Tarea 2: Procesamiento distribuido y redes neuronales profundas

Camila Gómez Nazal Ignacio Zurita Tapia

***Resumen*—El presente informe tiene como objetivo resolver un problema de clasificación binaria de imágenes, utilizando redes neuronales mediante el framework pytorch, específicamente imágenes torácicas de rayos X de pacientes sanos y pacientes con neumonía. Se comienza dando una introducción al problema y en las secciones siguientes se detallan tanto el procedimiento, como los resultados obtenidos.**

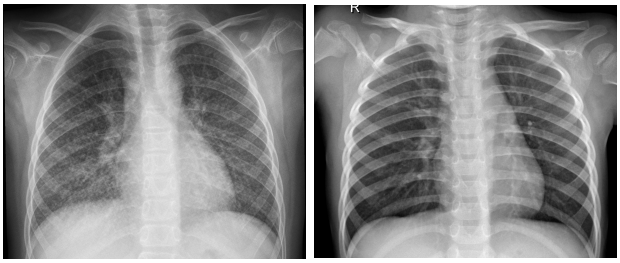
I. Introducción

Las aplicaciones de la inteligencia artificial abarcan un sin fin de campos al día de hoy, entre ellos, una área importante es la salud, en la que pueden haber soluciones que automatizen o agilicen ciertas tareas con el fin de quitar carga al personal especializado. Un ejemplo de lo anterior, es el análisis de imágenes médicas como resonancias o rayos X, con la finalidad de detectar anomalías en los pacientes. En éste trabajo, se aborda un problema de detección de pacientes con neumonía, a partir de imágenes de rayos X del tórax. Para ello, se cuenta con una base de datos que posee imágenes con la siguiente distribución:

	Normal	Neumonía
Entrenamiento	1349	3884
Prueba	234	390

Tabla I: Distribución de los datos en los conjuntos de prueba y entrenamiento.

En la figura 14, se muestra un ejemplo de imágenes de la base de datos, de pacientes sanos y con neumonía.



(a) Paciente con neumonía (b) Paciente normal

Figura 1: Ejemplos de la base de datos.

II. Carga y transformación de datos

En primer lugar, las imágenes de la base de datos poseen diferentes tamaños, por lo que es necesario re-dimensionarlas para poder utilizar una red neuronal. Además, como la base de datos es pequeña, se utilizaran técnicas de aumento de datos, por medio de un *DatasetFolder* con las transformaciones que se detallan a continuación:

- Redimensionamiento de la imagen a un tamaño de 224x224 píxeles.
- Escalamiento de los píxeles a valores entre 0 y 1 dividiendo por el máximo valor de píxel.
- Volteo horizontal de la imagen aleatorio, con probabilidad de $\frac{1}{2}$.
- Rotación de la imagen con un ángulo aleatorio entre -20° y 20°.
- Multiplicación de cada píxel por un número aleatorio entre 1.2 y 1.5.

Como se puede observar de la tabla I, se tienen diferentes distribuciones de los datos para los conjuntos de entrenamiento y de prueba. En el primer caso, se tiene que un $25{,}77\,\%$ de los pacientes están sanos, mientras que en el segundo caso el $37{,}5\,\%$ de los pacientes están sanos. La diferencia de distribuciones en éstos dos conjuntos, puede llevar a errores en la estimación del rendimiento en producción, esto pues, el modelo puede sesgarse durante el entrenamiento con la distribución del conjunto de train.

Otro posible problema, es el desbalance de clases existente en los datos, lo que, nuevamente puede sesgar el modelo durante el entrenamiento, obteniendo aparentes buenos resultados. Por ejemplo, el modelo puede decidir estimar que todas las imágenes entrantes corresponden a neumonía, y se obtendría casi un $75\,\%$ de accuracy, resultado relativamente bueno, pero que en realidad corresponde a un mal modelo. Para resolver esto, existen técnicas de balanceo de clases, que para este trabajo no fueron aplicadas, pero para obtener una mejor estimación de la calidad del modelo, se utilizará además del *accuracy*, la métrica *f1-score*, que se computa teniendo en cuenta la distribución de cada clase.

