
Informe Tarea 2

Procesamiento distribuido y redes neuronales profundas

Fabián Badilla¹ Francisco Vásquez¹ Javier Santibáñez¹
Nicolas Caro² Rodrigo Lara³

Abstract

Se trabajó el problema de clasificación binaria de imágenes torácicas de rayos X, donde se buscó predecir si la imagen corresponde a neumonía o no. Además, se implementó un modelo de interpretabilidad local (LIME) sobre la red VGG y la red InceptionV3.

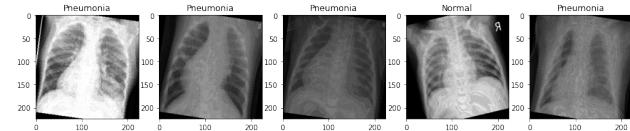


Figura 1.1. Imágenes del DatasetFolder del conjunto de entrenamiento.

1. Carga y transformación de datos

En esta sección se describe la carga y transformación de los datos, además de la creación de los objetos necesarios para luego entrenar el modelo. En la base de datos, se tienen 5232 imágenes en el conjunto de entrenamiento y 624 en el de prueba. Para optimizar el tiempo de carga de los datos se construye un generador y como se tienen pocos datos se emplean técnicas de aumentación de datos.

Primero, se crea la función `loader` que carga una imagen de la base, mediante la librería `PIL`, y si es de 1 canal la pasa a 3 canales. Luego, se instancian transformaciones mediante la librería `torchvision` que se detallan a continuación:

- Escala la imagen a un tamaño de 224×224 pixeles.
- Escala los valores de brillo de los pixeles a valores entre 0 y 1.
- Con probabilidad $\frac{1}{2}$, volteá la imagen en el eje horizontal.
- Rota la imagen, con respecto a su centro, con un ángulo aleatorio entre -20° y 20° .
- Multiplica los valores de brillo de cada canal por un número aleatorio entre 1.2 y 1.5. Donde cada pixel es multiplicado por un número potencialmente distinto.
- Transforma las imágenes a Tensor. Esto pues después es más fácil de trabajar con otros objetos de la librería.

Con esto, se instancian objetos de la clase `torchvision.datasets.DatasetFolder` el cual carga las imágenes aplicándole las transformaciones anteriores. Se pueden observar las transformaciones obtenidas en la Figura 1.1. Como se puede apreciar, para cada imagen se realizan las mismas transformaciones solo que con distintos valores, es decir, varían los valores de

brillo y el ángulo de rotación, entre otros, como era de esperarse.

Asimismo, se crean la función de carga de imagen y las transformaciones mediante la librería `skimage`, esto pues se desea estudiar el tiempo de cómputo para ambas librerías. Mediante perfilamiento de tiempo se obtiene que usando `torchvision` una imagen demora aproximadamente 160 ms ± 1.6 ms en cargar, por otro lado usando `skimage` esta demora 3.19 s ± 77.8 ms aproximadamente. Por lo tanto utilizar las transformaciones mediante `torchvision` es más óptimo para implementar lo deseado. Esto puede deberse a que esta librería tiene las transformaciones más dedicadas y más variedad de estas, en cambio `skimage` no posee todas las transformaciones que se deseaban por lo que se tuvo que usar `numpy` en algunas, lo cual influye bastante en el tiempo de carga.

Se crea la función `suma_clases`, la cual dado un `DatasetFolder` retorna la cantidad de datos en clase 0 y clase 1. Con esto, se hace un gráfico de barra para visualizar de mejor manera la cantidad de muestras de cada clase y de cada conjunto, el cual se muestra en la Figura 1.2.

Se observa que en el conjunto de entrenamiento hay una diferencia muy grande entre las imágenes normales y aquellas con neumonía, a diferencia del conjunto de prueba donde las cantidades son bastante cercanas. Esto se puede deber a que la gente que se hace esta radiografía es porque ya sospecha que puede tener neumonía, por eso la cantidad para esta clase es mayor. Además, al tener más imágenes de clase 1 en el conjunto de entrenamiento puede ocurrir que el modelo a entrenar quede sesgado hacia esa clase.

Debido a este desbalance en las clases para los conjuntos de train y test, se asume que la distribución del problema es la presentada en el conjunto test, es decir, en un entorno de producción se espera recibir muestras distribuidas de manera similar a tal conjunto. Por esto, se separan los

¹Integrantes grupo 8 ²Profesor ³Auxiliar .

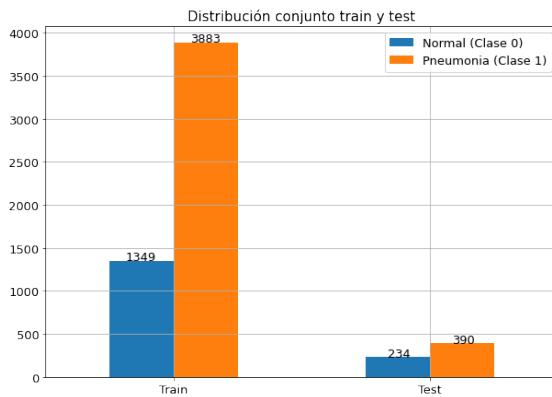


Figura 1.2. Distribución conjunto de entrenamiento y de prueba para ambas clases

índices del `DatasetFolder` asociado a la carpeta `train` en un conjunto de entrenamiento y otro de validación, además de construir la clase `ReplicarMuestreoDePrueba` la cual permite iterar sobre el conjunto de validación de tal forma que replica la distribución de clases del conjunto de prueba, mediante un sobremuestreo de la clase minoritaria.

Se tiene que en el conjunto de test la probabilidad de ser 1 es de 0.6250 e inicialmente en el conjunto de validación este valor corresponde a 0.7612. Para tener una menor diferencia se instancia un objeto de la clase anterior construida y así al conjunto de validación se le agregan 227 índices (repetidos de los que ya hay). Con esto se tiene que la probabilidad de ser 1 corresponde a 0.6256 aproximadamente, es decir, la clase funciona como se desea.

