasificador 3 se obtuvo siguiendo el mismo procedimiento que el realizado para obtener el Clasificador 2. La diferencia radica que los descriptores se obtuvieron mediante el Clasificador Auxiliar en lugar del Clasificador 1. Esto último explicado se puede visualizar en la figura 3.44.

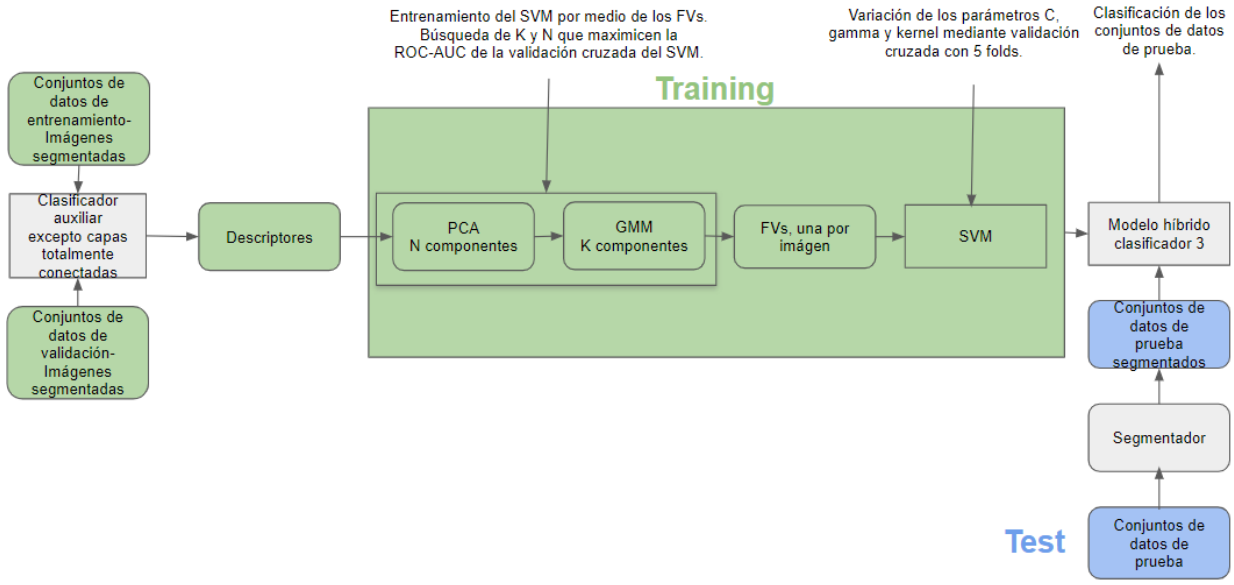


Figura 3.44: Modelo Híbrido con los descriptores de las imágenes segmentadas.

En la figura 3.44 se observa que los descriptores se obtienen con el Clasificador Auxiliar, y constituyen la entrada para la generación de los vectores de Fisher (FVs). En la generación de los FVs, se encontraron los hiperparámetros N (componentes principales) y K (componentes gaussianas) que maximizaron el ROC-AUC en la validación cruzada del SVM. Por ende, al igual que en la sección 3.3.3, el primer paso fue elegir el rango de valores de las componentes principales para los cuales se probó el modelo. En la figura 3.45 se observa el gráfico de sedimentación en el cual se obtiene, prácticamente, la misma curva que en la figura 3.37 de la sección 3.3.3, en donde el punto de quiebre se encuentra en 128 componentes principales.

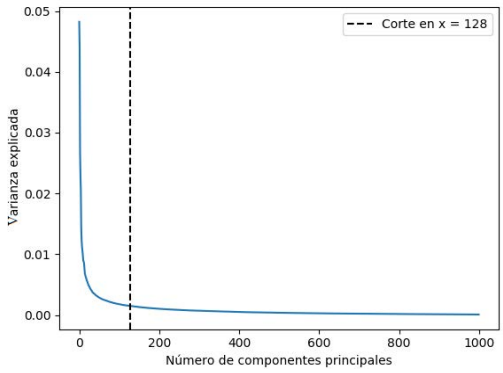


Figura 3.45: Varianza explicada (de los descriptores de las imágenes segmentadas) en función del número de componentes (gráfico de sedimentación). Se observa que para 128 componentes principales se produce un punto de quiebre.

Luego, siguiendo el mismo proceso que el realizado para el Clasificador 2, observamos en la figura 3.46 la varianza explicada en función de las componentes principales. Con lo cual, si tomamos las componentes que se encuentran en el intervalo [128,600] explicaremos entre el 60% y el 85% de la varianza total de los descriptores. Por ende, se entrenó el modelo con las componentes incluidas en el siguiente conjunto: N={128,256,400,500,600}.

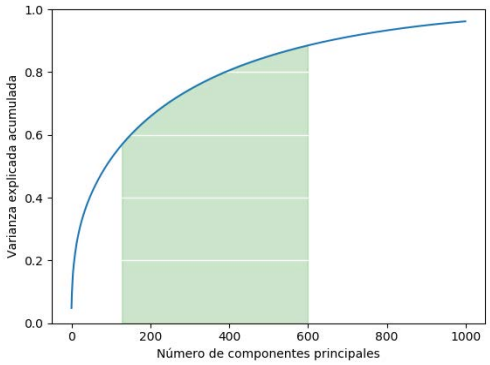


Figura 3.46: Varianza explicada acumulada (de los descriptores de las imágenes segmentadas) según el número de componentes principales.

Posteriormente, se determinó el conjunto de valores que va a tomar K, donde cada elemento de dicho conjunto (k_i) representa el número de componentes gaussianas. Siguiendo el mismo procedimiento que el utilizado en el Clasificador 2, se aplicó PCA con 1000 componentes principales al conjunto de datos de los descriptores. El objetivo fue obtener un conjunto de datos de menor dimensión y que explica el 95% de la varianza (prácticamente el total de la varianza) de los descriptores. Luego, tomando este nuevo conjunto de datos de menor dimensión, se generaron 29 modelos GMM, cada uno de ellos con un número de componentes gaussianas comprendido en el intervalo [1,29]. Es decir, el primer modelo tendrá una sola componente gaussiana y el último tendrá 29 componentes gaussianas. Por cada uno de estos modelos se calcularon los indicadores AIC y BIC, los cuales se visualizan en la figura 3.47. Además, se observa que para k=3 se obtiene un bajo puntaje tanto en AIC como en BIC. No obstante, dado que el modelo de clasificación involucra tanto el modelo GMM como el SVM, se varió el número de componentes gaussianas en los alrededores de k=3 con el objetivo de encontrar la mejor clasificación en la validación cruzada en el SVM. Además, por problemas de performance y como ocurrió en el procedimiento para obtener el Clasificador 2, no se incluyeron los números de componentes gaussianas mayores a 29 en lo que respecta a la visualización de la curva (ver figura 3.47). Sin embargo, se pudieron probar grandes números de componentes gaussianas en forma individual. Teniendo en cuenta lo anterior, se tomaron las componentes gaussianas que están incluidas en el siguiente conjunto: K={3,15,20,40,45,50,60,100}.

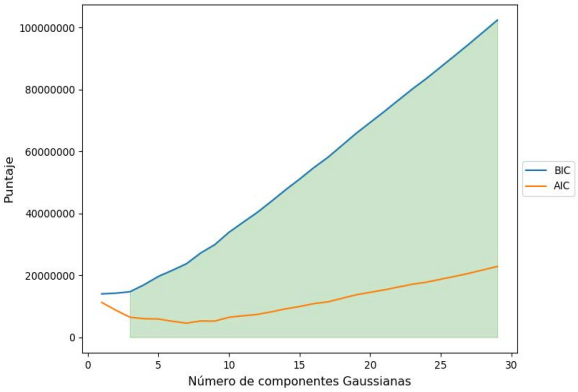


Figura 3.47: Puntaje del AIC y BIC según el número de componentes gaussianas.

A continuación, por cada par (n_i, k_i) que surge de la combinación de valores entre los conjuntos N (cada elemento corresponde a un número de componentes principales) y K (cada elemento corresponde a un número de componentes gaussianas), se realizó la validación cruzada (con 5 folds) en el SVM. Para dicha validación cruzada se combinaron los hiperparámetros mostrados en la tabla 3.6 para obtener la máxima ROC-AUC.

Parámetro	Valor	Comentario
Costo(C)	{1, 10, 100, 1000}	Se encontraron los parámetros óptimos del SVM (para obtener una ROC-AUC máxima) por cada una de los 40 combinaciones entre K y N.Donde: K={3,15,20,40,45,50,60,100} N={128,256,400,500,600}
Gamma	$\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$	
Kernel	{Lineal, RBF}	

Tabla 3.6: Hiperparámetros del SVM para el Clasificador 3.

Los hiperparámetros óptimos encontrados para el Clasificador 3 que hacen que el ROC-AUC sea máxima en la validación cruzada del SVM son: 15 componentes gaussianas, 256 componentes principales, kernel= Lineal o RBF, C = 10 y gamma = 0.1. Esto último se puede observar en la figura 3.48.

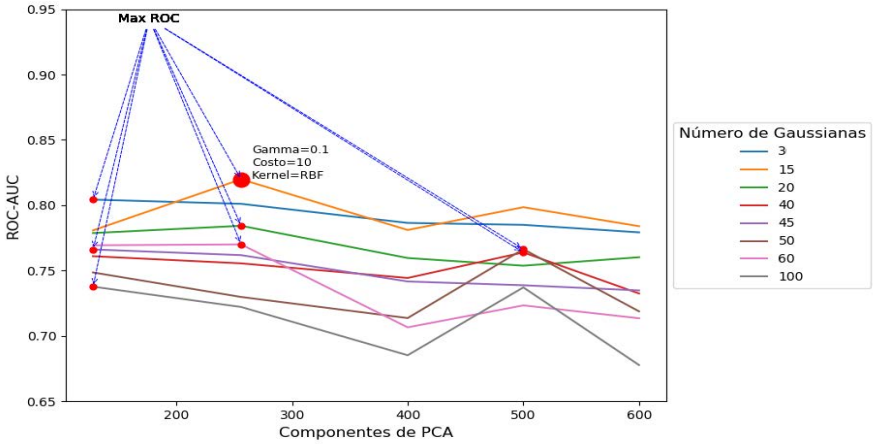


Figura 3.48: ROC-AUC máxima del SVM (del Clasificador 3) por cada combinación del número de componentes gaussianas (k_i) y número de componentes principales (n_i).

Luego, en las figuras 3.49 y 3.50, se observan los valores que se obtienen de ROC-AUC para la combinación de los hiperparámetros gamma y costo, tanto para un kernel Lineal como RBF. Para realizar estas combinaciones, se fijó el número de componentes gaussianas en k=15 y el número de componentes principales en n = 256 ya que con estos valores se obtuvo la máxima ROC-AUC en la figura 3.48. En dichas combinaciones, efectivamente, se ratifica que para C=10, kernel=RBF y gamma=0.1 se obtiene una máxima ROC-AUC de 0.819 (que es la que se visualiza en la figura 3.48). Se aclara los valores de ROC-AUC (en la figura 3.50) no se encuentran en función del parámetro gamma ya que este parámetro es un parámetro del kernel RBF (y no del kernel lineal).

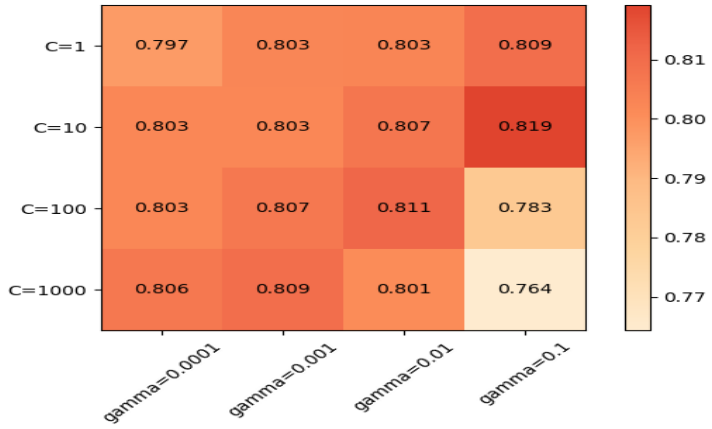


Figura 3.49: ROC-AUC del SVM (del Clasificador 3) para la combinación de los parámetros gamma y costo, utilizando un kernel RBF.

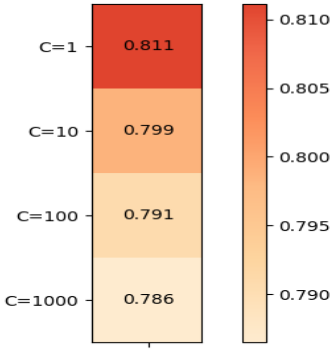


Figura 3.50: ROC-AUC del SVM (del Clasificador 3) para los distintos valores de costo, utilizando un kernel lineal.

Por último, en la figura 3.51 se observa el indicador ROC-AUC obtenido en los distintos folds (de la validación cruzada) para la máxima ROC-AUC (0.819) con k=15, n=256, C=10, gamma=0.1 y kernel RBF.

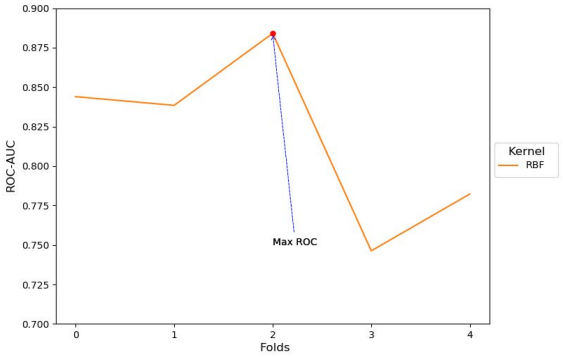


Figura 3.51: Valores del ROC-AUC (del Clasificador 3), en las distintos folds de la validación cruzada, para la máxima ROC-AUC obtenida (0.819).

3.3.5 Ensamble

Una vez realizado el entrenamiento de los 3 clasificadores explicados en las secciones anteriores, se creó otro clasificador que es el Ensamble de los mismos. Es decir, en este Ensamble, la probabilidad de que una lesión pigmentada sea melanoma está determinada por el promedio aritmético de las probabilidades que cada uno de los 3 clasificadores (integrantes del Ensamble) le asigna a dicha lesión. Como el Clasificador 3 requiere que las imágenes estén segmentadas, la entrada del Ensamble está compuesto tanto por las imágenes originales como las segmentadas. Además, se aplicó Max-RGB (explicado en la sección 2.3.4) a todas las imágenes. Lo anteriormente mencionado se vé reflejado en la figura 3.52.

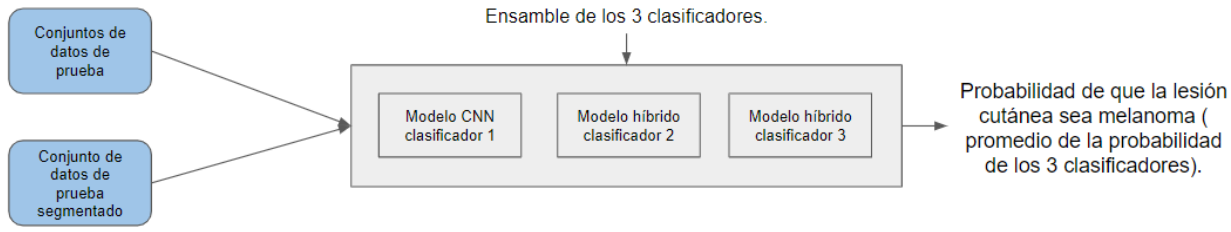


Figura 3.52: Ensamble de los 3 clasificadores para obtener la probabilidad de que la lesión pigmentada sea melanoma.

4 Resultados

Organización de los resultados

A lo largo de la sección 3.3 (Métodos), se explicó cómo se entrenaron cada uno de los clasificadores. La selección de los hiperparámetros óptimos involucrados en la generación de los vectores de Fisher a partir de los descriptores fue explicada en las secciones 3.3.3 y 3.3.4. Sin embargo, no se indicó cómo se eligieron los hiperparámetros de las redes convolucionales utilizadas. Con lo cual, en este apartado se realizarán varios experimentos para justificar la elección de los hiperparámetros seleccionados para el Clasificador 1 y el Clasificador Auxiliar. Además se explicarán los experimentos realizados para seleccionar los hiperparámetros del Segmentador.

Por otro lado, se aclara que los resultados obtenidos con el Clasificador 2 (Modelo Híbrido con las imágenes sin segmentar) y el Clasificador 3 (Modelo Híbrido con imágenes segmentadas) son muy similares. Por ende, teniendo en cuenta que el objetivo es comparar el rendimiento del modelo convolucional (Clasificador 1) y el del Modelo Híbrido, se tomará al Clasificador 2 como "representante" del Modelo Híbrido en los resultados de esta sección. Además se mostrarán los resultados del Ensamble (ver sección 3.3.5) ya que es el modelo con el que se obtuvieron los mejores resultados en los distintos conjuntos de prueba.

Con lo cual, se mostrarán los resultados de los siguientes clasificadores cuando son aplicados a los conjuntos de datos de prueba (ISIC, HI, ROT, XY e ISIC 2016):

- **Clasificador 1** : Constituido por la mejor arquitectura VGG-16 que se halló en base a los experimentos que se expondrán en esta sección.

- **Clasificador 2**: Es el Modelo Híbrido óptimo en base a los descriptores del mejor Clasificador 1 hallado, según los experimentos que se mencionarán en esta sección.

- **Ensamble**: Constituido por el promedio de la probabilidad, que una lesión pigmentada sea melanoma, de los clasificadores: Clasificador 1, Clasificador 2 y Clasificador 3.

Se aclara que el Clasificador 3 que se encuentra dentro del Ensamble es el Modelo Híbrido óptimo en base a los descriptores del mejor Clasificador Auxiliar hallado, según los experimentos que se mencionarán en esta sección.

4.1 Selección de los hiperparámetros de las redes convolucionales

Para la determinación de los hiperparámetros óptimos, no nos basamos solamente en los resultados de las métricas ROC-AUC y PR-AUC sobre los conjuntos de datos de prueba, sino que también se observó la forma de las curvas generadas (con los conjuntos de datos de

entrenamiento y validación) en el entrenamiento de la red. A continuación, y a modo ilustrativo, se muestran los conceptos que se tuvieron en consideración (durante el entrenamiento) en relación a las curvas que corresponden a la función de pérdida en función de las épocas y a la Exactitud en función de las épocas.

En la figura 4.1 se observa la función de pérdida en función de las épocas. Se visualiza que dependiendo de la tasa de aprendizaje aplicada, la función de pérdida desciende o asciende a una mayor velocidad. Con lo cual, se busca una velocidad tal que la función converja en un número razonable de épocas. Además, se observa que según el batch size aplicado, la función de pérdida puede llegar a ser inestable entre las épocas. Por ende, se trata de aplicar un batch size, en el cual se logre cierta estabilidad en la función de pérdida.

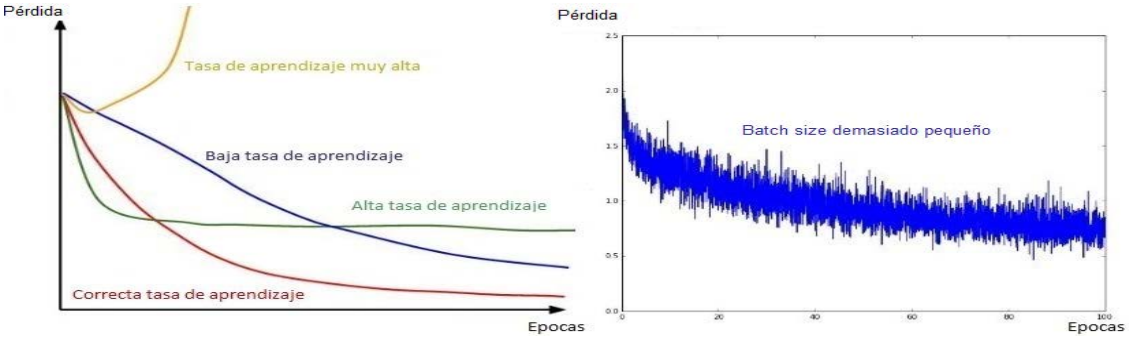


Figura 4.1: (Izquierda) Influencia de la tasa de aprendizaje en la función de pérdida. (Derecha) Influencia del batch size en la función de pérdida.

Durante el entrenamiento se trata de obtener un bajo sobreajuste, no obstante, en la figura 4.2 se observa un alto sobreajuste. Esto ocurre cuando la red empieza a memorizar los patrones de entrada en lugar de aprenderlos, ocasionando un incremento de la Exactitud en el conjunto de entrenamiento y un decremento de la Exactitud en el conjunto de validación.

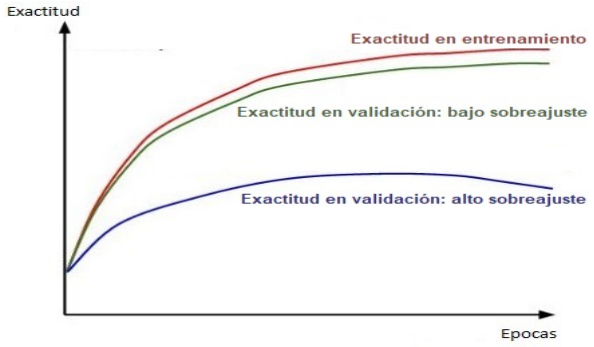


Figura 4.2: Tipos de sobreajuste en el entrenamiento de la red.

4.1.1 Hiperparámetros del Segmentador

Los hiperparámetros que se encuentran en la tabla 4.1 en color verde corresponden a los utilizados en el modelo Segmentador de las imágenes.

	Deep Learning								FV		SVM	
Modelo	**Opt. U-Net**	**Batch Size U-Net**	**Epochs U-Net**	**Tasa de aprendiz. U-Net**	**Opt. VGG-16**	**Batch Size VGG-16**	**Epochs VGG-16**	**Tasa de aprendiz. VGG-16**	**PCA (N)**	**GMM (K)**	**Cost (C)**	**Gamma**
Clasificador 1 (VGG-16)					SGD	16	140	0.00001				
Clasificador 2 (Híbrido)					SGD	16	140	0.00001	256	15	1	
Clasificador 3 (Híbrido)	Adam	4	260	0.00001	SGD	16	140	0.00001	256	15	10	0.1

Tabla 4.1: Hiperparámetros del sistema clasificador de lesiones pigmentadas.

Para llegar a determinar los hiperparámetros indicados en color verde en la tabla 4.1 se realizaron varias pruebas durante el entrenamiento de la red TernausNet-16, entre ellas:

- Se utilizó un optimizador SGD con momento Nesterov y tasas de aprendizaje iguales a 0.00001 y 0.00002. Además, se utilizó el optimizador Adam con tasas de aprendizaje iniciales de 0.00001 y 0.00002.
- Se utilizaron diferentes batch size: de 4, 8 y 12 imágenes.
- Dada una imagen, a cada píxel se le restó un "píxel medio" conformado por la media de los píxeles de cada canal (RGB) de dicha imágen. Además se probó restar a cada píxel otro "píxel medio" pero esta vez calculado como la media de los píxeles de cada canal de todas las imágenes de ImageNet. Estas 2 pruebas se repitieron para todas las imágenes que pertenecen al conjunto de entrenamiento y de validación.
- Se utilizó como función de pérdida al Coeficiente de Dice y a la Binary cross entropy.

Entre las combinaciones de estos hiperparámetros, se tienen 48 modelos con resultados comparables (utilizando los conjuntos de datos de entrenamiento y validación). Para comparar dichos modelos, además de la evaluación del Coeficiente de Jaccard, se tuvo en consideración la forma de la función de pérdida y si existe o no sobreajuste en el entrenamiento (como fue explicado en la sección 4.1). El modelo que obtuvo un Coeficiente de Jaccard máximo en el conjunto de datos de validación corresponde a la configuración indicada en la tabla 4.1. La función de pérdida utilizada corresponde al Coeficiente de Dice y a cada píxel de cada imagen se le restó un "píxel medio" calculado como la media de los píxeles de cada canal de todas las imágenes de ImageNet. Empleando este último modelo, se verificó su rendimiento aplicándolo al conjunto de datos de prueba ISIC 2017. Se utilizó este último conjunto de datos, que es el oficial de la competencia ISIC 2017 para la tarea de segmentación, con el objetivo de comparar los resultados obtenidos con el de otros participantes. Tal cual se explicó en la sección 3.3.1 se logró la octava posición con un Coeficiente de Jaccard de 0.75. Se aclara que además de las pruebas mencionadas, se realizaron otros experimentos con resultados no satisfactorios, entre ellos, utilizar una taza de aprendizaje más alta (por ejemplo, 0.00009) o la utilización de tamaños de lotes (batch size) mayores o iguales a 32.

4.1.2 Hiperparámetros del Clasificador 1 y del Clasificador Auxiliar

Los hiperparámetros óptimos indicados en color verde (correspondientes al fine tuning) de la tabla 4.2 serán justificados en base a los experimentos que se realizarán en la presente

sección y los que se encuentran en el apéndice. Los hiperparámetros indicados en las columnas FV y SVM de dicha tabla, surgen del proceso explicado en la sección 3.

	Deep Learning								FV		SVM (RBF)	
Modelo	**Opt. U-Net**	**Batch Size U-Net**	**Epochs U-Net**	**Tasa de aprendiz. U-Net**	**Opt. VGG-16**	**Batch Size VGG-16**	**Epochs VGG-16**	**Tasa de aprendiz. VGG-16**	**PCA (N)**	**GMM (K)**	**Cost (C)**	**Gamma**
Clasificador 1 (VGG-16)					SGD	16	140	0.00001				
Clasificador 2 (Híbrido)					SGD	16	140	0.00001	256	15	1	
Clasificador 3 (Híbrido)	Adam	4	260	0.00001	SGD	16	140	0.00001	256	15	10	0.1

Tabla 4.2: Hiperparámetros del sistema clasificador de lesiones pigmentadas.

Si bien en la tabla 4.2 se observan 3 clasificadores, los hiperparámetros en color verde surgen del Clasificador 1 y Clasificador Auxiliar. El Clasificador 1 además de clasificar las imágenes, es el generador de los descriptores que serán utilizados por el Clasificador 2 y el Clasificador Auxiliar generará los descriptores que serán utilizados por el Clasificador 3.

La tasa de aprendizaje y el batch size constituyen 2 de los principales hiperparámetros en las redes convolucionales. Además, otro componente clave en la performance de este tipo de redes es el optimizador u algoritmo utilizado para la actualización de los pesos. Estos 3 elementos fueron explicados en la sección 2.2.4 y serán los hiperparámetros en las distintos experimentos. Con lo cual, los 36 experimentos realizados (18 experimentos para el Clasificador 1 y 18 experimentos para el Clasificador Auxiliar), surgen con la combinatoria de los siguientes tres conjuntos:

- Batch size={16, 32}
- Tasa de aprendizaje={0.00003, 0.00002, 0.00001}
- Optimizador={Adam, SGD con momento Nesterov(Con y sin “step decay”)}

Llamamos “step decay” al decremento de la tasa de aprendizaje en función de la época del entrenamiento en que nos situemos. En el step decay que se implementó, se especificó que cada 35 épocas la tasa de aprendizaje tome la mitad del valor de la época anterior. En la sección “A” del apéndice se detallan los 18 experimentos para el Clasificador 1 y en la sección “B” los 18 experimentos para el Clasificador Auxiliar. En dichas secciones se grafican la Exactitud, Binary cross entropy y tasa de aprendizaje, todas ellas, en función de las épocas.

A continuación se mostrarán los siguientes gráficos para cada clasificador (Clasificador 1 y Clasificador Auxiliar):

- Las curvas de la Exactitud en función de las épocas de las 18 configuraciones sobre el conjunto de datos de validación (A medida que se entrena cada modelo con su respectiva configuración).
- Las curvas de Precisión-Recall (PR) y ROC de los 18 modelos (configuraciones) entrenados aplicados en los conjuntos de datos de prueba ISIC y prueba HI.