Considerando lo anterior, se asumirá que la distribución del problema será similar a la del conjunto de prueba. Así, se separan los datos de la carpeta *train*, en dos subconjuntos: entrenamiento y validación, en una proporción 80/20 respectivamente. Además, se replicará la distribución de los datos de test sobre el conjunto de validación, mediante sobremuestreo de la clase minoritaria, para obtener una mejor estimación del rendimiento del modelo durante el entrenamiento. Este sobremuestreo se aplica mediante objetos de la clase *torch.utils.data.Sampler*, dentro de Dataloaders (torch.utils.data.Dataloader).

Se midió el tiempo de carga de datos al variar el parámetro num_workers, del Dataloader. Específicamente, se computó el tiempo de iteración sobre todos los datos del dataloader

de entrenamiento. Utilizando $num\_workers = 0$, el mejor tiempo de 3 ejecuciones fue de 70 segundos, mientras que con $num\_workers = 5$, se obtuvieron 58.8 segundos. Además se intentó con $num\_workers = 10$, pero se obtuvo un tiempo de 58.9 segundos.

III. Entrenamiento utilizando CNN's

Se comienza contruyendo una clase que implementa una convolución separable (*Depthwise Separable Convolution*), para luego utilizarla en la red convolucional profunda.

La ventaja obtenida al usar este tipo de convolución es que disminuye el número de multiplicaciones al compararlo con con una convolución tradicional, y por ende tiene menos costo computacional.

Utilizando la *Depthwise Separable Convolution*, se construye una red similar a la VGG16, llamada VGG16DWSep cuya arquitectura se muestra en la tabla II.

Capa	Kernel size	Padding	Stride	# filtros	F. activación
Conv2D	3	1	1	64	ReLU
Conv2D	3	1	1	64	ReLU
MaxPool2D	2	0	2	-	-
DWSepConv2D	3	1	1	128	ReLU
DWSepConv2D	3	1	1	128	ReLU
MaxPool2D	2	0	2	-	-
DWSepConv2D	3	1	1	256	ReLU
BatchNorm2D	-	-	-	-	-
DWSepConv2D	3	1	1	256	ReLU
BatchNorm2D	-	-	-	-	-
DWSepConv2D	3	1	1	256	ReLU
MaxPool2D	2	0	2	-	-
DWSepConv2D	3	1	1	512	ReLU
BatchNorm2D	-	-	-	-	-
DWSepConv2D	3	1	1	512	ReLU
BatchNorm2D	-	-	-	-	-
DWSepConv2D	3	1	1	512	ReLU
MaxPool2D	2	0	2	-	-
Flatten	-	-	-	-	-
Linear	-	-	-	1024	ReLU
Dropout (0.7)	-	-	-	-	-
Linear	-	-	-	512	ReLU
Dropout (0.5)	-	-	-	-	-
Linear	-	-	-	2	ReLU

Tabla II: Arquitectura VGG16DWSep

Con ésta arquitectura, se obtiene un total de 103927746 parámetros entrenables, mientras que, si reemplazamos las capas DWSep por Conv2D, se obtienen 108564802, es decir 4637056 parámetros adicionales. Por otro lado, se midieron los tiempos de inferencia en ambos casos sobre un batch de 16 imágenes y como era de esperarse, la red VGG16DWSep obtuvo un menor tiempo de inferencia que la misma reemplazando las capas DWSep por Conv2D, tomando un total de 28.8 ms y 36.2 ms respectivamente.

Además, se implementa la clase EarlyStopping, la cual representa a la heurística de regularización del mismo nombre, que consiste en detener el entrenamiento en el caso de que alguna métrica determinada empeore (o no mejore) de forma sostenida.