Finalmente, se instancian objetos de la clase `torch.utils.data.DataLoader` para recorrer los conjuntos de entrenamiento, validación y prueba, donde se entregan los índices separados en entrenamiento y validación además de la clase creada donde corresponde. Esta clase también recibe como parámetro `num_workers`, que corresponde al número de subprocesadores por usar para la carga de los datos, como `default` es 0 que significa que van a ser cargados en el procesador principal. Además, para comparar tiempos de cómputo de este parámetro, se crea la función `DL_num_workers` la cual recorre una vez el `DataLoader` asociado al conjunto de `train`, usando la cantidad de `num_workers` entregada. Los resultados de tiempo fueron obtenidos en Google Colab después de varias iteraciones y se muestran en la siguiente tabla.

<code>num_workers</code>	Tiempo (s)
0	5.8
1	7.49
2	5.71
3	4.5
4	5.14

Tabla 1.1. Tiempo de cómputo para diferentes `num_workers`

De la tabla se tiene que al usar un sólo subprocesador este demora más que al usar el procesador principal, lo cual era de esperarse ya que los subprocesadores tienen menor capacidad de cómputo. Pero al aumentar la cantidad de `num_workers` se consigue optimizar el proceso de carga al hacerlo de manera paralela en la CPU, logrando un mejor tiempo que usando el procesador principal.

2. Redes convolucionales profundas

Con el objetivo de clasificar las imágenes de rayos X, de manera que se pueda distinguir entre pacientes con neumonía y sanos, se construye una red neuronal profunda. Esto se realiza en diferentes etapas, las cuales se describen a continuacion.

Separación Depthwise

Mediante la librería `Pytorch`, se implementa una capa que básicamente separa el proceso de una capa de convolución estándar en 2 partes; *Depthwise* y *Pointwise*. Dada una capa de convolución con k filtros de tamaño $n \times n \times c$, con c el número de canales, se definen las capas:

- **Depthwise:** Con c filtros de tamaño $n \times n \times 1$ generando un output de c canales
- **Pointwise:** Con k filtros de tamaño $1 \times 1 \times c$ aplicado al output de la capa anterior. Sobre esta capa se aplica la función de activación.

Con esto se obtiene el mismo volumen de salida que se tendría en el caso original, pero necesitando mucho menos parámetros. De esta manera se gana en tiempo de ejecución, pero con el costo de perder algo de precisión.

La implementación de esta capa, se realiza creando la clase `DWSepConv2d`, la cual hereda los métodos de `torch.nn`, anulando `__init__` de manera que reciba los parámetros `in_channel`, `out_channel`, `kernel_size`, `padding`, `bias`, los cuales son autoexplicativos. También se anula `forward` siguiendo la estructura antes mencionada, a partir de los parámetros originales.

Creación de la red convolucional profunda

En esta parte, se define la arquitectura de la red neuronal en la clase `VGG16DWSep`, la cual utiliza la capa `DWSepConv2d` entre otras, siguiendo la estructura de la Tabla A.2.1. Esta clase se define nuevamente a partir de `torch.nn` anulando sus métodos, para incorporar la arquitectura deseada. Luego con la función `summary` de `torchsummary`, se ve cuantos parámetros posee la red al usar `DWSepConv2d` obteniendo 104,196,546 parámetros entrenables. Mientras que la red `VGG16_Conv`, que utiliza en cambio capas de convolución normales, reporta 110,925,634 parámetros necesarios, aproximadamente un 6.5% mas. Cabe destacar, que en ambas redes, la mayor parte de los parámetros son de la primera capa `Linear` que reduce 100,352 valores a 1,024. Esto genera unos 102,761,472 parámetros, ya que `Linear` es una capa *fully connected*.

Entrenamiento de la red

Para entrenar la red, se importa un modelo VGG16 pre-entrenado en *ImageNet* y se transfieren los pesos de sus dos primeras capas a la red construida. Luego, estos parámetros (38,774) se dejan constantes durante la fase de entrenamiento. De esta manera se pretende mejorar la curva de aprendizaje de nuestra red, pues estas dos capas detectan bien las características principales de las imágenes. Una vez transferido los pesos, se emplea la heurística descrita en listing A.3.1, mediante la creación de la clase *EarlyStopping*. Esta clase recibe como *inputs* los siguientes parámetros:

- **modo:** Indica si es mejor que el valor de pérdida suba o baje ('max' o 'min')
- **paciencia:** Entero que indica cuanta paciencia tiene la heurística.
- **porcentaje:** Booleano que indica si se debe medir el cambio en porcentaje o en valor absoluto
- **tol:** Tolerancia admitida por la heurística en los valores.

El método *mejor* de la clase, sirve para discernir si el valor actual es mejor que todos los registrados, permitiendo también guardar el valor evaluado o no. A partir de él se construye el método *deberia_parar*, el cual revisa si la métrica actual es mejor a las registradas y va actualizando las "vidas" del proceso. Así chequea si se ha agotado la paciencia o no, retornando *True* o *False* respectivamente.

Se utilizan algunas funciones vistas en clases como *loss_batch* y *register*, que sirven para calcular pérdidas promedio y su desviación estándar por *batches*. Se modifica la función *fit* vista en clases, de manera que ahora incluya la heurística de regularización, además de otras utilidades, como calcular métricas *f1* y *accuracy* de manera global, en cada iteración del proceso, todo esto es guardado en un *DataFrame* que contiene toda la información del proceso de aprendizaje.

Se agrega la función *evaluar* que calcula los vectores de probabilidad entregados por el modelo y la clase más probable, de manera que sea más legible al momento de predecir. Se inicializan la función de perdida y el algoritmo optimizador, en este caso *cross_entropy* y *Adam* respectivamente con parámetros *lr=1e-4* y *weight_decay=1e-5*.

Por último, se traspasa el modelo y las imágenes a la GPU para obtener el máximo rendimiento y se entrena el modelo, obteniendo los resultados reportados en la Figura 2.1, luego de realizar 23 iteraciones en un total de 1 hora y 26 minutos.

Se guardan los mejores pesos registrados en el proceso, en un archivo *modelo.h5* para uso futuro. También se generan los gráficos de la Figuras A.1.1 y A.1.2, donde se observa como van mejorando las métricas *accuracy* y *f1* en los conjuntos de entrenamiento y validación, a través de las épocas.