Cada gráfico anterior contendrá 9 configuraciones, ya que las 18 configuraciones fueron divididas por batch size (mediante lotes de 16 y 32) para que las curvas sean más legibles.

Clasificador 1

Rendimiento de las 18 configuraciones durante el entrenamiento

En las figuras 4.3 y 4.4 se muestra la Exactitud del Clasificador 1 en el conjunto de validación durante el entrenamiento del mismo para cada una de sus 18 configuraciones. Estas configuraciones, para una mejor visualización de las curvas, fueron divididas por batch size (16 y 32). Si se requiere más detalles de cada uno de estos modelos, ver la sección “A” del apéndice donde se graficó cada curva en forma individual.

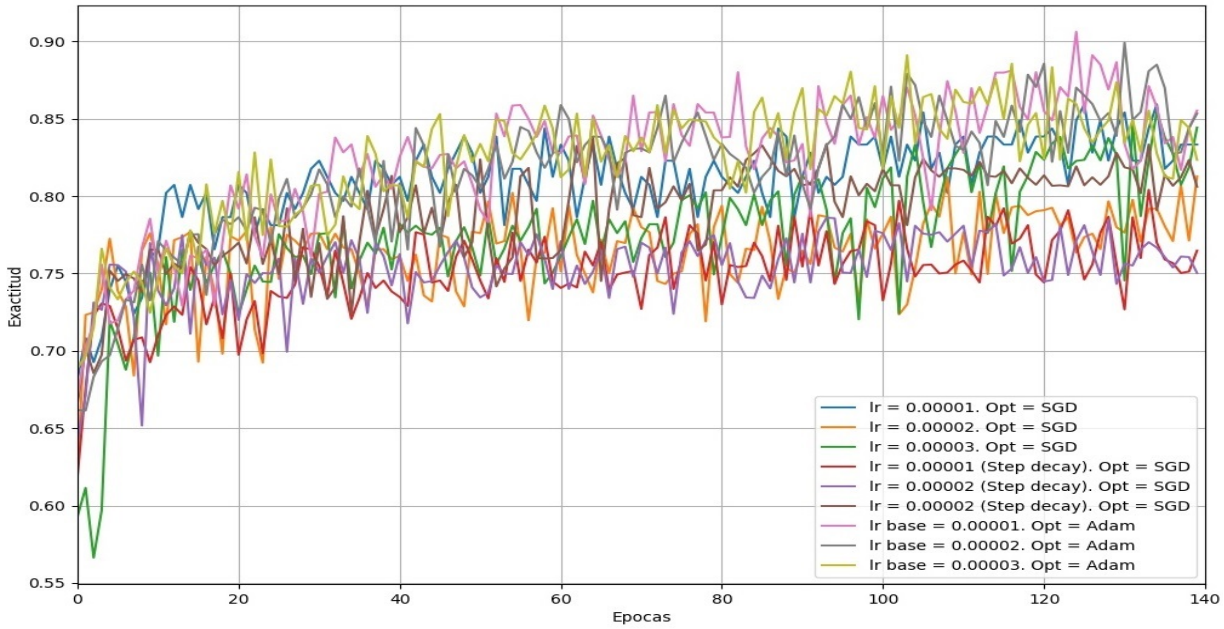


Figura 4.3: Exactitud del Clasificador 1, con sus diferentes configuraciones, en el conjunto de datos de validación con un batch size de 16.

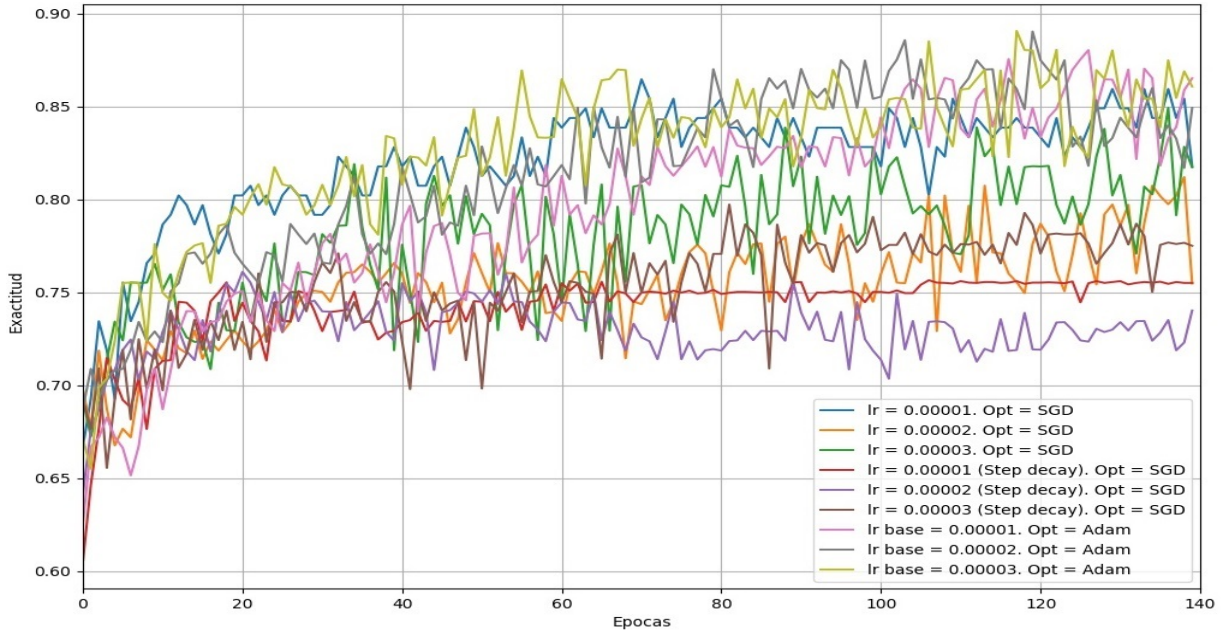


Figura 4.4: Exactitud del Clasificador 1, con sus diferentes configuraciones, en el conjunto de datos de validación con un batch size de 32.

En las figuras 4.3 y 4.4 observamos que los modelos creados a partir del Clasificador 1 con las configuraciones de optimizador Adam, batch size de 16 y 32, y tasas de aprendizaje de 0.00003 y 0.00001 producen los mejores resultados en el conjunto de datos de validación. No obstante, si se observan las curvas de la Binary cross entropy en función de las épocas de estos 4 modelos en la sección “A” del apéndice (ver experimentos 7,9,16 y 18), se visualiza que el modelo con tasa de aprendizaje igual a 0.00001 y batch size de 32 es el más estable. Además, este modelo cuenta con una velocidad razonable de decrecimiento de dicha función de pérdida (Binary cross entropy) a medida que transcurren las épocas y no se observa sobreajuste cuando se visualiza la Exactitud en las curvas de entrenamiento y validación.

Rendimiento de las 18 configuraciones en los conjuntos de: Prueba ISIC y HI

A continuación graficamos las curvas PR y ROC en los conjuntos de datos de prueba ISIC y HI con el objetivo de visualizar el rendimiento del Clasificador 1 con sus 18 configuraciones. Como se dijo anteriormente, por una cuestión de visualización, las 18 configuraciones fueron divididas por batch size (16 y 32) para que los gráficos sean más legibles.

Se aclara que las curvas PR, como se explicó en la sección 3.1, son complementarias a las curvas ROC. Por ende, el ROC-AUC es el indicador principal a evaluar en las distintas configuraciones del Clasificador 1. No obstante, el indicador PR-AUC o AP se tomará en consideración cuando se obtengan puntajes de ROC-AUC similares entre las diferentes configuraciones del Clasificador 1.

Conjunto de datos de Prueba ISIC

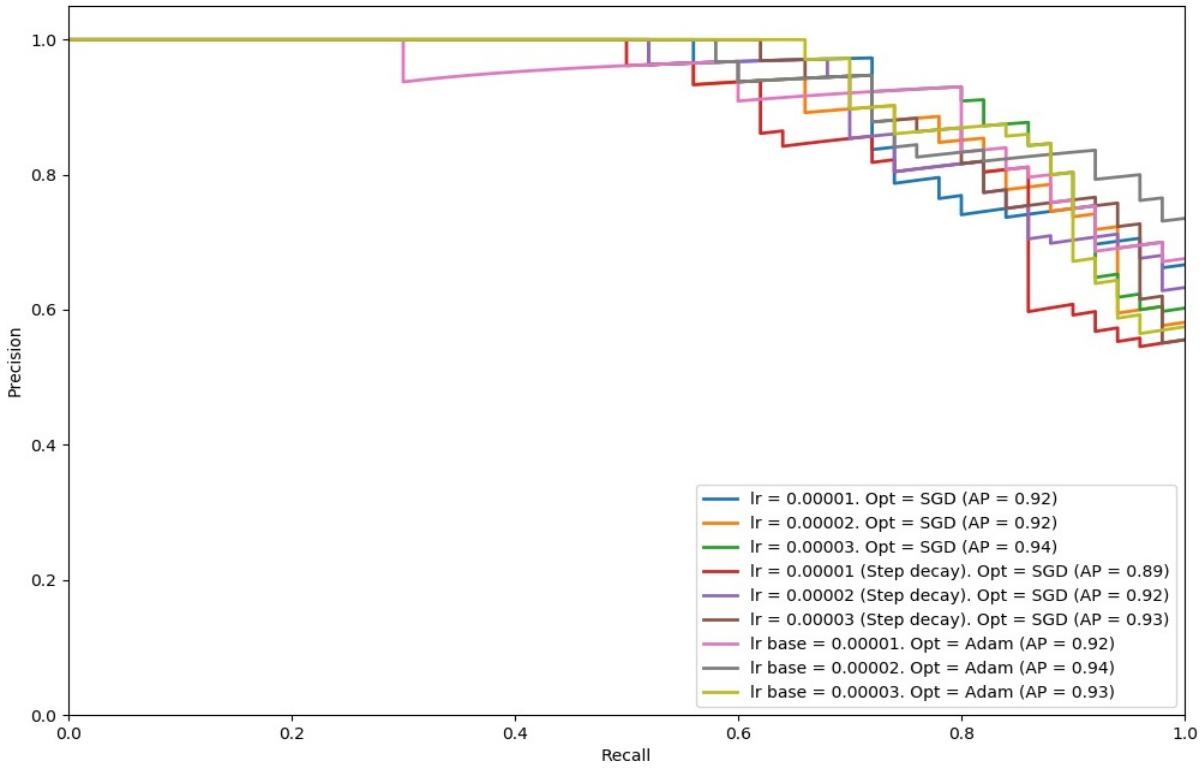


Figura 4.5: Curvas PR de las diferentes configuraciones del Clasificador 1, con un batch size de 16, en el conjunto de datos de Prueba ISIC.

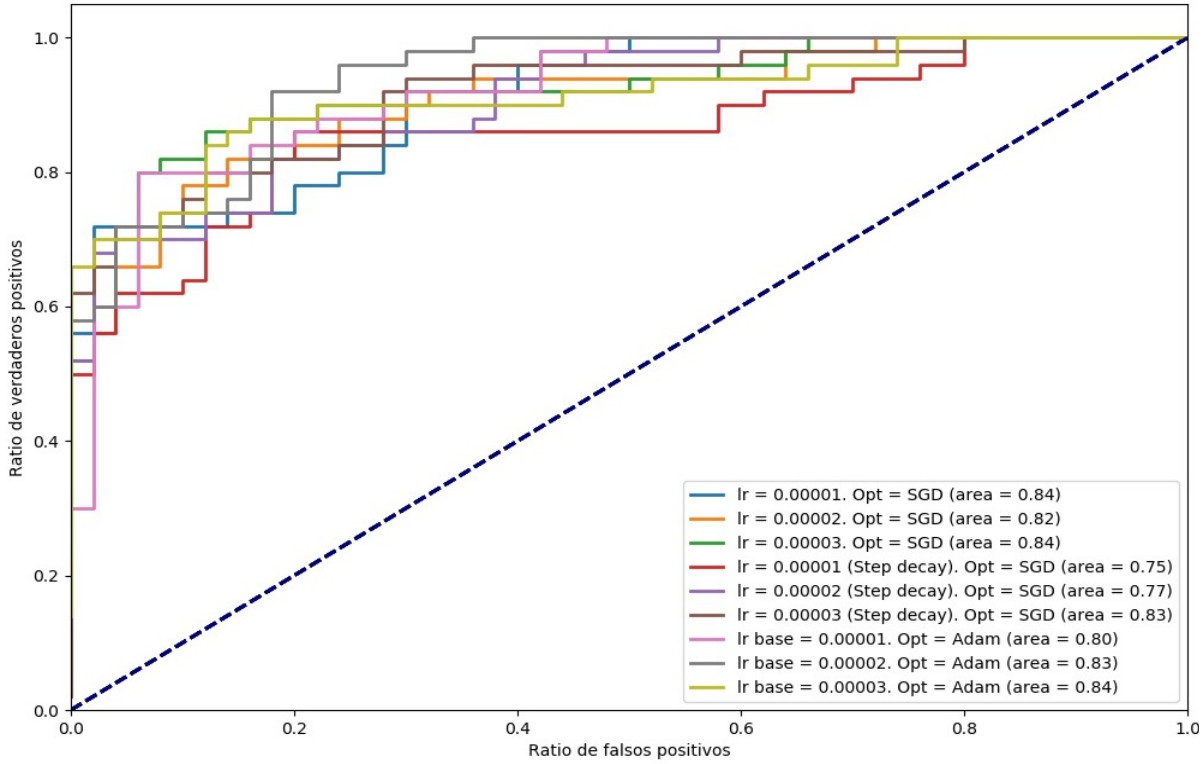


Figura 4.6: Curvas ROC de las diferentes configuraciones del Clasificador 1, con un batch size de 16, en el conjunto de datos de Prueba ISIC.

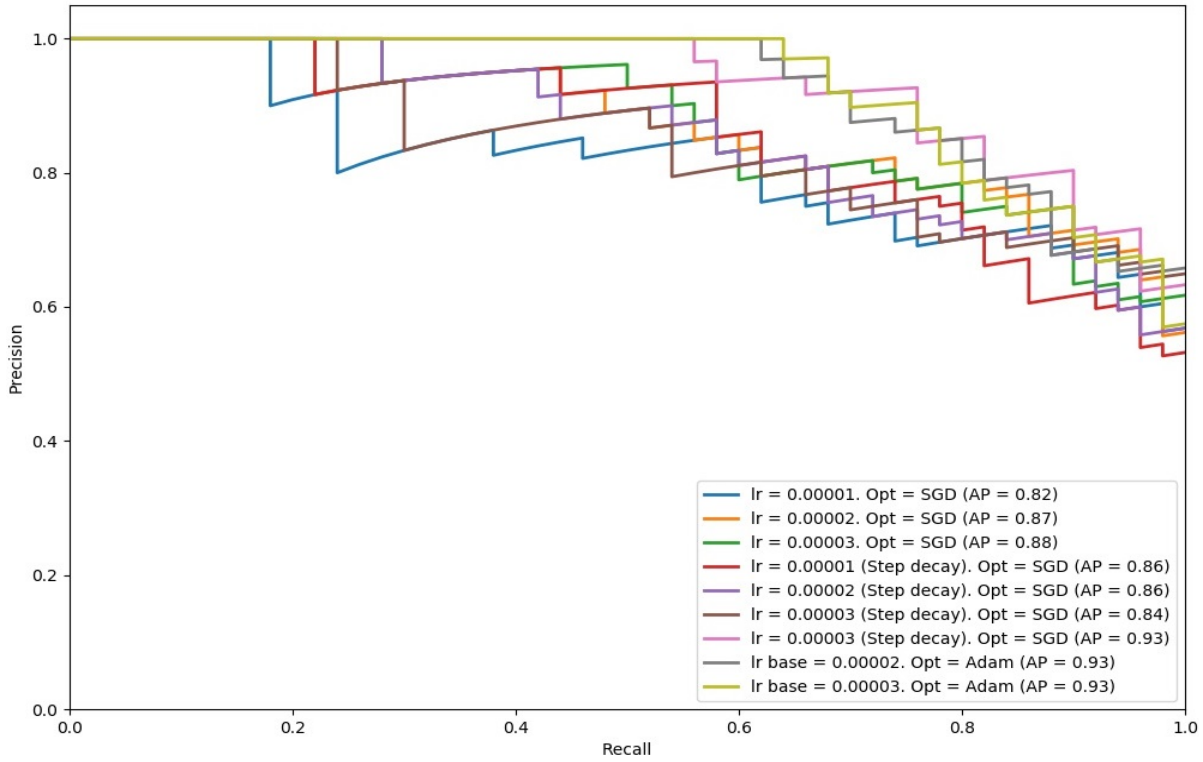


Figura 4.7: Curvas PR de las diferentes configuraciones del Clasificador 1, con un batch size de 32, en el conjunto de datos de Prueba ISIC.

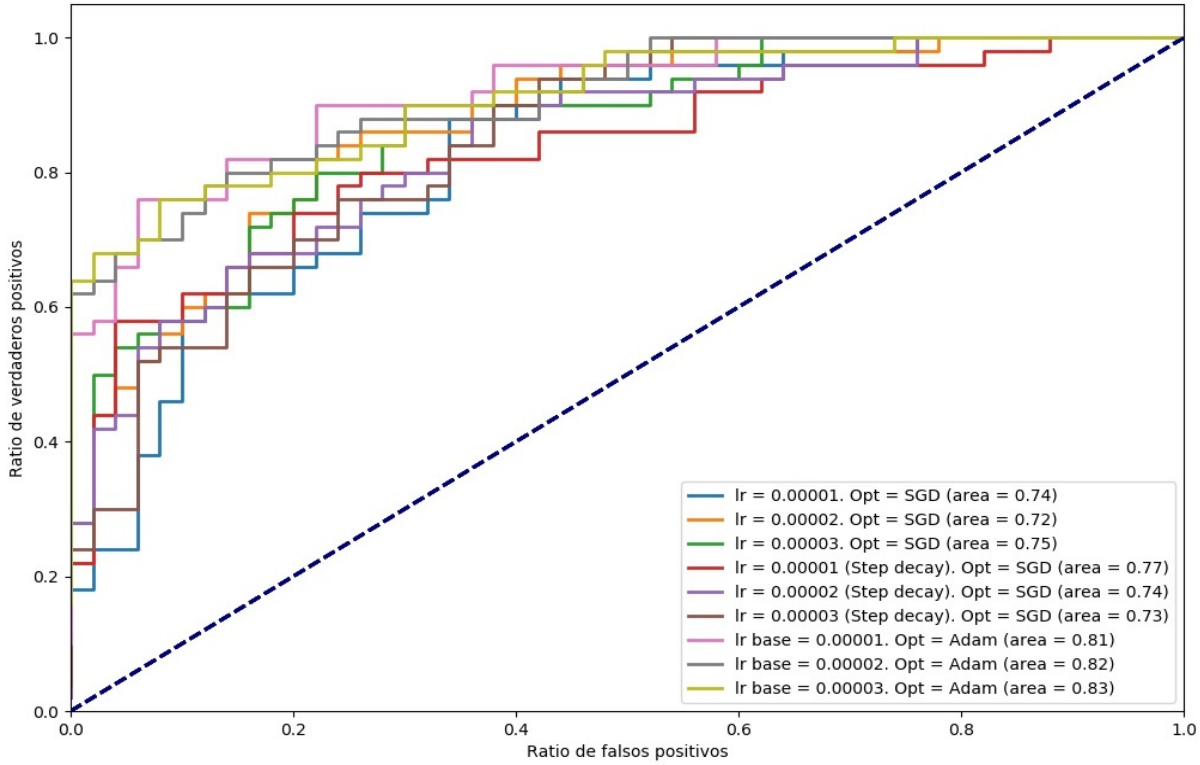


Figura 4.8: Curvas ROC de las diferentes configuraciones del Clasificador 1, con un batch size de 32, en el conjunto de datos de Prueba ISIC.

Conjunto de datos de Prueba HI

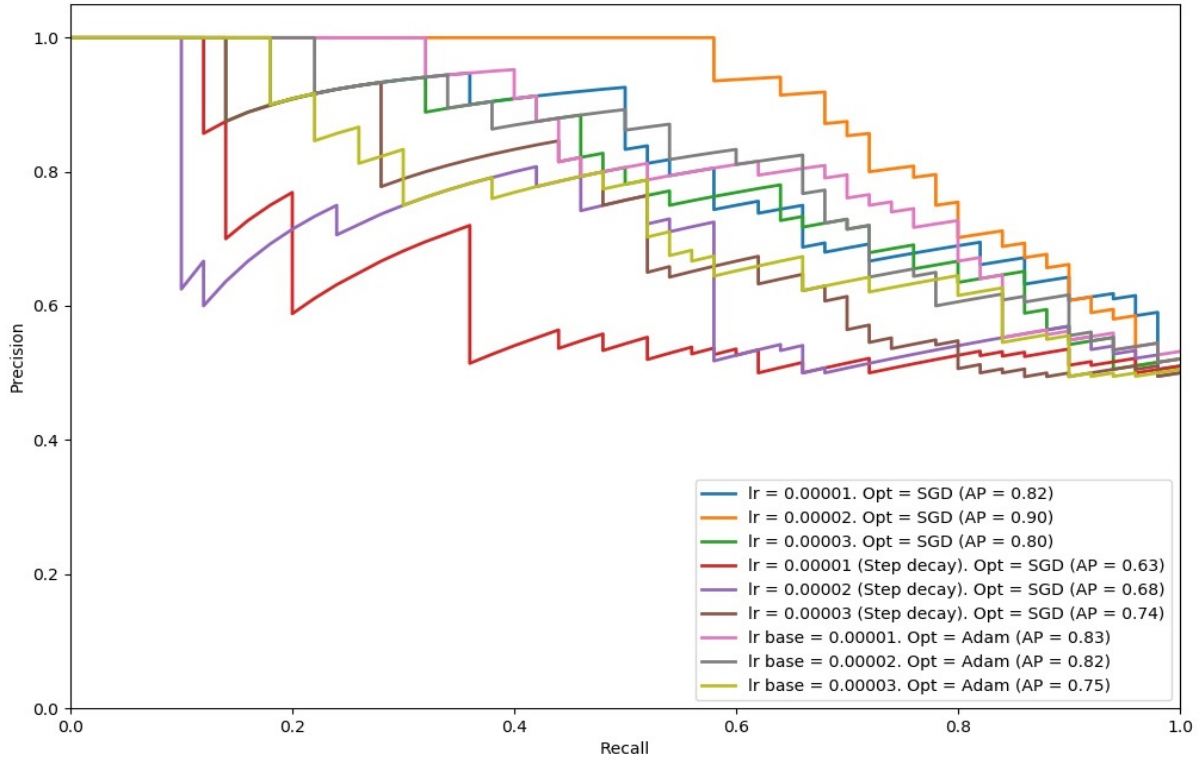


Figura 4.9: Curvas PR de las diferentes configuraciones del Clasificador 1, con un batch size de 16, en el conjunto de datos de Prueba HI.

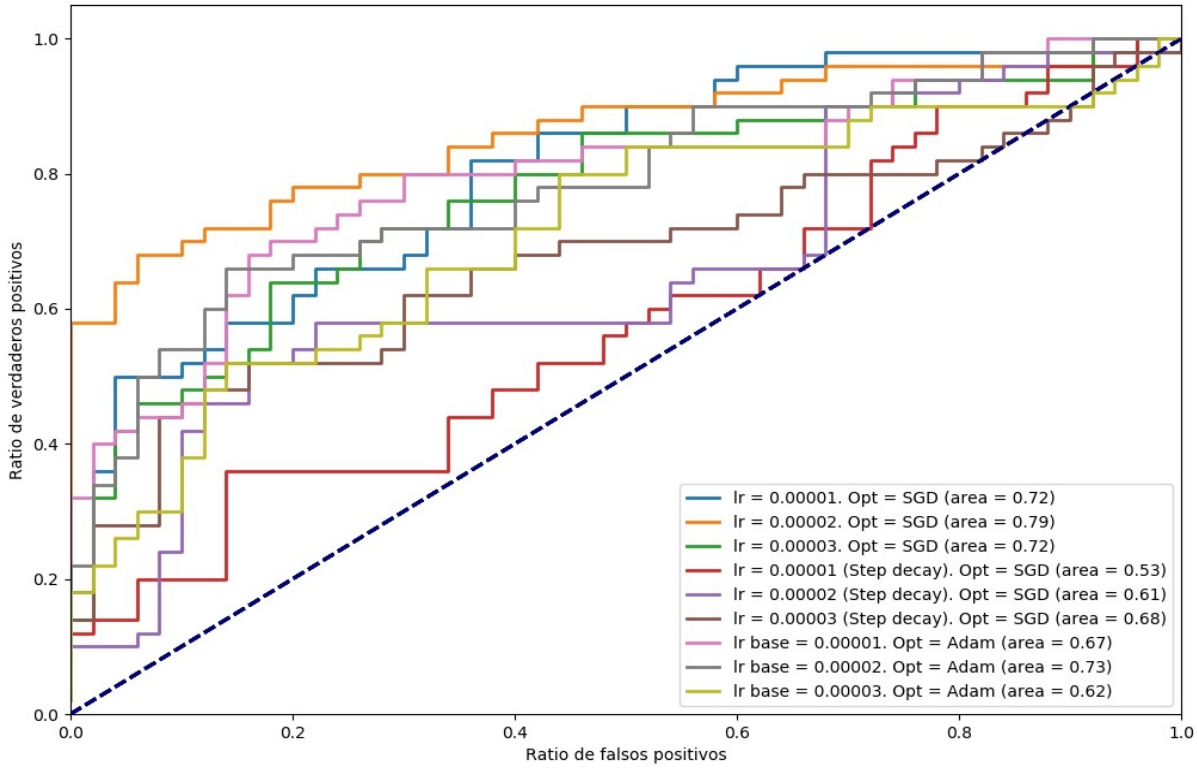


Figura 4.10: Curvas ROC de las diferentes configuraciones del Clasificador 1, con un batch size de 16, en el conjunto de datos de Prueba HI.

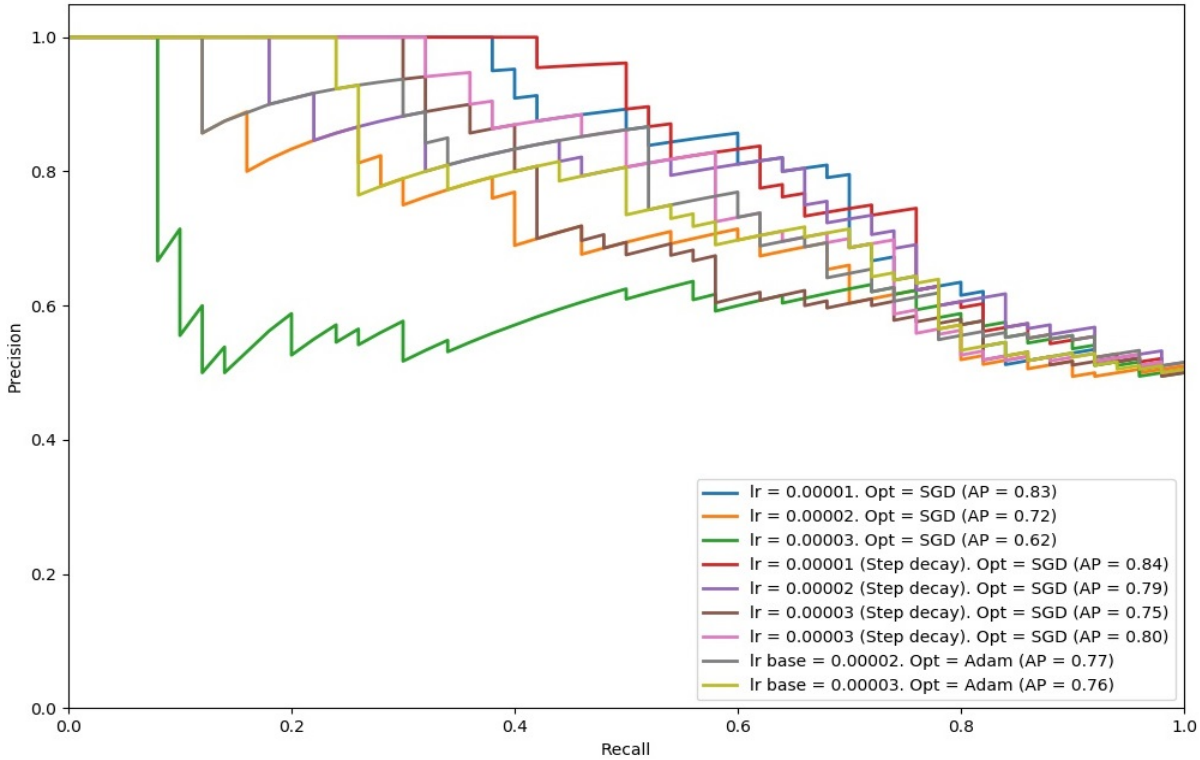


Figura 4.11: Curvas PR de las diferentes configuraciones del Clasificador 1, con un batch size de 32, en el conjunto de datos de Prueba HI.