Para la optimización de la red, se utiliza la regla de actualización de pesos *Adam*, y entropía cruzada como función de costos. Los hiper-parámetros utilizados para el entrenamiento de la red, se muestran en la tabla III.

Hiper-parámetro	Valor
Optimizador	Adam
Learning rate	1e-4
Weight decay	1e-5
Función de costos	Entropía cruzada
Early stopping	Sí
Data augmentation	Sí
Batch normalization	Sí
Dropout	0.5 y 0.7
Batch size	16
Función de salida	Softmax

Tabla III: Hiper-parámetros.

A pesar de que se debía establecer un early stopping con su parámetro *paciencia* = 5, se aumentó éste valor a 8, debido a que, en general, durante aproximadamente las 15 primeras épocas del entrenamiento, el modelo no mejoraba prácticamente en nada, y luego existía un *punto de quiebre*, en donde empezaba a mejorar. Por lo anterior, si se dejaba la *paciencia* = 5, se terminaba el entrenamiento antes de este punto de quiebre.

III-A. Resultados del entrenamiento

El modelo se entrenó durante 33 épocas, hasta que se detuvo debido al early stopping. En las figuras 2, 3, 4, se muestran el accuracy, el f1-score y loss, respectivamente, obtenidos a lo largo del entrenamiento, para los conjuntos de entrenamiento y validación.

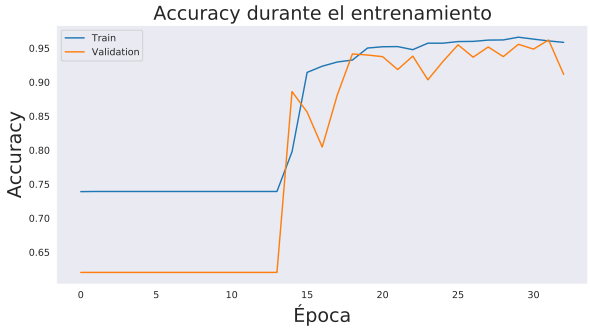


Figura 2: Accuracy durante el entrenamiento.

Junto con lo anterior, se obtuvo para el conjunto de prueba, un accuracy del $86{,}86\,\%$ y f1-score del $89{,}95\,\%$.

Como se puede observar en los gráficos, se obtuvo un comportamiento algo extraño durante el entrenamiento, debido a que, durante las primeras épocas, parece no haber una mejoría considerable en el modelo. De hecho se nota un punto de inflexión en la época 14, etapa bastante avanzada del loop. Se cree que esto sucede ya que la cantidad de parámetros es demasiado grande, y pueden existir muchos mínimos locales en los cuales la red se queda estancada.

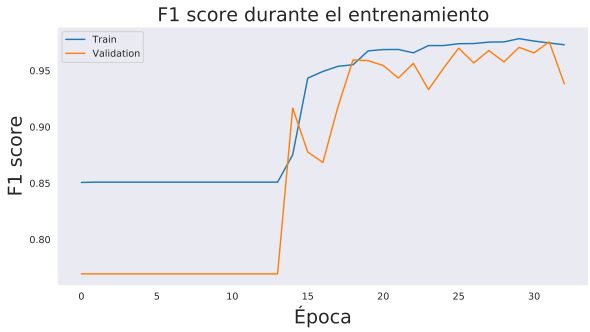


Figura 3: F1-score durante el entrenamiento.

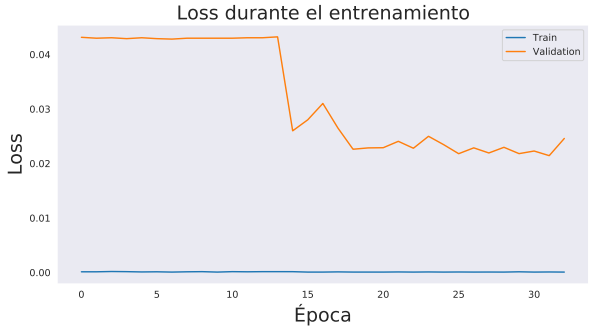


Figura 4: Loss durante el entrenamiento.