Luego se evalúa en el conjunto de test, para ver como se comporta la red a la hora de predecir, obteniendo los resul-

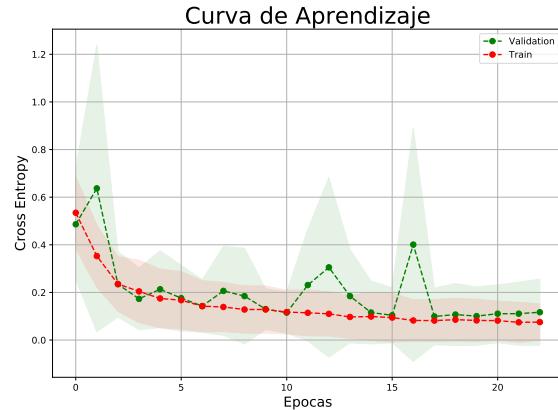


Figura 2.1. Curva de aprendizaje de la red neuronal profunda

tados de la Tabla 2.1. Estos valores son coherentes con lo esperado, por enunciado.

MÉTRICA	VALOR APROX
ACCURACY	0.755
F1	0.835

Tabla 2.1. Resultados reportados por la red VGG16DWSep en el conjunto de prueba

Evaluación alternativa

Finalmente, se implementa aumentación de datos en el conjunto de prueba, con el fin de mejorar aún más la predicción. Para ello, dada una observación x_i se generan d muestras $\{x_i^{(j)}\}_{j=1}^d$ a través de las mismas transformaciones detalladas en la primera parte del informe. Esto se lleva a cabo con la clase *AumentaSample*, la cual de manera aleatoria obtiene índices del dataset y los repite d veces, de esta manera el *DataLoader* ocupa una misma observación d veces, las transforma y obtiene las muestras requeridas. Luego se calcula $y_i = \arg \max_{k=0,1} \sum_{j=1}^d \mathbb{P}(x_i^{(j)} \in C_k)$ mediante el uso de *sum*, *argmax*, *softmax* de *torch*, implementadas en la función auxiliar *evaluar2* y finalmente se obtienen los resultados de la Tabla 2.2, donde se observa una leve mejoría. Cabe destacar, que este proceso de evaluación demora d veces más que el original.

MÉTRICA	VALOR APROX
ACCURACY	0.760
F1	0.842

Tabla 2.2. Resultados reportados al usar la segunda técnica de evaluación

3. Interpretabilidad

La finalidad de esta sección fue implementar un modelo auxiliar de interpretabilidad local que nos permitiera entender de mejor forma la importancia de las variables en los procesos de predicción. El método implementado es

el método Local Interpretable Model-Agnostic Explanations (LIME).

Lo primero a realizar es componer una serie de transformaciones con la ayuda de funciones del módulo `torchvision.transforms.Compose` las cuales nos permitirán ajustar las imágenes para usar el modelo InceptionV3, modelo de clasificación de imágenes entrenado sobre el famoso dataset *ImageNet*, con tal de poder aplicar el método LIME sobre él. Se reescalía la imagen con la función `resize` a un tamaño de 299 píxeles de ancho y alto, se corta la imagen con `CenterCrop` en 299 igualmente, se transforma a tensor con `ToTensor` y se genera estandariza la imagen con el uso de `Normalize` con medias [0.485, 0.456, 0.406] y desviaciones estándar de [0.229, 0.224, 0.225].

Se carga la red `inception_v3` desde la librería `torchvision.models` preentrenada. Al trabajar en Colab fue posible hacer uso de la GPU, por lo cual el `device` en el cual se trabajó tanto en el modelo como las variables fue `cuda`. Se fijan los gradientes del modelo para usarlo solo en evaluación.

Para la selección de la imagen, se procuró utilizar una imagen que fuese de interés para el estudio, en imágenes anteriores a la seleccionada, ocurría que la InceptionV3 tenía muy buenos resultados aún cuando la imagen se encontraba perturbada, además, se decide por no utilizar una imagen con fondo blanco dado que tampoco es muy real su utilización. La imagen de control seleccionada es una imagen de un tigre con un fondo no homogéneo que se puede observar como la *Imagen Original* en la Figura 3.1, también se puede observar uno de los 3 canales obtenidos en su transformación como *Canal N°1 Transformación* en la misma figura. Con la imagen ya transformada, se procede a realizar una predicción sobre la imagen con el modelo InceptionV3, la predicción obtenida del modelo es decodificada con la ayuda de la función `decode_prediction` de la librería keras. Los resultados se pueden observar en la Tabla 3.1

Label Predecido	Porcentaje
Tigre	9.2
Gato tigre	7.7
Lince	2.4
Transbordador	2.1
Jaguar	1.9

Tabla 3.1. 5 primeros resultados de predicción de Inception_V3 sobre imagen de control transformada

Se puede observar que salvo por la etiqueta transbordador que hace referencia a un transbordador espacial todas hacen referencia a la familia de felinos o similares y que la etiqueta top 1 es la de un tigre, por lo cual se tiene que el modelo predice bien la imagen de control.

3.1. Segmentación Slic

Para poder estudiar bien el comportamiento de las variables, es necesario generar perturbaciones sobre la imagen. Estas perturbaciones se realizarán segmentando la imagen para después apagar los segmentos. Para realizar la segmentación de la imagen se utilizó la función `slic` del módulo `skimage.segmentation` sobre la imagen de control solo escalada. Para visualizar los segmentos obtenidos, se utilizó la función `mark_boundaries` del mismo módulo mencionado anteriormente con un número de segmentos igual a 80 (`n_segments = 80`), la visualización se encuentra en la Figura 3.2 en donde se puede observar la imagen de control escalada junto con su segmentación marcada por líneas amarillas. Notamos que la segmentación obtenida no logra reconocer grandes parches de colores y tiende a generar parches de igual tamaño.

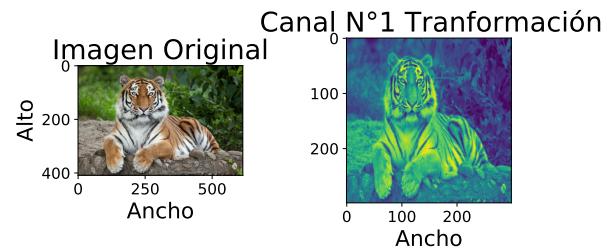


Figura 3.1. Imagen de control y su transformación (se muestra solo un canal).