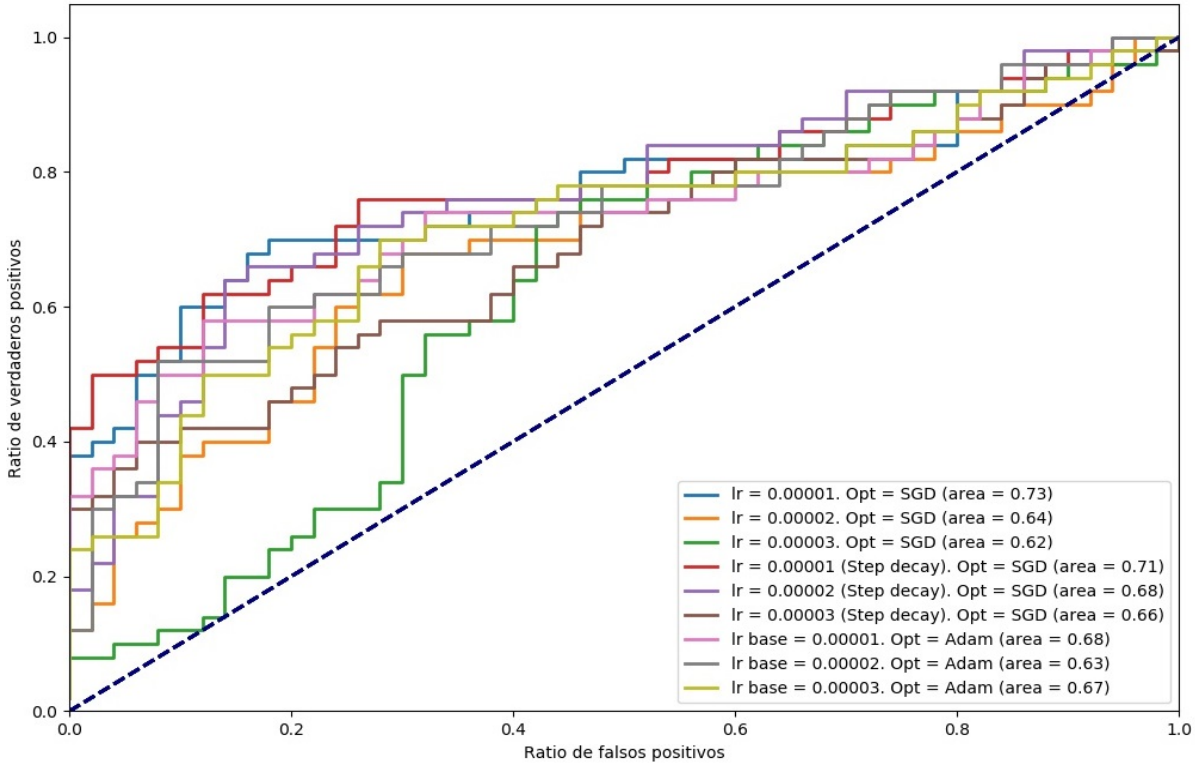


Figura 4.12: Curvas ROC de las diferentes configuraciones del Clasificador 1, con un batch size de 32, en el conjunto de datos de Prueba HI.

Observando las curvas ROC (y las curvas PR cuando se obtienen valores similares en la métrica ROC-AUC) de las figuras anteriores (4.5 a la 4.12), podemos mencionar lo siguiente:

- En el conjunto de datos de Prueba ISIC, el modelo que obtiene el mejor rendimiento es el que tiene un batch size de 16, un optimizador SGD con momento de Nesterov y una tasa de aprendizaje de 0.00003.

- En el conjunto de datos de Prueba HI, el modelo que obtiene el mejor rendimiento es el que tiene un batch size de 16, un optimizador SGD con momento Nesterov y una tasa de aprendizaje de 0.00002.

Sin embargo, tomando en consideración la Exactitud en el conjunto de datos de validación y los puntajes de PR-AUC y ROC-AUC en los conjuntos de datos de pruebas (ISIC y HI) en **forma simultánea**, la configuración del Clasificador 1 que obtiene el mejor rendimiento es la que tiene un optimizador SGD con momento Nesterov, una tasa de aprendizaje de 0.00001 y un batch size de 16. En esta configuración, que corresponde al experimento número 1 de la sección “A” del apéndice, se observa que el modelo tiene una función de pérdida (Binary cross entropy) que decrece a una velocidad aceptable, no tiene grandes variaciones (inestabilidad) entre las épocas y no se observa sobreajuste cuando se evalúa la Exactitud en los conjuntos de entrenamiento y validación. Con lo cual, confirmamos que la configuración del Clasificador 1 indicada en la tabla 4.2 es la óptima dentro de los modelos probados y expuestos en esta sección.

Clasificador auxiliar

Rendimiento de las 18 configuraciones durante el entrenamiento

En las figuras 4.13 y 4.14 se visualiza la Exactitud del Clasificador Auxiliar cuando es entrenado en sus 18 configuraciones y aplicada cada una de estas al conjunto de datos de validación. Por un tema de visualización de las curvas, las 18 configuraciones fueron divididas por batch size (16 y 32). Si se desea más detalles sobre una configuración, ver la sección “B” del apéndice donde cada curva se encuentra graficada en forma individual.

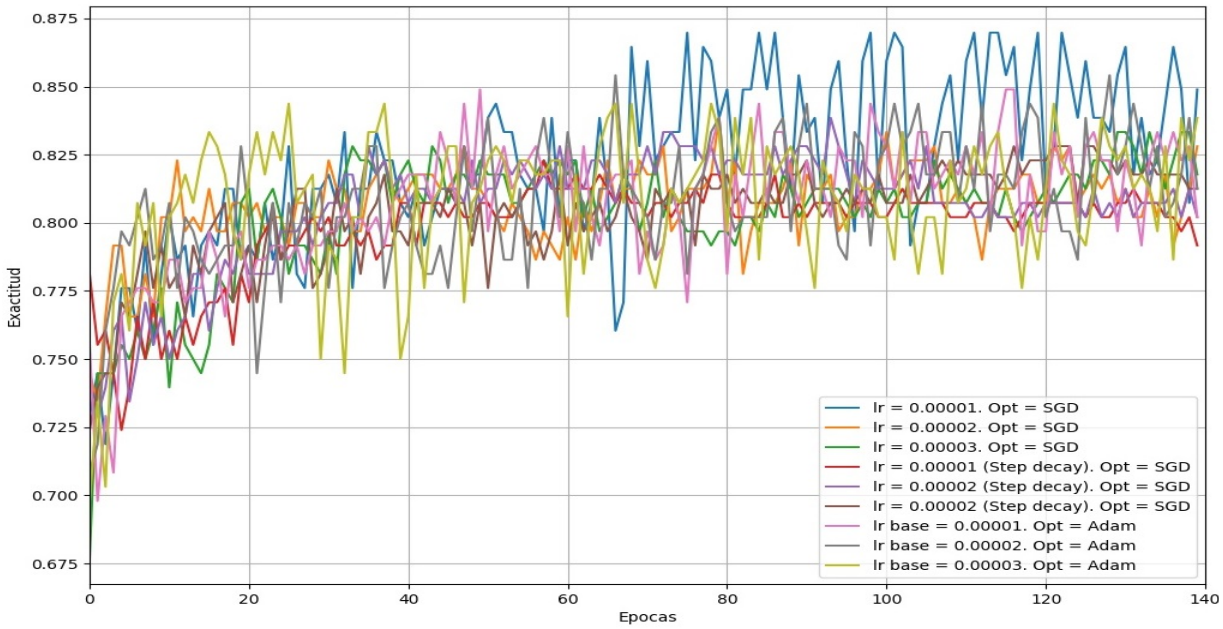


Figura 4.13: Exactitud del Clasificador Auxiliar, con sus diferentes configuraciones, en el conjunto de datos de validación con un batch size de 16.

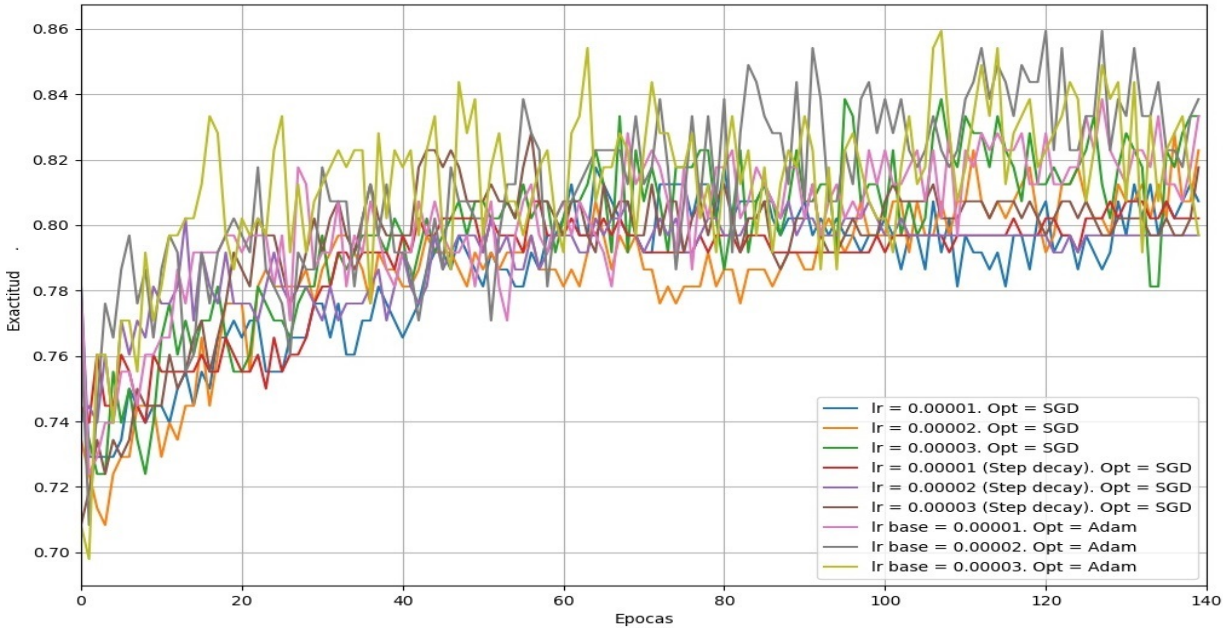


Figura 4.14: Exactitud del Clasificador Auxiliar, con sus diferentes configuraciones, en el conjunto de datos de validación con un batch size de 32.

En las figuras 4.13 y 4.14 se observa que los modelos tienen un rendimiento similar. No obstante, si visualizamos las curvas de la Binary cross entropy en función de las épocas en la sección “B” del apéndice, vemos que el modelo del experimento 11 es el que tiene la curva con menor ruido y una aceptable velocidad de decrecimiento. Además, en este modelo (compuesto por un optimizador SGD con momento Nesterov, un batch size de 32, una tasa de aprendizaje de 0.00001 y un step decay cada 35 épocas) no se observa sobreajuste al visualizar la Exactitud en las curvas de entrenamiento y validación.

Rendimiento de las 18 configuraciones en los conjuntos de: Prueba ISIC y HI

A continuación graficamos las curvas PR y ROC en los conjuntos de datos de Prueba ISIC y Prueba HI con el objetivo de visualizar el rendimiento de las configuraciones del Clasificador Auxiliar. Al igual que en el Clasificador 1, las 18 configuraciones fueron divididas por batch size (16 y 32) para que las curvas de los gráficos sean más comprensibles. Nuevamente se toma a la métrica ROC-AUC como indicador principal para evaluar las diferentes configuraciones y a la métrica PR-AUC (o AP) como indicador complementario. Es decir, el PR-AUC se tendrá en cuenta cuando se obtengan puntajes similares con la métrica ROC-AUC en las diferentes configuraciones del Clasificador Auxiliar.

Conjunto de datos de Prueba ISIC

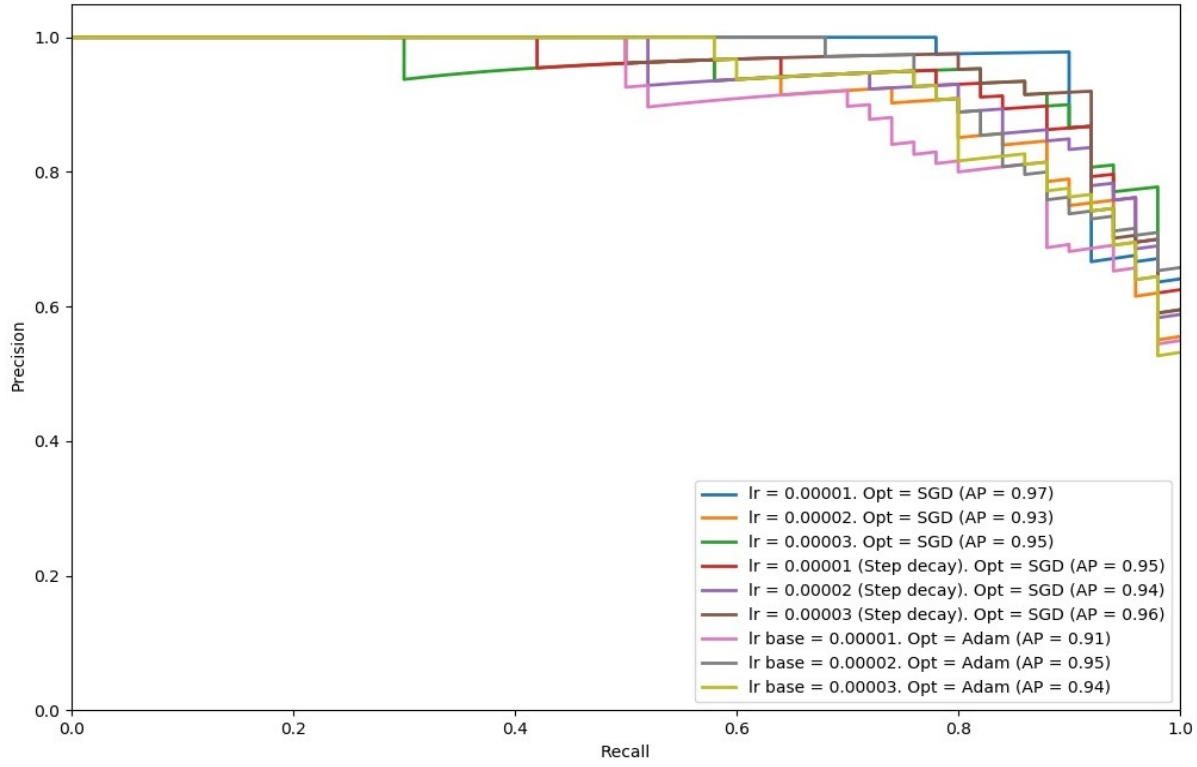


Figura 4.15: Curvas PR de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 16, en el conjunto de datos de Prueba ISIC.

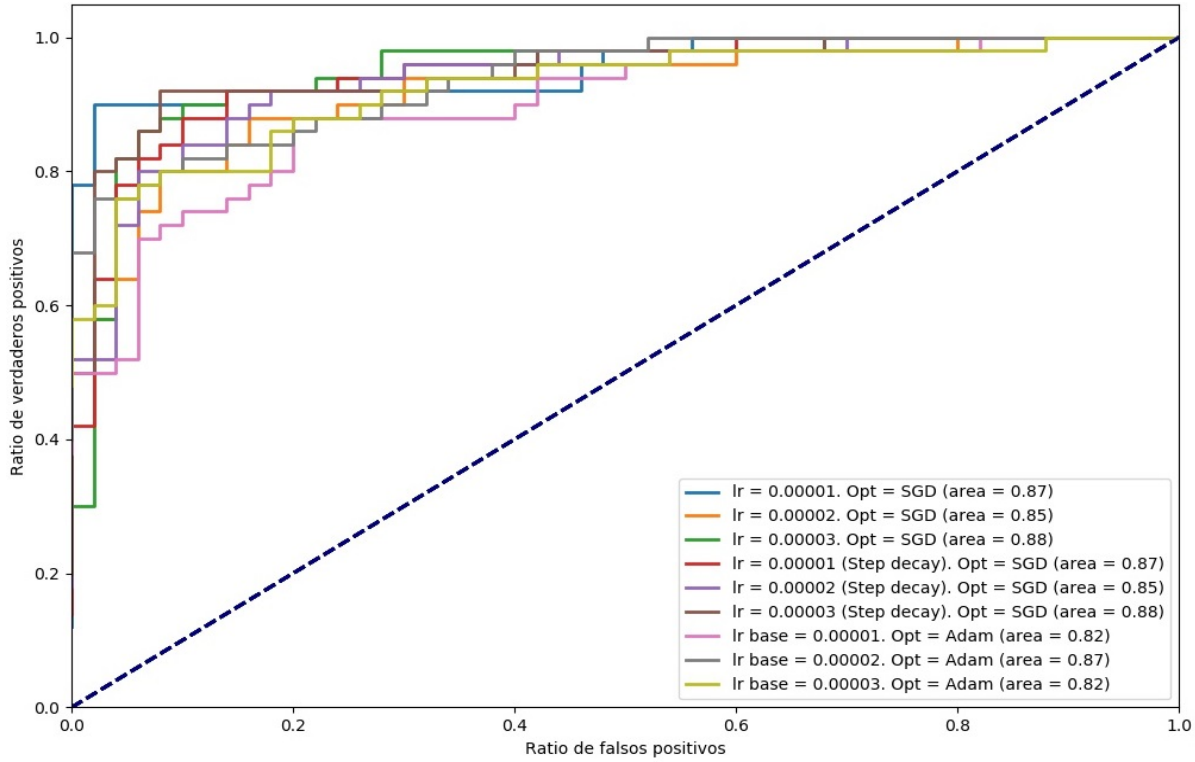


Figura 4.16: Curvas ROC de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 16, en el conjunto de datos de Prueba ISIC.

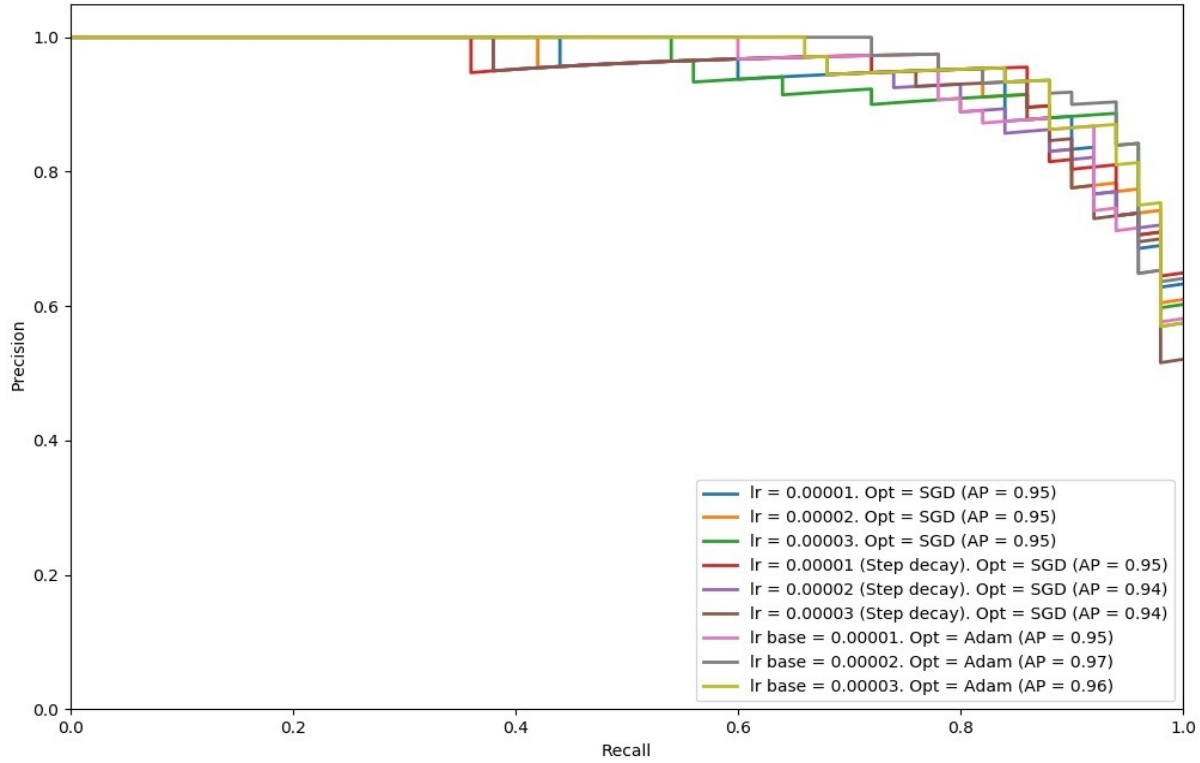


Figura 4.17: Curvas PR de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 32, en el conjunto de datos de Prueba ISIC.

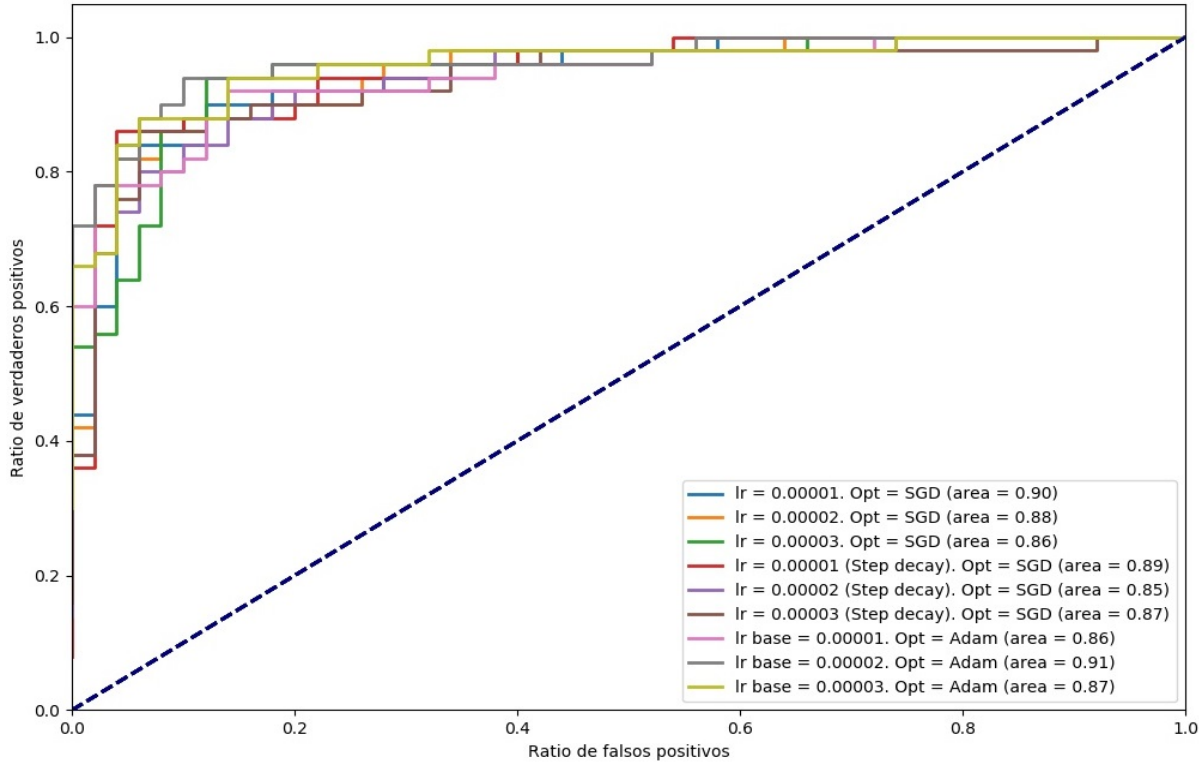


Figura 4.18: Curvas ROC de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 32, en el conjunto de datos de Prueba ISIC.

Conjunto de datos de Prueba HI

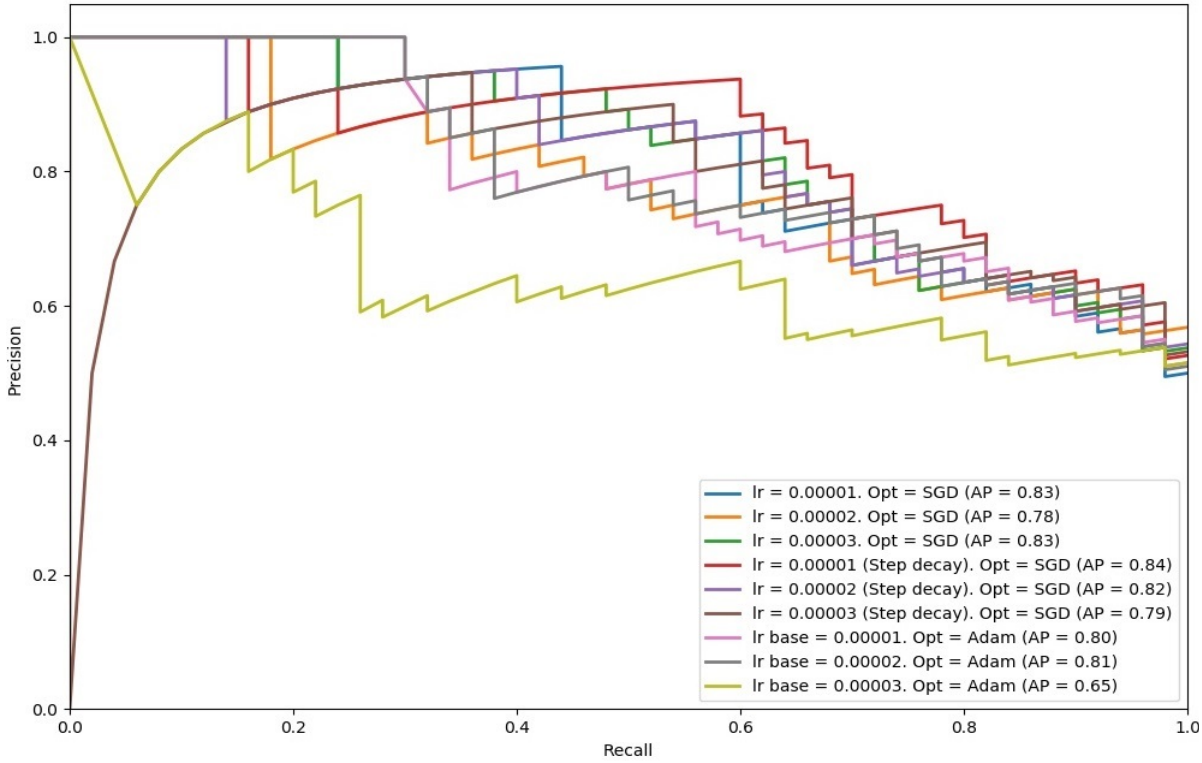


Figura 4.19: Curvas PR de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 16, en el conjunto de datos de Prueba HI.