Además, la aumentación de datos puede estar provocando que las imágenes varíen demasiado, y a la red le cueste aprender esta variabilidad.

No se alcanzó a implementar el data augmentation sobre el conjunto de prueba, pero se estima que puede llegar a ser un buen método para incrementar el rendimiento del modelo, si se escogen cuidadosamente las transformaciones hechas sobre las imágenes. Se piensa que ciertas transformaciones pueden ser útiles para un problema en particular, pero no de manera general.

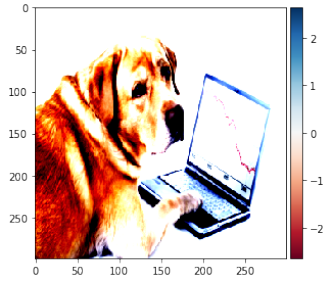
IV. Interpretabilidad de los modelos

En esta sección se implementó un modelo auxiliar de interpretabilidad local sobre las predicciones que genera una red neuronal, para lo cual se utilizó el procedimiento LIME (*Local Interpretable Model-Agnostic Explanations*).

El objetivo de este modelo es entender el por qué de la predicción de la red, es decir, se busca saber cuales fueron las secciones de la imagen procesada que más contribuyeron a la decisión final del clasificador.

La imagen original de control utilizada se puede ver en la Figura 5.a. Se hizo necesario transformarla para aplicar el procedimiento LIME, por lo que se le aplicó un escalamiento de $299 \cdot 299$ píxeles, luego se centró y normalizó. La imagen transformada se puede observar en la Figura 5.b.

(a) Imagen de Control Original



(b) Imagen de Control Transformada

Figura 5: Imagenes de Control

IV-A. Procedimiento LIME.

A continuación se definen los pasos realizadas para aplicar el procedimiento LIME.

IV-A1. Segmentación de la Imagen de Control: Primeramente se cargó la red *inception_v3* entrenada con la base de datos de *imageNet*, para después procesar y clasificar la imagen de control. El resultado de la clasificación fue la etiqueta *"Labrador Retriever"*.

Posteriormente, se segmenta la imagen en 80 segmentos, para generar súper-píxeles. Estos inducen bordes en la imagen, los cuales se pueden ver la Figura 6.

IV-A2. Perturbación de la Imagen: Se procede a perturbar la imagen de control, para lo cual se genera un arreglo binario de perturbación mediante una variable aleatoria *Bernoulli* con probabilidad de éxito $p = 0{,}5$. La imagen perturbada consiste en asignar el valor 0 en cada canal de color en aquellos píxeles cuyos super-píxeles asociados tengan su componente nula en el arreglo de perturbación.

El proceso de generar una imagen perturbada descrito anteriormente se realizó 1000 veces, obteniendo una base de

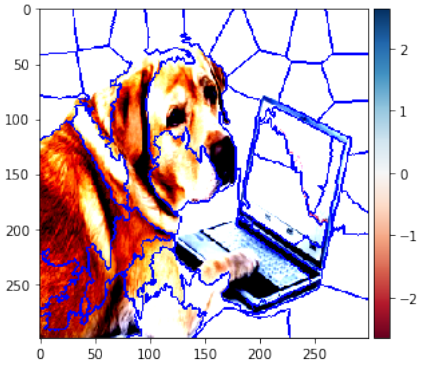
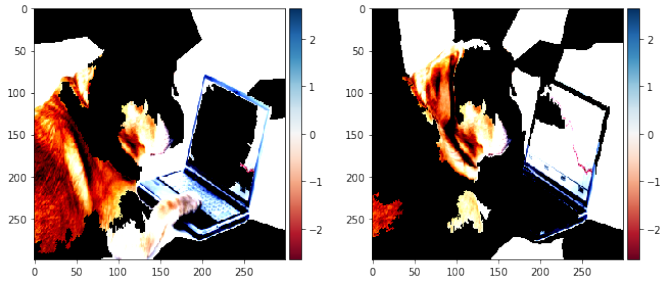


Figura 6: Imagen Segmentada

datos de mil imágenes perturbadas cada una con su arreglo binario de perturbación asociado. A continuación, se muestran 2 de ellas:



(a) Imagen Perturbada 1 (b) Imagen Perturbada 2

Figura 7: Ejemplos de imágenes perturbadas.