Con los segmentos obtenidos anteriormente, se procedió a realizar perturbaciones sobre estos apagando algunos superpíxeles, traducido en colocar todos los canales de la imagen definidos por un segmento en 0. Dado el número de superpíxeles definidos por `slic`, se crean vectores aleatorios de largo el número de segmentos definidos por una variable aleatoria Bernoulli con $p=0.5$, de estos vectores, se crean un número de vectores igual a `n_perturbaciones = 2000` y se genera una matriz de estos, donde las columnas corresponden a los vectores definidos para perturbar una imagen. Con lo anterior, se generan 2000 imágenes perturbadas. Se puede observar una imagen perturbada y su transformación perturbada en la Figura 3.2, en donde es posible observar manchas las cuales corresponden a estos segmentos apagados.

Con el vector de imágenes perturbadas creado, se realizó una predicción sobre cada una de las imágenes con el modelo `inception_v3`, si es que el top label predecido es igual al obtenido sobre la imagen de control (Tigre) se guarda un 1 en un vector, en caso contrario, un 0. Lo anterior creó un vector binario de largo 2000, llamado y . Este vector cuenta con 1409 casos positivos (1) y 591 casos negativos (0)

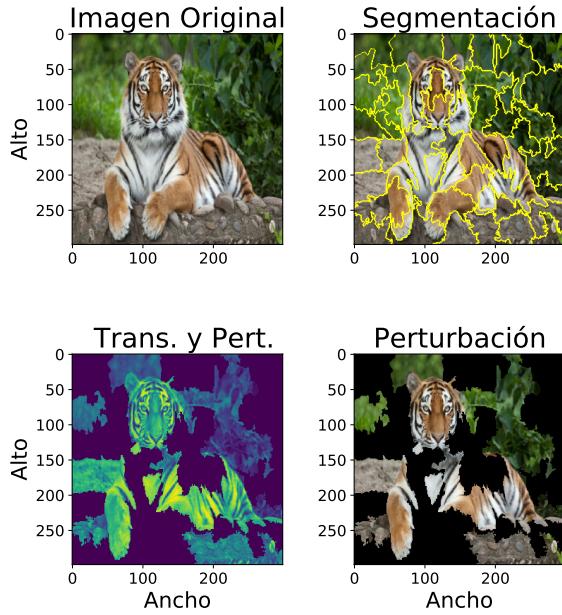


Figura 3.2. Imagen de control reescalada con su segmentación respectiva a través de la función `slic`. Se observa en la segunda fila una imagen perturbada y transformada, además de la reconstrucción de esta en RGB

Cada uno de los vectores de perturbación correspondientes a la columna de la matriz de perturbaciones son las representaciones interpretables de las imágenes perturbadas. El valor de representación x' en este caso corresponde al vector $x' = [1, \dots, 1]$ dado que es la representación interpretable completa de la imagen dados sus superpixeles. Para calcular la distancia de cada vector de representación de las perturbaciones z' correspondientes a las columnas de la matriz mencionada, se utiliza la distancia del coseno entre los vectores y el vector de representación completo de la imagen de control x' para obtener un vector de pesos de la siguiente forma:

$$\pi_x(z') = \exp\left(\frac{-1 - \left(\frac{x'z'}{\|x'\|\|z'\|}\right)^2}{\sigma^2}\right) \quad (1)$$

Con lo anterior, se procede a generar la minimización de la función de fidelidad. Para esto se considera la regresión logística como familia de explicaciones G. Definida la función, se considera como conjunto de entrenamiento la matriz de perturbaciones a la cual denotamos como D_p , la variable target a y y como vector de pesos sobre el conjunto de datos al vector π_x , además de seleccionar las clases balanceadas a través del parámetro `stratify` con el vector y . Lamentablemente, para la implementación de `LogisticRegression` que se encuentra en `scikit-learn` no es posible agregar una medida

de complejidad $\Omega(g)$ dado que la implementación solo minimiza localmente a la ecuación (2) del enunciado sin tener en consideración esta medida, para poder implementarla habría que modificar la implementación del modelo cambiando el método `fit`. Dados esto, se separan los datos en conjuntos de entrenamiento (80% de los datos) y validación (20% de los datos) para poder validar el valor predictivo del modelo entrenado, además de validar los resultados obtenidos. El accuracy ponderado obtenido fue de 0.91 y el f1-score ponderado fue de 0.91, con lo cual el modelo entrenado es bueno en nivel predictivo, a pesar de que las clases se encuentren desbalanceadas, por lo tanto los resultados obtenidos son de fiar.

Al ser todos los valores positivos del dataset completo (solo ceros o unos) se tiene que los coeficientes obtenidos se pueden interpretar fácilmente, esto se debe a que en la regresión logística, la función sigmoide que es la que genera las probabilidades respectivas de un dato a pertenecer a la clase positiva tiene en su denominador a $1 + \exp(-\beta X)$ en donde como todos los datos de X son positivos y la función exponencial biyectiva, continua y estrictamente creciente, la forma de que la función sigmoide tenga más probabilidad es que la exponencial sea más pequeña, esto se logra cuando los coeficientes son positivos y lejanos al cero. Se seleccionan los mejores coeficientes con respecto a lo anterior mencionado de la siguiente forma: $MC = \{j | \beta_j > \frac{1}{N} \sum_i |\beta_i|\}$. En donde solo se toma como referencia al promedio de los valores absolutos para considerar que sobre ese valor los coeficientes son buenos para predecir la categoría. Los coeficientes se pueden observar en la Figura A.1.5 del Anexo A, se tiene que hay 10 coeficientes que cumplen la condición anterior. Como cada coeficiente tiene un superpixel asociado, se obtienen los números de los segmentos correspondientes para poder visualizarlos. Con lo anterior, tenemos los superpixeles que mejor representan la imagen para diferenciar del caso positivo, los resultados se pueden observar en la figura Imp. Slic de la Figura 3.3. Se puede notar que en los superpixeles observados no aparece el tigre completo, ni siquiera su cara y tampoco aparecen solo segmentos asociados al tigre, esto nos da a entender que la red necesita, de cierta forma, un contexto de la imagen además de poder observar el tigre. Además, que agregando la cara del tigre, se puede reconocer algún otra categoría, por lo cual no es tan buen predictor por si solo para el modelo.