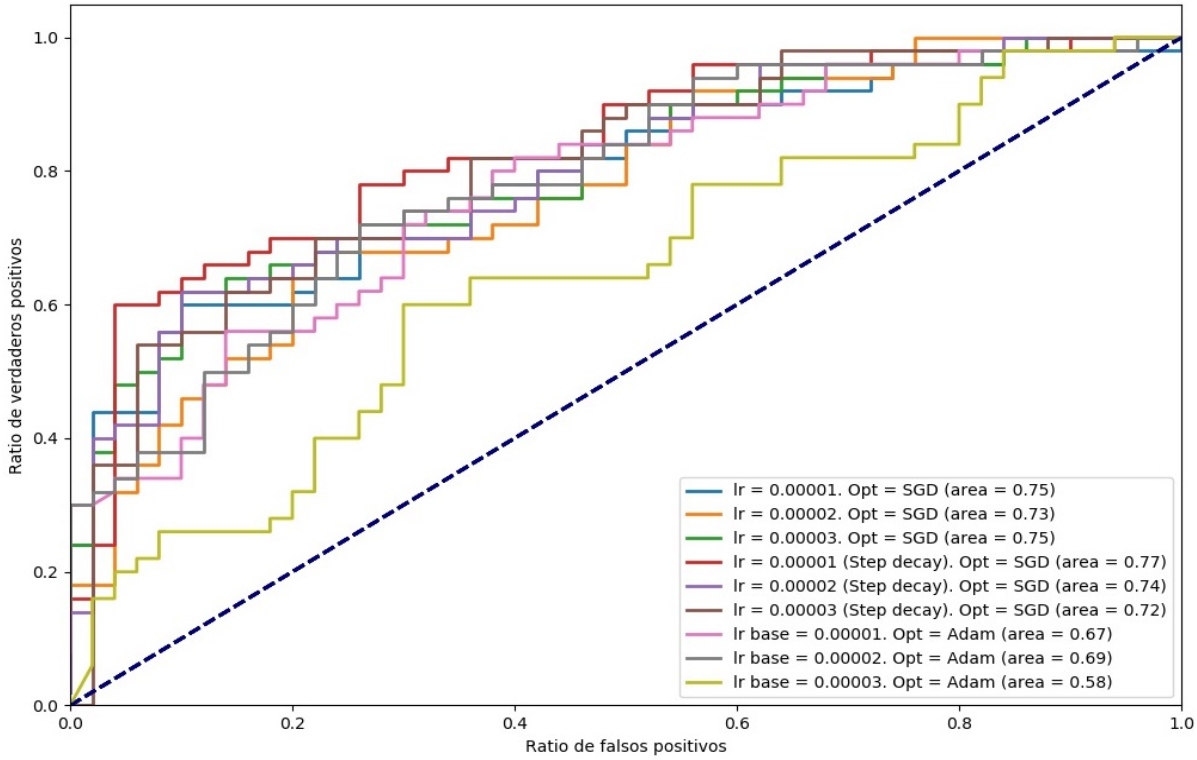


Figura 4.20: Curvas ROC de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 16, en el conjunto de datos de Prueba HI.

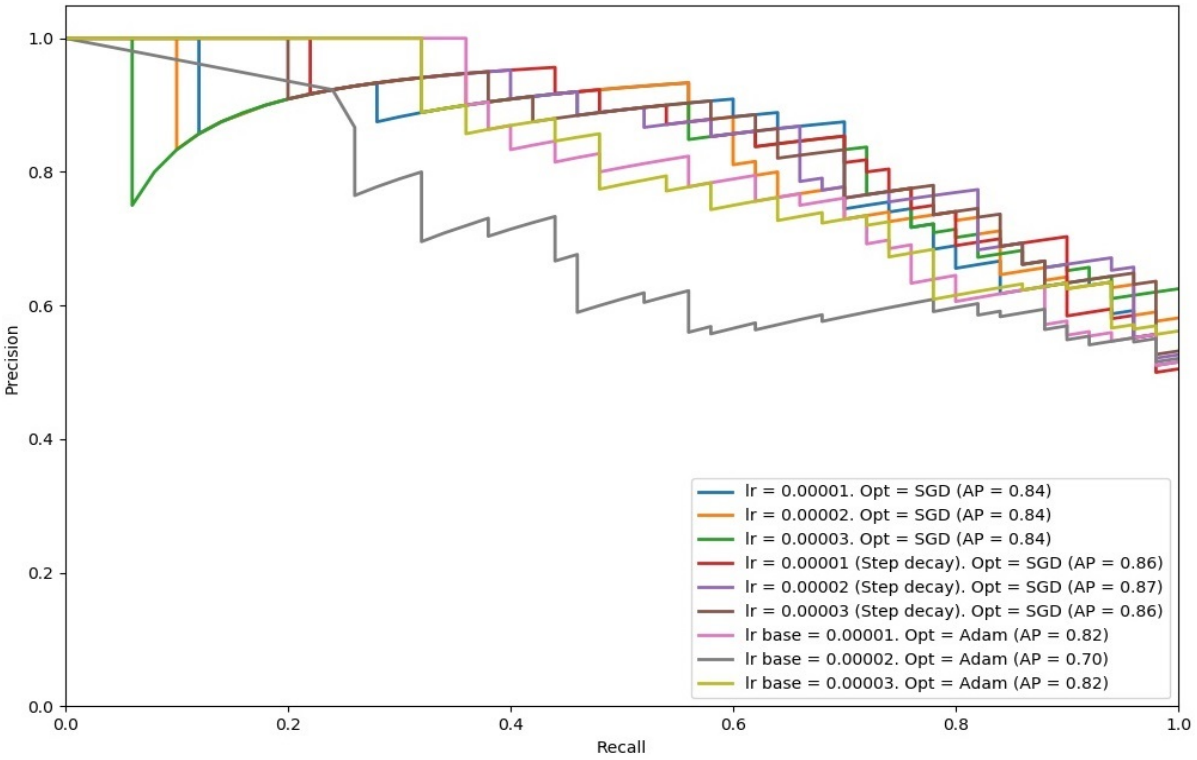


Figura 4.21: Curvas PR de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 32, en el conjunto de datos de Prueba HI.

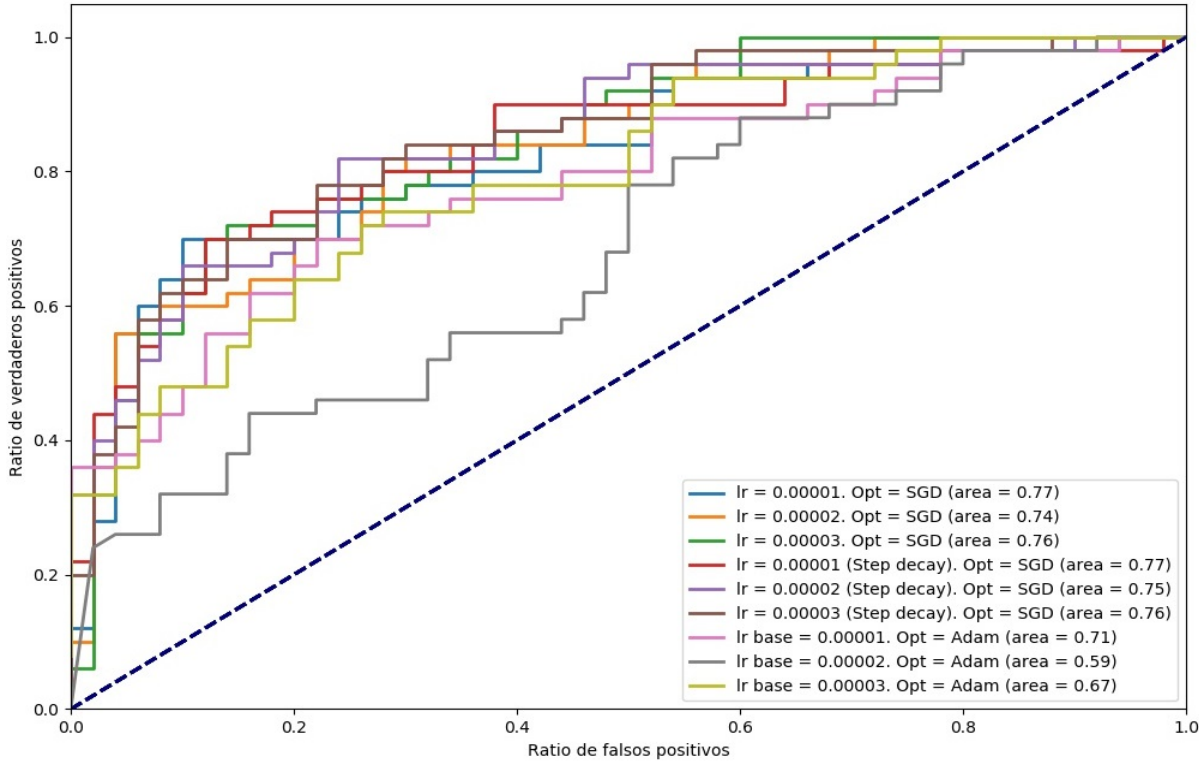


Figura 4.22: Curvas ROC de las diferentes configuraciones del Clasificador Auxiliar, con un batch size de 32, en el conjunto de datos de Prueba HI.

Observando las curvas ROC (y las curvas PR cuando se obtienen valores similares en la métrica ROC-AUC) de las figuras anteriores (4.15 al 4.22), podemos mencionar lo siguiente:

- En el conjunto de datos de Prueba ISIC, el modelo que obtiene el mejor rendimiento es el que tiene un batch size de 32, un optimizador Adam y una tasa de aprendizaje de 0.00002.

- En el conjunto de datos de Prueba HI, el modelo que obtiene el mejor rendimiento es el que tiene un batch size de 32, un optimizador SGD con momento Nesterov (con step decay) y una tasa de aprendizaje de 0.00001.

Analizando **conjuntamente** la Exactitud en el conjunto de datos de validación y los puntajes de PR-AUC y ROC-AUC en los conjuntos de datos de Prueba ISIC y Prueba HI, el mejor rendimiento se obtiene con la configuración del Clasificador Auxiliar compuesta por un optimizador SGD con momento Nesterov, una tasa de aprendizaje 0.00001 y un batch size de 16. Para esta configuración (ver experimento número 1 de la sección “B” del apéndice), el modelo tiene una función de pérdida (Binary cross entropy) que decrece a una velocidad aceptable, no se visualiza ruido entre las épocas y no se observa sobreajuste cuando se evalúa la Exactitud en las curvas de entrenamiento y validación. Por ende, verificamos que los hiperparámetros indicados en la tabla 4.2 son los óptimos teniendo en cuenta los 18 modelos que fueron probados y expuestos en esta sección.

4.2 Resultados en los conjuntos de datos de prueba

Una vez seleccionados los hiperparámetros, en este subsección se analizarán los resultados obtenidos del Clasificador 1, del Clasificador 2 y del Ensamble de la siguiente manera:

- Se analizará el rendimiento de cada modelo (Clasificador 1, Clasificador 2 y el Ensamble) en los distintos conjuntos de datos de prueba (ISIC, HI, ROT y XY). Como se explicó en la sección 3.1, el rendimiento será evaluado mediante la métrica ROC-AUC y el indicador PR-AUC. Además se calcularán algunas métricas complementarias (explicadas en la sección 3.1) derivadas de la matriz de confusión.
- Se comparará, en forma conjunta, el rendimiento de los modelos mencionados en los conjuntos de datos de prueba (ISIC, HI, ROT y XY). Para evaluar el rendimiento, además de utilizar las métricas mencionadas en el punto anterior, se calcularán puntos de Especificidad para una Sensibilidad determinada.
- Se comparará el rendimiento obtenido de los mencionados modelos con el de un modelo externo (que utiliza redes convolucionales y vectores de Fisher). El objetivo es verificar que los resultados hallados se encuentran “en el orden” en relación a los que se obtienen con un modelo externo (cuya arquitectura es similar a los Modelos Híbridos que se utilizaron en el presente trabajo).

4.2.1 Rendimiento de los modelos en los conjuntos de datos de prueba

Ensamble

En la figura 4.23 se observan las curvas ROC del Ensamble en los distintos conjuntos de datos de prueba (ISIC, HI, ROT y XY). El mayor rendimiento se obtiene en el conjunto de Prueba ISIC cuya ROC-AUC es de 0.9106. En los conjuntos de datos de prueba sintéticos (ROT y XY) se visualiza una baja de rendimiento (en relación al obtenido en Prueba ISIC). Además, se visualiza que en el conjunto de datos de Prueba ROT, el rendimiento en base a la métrica ROC-AUC se incrementa en aproximadamente 10 puntos en relación a la obtenida en el conjunto de datos de Prueba XY (0.8452 versus 0.7517). Este incremento puede deberse que la rotación fue una operación incluida en el data augmentation del conjunto de datos de entrenamiento pero no se incluyó la traslación sobre los ejes X e Y. Por último, observamos que el ROC-AUC en el conjunto de datos de Prueba HI es de 0.8238. La resolución de las imágenes incluidas en Prueba HI es inferior a las imágenes en Prueba ISIC, con lo cual, esto puede ser un motivo de la diferencia de performance (0.9106 versus 0.8238) cuando se aplica el Ensamble en estos 2 conjuntos de datos de prueba.

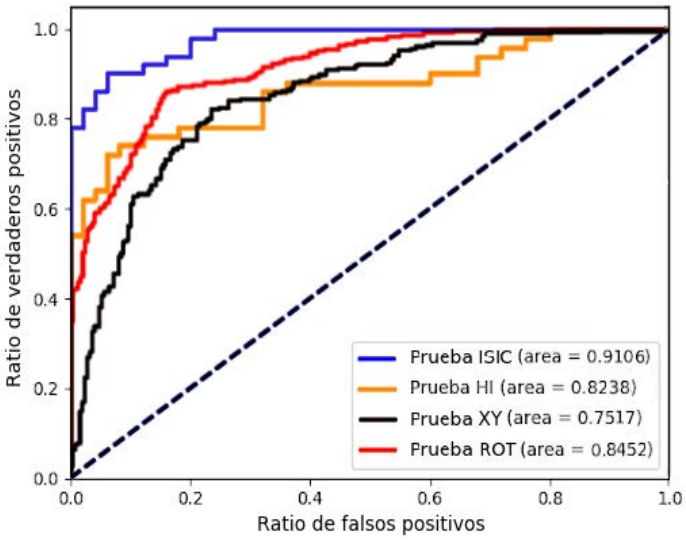


Figura 4.23: Curvas ROC del Ensamble en los distintos conjuntos de datos de Prueba.

Luego, como métrica de soporte al ROC-AUC graficamos las curvas PR del Ensamble en los distintos conjuntos de datos de prueba con el objetivo de obtener el PR-AUC o AP. Esto puede visualizarse en la figura 4.24. Si se ordena el rendimiento por el AP obtenido en cada conjunto de datos de prueba, obtenemos el mismo orden que si lo ordenamos por el ROC-AUC. Es decir, el mejor rendimiento se obtiene en el conjunto de datos de Prueba ISIC, luego en Prueba ROT seguido por Prueba HI y por último en Prueba XY. Siendo consistente el empleo de ambas métricas (ROC-AUC y PR-AUC) para evaluar el rendimiento de este clasificador.

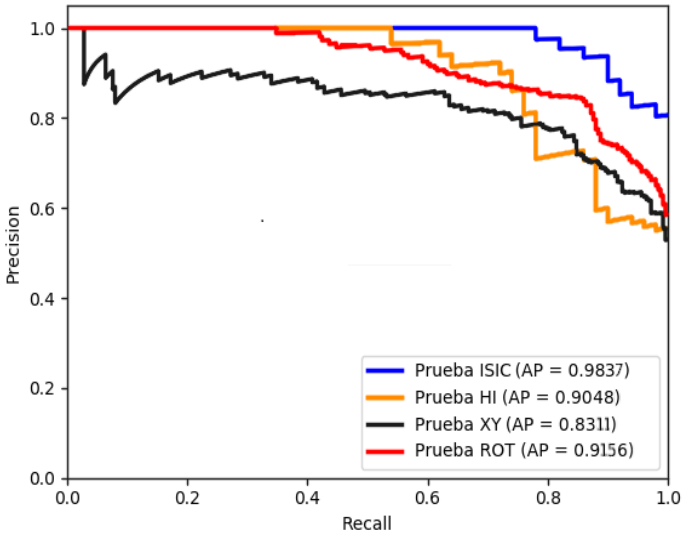


Figura 4.24: Curvas PR del Ensamble cuando es aplicado en los distintos conjuntos de datos de Prueba.

Por último, en la tabla 4.3 observamos otras métricas complementarias al ROC-AUC. Las mismas fueron obtenidas con un punto de corte de 0.5 y se calculan según lo explicado en la sección 3.1. Estás métricas complementarias son menos informativas que el ROC-AUC o la PR-AUC ya que dependen fuertemente del punto de corte. No obstante, se observa que la métrica F1-Score, que es el promedio armónico entre la Precisión y el Recall, tiene su valor máximo (0.9120) cuando se aplica el Ensamble al conjunto de datos de Prueba ISIC, siendo este mismo conjunto de datos en el cual se obtiene la máxima ROC-AUC.

Conjunto de datos	Lesión pigmentada	Precision	Recall	F1-Score	Soporte	Average Precision (AP)	ROC-AUC
Prueba ISIC	Melanoma	0.8664	0.9627	**0.9120**	50	**0.9837**	**0.9106**
	No Melanoma	0.9648	0.8610	**0.9099**	50		
Prueba HI	Melanoma	0.8641	0.7590	0.8081	50	0.9048	0.8238
	No Melanoma	0.7872	0.7816	0.7843	50		
Prueba ROT	Melanoma	0.8539	0.8365	0.8451	250	0.9156	0.8452
	No Melanoma	0.8414	0.8619	0.8515	250		
Prueba XY	Melanoma	0.8508	0.6012	0.7045	250	0.8311	0.7517
	No Melanoma	0.6943	0.8870	0.7789	250		

Tabla 4.3:Métricas correspondientes al Ensamble cuando es aplicado en los distintos conjuntos de datos de prueba.

Clasificador 1 (Modelo CNN)

En la figura 4.25 se observa el rendimiento (en base a las curvas ROC) del Clasificador 1 en los distintos conjuntos de prueba. Si ordenamos dicho rendimiento de acuerdo al puntaje de la métrica ROC-AUC, se obtiene el mismo orden que el hallado en el Ensamble (evaluando esta misma métrica, ver figura 4.23). Esto último confirma lo dicho previamente, donde la causa de este orden probablemente se encuentre en que la calidad de las imágenes del conjunto de datos de Prueba HI es inferior al de Prueba ISIC y que la operación de traslación no fue incluida en el data augmentation durante el entrenamiento (mientras que la de rotación si se incluyó).

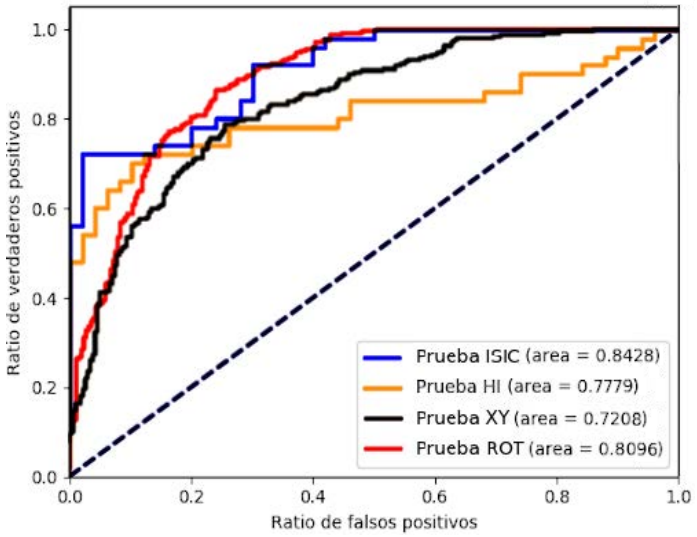


Figura 4.25: Curvas ROC del Clasificador 1 (Modelo CNN) en los conjuntos de datos de prueba.

Nuevamente, graficamos las curvas PR y calculamos la métrica AP como complemento de la curva ROC y la métrica ROC-AUC. Esto último puede observarse en la figura 4.26. En esta figura se visualiza que si se ordena el rendimiento de este clasificador en base al AP en los distintos conjuntos de datos de prueba, se obtiene el mismo ordenamiento que si utilizamos el ROC-AUC (ver figura 4.25). Ratificando una vez más la influencia de la baja resolución de las imágenes en el conjunto de datos de Prueba HI y de la aplicación del data augmentation (incluyendo la operación de rotación de las imágenes) en la performance de este clasificador.

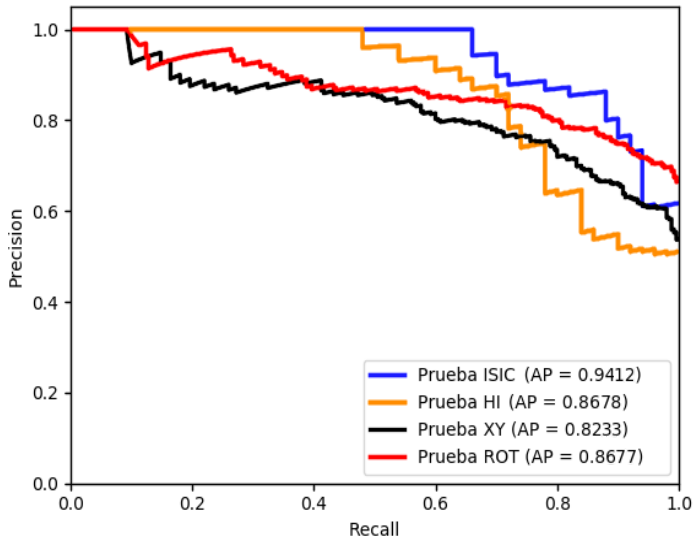


Figura 4.26: Curvas PR del Clasificador 1 (Modelo CNN) cuando es aplicado en los distintos conjuntos de datos de prueba.

Por último, en la tabla 4.4 observamos las métricas complementarias (tomando un punto de corte de 0.5) del ROC-AUC. Al igual que en el Ensamble, la métrica F1-Score en este clasificador toma su valor máximo en el conjunto de datos de Prueba ISIC, siendo este conjunto de datos donde se obtiene la máxima ROC-AUC.

Conjunto de datos	Lesión pigmentada	Precision	Recall	F1-Score	Soporte	Average Precision (AP)	ROC-AUC
Prueba ISIC	Melanoma	0.9688	0.7044	**0.8157**	50	**0.9412**	**0.8428**
	No Melanoma	0.7734	0.9703	**0.8607**	50		
Prueba HI	Melanoma	0.9418	0.6020	0.7345	50	0.8678	0.7779
	No Melanoma	0.7095	0.9633	0.8171	50		
Prueba ROT	Melanoma	0.7826	0.8472	0.8137	250	0.8677	0.8096
	No Melanoma	0.8395	0.7610	0.7983	250		
Prueba XY	Melanoma	0.8159	0.5623	0.6658	250	0.8233	0.7208
	No Melanoma	0.6721	0.8792	0.7618	250		

Tabla 4.4: Métricas correspondientes al Clasificador 1 (Modelo CNN) cuando es aplicado en los distintos conjuntos de datos de prueba.

Clasificador 2 (Modelo Híbrido sin segmentación)

En la figura 4.27 se observan las curvas ROC del Clasificador 2 en los distintos conjuntos de datos de prueba. Nuevamente, si ordenamos el rendimiento de este clasificador en base a la métrica ROC-AUC, obtenemos el mismo orden que el hallado en el Clasificador 1 y en el Ensamble (también evaluados con la métrica ROC-AUC). Ratificando la conclusión expuesta anteriormente, donde la resolución de las imágenes y las operaciones que se incluyen en el data augmentation tienen una relación directa en la performance del clasificador.

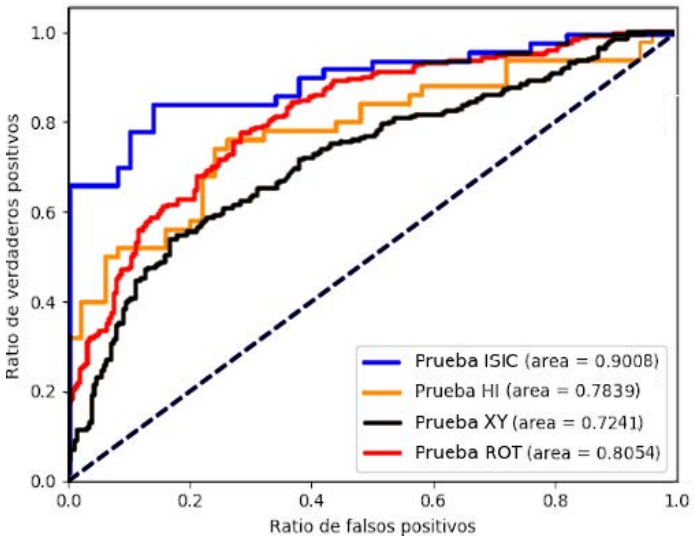


Figura 4.27: Curvas ROC del Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) en los distintos conjuntos de datos de prueba.

Tal cual lo realizado en el Ensamble y en el Clasificador 1, se grafican las curvas PR (con su correspondiente métrica AP) en cada conjunto de datos de prueba (ver figura 4.28). Si bien el AP que se obtiene en el conjunto de datos de Prueba XY es mayor al que se obtiene en Prueba HI (0.8280 versus 0.8231), diremos que el Clasificador 2 tiene un mejor rendimiento en el conjunto de Prueba HI que en Prueba XY dado que, tal cual establecimos en la sección 3.1, la métrica primaria es el ROC-AUC y la métrica AP es complementaria. Utilizando las métricas complementarias sólo cuando se obtienen puntajes del ROC-AUC similares y este no es el caso ya que los puntajes de la métrica ROC-AUC son de 0.7839 y 0.7241 para los conjuntos de datos de Prueba HI y Prueba XY respectivamente.

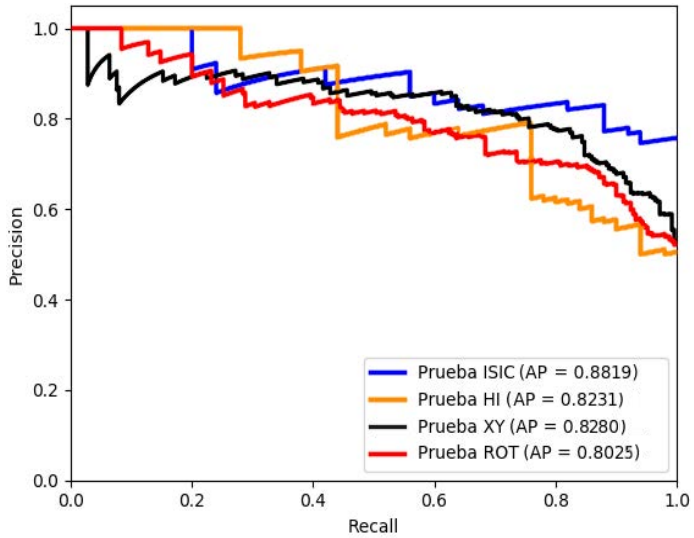


Figura 4.28: Curvas PR del Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) cuando es aplicado en los distintos conjuntos de datos de prueba.

Luego, en la tabla 4.5 se observa que la métrica F1-Score, complementaria del ROC-AUC, toma su valor máximo en el conjunto de datos de Prueba ISIC. Siendo este conjunto de datos donde se obtiene la máxima ROC-AUC (0.9008).