3.2. Segmentación K-Means y Gaussian Mixture

Para poder comparar los resultados obtenidos con la segmentación realizada por la función `slic` se generan otras segmentaciones de la imagen con los métodos de clusterización de K-Means y GaussianMixture. La segmentación a realizar es una segmentación espacial, lo que quiere decir que la clusterización se lleva a cabo a través sobre la escala de grises según su posición en la imagen. No se escalan los datos dado que se encuentran en escalas similares. Para Kmeans se utiliza el método del codo so-

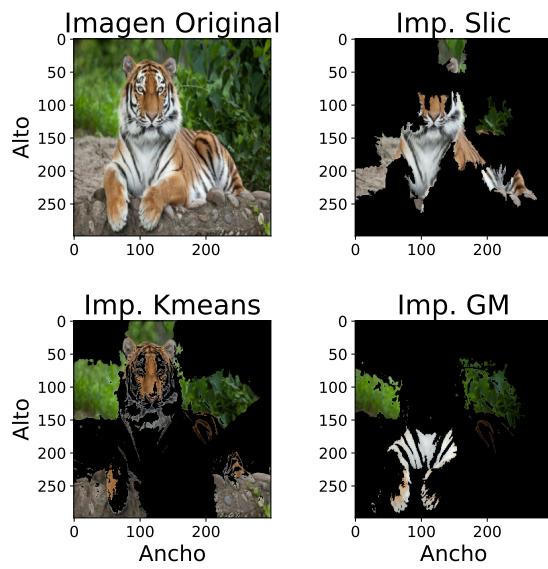


Figura 3.3. Imagen original junto con los segmentos más relevantes encontrados para cada segmentación

bre los WSS además del método de los promedios de las siluetas para poder obtener el número de clusters y para G.M. se utiliza el método del codo sobre su score BIC, para Kmeans se selecciona un número de 17 clusters y para G.M. se selecciona un número de 17 componentes. Las segmentaciones obtenidas se pueden observar en la Figura 3.4 y los métodos de selección de cluster en las Figuras A.1.3 y A.1.4 del Anexo A.

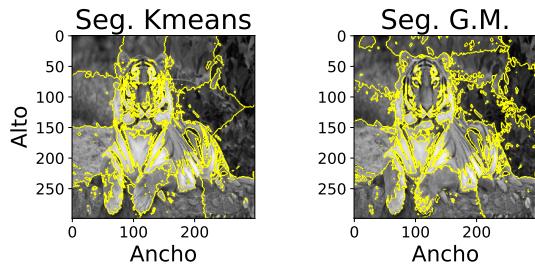


Figura 3.4. Segmentaciones obtenidas con el método KMeans y Gaussian Mixture

Se tienen que las segmentaciones obtenidas varían con las obtenidas por el método `slic`, esto se debe a que el color pondera más en estos casos con lo cual alguno de los segmentos que se tienen no son contiguos, agrupándose principalmente por color que por posición de los píxeles, lo anterior lleva a que en las partes homogéneas de la imagen, en este caso el fondo, se tiene que hay pocos segmentos asociados, en cambio para el tigre, se separa por colores, rayas, cara, etc. Lo anterior se debe también a como clusterizan

los modelos utilizados, KMeans ajusta bolas con la métrica euclíadiana, con lo cual los cambios en los colores se puede notar en la posición de estos en el espacio lo cual conlleva a que los colores ponderen de igual forma que la posición de los píxeles. Algo similar ocurre con Gaussian Mixture, que a pesar de ser un modelo basado en densidades, tiende a encontrar bolas o bolas achatadas en el espacio sumandole un poco de ruido atribuido a la probabilidad de estar lejos de la media, lo que se puede ver en la forma de los segmentos que son un menos definidos que para Kmeans. Para ambas segmentaciones se sigue el mismo procedimiento antes mencionado de generar las perturbaciones sobre la imagen con los superpixeles definidos, generar los conjuntos de entrenamiento y la variable target además de sus pesos y luego aplicar la regresión logística. Los superpixeles más importantes para el modelo se seleccionan de la misma forma descrita anteriormente para la segmentación con `slic`.

Para KMeans se obtiene que de las 2000 perturbaciones generadas sobre la imagen, 1738 fueron predecidas positivamente y 262 negativamente. Al realizar el entrenamiento y el testeо, se tiene que el accuracy obtenido fue de un 0.93, lo cual es bastante alto, sin embargo, los f1-score para la clase positiva fue de un 0.96 y de la clase negativa un 0.69, lo cual se debe al gran desbalanceo de clases que hay. La visualización completa de los valores de los coeficientes obtenidos se puede observar en la FiguraA.1.6 en el Anexo A. Se tiene que existen 7 coeficientes que cumplen la condición de estar sobre el promedio de los valores absolutos, los segmentos asociados se pueden observar en *Imp. Kmeans* en la Figura 3.3.

Para la segmentación realizada por Gaussian Mixture, se generan 2000 imágenes perturbadas de igual forma, en las cuales 1716 fueron predecidas correctamente y 284 no. Los resultados luego de aplicar la regresión logística tuvieron un accuracy de 0.97 y un valor de f1-score de 0.88 en la clase negativa y de 0.98 en la clase positiva, lo cuales son altos considerando el gran desbalanceo de clases que hay. La visualización completa de los valores de los coeficientes obtenidos se pueden observar en la FiguraA.1.7 en el anexo A. Se tiene que existen 3 segmentos de 20 que cumplen la condición de estar sobre el promedio de los valores absolutos, los segmentos asociados se pueden observar en *Imp. GM* en la Figura 3.3.

A modo de comparación de todos los resultados obtenidos, se tiene que mirando la Figura 3.3 se tiene que para las tres segmentaciones realizadas, los superpixeles más importantes son distintos y representan distintas partes de la imagen. Para `Slic` dado que las segmentaciones son más espaciales sobre la posición de la imagen, se tiene que se tiende a escoger una mayor parte del felino, que también viene de la mano con el número de segmentos que hay. Para KMeans las segmentaciones realizadas diferencian mucho por el color, por lo cual se puede notar que los segmentos que se ven en la imagen contienen pintas negras, pudiendo observar de buena forma la imagen de un león. Al contrario de Kmeans, GM solo pondera en gran parte en 3 de sus segmentos, que se puede notar que son las líneas significati-

vas blancas del león. En todas las segmentaciones existen segmentos asociados al fondo de la imagen, lo cual puede asociarse a que es necesario un contexto del individuo para que el modelo pueda diferencias entre casos. A lo anterior se suma, que en ningún caso se muestra la figura completa del león, solo partes de este.

3.3. Aplicación de LIME sobre VGG16

Se procedió a realizar el mismo esquema LIME realizado para la imagen y modelo anterior sobre la red VGG16Sep con separación Depthwise sobre una nueva imagen de control del dataset de Mendeley de imágenes de radiografías de neumonía.