Conjunto de datos	Lesión pigmentada	Precision	Recall	F1-Score	Soporte	Average Precision (AP)	ROC-AUC
Prueba ISIC	Melanoma	0.9246	0.6771	**0.7817**	50	**0.8819**	**0.9008**
	No Melanoma	0.7518	0.9406	**0.8357**	50		
Prueba HI	Melanoma	0.7662	0.7384	0.7520	50	0.8231	0.7839
	No Melanoma	0.7461	0.7632	0.7545	50		
Prueba ROT	Melanoma	0.7711	0.6691	0.7165	250	0.8025	0.8054
	No Melanoma	0.7097	0.8028	0.7534	250		
Prueba XY	Melanoma	0.6704	0.6346	0.6502	250	0.8280	0.7241
	No Melanoma	0.6536	0.6979	0.6750	250		

Tabla 4.5: Métricas correspondientes al Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) cuando es aplicado en los distintos conjuntos de datos de prueba.

4.2.2 Comparación de los modelos en cada conjunto de datos de prueba

Conjunto de datos de Prueba ISIC

En la tabla 4.6 se observa que el Ensamble es el clasificador que tiene el mejor rendimiento en este conjunto de datos, con un ROC-AUC de 0.9106. Además se verifica que en el Ensamble, los valores de los 3 puntos de Especificidad para una Sensibilidad determinada, el AP y el F1-Score son superiores a los obtenidos por el Clasificador 1 y el Clasificador 2. Esto último confirma la consistencia entre las métricas complementarias y primarias.

Modelos	Threshold 0.5				Especif. con sensib. al 82%	Especif. con sensib. al 89%	Especif. con sensib. al 95%	Average Precision (AP)	ROC-AUC
	Precision	F1-Score	Especificidad	Sensibilidad					
Clasificador 1 (CNN)	0.9688	0.8157	0.8692	0.7044	0.7313	0.6846	0.5722	0.9412	0.8428
Clasificador 2 (Híbrido)	0.9246	0.7817	0.9306	0.6771	0.7478	0.6018	0.2811	0.8819	0.9008
Ensamble	0.8664	0.9120	0.8146	0.9627	0.9744	0.8733	0.8195	0.9837	**0.9106**

Tabla 4.6: Métricas correspondientes a los distintos clasificadores cuando son aplicados al conjunto de datos de Prueba ISIC.

Conjunto de datos de Prueba HI

En la tabla 4.7 se observa que la máxima ROC-AUC (0.8238) se obtiene, nuevamente, con el Ensamble. Además se vuelve a ratificar que las métricas complementarias “acompañan” al máximo valor del ROC-AUC, ya que con el Ensamble se obtienen máximos (en relación al Clasificador 1 y el Clasificador 2) en los valores de los 3 puntos de Especificidad para una Sensibilidad determinada, en el AP y en el F1-Score.

Modelos	Threshold 0.5				Especif. con sensib. al 82%	Especif. con sensib. al 89%	Especif. con sensib. al 95%	Average Precision (AP)	ROC-AUC
	Precision	F1-Score	Especificidad	Sensibilidad					
Clasificador 1 (CNN)	0.9418	0.7345	0.9516	0.6020	0.5323	0.2284	0.0941	0.8678	0.7779
Clasificador 2 (Híbrido)	0.7662	0.7520	0.7336	0.7384	0.5317	0.2879	0.0837	0.8231	0.7839
Ensamble	0.8641	0.8081	0.8772	0.7590	0.6925	0.3531	0.2698	0.9048	**0.8238**

Tabla 4.7: Métricas correspondientes a los distintos clasificadores cuando son aplicados al conjunto de datos de Prueba HI.

Conjunto de datos de Prueba ROT

En la tabla 4.8 se observa que el Ensamble resulta ser el mejor clasificador en este conjunto de datos, ya que obtiene la máxima ROC-AUC (0.8452). Además, las métricas complementarias también obtienen sus máximos en el Ensamble, reforzando el criterio de selección de dicho clasificador utilizando solo la métrica ROC-AUC.

Por otro lado, se observa que si bien el Clasificador 1 y el Clasificador 2 tienen prácticamente el mismo ROC-AUC (0.8096 y 0.8054), el Clasificador 1 tiene una curva ROC de ascenso más rápido. Esto puede inferirse dado que para distintos valores de Sensibilidad (82%, 89% y 95%), se obtienen valores de Especificidad más altos en el Clasificador 1 que en el Clasificador 2. Por lo mencionado anteriormente y observando el AP obtenido en estos 2 clasificadores (0.8677 versus 0.8025) concluimos que el Clasificador 1 tiene un mejor rendimiento que el Clasificador 2 en este conjunto de datos de prueba.

Modelos	Threshold 0.5				Especif. con sensib. al 82%	Especif. con sensib. al 89%	Especif. con sensib. al 95%	Average Precision (AP)	ROC-AUC
	Precision	F1-Score	Especificidad	Sensibilidad					
Clasificador 1 (CNN)	0.7826	0.8137	0.7617	0.8472	0.7809	0.6912	0.6073	0.8677	0.8096
Clasificador 2 (Híbrido)	0.7711	0.7165	0.6689	0.6691	0.6774	0.5719	0.3246	0.8025	0.8054
Ensamble	0.8539	0.8451	0.8337	0.8365	0.8624	0.6762	0.5804	0.9156	**0.8452**

Tabla 4.8: Métricas correspondientes a los distintos clasificadores cuando son aplicados al conjunto de datos de Prueba ROT.

Conjunto de datos de Prueba XY

En la tabla 4.9 se observa que el Ensamble tiene el mejor rendimiento en este conjunto de datos tomando como métrica primaria el ROC-AUC (0.7517). Además, las métricas complementarias obtienen sus máximos en el Ensamble, avalando la selección del mismo por sobre el Clasificador 1 y el Clasificador 2.

Se observa que el Clasificador 1 y el Clasificador 2 poseen valores similares en la métrica ROC-AUC (0.7208 y 0.7241) y en el indicador AP (0.8233 versus 0.8280). Sin embargo, el Clasificador 1 tiene un ascenso más rápido en la curva ROC que el Clasificador 2 ya que alcanza mayores valores de Especificidad para los 3 puntos de Sensibilidad indicados (82%, 89% y 95%). Además, el Clasificador 1 tiene una métrica F1-Score mayor que la del Clasificador 2 (0.6658 versus 0.6502). Según lo anteriormente mencionado, concluimos que el Clasificador 1 tiene un mejor rendimiento que el Clasificador 2 en este conjunto de datos de prueba.

Modelos	Threshold 0.5				Especif. con sensib. al 82%	Especif. con sensib. al 89%	Especif. con sensib. al 95%	Average Precision (AP)	ROC-AUC
	Precision	F1-Score	Especificidad	Sensibilidad					
Clasificador 1 (CNN)	0.8159	0.6658	0.8896	0.5623	0.6516	0.5283	0.3684	0.8233	0.7208
Clasificador 2 (Híbrido)	0.6704	0.6502	0.7012	0.6346	0.3963	0.2300	0.1418	0.8280	0.7241
Ensamble	0.8508	0.7045	0.9047	0.6012	0.7463	0.5928	0.4441	0.8311	**0.7517**

Tabla 4.9: Métricas correspondientes a los distintos clasificadores cuando son aplicados al conjunto de datos de Prueba XY.

Por último, notamos que las variaciones de las métricas F1-Score y AP entre las tablas 4.8 y 4.9 son inferiores en el Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) que en el Clasificador 1 (CNN). Esto nos sugiere inferir que el Modelo Híbrido es menos variante en su rendimiento cuando a las imágenes se les aplican operaciones no “vistas” en la fase de entrenamiento de la red. Es decir, operaciones no efectuadas en la etapa de data augmentation, en este caso traslaciones sobre los ejes X e Y.

4.2.3 Comparación de los modelos con un modelo externo

Para validar el rendimiento de los modelos, se los comparará con un modelo externo cuya arquitectura es similar a la que emplean los clasificadores híbridos en el presente trabajo. Es decir, involucra redes convolucionales y vectores de Fisher. Para dicha validación, como se mencionó en la sección 3.2, se utilizó el conjunto de datos de Prueba ISIC 2016 (que pertenece al utilizado en la competencia ISIC 2016[15]).

[15] https://challenge.kitware.com/#phase/5667455bcad3a56fac786791

El modelo externo utiliza una arquitectura AlexNet y a continuación se describe brevemente el preprocesamiento empleado, su arquitectura y entrenamiento; para más detalles se sugiere ver [27]. Por último, se mostrarán los resultados que se obtuvieron al aplicar los clasificadores de esta tesis y el modelo externo sobre el mencionado conjunto de datos de prueba (Prueba ISIC 2016).

Preprocesamiento para el modelo externo

El conjunto de datos de entrenamiento utilizado por los autores de [27] está compuesto por 900 imágenes y el de prueba por 379 imágenes. Cada uno de estos conjuntos de datos tiene 2 clases: “Melanoma” y “No Melanoma”. En primer lugar las imágenes fueron reescaladas a 542 píxeles de ancho con el alto variable (de forma que conserven el aspect ratio). Luego aplicaron data augmentation, rotando dichas imágenes escaladas en 0, 90, 180 y 270 grados.

Posteriormente, a este conjunto de imágenes se le aplicó varias ventanas de muestreo de tamaño variable de modo que cada ventana captura una porción de la imagen, conformando una sub-imagen. Las sub-imágenes fueron reescaladas a 227 x 227 píxeles y a cada una de ellas se les restó un "píxel medio" calculado como la media de los píxeles de cada canal RGB de todas las imágenes del conjunto de datos de entrenamiento. Lo anteriormente explicado se visualiza en la figura 4.29.

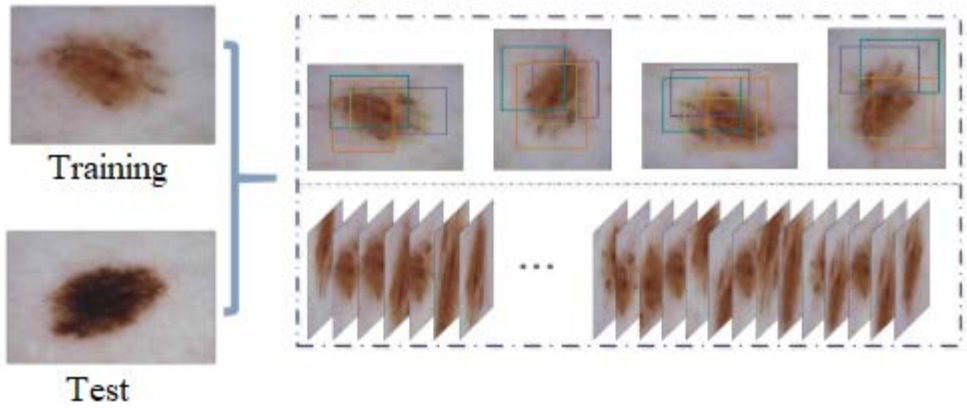


Figura 4.29: Rotación de las imágenes y conformación de las sub-imágenes con la ventana de muestreo de tamaño variable.

Arquitectura del modelo externo

La arquitectura empleada corresponde a la Alexnet y se utilizaron los descriptores correspondientes a la salida de las capas totalmente conectadas 6 y 7 para la codificación de los vectores de Fisher. Obteniendo el mejor resultado (utilizando la métrica ROC-AUC) con los descriptores (de 4096 dimensiones) pertenecientes a la salida de la capa totalmente conectada número 6. La arquitectura de la red utilizada se visualiza en la figura 4.30.

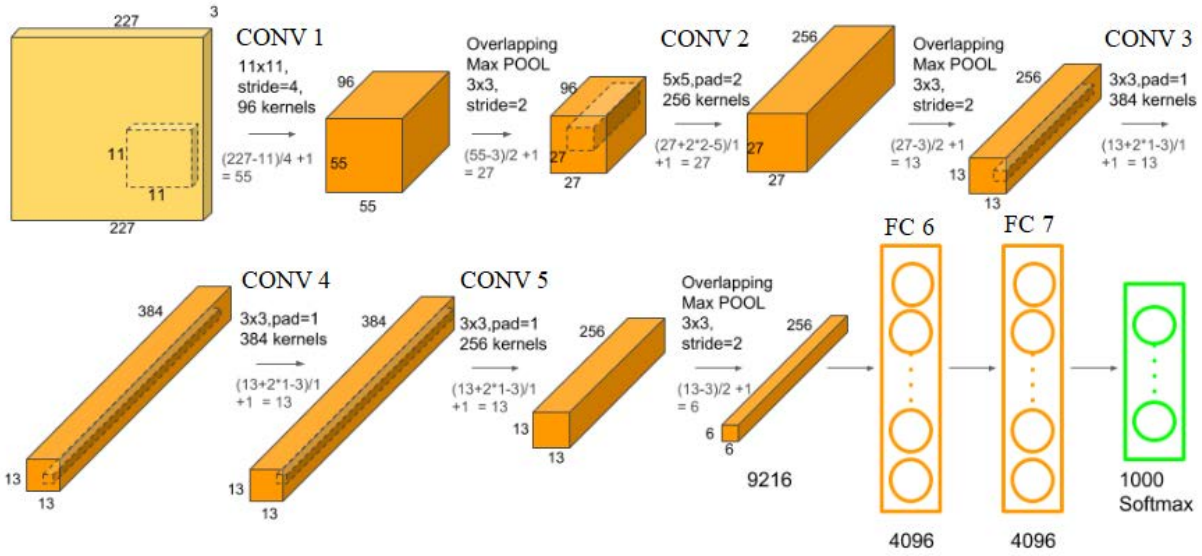


Figura 4.30: Arquitectura AlexeNet indicando las dimensiones de los kernels, los volúmenes de salida y los pads.

Codificación en los vectores de Fisher

Por cada imagen I habrá N sub-imágenes que serán ingresadas a la CNN de modo que la conformación del descriptor final F^I será la concatenación de los descriptores de cada sub-imagen:

$$F^I = \{f_1, f_2, \dots, f_N\}$$

Cada f_i tendrá una dimensión de 4096 y la cantidad de sub-imágenes (N) está fijado en 256. Teniendo en cuenta los descriptores F^I, procedieron a calcular los vectores de Fisher fv de la misma forma que en el presente trabajo, donde fijaron 128 componentes principales.

Para obtener el mejor resultado, por cada imagen I, generaron 4 imágenes adicionales con distinta escala. De forma que para cada imagen, generaron un vector de Fisher FV^I, que es la concatenación de los 5 vectores de Fisher individuales fv_i (un vector para la imagen original y 4 vectores para las 4 imágenes adicionales de diferente escala):

$$FV^I = \{fv_1, fv_2, fv_3, fv_4, fv_5\}$$

A este último método de concatenación lo llamaron “fusión”. Por último, estos FV^I constituyen el conjunto de entrenamiento de un modelo SVM con kernel lineal.

Resultados

En la tabla 4.10 se pueden observar los resultados obtenidos al aplicar los clasificadores del presente trabajo y el modelo externo en el conjunto de datos de Prueba ISIC 2016. Para la

evaluación, se tomó como métrica principal el ROC-AUC y como métricas complementarias la Exactitud y el AP. Se observa que los Clasificadores 2 y 3 obtienen una ROC-AUC sensiblemente inferior (pero que se encuentra en el orden) del obtenido por el Modelo Externo. Además se visualiza que el Clasificador 1 tuvo un buen rendimiento (ROC-AUC = 0.7998), superando al Modelo Externo. Por otro lado, con el Ensamble se obtuvo la mejor ROC-AUC y el mejor AP dentro de los modelos evaluados, 0.8211 y 0.6813 respectivamente. Con lo cual puede inferirse que los Modelos Híbridos (Clasificadores 2 y 3) complementan en cierta forma al modelo CNN convencional (Clasificador 1). Si bien la mejor Exactitud corresponde al Modelo Externo, dicha métrica no debe ser tomada con gran relevancia dado que el conjunto de datos de Prueba ISIC 2016 se encuentra desbalanceado (305 “No Melanomas” y 71 “Melanomas”). Es decir, se podría obtener un alto valor de Exactitud clasificando todas las lesiones pigmentadas como la clase mayoritaria (en este caso “No Melanoma”) y esto no indicaría un correcto funcionamiento del clasificador (ver la definición de la métrica Exactitud en la sección 3.1).

Clasificadores	ROC-AUC	Exactitud	Average Precision (AP)
Clasificador 1 (CNN)	0.7998	0.7542	0.6470
Clasificador 2 (Híbrido)	0.7631	0.7023	0.5098
Clasificador 3 (Híbrido)	0.7336	0.6486	0.4852
Ensamble	0.8211	0.8029	0.6813
Alexnet + FV (Modelo Externo)	0.7957	0.8309	0.5350

Tabla 4.10: Métricas correspondientes a los distintos clasificadores y al modelo externo cuando son aplicados al conjunto de datos de Prueba ISIC 2016.

5 Conclusiones y trabajos futuros

5.1 Conclusiones

El objetivo del presente trabajo es elaborar un sistema que identifique las imágenes de los melanomas dentro de un conjunto de imágenes que incluye otros tipos de lesiones pigmentadas. La posibilidad del dermatólogo en disponer de una clasificación (del sistema) suplementaria de la lesión, permitirá mejorar la exactitud del diagnóstico de la lesión. Las imágenes utilizadas corresponden a las enviadas por la médica dermatóloga Viviana Kowalckzuc del Hospital Italiano y las que se obtuvieron en los sitios Dermnet e ISIC. Además se generaron imágenes sintéticas en base a las imágenes originales y se redujo, mediante el algoritmo Max-RGB, el efecto de la posible distorsión del iluminante en las imágenes. Este sistema consta de una etapa de preprocesamiento de imágenes, otra de segmentación de imágenes y por último una de clasificación. Es en la etapa de clasificación donde se construyeron los Modelos Híbridos y los modelos convencionales CNN. Los Modelos Híbridos utilizan los descriptores locales de los clasificadores del tipo CNN para codificarlos en los vectores de Fisher, con el objetivo de clasificarlos mediante una máquina de vectores de soporte. Además, se realizó un Ensamble entre los clasificadores construidos, alcanzando el mayor rendimiento (frente a los clasificadores individuales) cuando fue evaluado en los distintos conjuntos de datos de prueba. Por último, se aplicó el método GRAD-CAM para evaluar la confiabilidad de la predicción de los modelos convencionales CNN (Clasificador 1 y Clasificador Auxiliar).

Las conclusiones, en base a los resultados, fueron las siguientes:

- En el conjunto de datos de Prueba HI se obtuvo un rendimiento inferior que en el de Prueba ISIC. Se induce que la baja resolución de las imágenes del Hospital Italiano puede ocasionar este declive en el rendimiento del Clasificador 1 y del Clasificador Auxiliar, los cuales fueron entrenados con imágenes de mayor resolución pertenecientes a los sitios Web de Dermnet e ISIC.

- Se recuerda que en el data augmentation se incluyeron las operaciones de las rotaciones de las imágenes pero no las traslaciones de las mismas sobre los ejes X e Y. Con lo cual, cuando se evalúan las variaciones de las métricas F1-Score y AP entre los conjuntos de datos de Prueba ROT y Prueba XY, se observa que dichas variaciones son inferiores en el Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) que en el Clasificador 1 (CNN). Esto nos sugiere inferir que el Modelo Híbrido es menos variable en su rendimiento cuando a las imágenes del conjunto de prueba se les aplican operaciones no “vistas” en la fase de entrenamiento de la red. Es decir, operaciones no efectuadas en la etapa de data augmentation, en este caso traslaciones sobre los ejes X e Y.

- La performance global del Clasificador 2 (Modelo Híbrido sin segmentación de imágenes) en los conjuntos de datos de prueba es levemente inferior al que se obtiene

en el Clasificador 1. No obstante, al construir el Ensamble, el rendimiento se eleva (en comparación al Clasificador 1) entre 3 y 7 puntos en la métrica ROC-AUC (ver figuras 4.23 y 4.25) y entre 1 y 5 puntos en la métrica PR-AUC (ver figuras 4.24 y 4.26) dependiendo el conjunto de datos de Prueba que se tome de análisis. Esta misma conclusión se obtuvo en la validación de los modelos con el conjunto de datos externo Prueba ISIC 2016, donde la métrica ROC-AUC del ensamble se incrementa en 2 puntos frente a la obtenida por los clasificadores individuales (ver tabla 4.10). Lo que indicaría que los Modelos Híbridos (Clasificadores 2 y 3) complementan de alguna forma al clasificador CNN (Clasificador 1), con lo cual se puede inferir que dichos modelos se encuentran levemente correlacionados.

- Por otro lado se observó que cuando los mapas de calor se encuentran fuera del área de la lesión, la performance del Clasificador 1 y el Clasificador Auxiliar disminuye. De hecho, en el 63% de los casos en que la mayor parte de la superficie del mapa de calor se encuentra fuera del área de la lesión, la predicción fue incorrecta. Con lo cual, cuando se realiza una predicción sobre una lesión pigmentada nueva, se tiene un indicio de la confiabilidad de la predicción dependiendo en donde se encuentre ubicado dicho mapa.

5.2 Trabajos futuros

Si bien existen varios experimentos que se podrían realizar para verificar si se incrementa la performance de los modelos clasificadores, se seleccionaron los siguientes:

- Se podría investigar la implementación de la red RESNET [117] como modelo clasificador y además se la podría utilizar para obtener los descriptores locales de las imágenes. Luego, con estos descriptores se construiría el Modelo Híbrido en el cual se codificarían los vectores Fisher para clasificarlos mediante un SVM. Por último se realizaría el ensamble de estos 2 clasificadores para verificar si, como si sucedió en el presente trabajo, tiene una mejor performance que los clasificadores individuales. Es decir, se podría verificar si en este escenario, nuevamente los clasificadores individuales (CNN e Híbrido) se complementan en cierta forma.

- Además se podría probar la utilización de los vectores de Fisher con estructura piramidal. Es decir, se obtendría un FV por cada región (obtenida en forma aleatoria) de la imagen y se conformaría un vector de Fisher final que sería el resultado de concatenar cada FV. Luego, utilizando los vectores de Fisher finales, se realizaría la clasificación de la imagen como se realizó en el presente trabajo (por medio de un SVM).

- Otro de los métodos que se podría probar, ya que en el presente trabajo se utilizó el método GRAD-CAM, es el que propone Xi Jia and Linlin Shen en [118]. En dicho trabajo la clasificación de una imagen se logra en 2 etapas. En la primer etapa el clasificador predice la imagen de una lesión pigmentada con el mapa de calor asociado (ya que utilizaron el método “Class Activation Mapping” explicado en la sección 2.5.2). En la segunda etapa se procede a recortar la imagen en los sectores

donde se superpone con el mapa de calor, con el objetivo de obtener una nueva imágen. Esta nueva imagen es ingresada en el modelo clasificador para obtener la predicción final. Con esta técnica, en el trabajo indicado, hubo una mejora del ROC-AUC de 0.821 (clasificación en 1 etapa) a 0.857 (clasificación en 2 etapas).

- Con respecto al Modelo Híbrido, existe un interesante trabajo propuesto por Patrick Wieschollek, Fabian Groh y Hendrik P.A. Lensch [119]. Los mencionados autores demuestran que los vectores de Fisher pueden ser implementados como una capa más en las redes neuronales y por ende se podrían aprender sus parámetros por medio del algoritmo backpropagation. Los parámetros son los que corresponden a las técnicas de PCA y GMM utilizadas para codificar los vectores de Fisher.

6 Referencias

[1] Palacios Julia, "Tejidos. Membrana. Piel. Derivados de la piel. [Online]. Available: https://www.infermeravirtual.com/files/media/file/95/Tejidos%2C%20membranas%2C%20piel%20y%20derivados.pdf?1358605323. Accedido: 08/11/2018.

[2] Vallejos, S. "American academy of dermatology". [Online]. Available : https://www.cancer.org/es/cancer/cancer-de-piel-tipo-melanoma/acerca/que-es-melanoma.html.Accedido: 05/11/2018.

[3] Magliano, J. "Cáncer de piel - Patología y abordaje terapéutico". [Online]. Available: http://tendenciasenmedicina.com/Imagenes/imágenes46/art_08.pdf. Accedido: 05/11/2018.

[4] Perez, G. "Síntomas de cáncer de piel". [Online]. Available: https://cancersintomas.com/cancer-de-piel. Accedido: 08/11/2018.

[5] Duce, Martín. Patología quirúrgica. Editorial Elsevier España; 2004.

[6] Gijsenij, Arjan & Gevers, T & Weijer, Joost. (2011). Computational Color Constancy: Survey and Experiments. IEEE transactions on image processing : a publication of the IEEE Signal Processing Society. 20. 2475-89. 10.1109/TIP.2011.2118224.

[7] Ronneberger, Olaf & Fischer, Philipp & Brox, Thomas. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.

[8] Simonyan, Karen & Zisserman, Andrew. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv 1409.1556.

[9] Sánchez, Jorge & Perronnin, Florent & Mensink, Thomas & Verbeek, Jakob. (2013). Image Classification with the Fisher Vector: Theory and Practice. International Journal of Computer Vision. 105. 10.1007/s11263-013-0636-x.

[10] R. Selvaraju, Ramprasaath & Das, Abhishek & Vedantam, Ramakrishna & Cogswell, Michael & Parikh, Devi & Batra, Dhruv. (2016). Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization.

[11] Stolz, Wilhelm & Riemann, A & B. Cognetta, A & Pillet, L & Abmayer, W & Holzel, D & Bilek, P & Nachbar, F & Landthaler, Michael & Braun-Falco, O. (1994). ABCD rule of dermatoscopy: A new practical method for early recognition of malignant melanoma. European Journal of Dermatology. 4. 521-527.

[12] Menzies, S. W. ; Ingvar, C. ; Crotty, K. A. ; McCarthy., W. H.: Frequency and Morphologic Characteristics of Invasive Melanoma Lacking Specific Surface Microscopic Features. En: Arch. Dermatology 132 (1996), p. 1178–1182.

[13] Argenziano G, Fabbrocini G, Carli P, De Giorgi V, Sammarco E, Delfino M. Epiluminescence Microscopy for the Diagnosis of Doubtful Melanocytic Skin Lesions Comparison of the ABCD Rule of Dermatoscopy and a New 7-Point Checklist Based on Pattern Analysis. Arch Dermatol. 1998.

[14] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66, Jan. 1979.

[15] Iyatomi, H. ; Oka, H. ; Hashimoto, M. ; Tanaka, M. ; Ogawa., K.: An internet based melanoma diagnostic system Toward the practical application. En: Proc. 2005 IEEE Symp. Computational Intelligence in Bioinformatics and Computational Biology., 2005, p. 1–4.

[16] Capdehourat, G. ; Corez, A. ; Bazzano, A. ; Musé., P.: Pigmented skin lesions classification using dermatoscopic images. En: CIARP, 2009.

[17] Alcón, J. F. ; Ciuhu, C. ; Kate, W. ; Heinrich, A. ; Uzunbajakava, N. ; Krekels, G. ; Siem, D. ; Haan., G.: Automatic Imaging System With Decision Support for Inspection of Pigmented Skin Lesions and Melanoma Diagnosis. En: IEEE Journal of Selected Topics in Signal Processing 3 (2009), p. 14–25.

[18] Leo, G. D. ; Paolillo, A. ; Sommella, P. ; Fabbrocini., G.: Automatic Diagnosis of Melanoma: a Software System based on the 7-Point Check-List. En: HICSS., 2010.

[19] Ruiz, et a.: A decision support system for the diagnosis of melanoma: A comparative approach. En: Expert Systems with Applications. 38 (2011), p. 15217–15223

[20] Capdehourat, G. ; Corez, A. ; Bazzano, A. ; Alonso, R. ; Musé., P.: Toward a combined tool to assist dermatologists in melanoma detection from dermoscopic images of pigmented skin lesions. En: Pattern Recognition Letters (2011), p. 1–10.

[21] Xie, Feng-ying & Fan, Haidi & Yang, Li & Jiang, Zhi-guo & Meng, Ru-song & Bovik, Alan. (2016). Melanoma Classification on Dermoscopy Images Using a Neural Network Ensemble Model. IEEE Transactions on Medical Imaging. PP. 1-1. 10.1109/TMI.2016.2633551.

[22] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” arXiv:1505.04597, 2015.

[23] N. Codella, J. Cai, M. Abedini, R. Garnavi, A. Halpern, and J. R. Smith, “Deep learning, sparse coding, and svm for melanoma recognition in dermoscopy images,” in MLMI, 2015, pp. 118–126.

[24] Demyanov, Sergey et al. “Classification of dermoscopy patterns using deep convolutional neural networks.” 2016 IEEE 13th International Symposium on Biomedical Imaging (ISBI) (2016): 364-368.

[25] J. Kawahara, A. BenTaieb, and G. Hamarneh, "Deep features to classify skin lesions," in ISBI, 2016, pp. 1397-1400.

[26] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, “Dermatologist-level classification of skin cancer with deep neural networks,” Nature, vol. 542, no. 7639, pp. 115–118, 2017.

[27] Yu, Z., Ni, D., Chen, S., Qin, J., Li, S., Wang, T., Lei, B.: Hybrid dermoscopy image classification framework based on deep convolutional neural network and Fisher vector. In: 14th IEEE International Symposium on Biomedical Imaging, ISBI 2017, Melbourne, Australia, April 18-21, 2017.

[28] Neural networks : algorithms, applications, and programming techniques / JamesA. Freeman, David M. Skapura

[29] Curso CS231n de la universidad de Stanford - Redes neuronales convolucionales para el reconocimiento visual - Módulo 1 [Online]. Disponible: http://cs231n.github.io. Accedido: 10/07/2018.

[30] Bishop, Christopher M. Pattern Recognition and Machine Learning - Pag. 227-229.

[31] Wu, Huaiqin (2009). "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions". Information Sciences. 179 (19): 3432–3441.

[32] Snyman, Jan (3 March 2005). Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Springer Science & Business Media.

[33] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016.Pag. 188-193.

[34] Mehryar Mohri, Afshin Rostamizadeh, Ameet Talwalkar (2012) Foundations of Machine Learning.

[35] Stuart Russell and Peter Norvig. Artificial Intelligence: A Modern Approach, 2009 (3rd edition). Pag. 709.

[36] Burnham, K. P.; Anderson, D. R. (2002), Model Selection and Multimodel Inference (2nd ed.), Springer-Verlag. Pag. 40-50.

[37] Burnham, K. P.; Anderson, D. R. (2002), Model Selection and Multimodel Inference (2nd ed.), Springer-Verlag. Pag. 32.

[38] Yosinski, Jason, Jeff Clune, Yoshua Bengio and Hod Lipson. “How transferable are features in deep neural networks?” NIPS (2014).

[39] Torrey, Lisa and Jude Shavlik. “Transfer Learning.” (2009).

[40] Curso CS231n de la universidad de Stanford - Redes neuronales convolucionales para el reconocimiento visual - Módulo 2 [Online]. Disponible: http://cs231n.github.io. Accedido: 09/08/2018.

[41] Chu, Brian, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman and Trevor Darrell. “Best Practices for Fine-Tuning Visual Classifiers to New Domains.” ECCV Workshops (2016).

[42] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016.Pag. 124-135.

[43] Hastie, T., Tibshirani, R., and J. Friedman The Elements of Statistical Learning Springer, 2017. Pag. 69-73.

[44] Michael A. Nielsen. Neural networks and deep learning. 2015. Capítulo 3.

[45] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016. Pag. 116.

[46] Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov. “Dropout: a simple way to prevent neural networks from overfitting.” Journal of Machine Learning Research 15 (2014): 1929-1958.

[47] S. Park and N. Kwak, “Analysis on the dropout effect in convolutional neural networks,” Asian Conference on Computer Vision, pp. 189–204, 2016.

[48] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” Proceedings of COMPSTAT 2010, pp. 177–177, 2010.

[49] Bottou, Léon. "Stochastic gradient descent tricks." Neural Networks: Tricks of the Trade. Springer Berlin Heidelberg, 2012. 421-436.

[50] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” International conference on machine learning, vol. 28, pp. III–1139–III–1147, 2013.

[51] Bengio, Yoshua, Nicolas Boulanger-Lewandowski and Razvan Pascanu. “Advances in optimizing recurrent networks.” 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (2013): 8624-8628.

[52] Sutskever, Ilya, James Martens, George E. Dahl and Geoffrey E. Hinton. “On the importance of initialization and momentum in deep learning.” ICML (2013).

[53] Duchi, John C., Elad Hazan and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.” COLT (2010).

[54] Zeiler, Matthew D.. “ADADELTA: An Adaptive Learning Rate Method.” CoRRabs/1212.5701 (2012).

[55] Geoff Hinton. “RMSPROP”. No posee publicación. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accedido: 20/09/2018.

[56] Kingma, Diederik P. and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” CoRR abs/1412.6980 (2014).

[57] Rumelhart, David E., Geoffrey E. Hinton and Ronald J. Williams. “Learning representations by back-propagating errors.” Nature 323 (1986): 533-536.

[58] Curso CS231n de la universidad de Stanford - Redes neuronales convolucionales para el reconocimiento visual - Lectura 04 [Online]. Disponible: http://cs231n.stanford.edu/syllabus.html. Accedido: 30/10/2018.

[59] R. Socher, “cs224d deep nlp, lecture 5 neural networks and backprop,” [On line]. Available: http://cs224d.stanford.edu/lectures/CS224d-Lecture5.pdf. Accedido: 01/11/2018.

[60] Zhang, Zhifei. “Derivation of Backpropagation in Convolutional Neural Network (CNN). (2016).

[61] Hubel DH, Wiesel TN (January 1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". The Journal of Physiology. 160 (1): 106–54.

[62] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016. Pag. 327.

[63] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016. Pag. 329-337.

[64] Masaaki Kurosu. Human-Computer Interaction. Part 2 - HCI International 2018. Pag. 385.

[65] Dumoulin, Vincent and Francesco Visin. “A guide to convolution arithmetic for deep learning.” CoRR abs/1603.07285 (2016).

[66] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016. Pag. 354.

[67] Maas, Andrew L.. “Rectifier Nonlinearities Improve Neural Network Acoustic Models.” (2013).

[68] Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke and Andrew Rabinovich. “Going deeper with convolutions.” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 1-9.

[69] Bishop, Christopher M. Pattern Recognition and Machine Learning - Pag. 198.

[70] El color es luz :http://proyectacolor.cl/teoria-de-los-colores/ el-color-es-luz/. Accedido: 01/09/2018.

[71] Rastislav Lukac. Computational photography - Pag. 104-106.

[72] M. Stone Journal of the Royal Statistical Society. Series B (Methodological) Vol. 39, No. 1 (1977), pp. 44-47.

[73] Rajalingappa Shanmugaman. Deep Learning for Computer Vision - Pag. 232-233.

[74] Sercu, Tom and Vaibhava Goel. “Dense Prediction on Sequences with Time-Dilated Convolutions for Speech Recognition.” CoRR abs/1611.09288 (2016).

[75] Farabet, Clément, Camille Couprie, Laurent Najman and Yann LeCun. “Learning Hierarchical Features for Scene Labeling.” IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (2013): 1915-1929.

[76] Giusti, Alessandro, Dan C. Ciresan, Jonathan Masci, Luca Maria Gambardella and Jürgen Schmidhuber. “Fast image scanning with deep max-pooling convolutional neural networks.” 2013 IEEE International Conference on Image Processing (2013): 4034-4038.

[77] Curso CS231n de la universidad de Stanford - Redes neuronales convolucionales para el reconocimiento visual - Lectura 11 [Online]. Disponible: http://cs231n.stanford.edu/syllabus.html. Accedido: 01/08/2018.

[78] Im, Daniel Jiwoong, Chris Dongjoo Kim, Hui Jiang and Roland Memisevic. “Generating images with recurrent adversarial networks.” CoRR abs/1602.05110 (2016). Explicación en materiales suplementarios.

[79] Gauthier, Jon. “Conditional generative adversarial nets for convolutional face generation.” (2015).

[80] Shelhamer, E., Long, J., & Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3431-3440.

[81] Krizhevsky, Alex, Ilya Sutskever and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks.” NIPS (2012).

[82] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI.

[83] G. Ian, B. Yoshua, and C. Aaron, Deep Learning. MIT Press, 2016. Pag. 499-500.

[84] Shvets, Alexey, Alexander Rakhlin, Alexandr A. Kalinin and Vladimir I. Iglovikov. “Automatic Instrument Segmentation in Robot-Assisted Surgery Using Deep Learning.” CoRR abs/1803.01207 (2018): n. pag.

[85] Zeiler, Matthew D. and Rob Fergus. “Visualizing and Understanding Convolutional Networks.” ECCV (2014).

[86] Lopez Adria Romero, Xavier Giró-i-Nieto, Jack Burdick and Oge Marques. “Skin lesion classification from dermoscopic images using deep learning techniques.” 2017 13th IASTED International Conference on Biomedical Engineering (BioMed) (2017): 49-54.

[87] Curso CS231n de la universidad de Stanford - Redes neuronales convolucionales para el reconocimiento visual - Lectura 9 [Online]. Disponible: http://cs231n.stanford.edu/syllabus.html. Accedido: 10/08/2018.

[88] Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh and Dhruv Batra. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization.” 2017 IEEE International Conference on Computer Vision (ICCV) (2017): 618-626.

[89] Zhou, Bolei, Aditya Khosla, Àgata Lapedriza, Aude Oliva and Antonio Torralba. “Learning Deep Features for Discriminative Localization.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 2921-2929.

[90] Lin, Min, Qiang Chen and Shuicheng Yan. “Network In Network.” CoRR abs/1312.4400 (2013): n. Pag.

[91] G. McLachlan and D. Peel, Finite Mixture Models. New York: John Wiley & Sons, 2000. Pag.1-10.

[92] G. McLachlan and D. Peel, Finite Mixture Models. New York: John Wiley & Sons, 2000. Pag.1-37.

[93] J. A. Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report TR-97-021, International Computer Science Institute and Computer Science Division, University of California at Berkeley, April 1998.

[94] C. Biernacki, G. Celeux, and G. Govaert, Choosing starting values for the EM algorithm for getting the highest likelihood in multivariate Gaussian mixture models, Comput. Stat. Data Anal., vol. 41, no. 3-4, pp. 561–575, Jan. 2003.

[95] V. Melnykov and I. Melnykov, Initializing the EM algorithm in Gaussian mixture models with an unknown number of components, Computational Statistics & Data Analysis, Nov. 2011.

[96] W. Kwedlo, A New Method for Random Initialization of the EM Algorithm for Multivariate Gaussian Mixture Learning, in Proceedings of the 8th International Conference on Computer Recognition Systems CORES 2013. Springer International Publishing, 2013, pp. 81–90.

[97] D. Arthur and S. Vassilvitskii, k-means++: the advantages of careful seeding, in SODA, N. Bansal, K. Pruhs, and C. Stein, Eds. SIAM, 2007, pp. 1027–1035.

[98] Jun Shao Statistica Sinica Vol. 7, No. 2 (April 1997), pp. 221-242.

[99] Hastie, T., Tibshirani, R., and J. Friedman The Elements of Statistical Learning Springer, 2001. Pag. 208.

[100] Pang-Ning Tan, Michael Steinbach, Vipin Kumar; “Introduction to Data Mining”; Pearson Education, 2006; ISBN 0-321-42052-7; 683-690.

[101] Pearson, K. (1901) On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine, 2, 559-572.

[102] Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 24, 417–441, and 498–520.

[103] Bishop, Christopher M. Pattern Recognition and Machine Learning - Pag. 561.

[104] Peres-Neto, Pedro R., Donald A. Jackson and Keith M. Somers. “How many principal components? stopping rules for determining the number of non-trivial axes revisited.” Computational Statistics & Data Analysis 49 (2005): 974-997.

[105] Kevin Dunn. Process Improvement Using Data. Chapter 6.

[106] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In ECCV, 2004.

[107] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In NIPS, 1999.

[108] Krapac J, Verbeek J, Jurie F (2011) Modeling spatial layout with fisher vectors for image categorization. In: ICCV.

[109] V. Vapnik y A. Lerner. Pattern recognition using generalized portraid method. Automation and remote control, 24. 1963.

[110] C. Burges. A tutorial on support vector machines for pattern recognition. Bell Laboratories, Lucent Technologies, USA. Data Mining and Knowledge Discovery, 2, 121–167. 1998.

[111] Pang-Ning Tan, Michael Steinbach, Vipin Kumar; “Introduction to Data Mining”; Pearson Education, 2006; ISBN 0-321-42052-7; 256-276.

[112] T.M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. IEEE Transaction on Electronics Computer. Vol. 14, 326-334. 1965 .

[113] Metz CE. Some practical issues of experimental design and data analysis in radiologic ROC studies. Invest Radiol 1989;24:234–245.

[114] J. Krzanowski, Wojtek & Hand, David. (2009). ROC Curves for continuous data. 10.1201/9781439800225. Paginas: 17-27.

[115] Saito T, Rehmsmeier M, "The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets". PLoS ONE 10(3): e0118432. 2015.

[116] He, Kaiming, Xiangyu Zhang, Shaoqing Ren and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.” 2015 IEEE International Conference on Computer Vision (ICCV) (2015): 1026-1034.

[117] He, Kaiming, Xiangyu Zhang, Shaoqing Ren and Jian Sun. “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 770-778.

[118] Xi Jia and Linlin Shen;”Skin Lesion Classification using Class Activation Map” (2018).

[119] Wieschollek, Patrick, Fabian Groh and Hendrik P. A. Lensch. “Backpropagation Training for Fisher Vectors within Neural Networks.” CoRR abs/1702.02549 (2017).

7 Apéndice

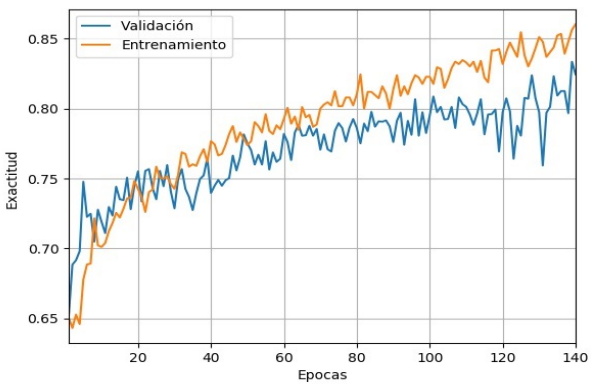
A continuación se muestran los diferentes entrenamientos efectuados tanto para el Clasificador 1 como para el Clasificador Auxiliar. En total son 36 entrenamientos, 18 para el Clasificador 1 (sección “A” del apéndice) y 18 para el Clasificador Auxiliar (sección “B” del apéndice). Los hiperparámetros que se variaron fueron: el optimizador, la tasa de aprendizaje, el batch size y se incluyó el step decay en la planificación de la tasa de aprendizaje. Es decir, se probó la combinatoria de los valores incluidos en los siguientes conjuntos:

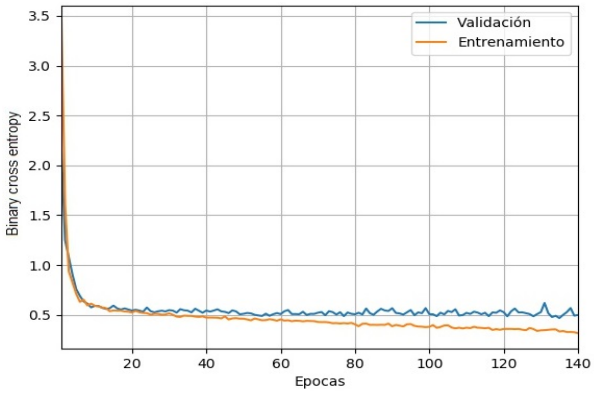
- Batch size = {16, 32}
- Tasa de aprendizaje = {0.00003, 0.00002, 0.00001}
- Optimizador = {Adam, SGD con momento Nesterov}

Cuando en la combinatoria aparece el optimizador SGD, se probó con y sin step decay. En el step decay se especificó que cada 35 épocas la tasa de aprendizaje tome la mitad del valor de la época anterior.

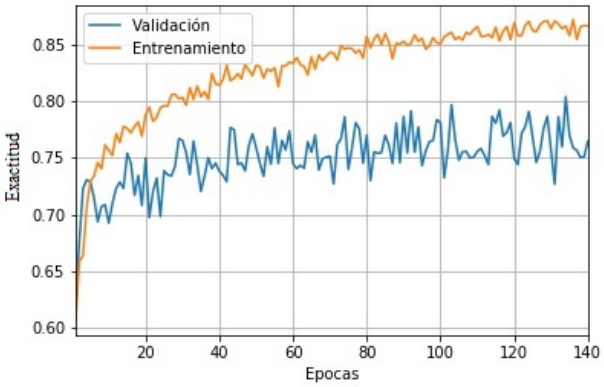
A Experimentos del Clasificador 1

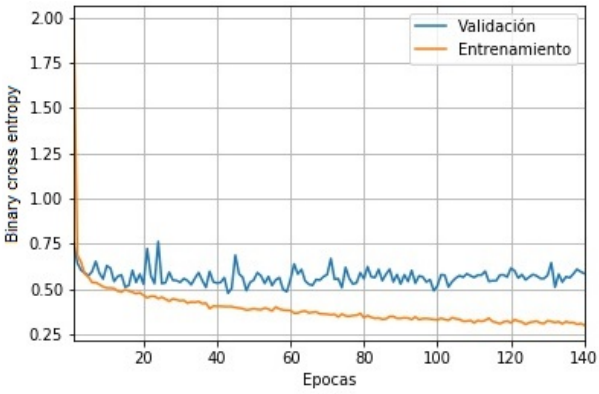
Experimento 1
Batch size = 16
Tasa de aprendizaje = 0.00001
Optimizador = SGD con momento Nesterov

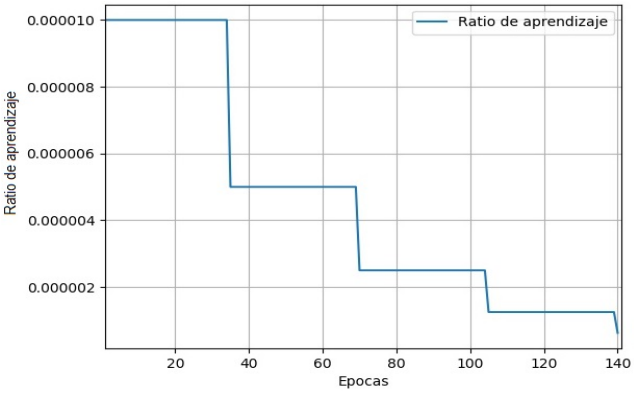




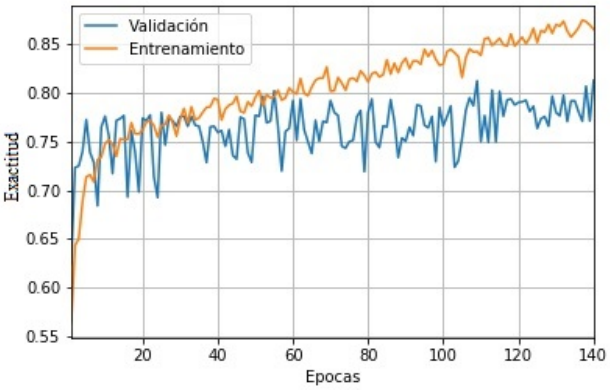
Experimento 2
Batch size = 16
Tasa de aprendizaje = 0.00001
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

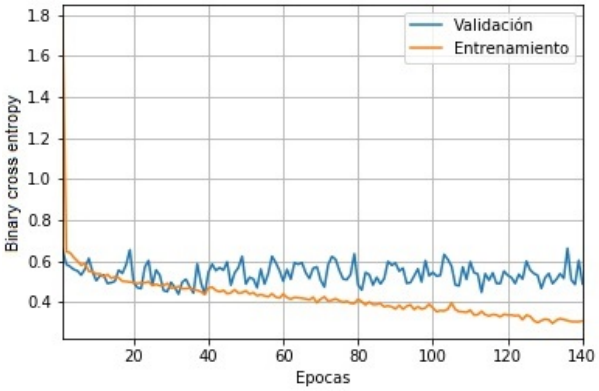




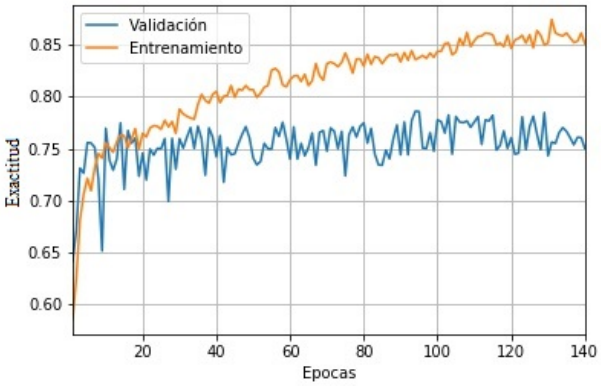


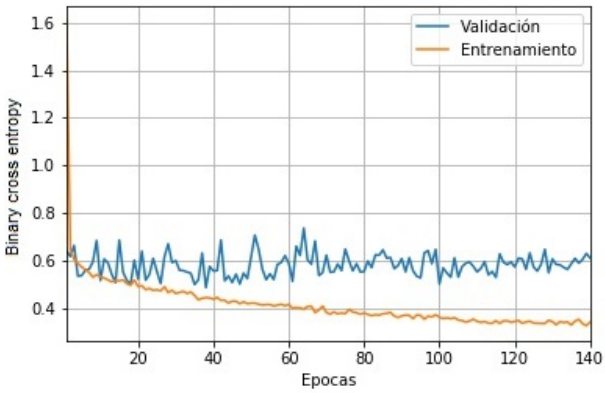
Experimento 3
Batch size = 16
Tasa de aprendizaje = 0.00002
Optimizador = SGD con momento Nesterov

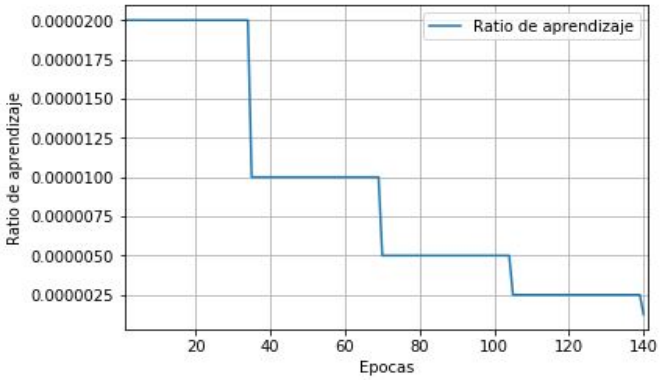




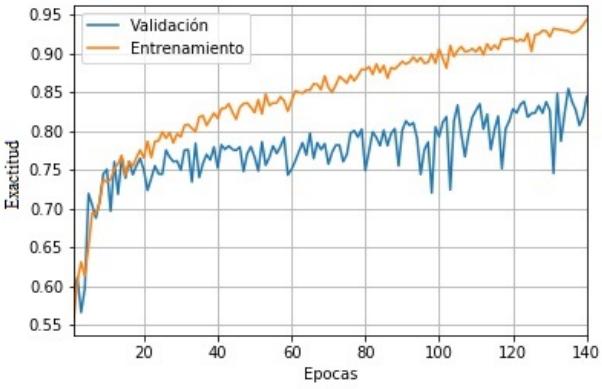
Experimento 4
Batch size = 16
Tasa de aprendizaje = 0.00002
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

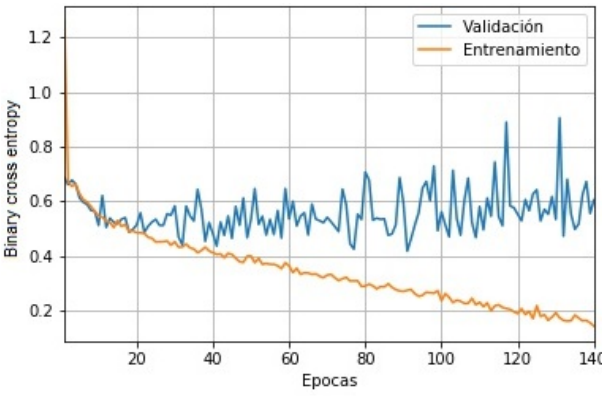




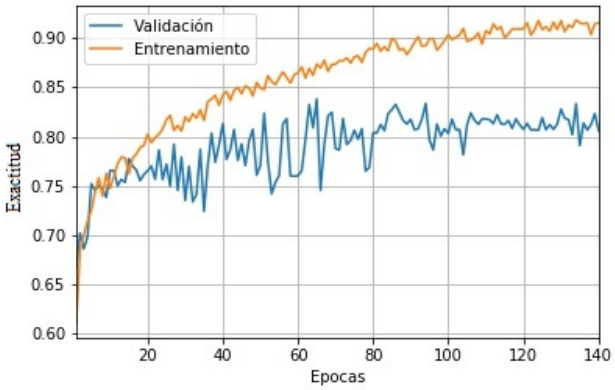


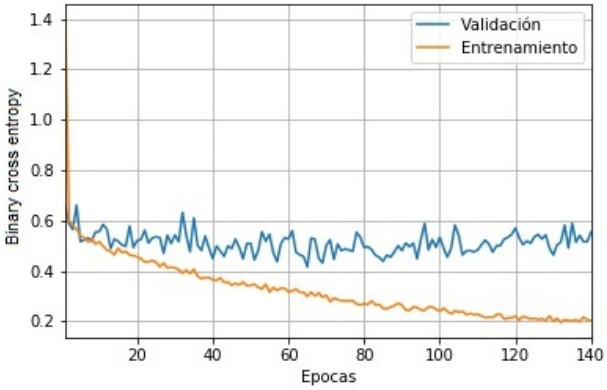
Experimento 5
Batch size = 16
Tasa de aprendizaje = 0.00003
Optimizador = SGD con momento Nesterov

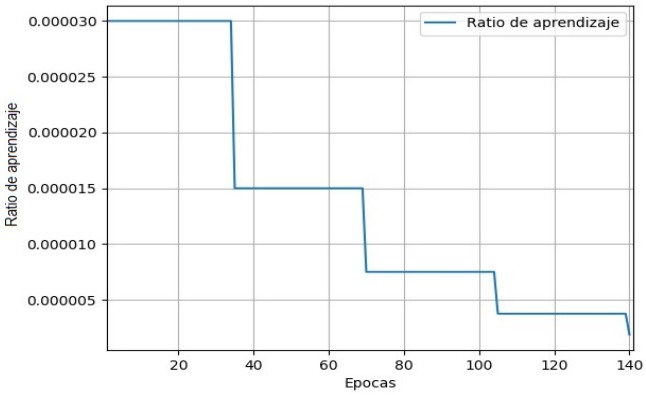




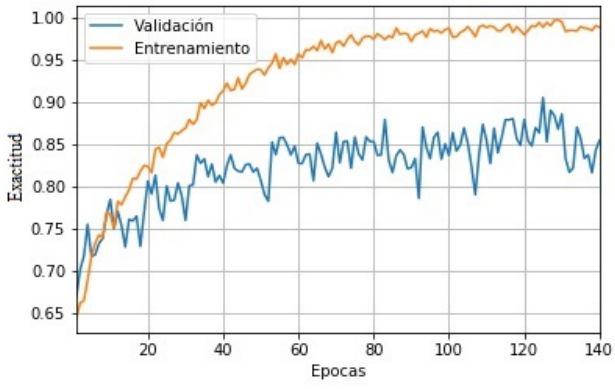
Experimento 6
Batch size = 16
Tasa de aprendizaje = 0.00003
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