Se ajustan las transformaciones realizadas a escalar la imagen en este caso a 244 píxeles de ancho y largo, además de ajustar el corte al mismo número. Los valores de estandarización de la imagen dejan iguales que la serie de transformaciones definidas al inicio de la sección. Se puede observar la imagen de control seleccionada en la Imagen Original de la Figura 3.5 y uno de sus canales con la transformación aplicada.

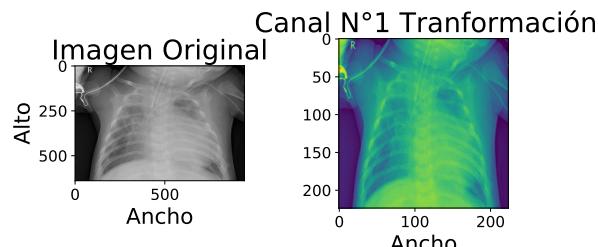


Figura 3.5. Imagen de control para el modelo VGG16 obtenida del dataset de Mendeley junto con una de sus transformaciones.

Se aplica la función `Slic` con un número de segmentos igual a 80 sobre la imagen de control para obtener nuevos superpixeles, los resultados se pueden ver en la imagen *Segmentación* de la Figura 3.6. Se generan 2000 perturbaciones de la imagen con los segmentos definidos anteriormente, se puede observar un resultado en la imagen *Perturbación* y su *Transformación* en la Figura 3.6

Sobre las 2000 imágenes perturbadas se generaron predicciones para crear el nuevo vector y , teniendo en este caso un total de 1372 casos positivos y 628 casos negativos. Dado el vector de pesos asociados a las perturbaciones, la matriz de datos de perturbaciones y la variable target asociada al vector y se aplica una regresión logística de la misma forma que con la segmentación `slic` en el modelo InceptionV3. Lo anterior da como resultado un total de accuracy de un 0.77 y f1-score sobre los casos negativos de un 0.56 y sobre los positivos de un 0.84. Los puntajes obtenidos son bastante bajos en comparación con los resultados anteriores, debiéndose principalmente al nuevo caso abordado además del desbalanceo de clases.

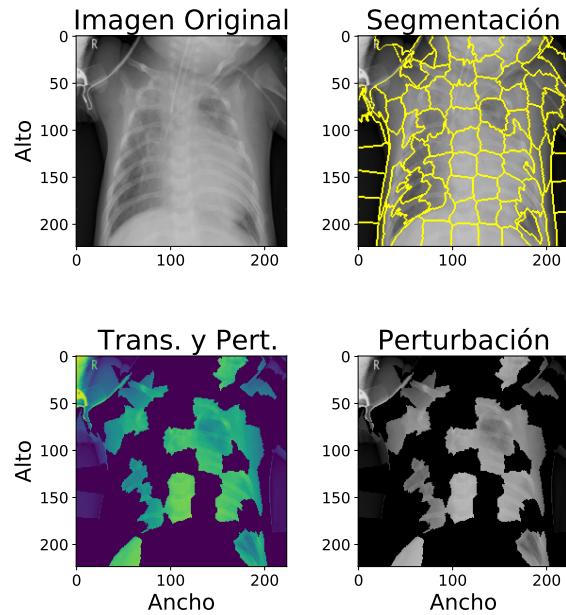


Figura 3.6. En la primera fila se encuentra la imagen de control junto con la segmentación obtenida con la función `slic`. En la segunda fila se encuentra una perturbación con una transformación y su imagen respectiva en RGB

Los valores de todos los coeficientes se pueden observar en la Figura A.1.8, en donde solo 6 coeficientes cumplen la condición de ser catalogados como buenos. Los segmentos asociados a los coeficientes anteriores se pueden observar en la Figura 3.7. Se tiene que es de gran importancia un segmento asociado a la zona del corazón del torax, el cual puede ser de gran importancia para el modelo VGG, además de 2 segmentos extremos a la zona de los pulmones. Sin embargo, también se pueden observar segmentos asociados a otras partes de la radiografía, los cuales a priori no es posible asociarlos a algo, salvo al contexto y posición de la imagen.

3.4. Segmentación por colores, Bonus

Se procede a realizar los mismos esquemas anteriores sobre una segmentación asociada a los colores de la imagen. A diferencia de las segmentaciones anteriores, se consideran ahora los 3 canales de la imagen asociados al RGB, es decir, se crea un conjunto de 299x299 datos con 5 variables, asociadas a las posiciones de los píxeles (2) y al RGB (3). Se aplica el modelo de KMeans sobre el conjunto anterior, se selecciona un número arbitrario de clusters, dado que en las segmentaciones anteriores no era claro un número óptimo de clusters, además, probando números superiores de clusters las segmentaciones eran muy buenas lo cual llevaba a que la imagen fuese clasificada el 100% de las veces bien, cosa que no es de interés en este caso. Se selecciona un número de 14 clusters, la segmentación obtenida de la imagen se

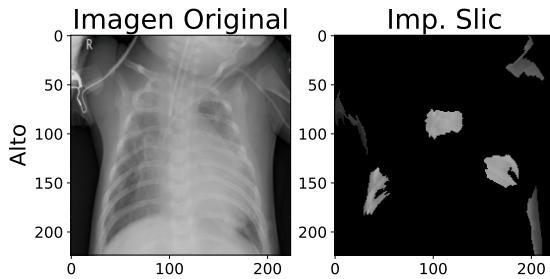


Figura 3.7. Imagen de control junto con los segmentos relevantes encontrados.

puede observar en la figura 3.8. Se tiene que en este caso las segmentaciones son mucho más separadas en cuanto a el color de la imagen, pudiendo separar de mejor forma las líneas blancas y cafés del tigre. Se generan 2000 perturbaciones sobre la imagen con los segmentos encontrados, se puede observar una imagen perturbada y su transformación en la Figura 3.8

Con las 2000 imágenes perturbadas se realizaron predicciones sobre ellas las cuales dieron como resultado 1698 imágenes correctamente etiquetadas y un total de 302 mal etiquetadas. Se calculan los vectores de pesos, se escoge la matriz de perturbaciones como dataset de entrenamiento y se selecciona el vector y generado con los labels bien y mal predecidos. Se escoge en esta oportunidad el modelo de clasificación XGBClassifier, modelo de la librería XGBoost. El modelo anterior pertenece a la familia de las funciones explicativas basadas en Random Forest, teniendo en su implementación una aceleración por gradiente, además de estar bien optimizado.

Se separa el dataset, la variable target y los pesos en entrenamiento y validación, con una proporción de 80/20. Sobre las predicciones en el conjunto de validación, se tiene un accuracy de 0.97 y f1-score de 0.89 para la clase negativa y 0.98 para la clase positiva, resultados que son muy superiores sobre los resultados obtenidos anteriormente, teniendo en cuenta el desbalanceo de clases que hay. En este caso, para poder medir las ponderaciones de las variables (segmento i encendido) es necesario utilizar otra métrica la cual es el Gain obtenido del atributo feature_importances_ del modelo XGBoost, que indica, a grandes rasgos, cuánto se gana en accuracy dado un split asociado a cada variable. Los resultados del Gain por

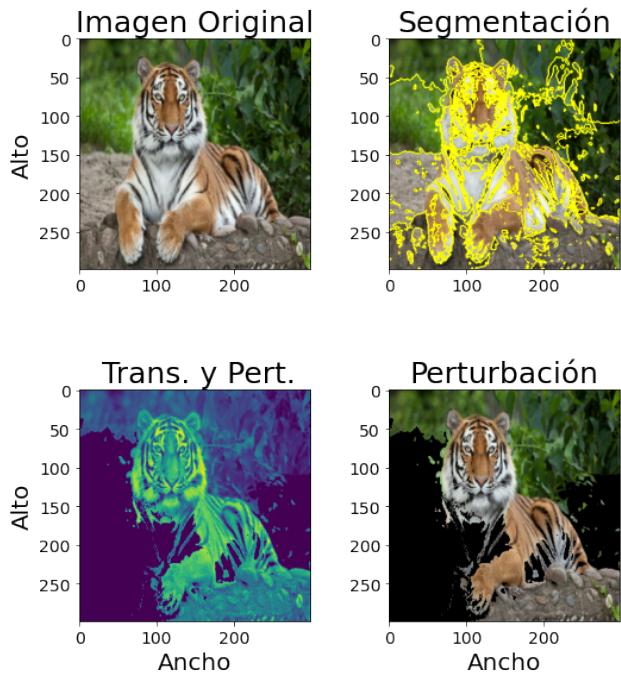


Figura 3.8. Imagen de control original junto con su segmentación obtenida con método de KMeans sobre escala de colores, en la segunda fila se encuentra una de sus perturbaciones son la transformación correspondiente

variable se puede observar en la Figura A.1.9 en el Anexo A.

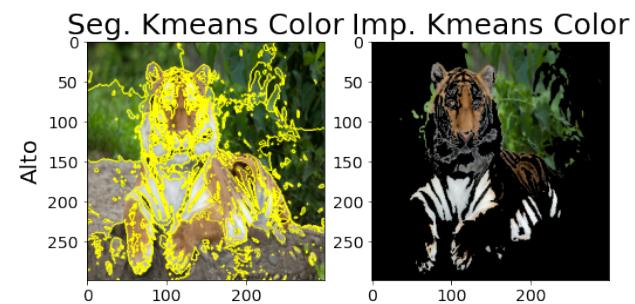


Figura 3.9. Segmentación de KMeans sobre la imagen junto con los segmentos más importantes encontrados con los coeficientes.

Para este caso, se consideran los 4 segmentos con los coeficientes más altos, los superpixeles asociados a estos se pueden observar en la figura 3.9. En la figura se puede observar de mejor forma como se realizó la clusterización, se puede ver que los colores blancos correspondientes a las líneas del tigre se encuentran en un solo segmento, que parte del lomo café se encuentra en otro y que se selecciona también parte del fondo de la imagen como contexto de la foto. En esta segmentación se tiene que se observa de mejor forma la figura de un tigre.

A. Anexo

A.1. Figuras

Figuras anexas, algunos títulos fueron removidos a propósito dado que se explica su contenido en el caption.

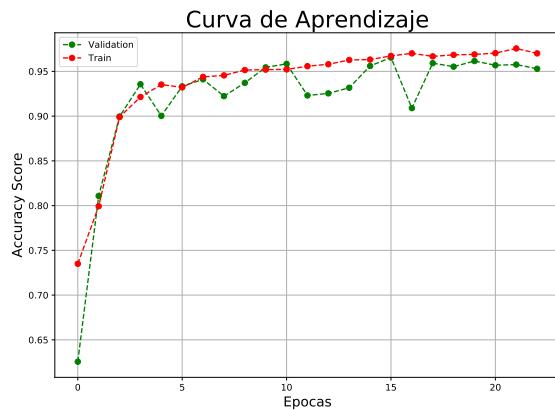


Figura A.1.1. Metrica *accuracy_score* a lo largo del proceso de aprendizaje

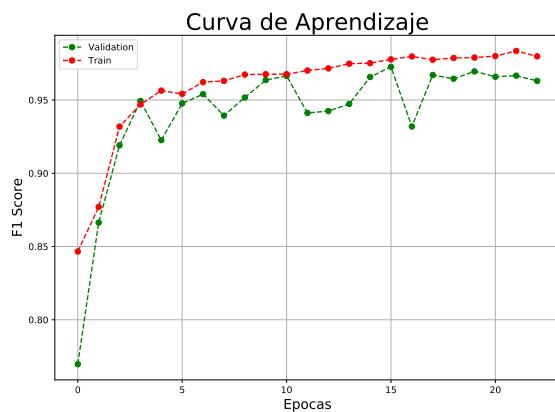


Figura A.1.2. Metrica *f1_score* a lo largo del proceso de aprendizaje

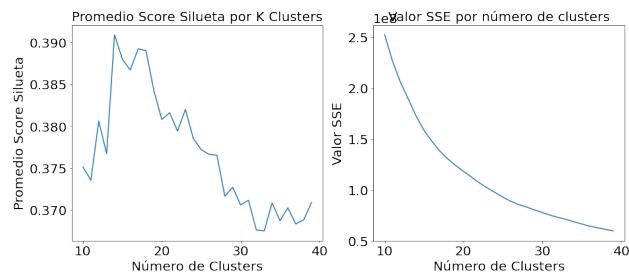


Figura A.1.3. Método del codo con WSS y Silhouette Score para KMeans.