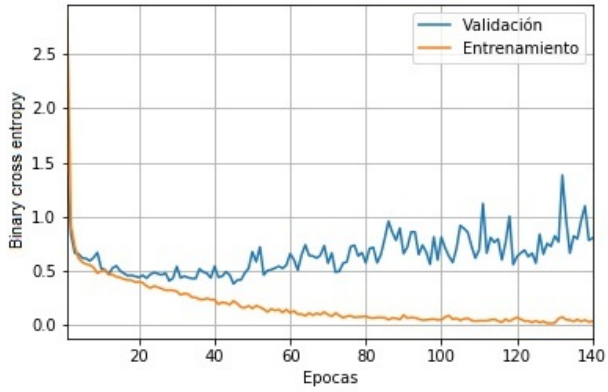




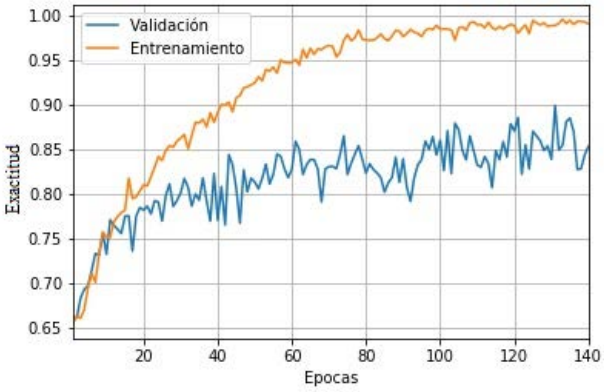


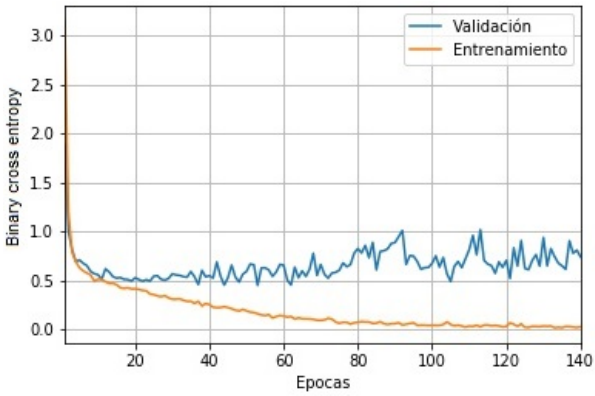
Experimento 7
Batch size = 16
Tasa de aprendizaje inicial = 0.00001
Optimizador = Adam



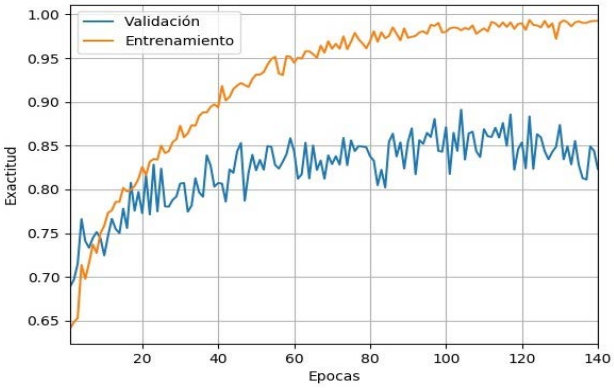


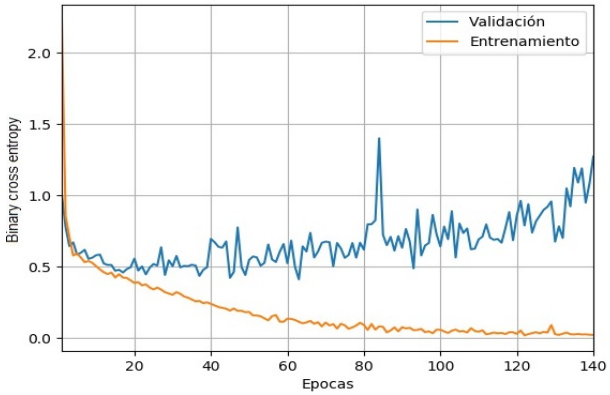
Experimento 8
Batch size = 16
Tasa de aprendizaje inicial = 0.00002
Optimizador = Adam



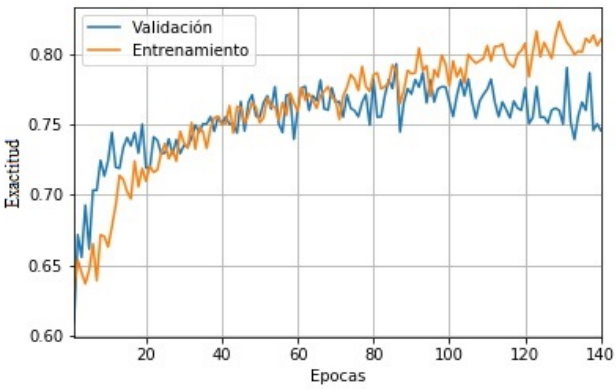


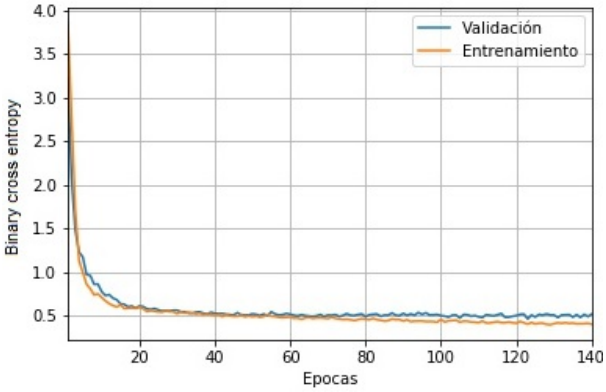
Experimento 9
Batch size = 16
Tasa de aprendizaje inicial = 0.00003
Optimizador = Adam



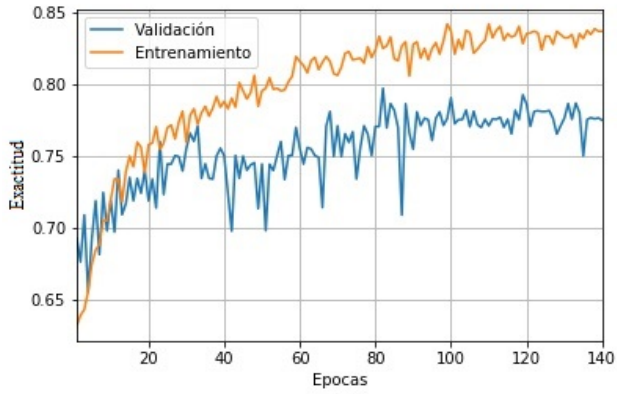


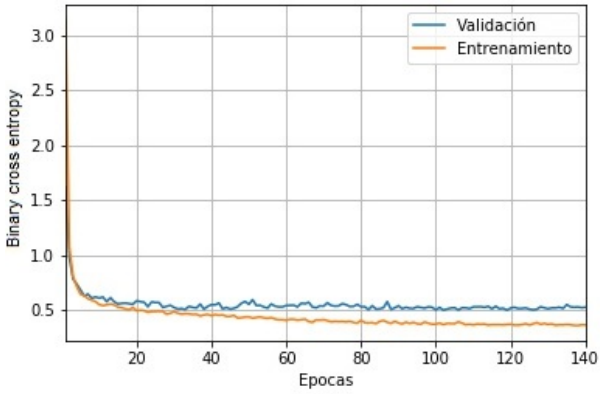
Experimento 10
Batch size = 32
Tasa de aprendizaje = 0.00001
Optimizador = SGD con momento Nesterov

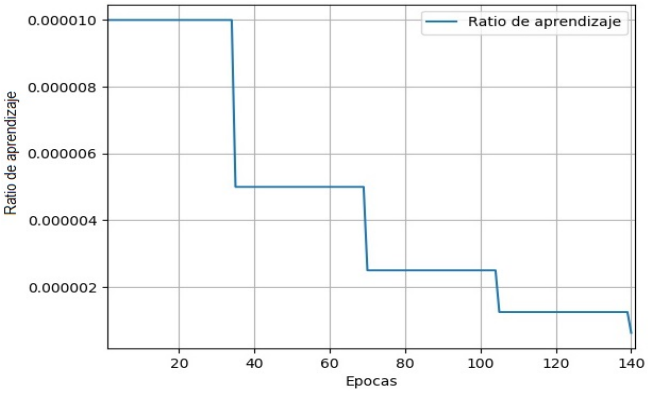




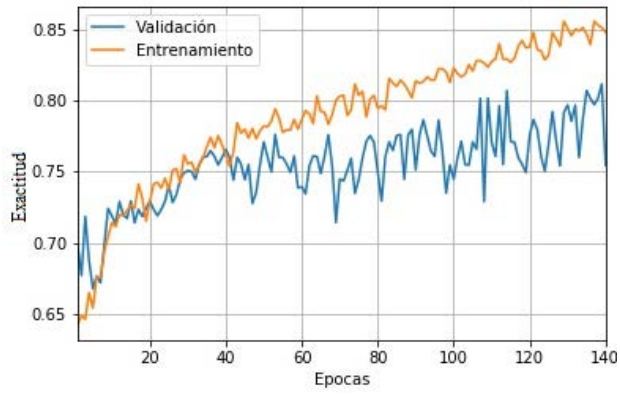
Experimento 11
Batch size = 32
Tasa de aprendizaje = 0.00001
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

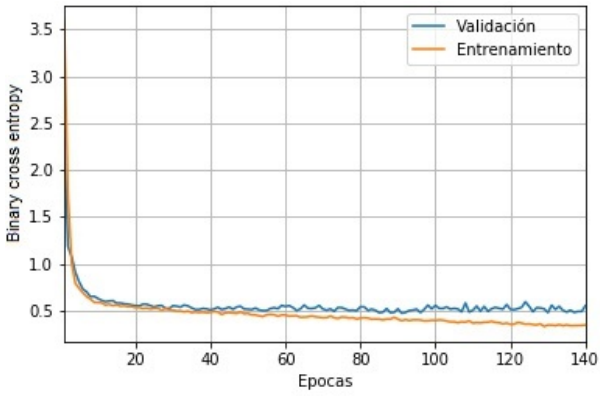




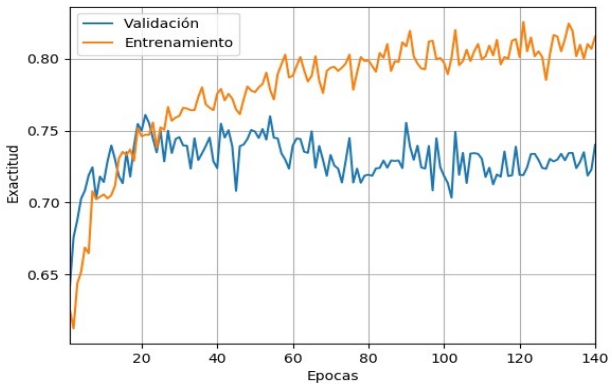


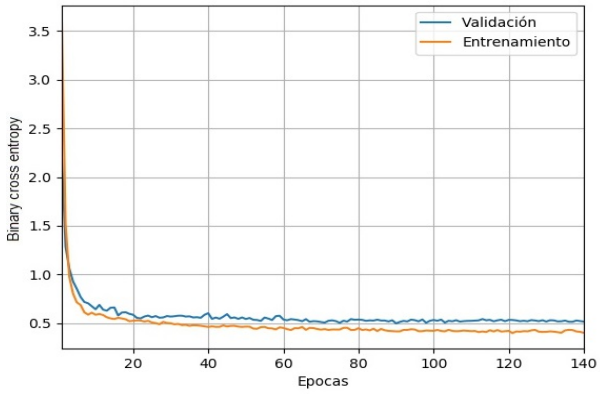
Experimento 12
Batch size = 32
Tasa de aprendizaje = 0.00002
Optimizador = SGD con momento Nesterov

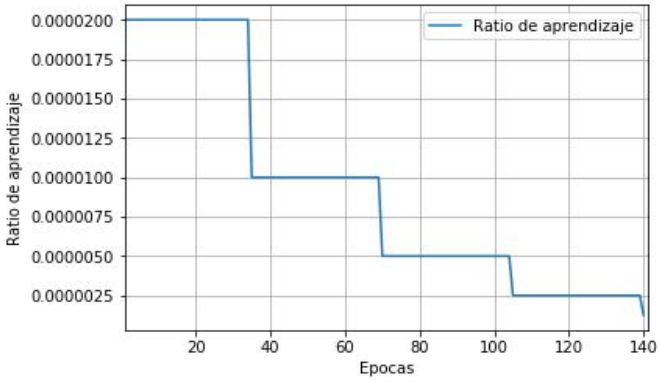




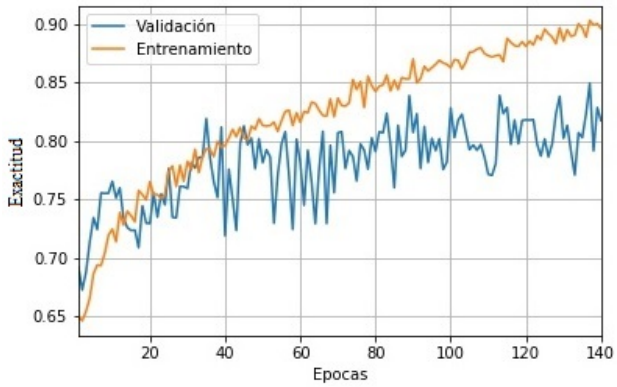
Experimento 13
Batch size = 32
Tasa de aprendizaje = 0.00002
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

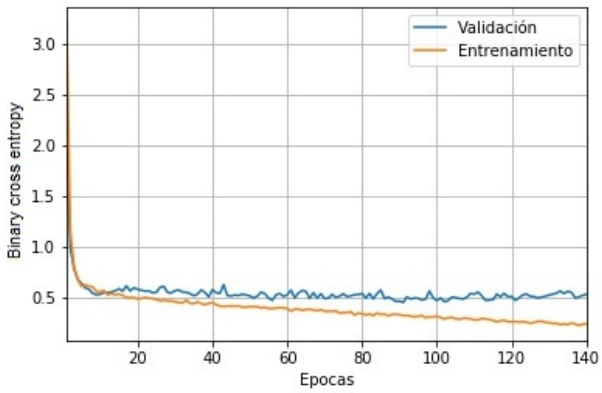




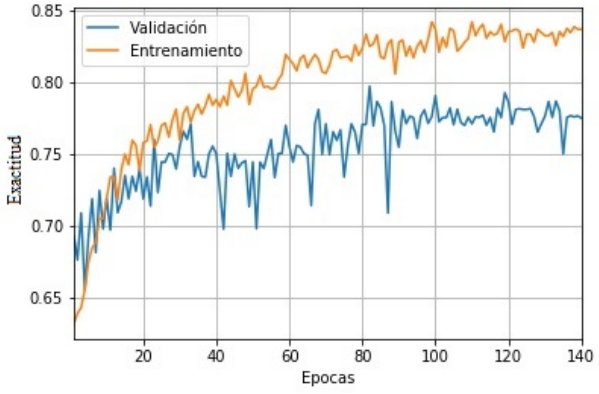


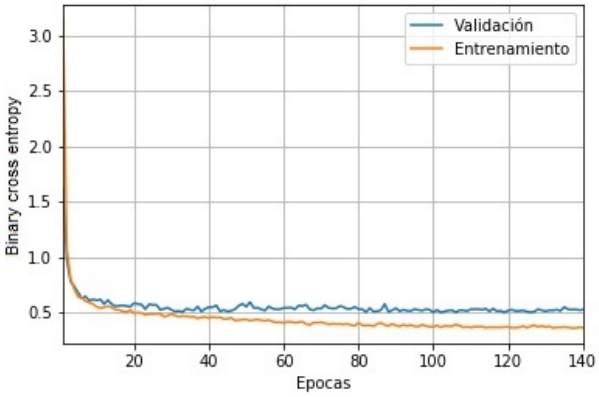
Experimento 14
Batch size = 32
Tasa de aprendizaje = 0.00003
Optimizador = SGD con momento Nesterov

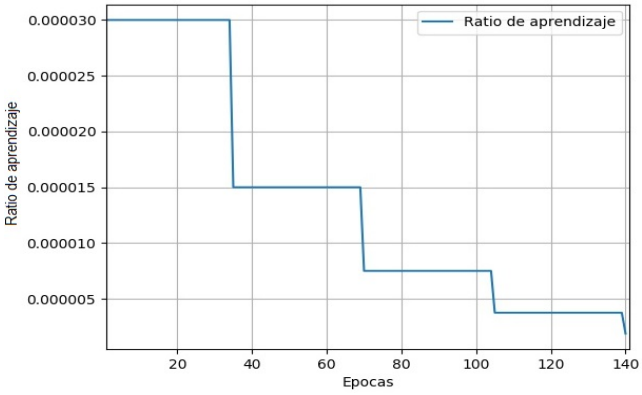




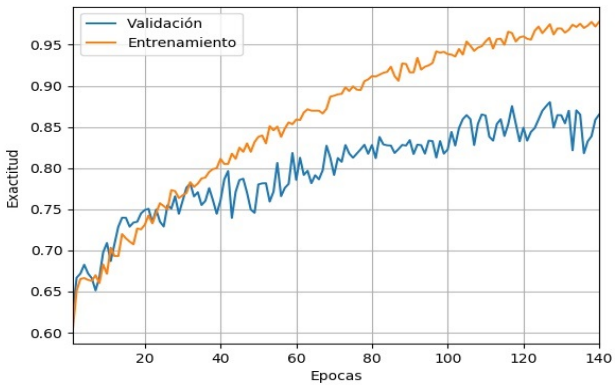
Experimento 15
Batch size = 32
Tasa de aprendizaje = 0.00003
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

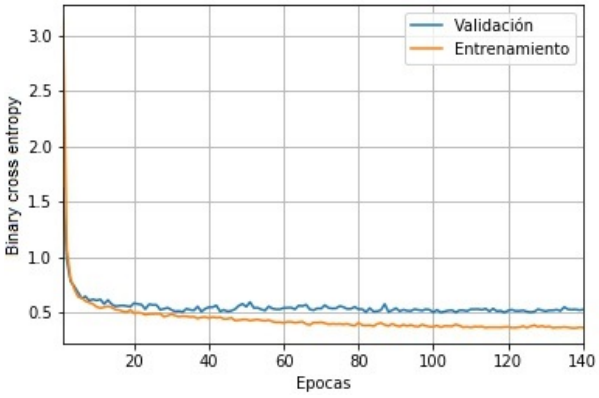




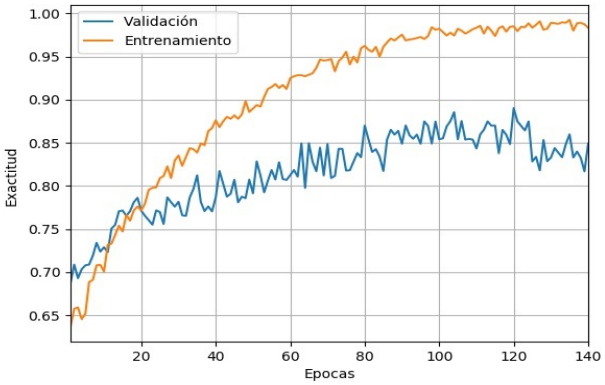


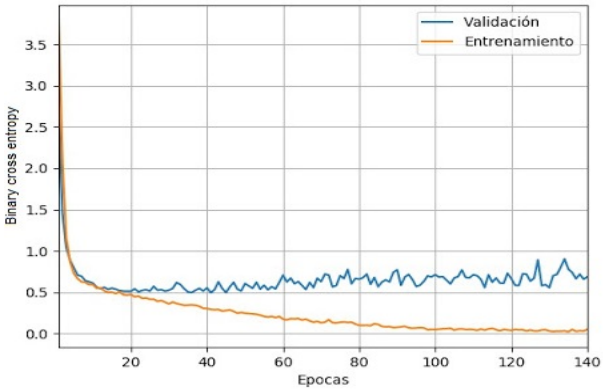
Experimento 16
Batch size = 32
Tasa de aprendizaje inicial = 0.00001
Optimizador = Adam



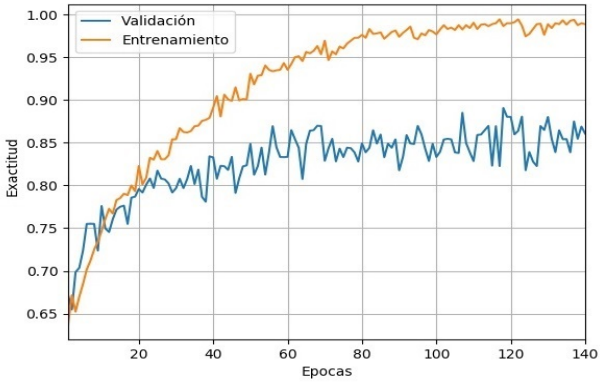


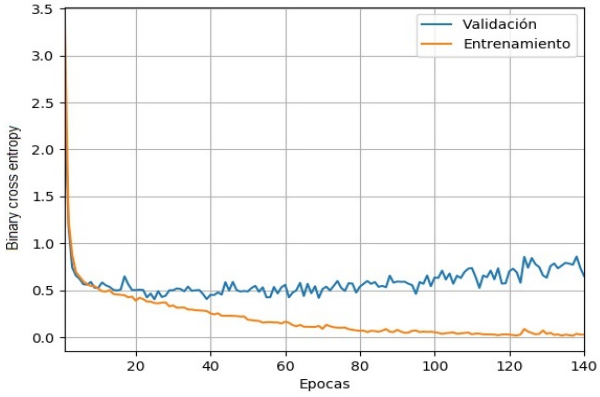
Experimento 17
Batch size = 32
Tasa de aprendizaje inicial = 0.00002
Optimizador = Adam





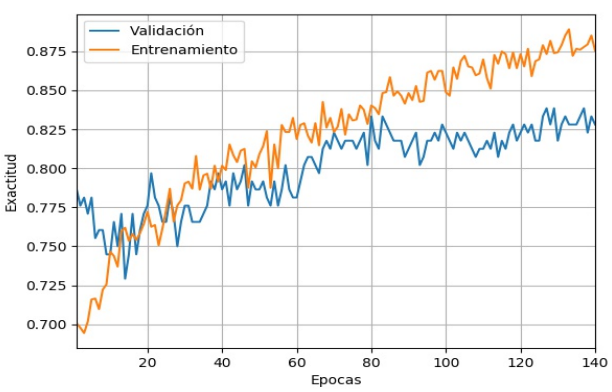
Experimento 18
Batch size = 32
Tasa de aprendizaje inicial = 0.00003
Optimizador = Adam

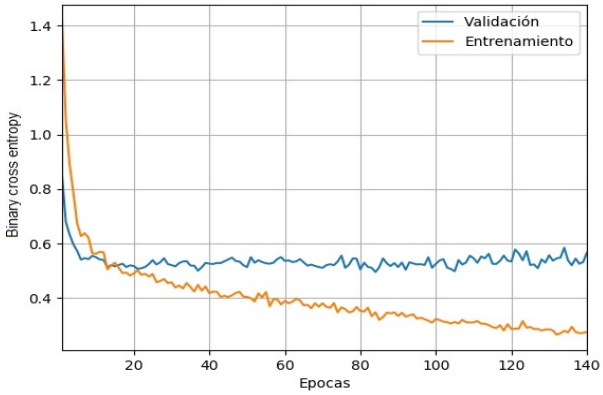




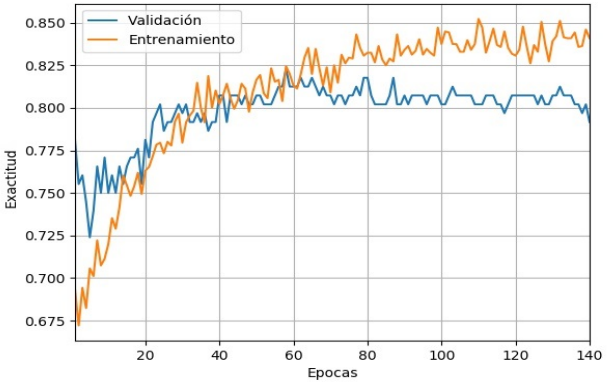
B Experimentos del Clasificador Auxiliar

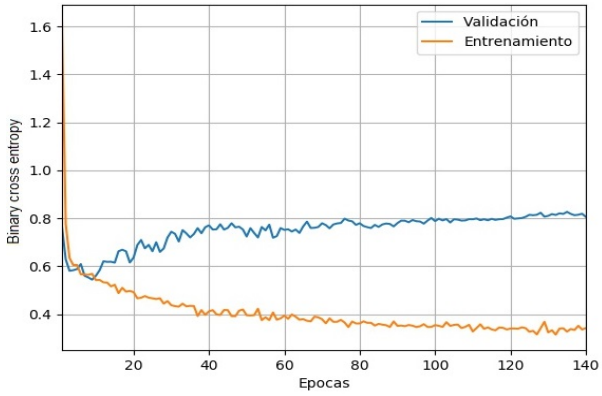
Experimento 1
Batch size = 16
Tasa de aprendizaje = 0.00001
Optimizador = SGD con momento Nesterov

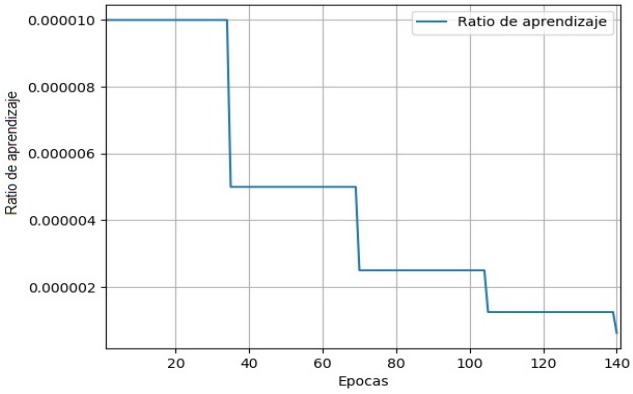




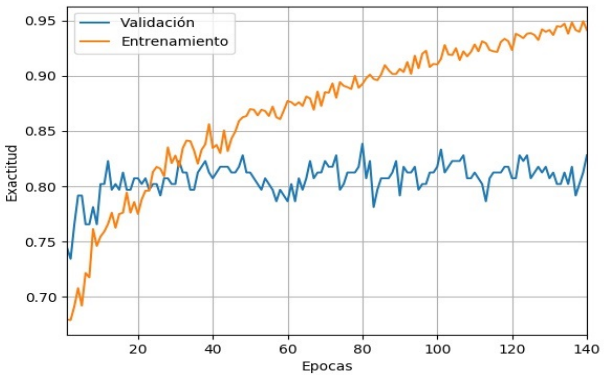
Experimento 2
Batch size = 16
Tasa de aprendizaje = 0.00001
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