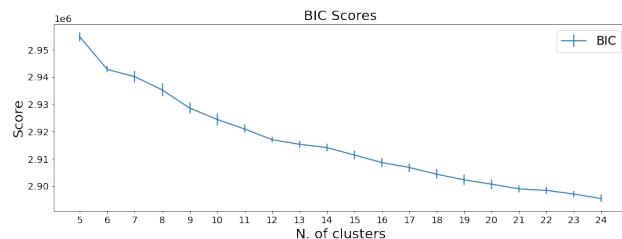


Figura A.1.4. Método del codo con BIC score para Gaussian Mixture.

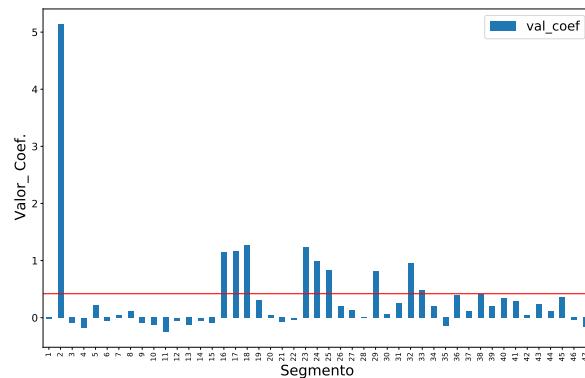


Figura A.1.5. Valores de los coeficientes de la regresión logística para LIME sobre imagen de control y **segmentación slic**, la línea roja indica el promedio de los valores absolutos de los coeficientes

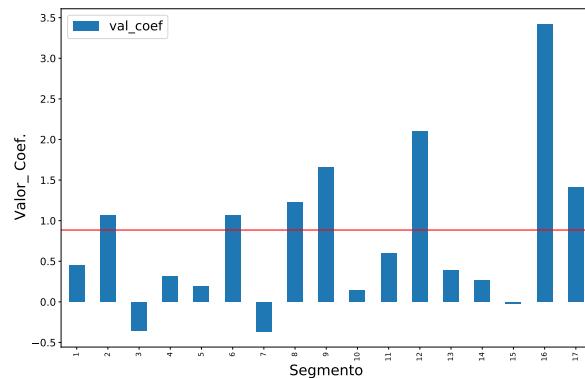


Figura A.1.6. Valores de los coeficientes de la regresión logística para LIME sobre imagen de control y **segmentación KMeans**

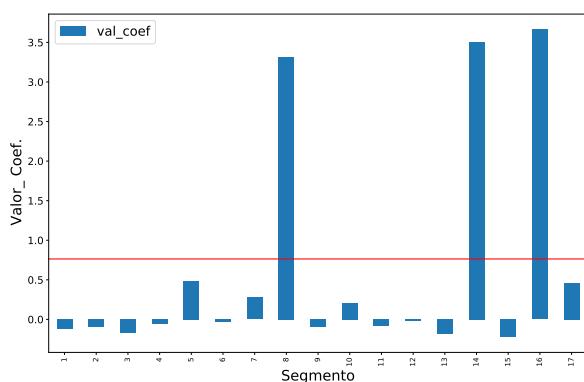


Figura A.1.7. Valores de los coeficientes de la regresión logística para LIME sobre imagen de control y **segmentación con Gaussian Mixture**

A.2. Tablas

CAPA	TAMAÑO FILTRO	PADDING	STRIDE	# FILTROS
CONV2D	3	1	1	64
CONV2D	3	1	1	64
MAXPOOL2D	2	0	2	-
DWSepConv2D	3	1	1	128
DWSepConv2D	3	1	1	128
MAXPOOL2D	2	0	2	-
DWSepConv2D	3	1	1	256
BATCHNORM2D	-	-	-	-
DWSepConv2D	3	1	1	256
BATCHNORM2D	-	-	-	-
DWSepConv2D	3	1	1	256
MAXPOOL2D	2	0	2	-
DWSepConv2D	3	1	1	512
BATCHNORM2D	-	-	-	-
DWSepConv2D	3	1	1	512
BATCHNORM2D	-	-	-	-
DWSepConv2D	3	1	1	512
MAXPOOL2D	2	0	2	-
FLATTEN	-	-	-	-
LINEAR	-	-	-	1024
DROPOUT(.7)	-	-	-	-
LINEAR	-	-	-	512
DROPOUT(.5)	-	-	-	-
LINEAR	-	-	-	2

Tabla A.2.1. Estructura de la red neuronal profunda implementado en la clase VGG16DWSep

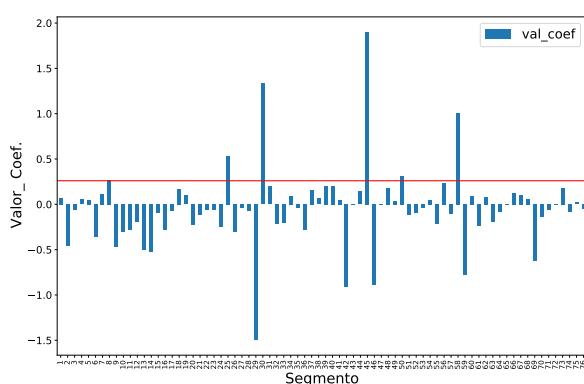


Figura A.1.8. Valores de los coeficientes de la regresión logística para LIME sobre radiografía y modelo VGG16 y **segmentación con Slic**

A.3. Listings

```

1 es = EarlyStopping (...)
2 for epoch in range(num_epochs):
3     # ciclo de entrenamiento
4     ...
5
6     # ciclo de validacion
7     ...
8     metrica_validacion = ...
9
10    if es.deberia_parar(metrica_validacion):
11        # se cumple el criterio de "early stop"
12        break

```

Listing A.3.1. Funcionamiento de la heurística Early Stopping

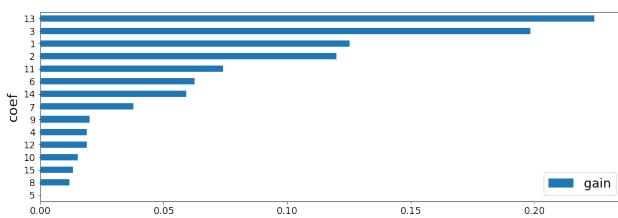


Figura A.1.9. Valores promedio Gain por variable para modelo XGBoost aplicado sobre la predicción de los segmentos