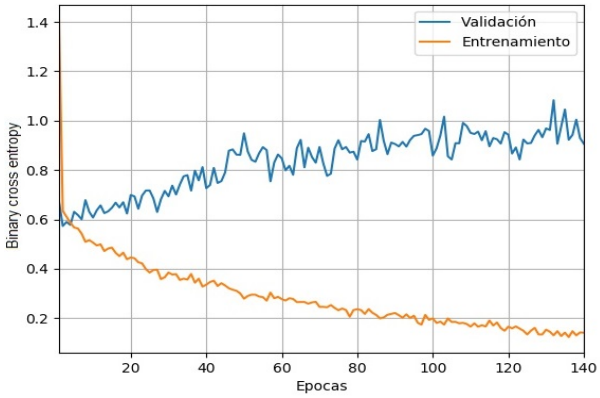




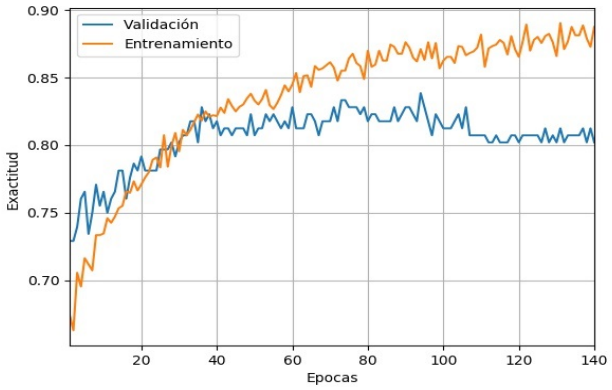


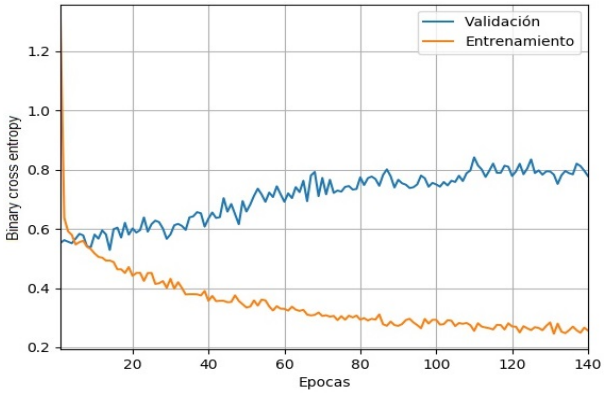
Experimento 3
Batch size = 16
Tasa de aprendizaje = 0.00002
Optimizador = SGD con momento Nesterov

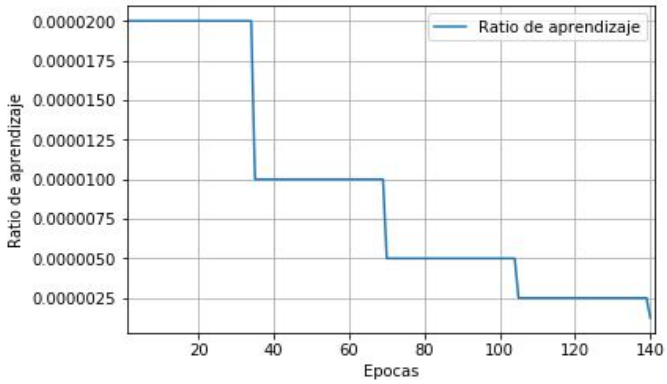




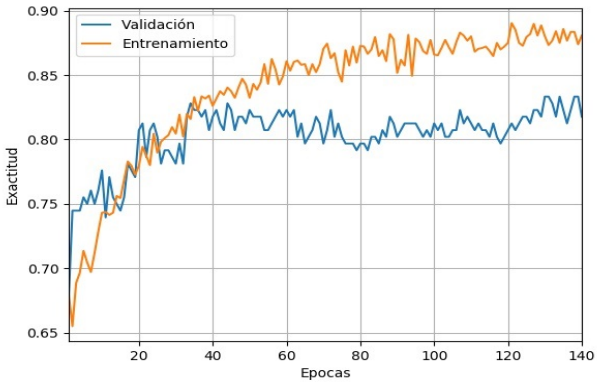
Experimento 4
Batch size = 16
Tasa de aprendizaje = 0.00002
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

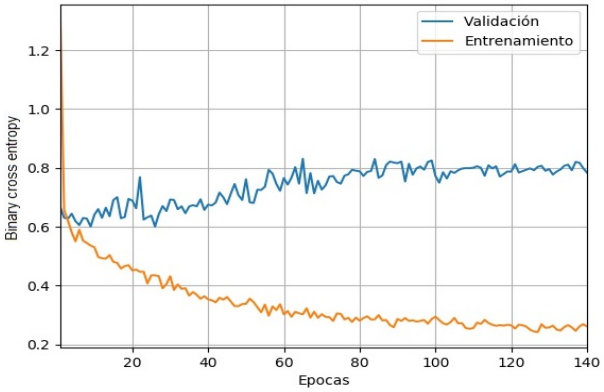




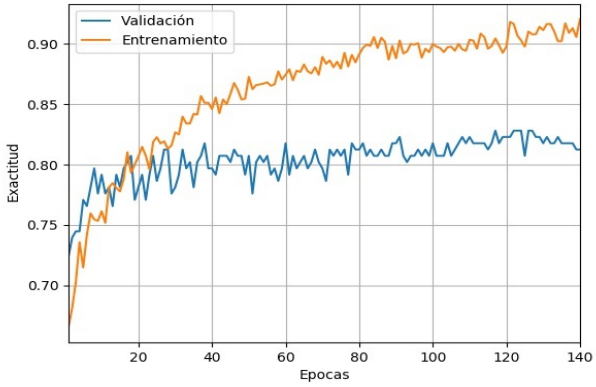


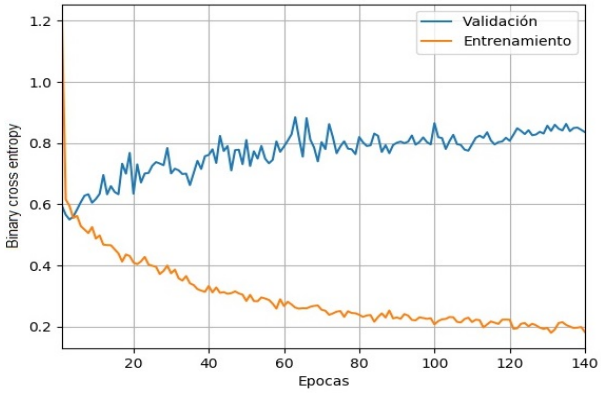
Experimento 5
Batch size = 16
Tasa de aprendizaje = 0.00003
Optimizador = SGD con momento Nesterov

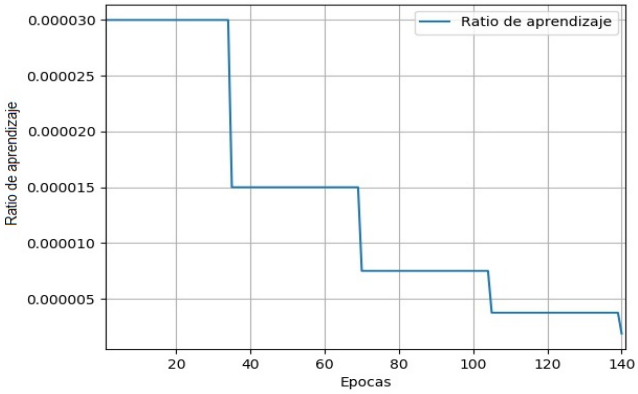




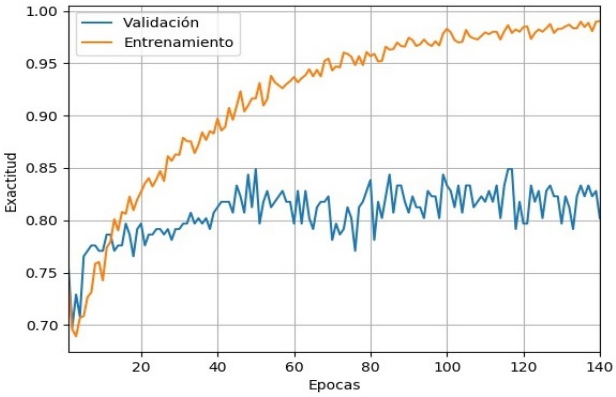
Experimento 6
Batch size = 16
Tasa de aprendizaje = 0.00003
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

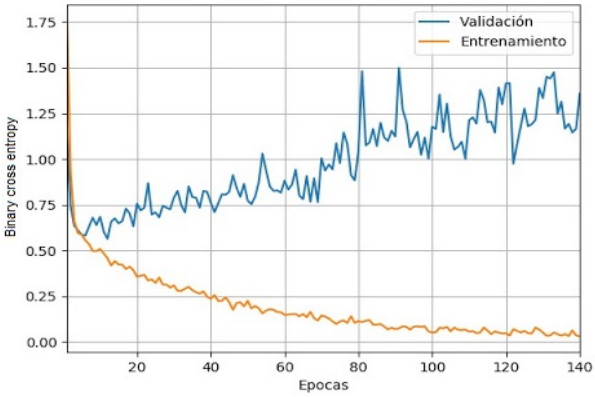




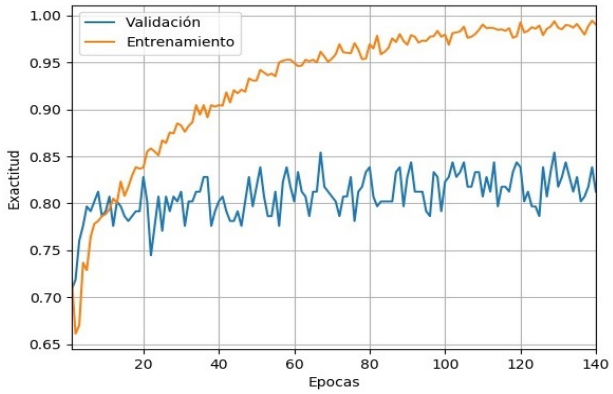


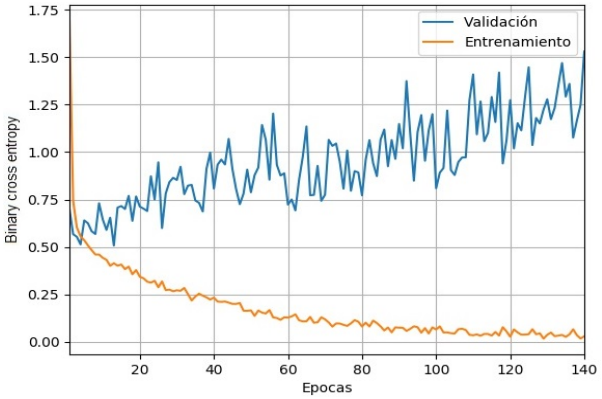
Experimento 7
Batch size = 16
Tasa de aprendizaje inicial = 0.00001
Optimizador = Adam



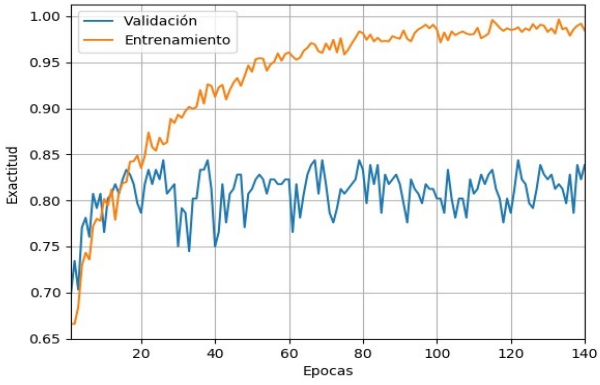


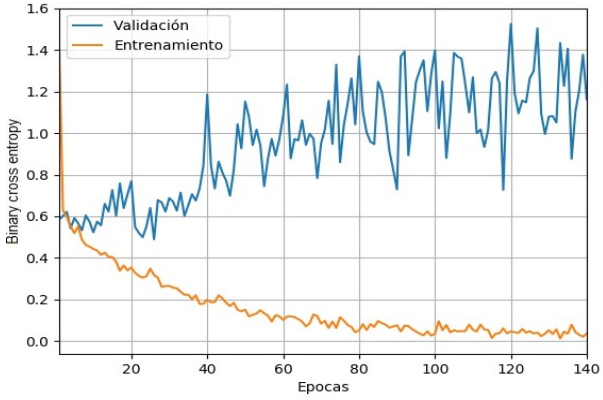
Experimento 8
Batch size = 16
Tasa de aprendizaje inicial = 0.00002
Optimizador = Adam



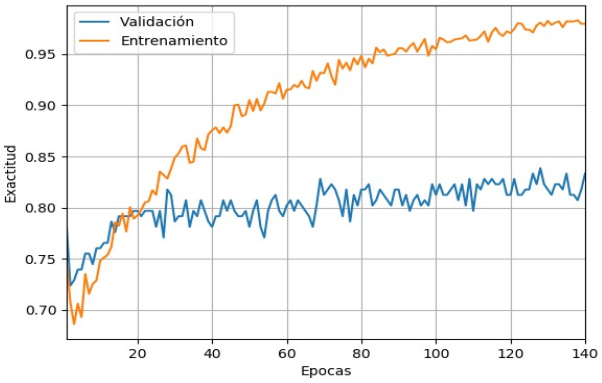


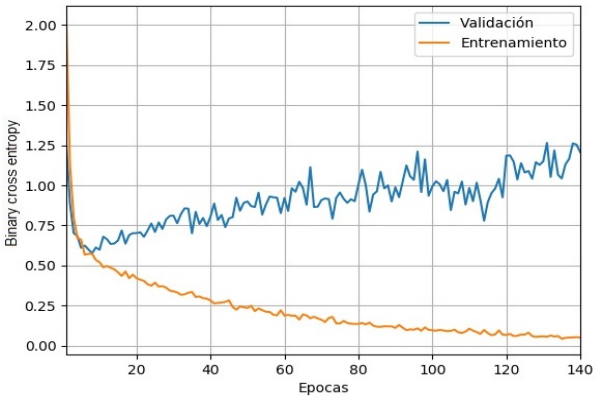
Experimento 9
Batch size = 16
Tasa de aprendizaje inicial=0.00003
Optimizador = Adam



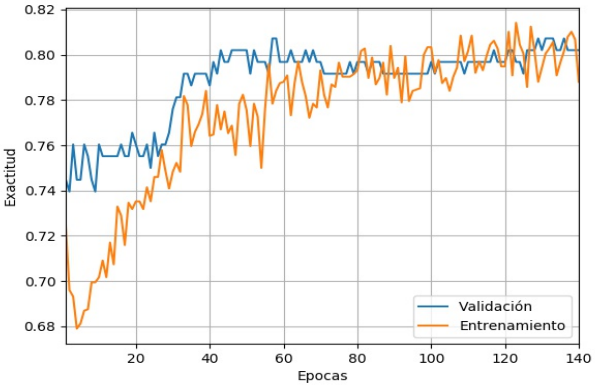


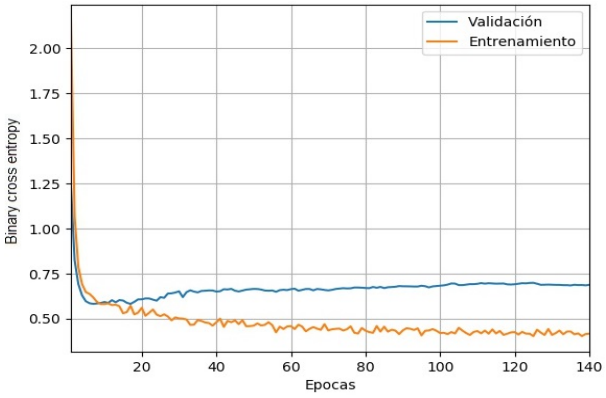
Experimento 10
Batch size = 32
Tasa de aprendizaje = 0.00001
Optimizador = SGD con momento Nesterov

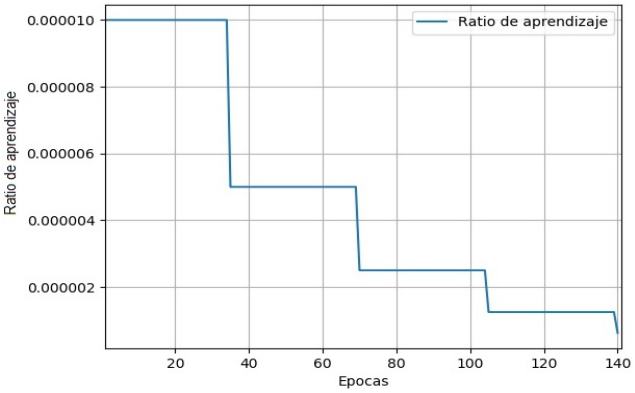




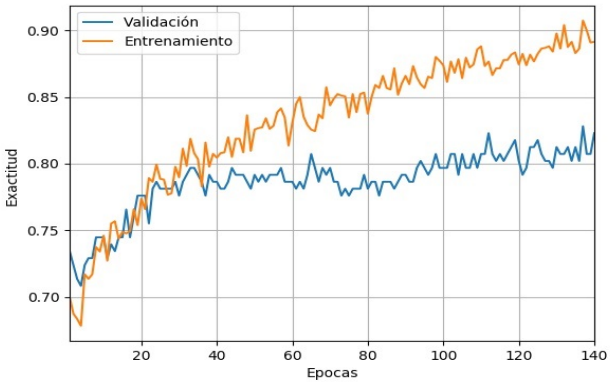
Experimento 11
Batch size = 32
Tasa de aprendizaje = 0.00001
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

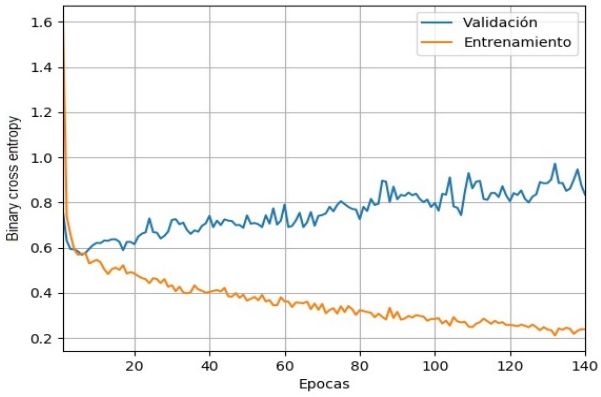




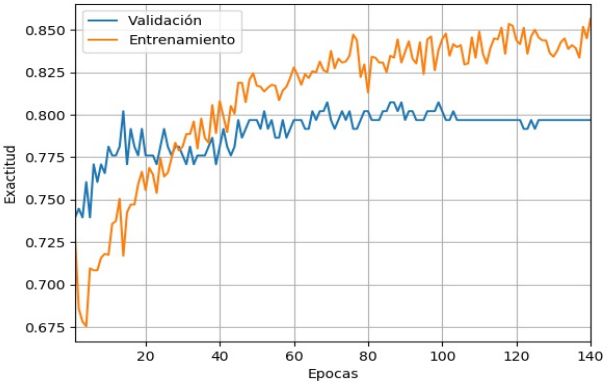


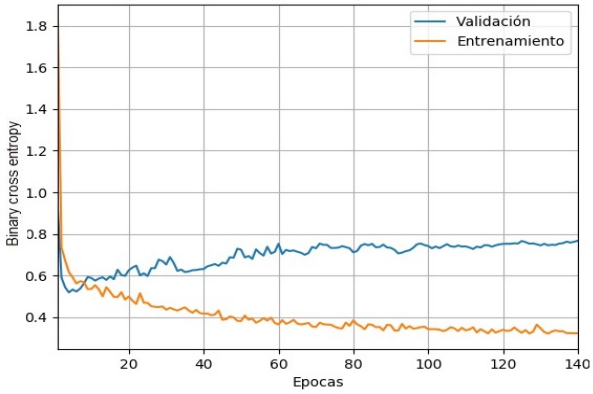
Experimento 12
Batch size = 32
Tasa de aprendizaje = 0.00002
Optimizador = SGD con momento Nesterov

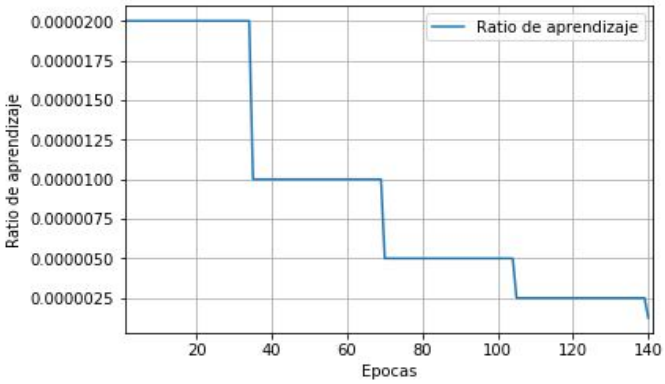




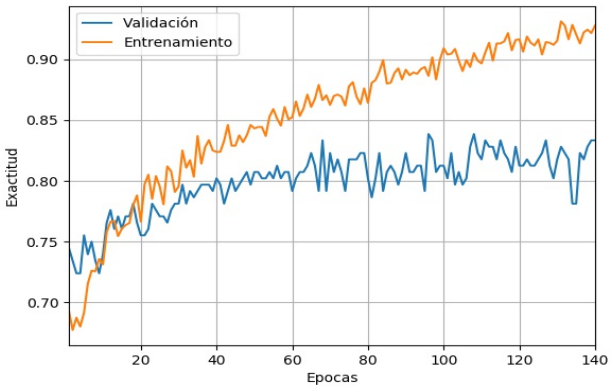
Experimento 13
Batch size = 32
Tasa de aprendizaje = 0.00002
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

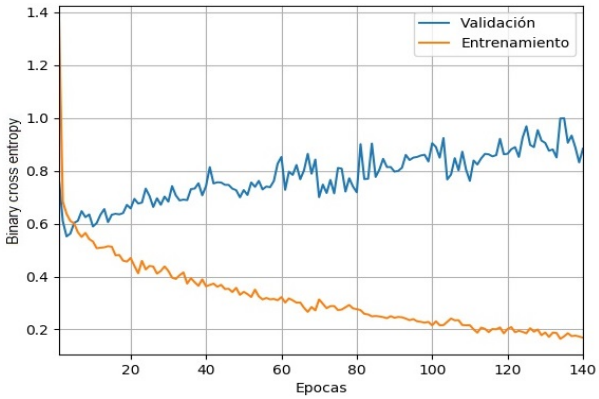




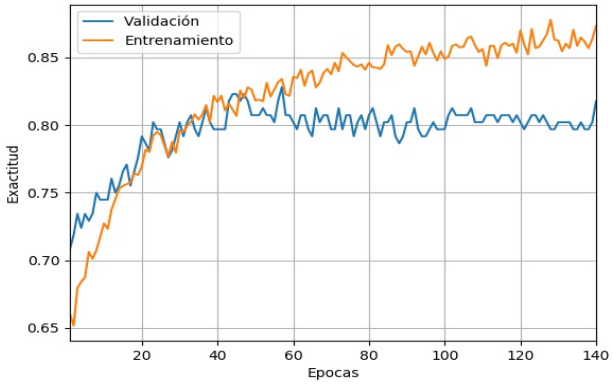


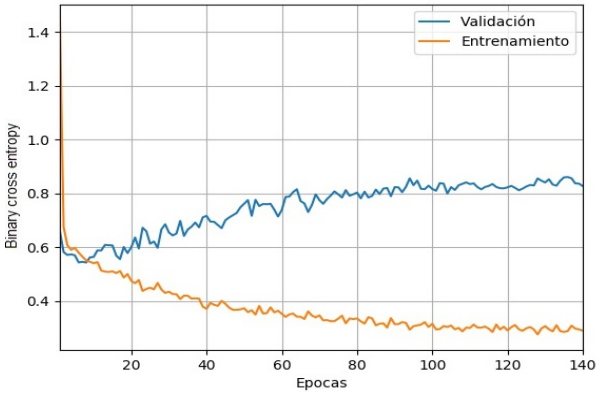
Experimento 14
Batch size = 32
Tasa de aprendizaje = 0.00003
Optimizador = SGD con momento Nesterov

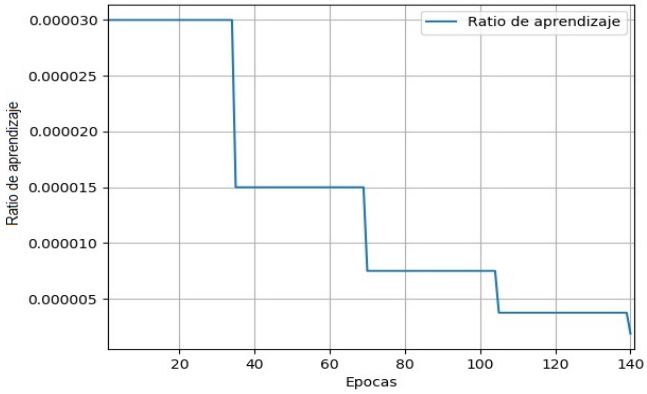




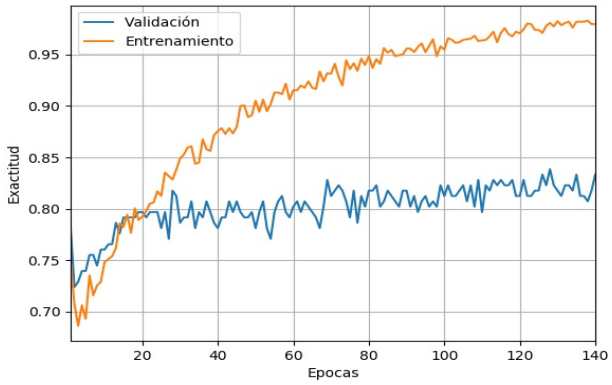
Experimento 15
Batch size = 32
Tasa de aprendizaje = 0.00003
Step decay = 50% cada 35 épocas
Optimizador = SGD con momento Nesterov

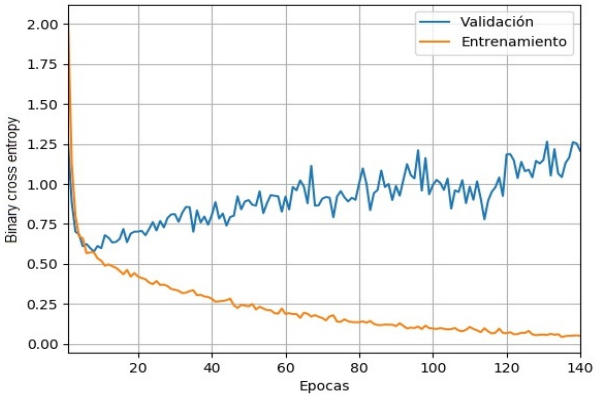




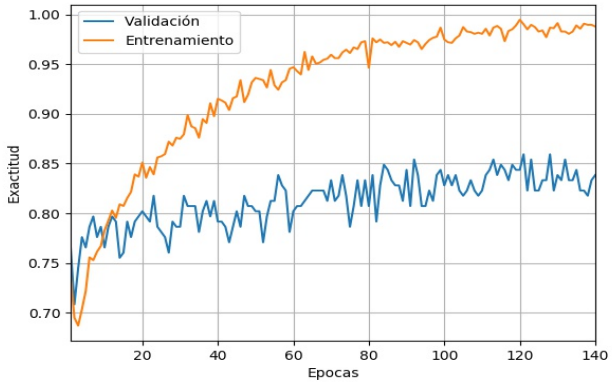


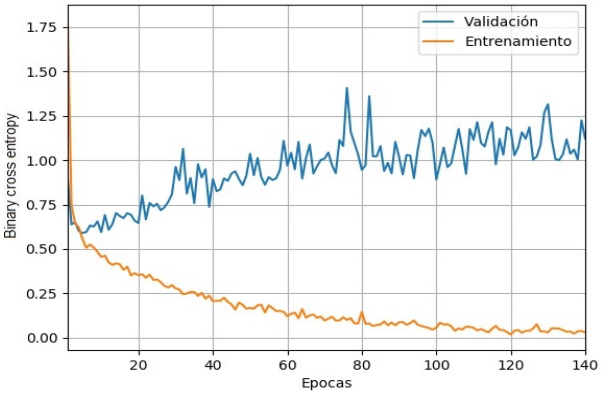
Experimento 16
Batch size = 32
Tasa de aprendizaje inicial = 0.00001
Optimizador = Adam





Experimento 17
Batch size = 32
Tasa de aprendizaje inicial = 0.00002
Optimizador = Adam





Experimento 18
Batch size = 32
Tasa de aprendizaje inicial = 0.00003
Optimizador = Adam