Utilizando la misma red cargada anteriormente (*inception_v3*) se clasificaron todas las imágenes de la base de datos generada. Así, se obtuvo un arreglo binario de etiquetas *"y"*, al cual se le asignó el valor 1 en caso de que la red clasificara la imagen perturbada con la misma etiqueta que la imagen de control original, y 0 en caso contrario.

IV-A3. Cálculo Kernel Exponencial: El siguiente paso consiste en calcular un kernel exponencial π_x definido como:

$$\pi_x = \exp(-\frac{d(x^{'}, z^{'})}{\sigma^2}) \quad (1)$$

El cual esta definido por una medida de similitud $d(\cdot)$, en este caso la distancia coseno, definida como:

$$d(x^{'}, z^{'}) = 1 - \frac{x^{'} \cdot z^{'}}{\|x^{'}\|\|z^{'}\|} \quad (2)$$

El kernel se calculó considerando $x^{'}$ como un vector de perturbación de la imagen de control y $z^{'}$ como cada vector de perturbación asociado a cada una de las imágenes perturbadas de la base de datos creada. Además, se consideró $\sigma = 0{,}2$. Así, se obtienen 1000 kernels distintos, con el objetivo de utilizarlos como pesos de un clasificador en el paso siguiente.

IV-A4. Clasificación con Regresión Logística de las perturbaciones: Se define un conjunto de entrenamiento D_p como el set de vectores de perturbación asociados a las imágenes perturbadas generadas anteriormente, y sus respectivas etiquetas las cuales corresponden al vector de etiquetas generado *"y"*.

Dado que se que se definió el conjunto de entrenamiento, se clasificará con el modelo de Regresión Logística de la librería *scikit-learn* de *Python*, asignándole como pesos los kernels π_x calculados.

IV-A5. Súper-píxeles de mayor importancia: Al obtener los coeficientes del modelo de Regresión, se utilizaron para inferir los súper-píxeles de mayor importancia en la clasificación de la imagen de control. Para esto, se decidió usar un umbral, de tal manera que se escogieran sólo los coeficientes mayor que este umbral, logrando que sólo se visualicen los súper-píxeles asociados a estos coeficientes. La imagen de control obtenida se puede ver en la Figura 8.

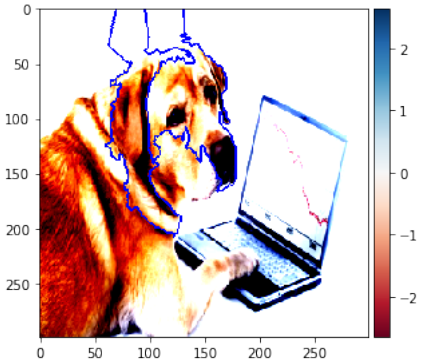


Figura 8: Imagen de Control con los súper-píxeles de mayor importancia marcados en azul

Al analizar la Figura 8, los súper-píxeles obtenidos tienen sentido en cuanto a cómo tomó la decisión la red al clasificar la imagen como *"Labrador Retriever"*, dado que a la hora de identificar la raza de un perro, la cabeza juega un papel fundamental en gran parte de los casos.

IV-B. LIME con Clustering.

En esta sección se aplicará el procedimiento LIME explicado en la sección IV-A pero intercambiando el paso IV-A1 de Segmentación por uno de Clustering.

Para hacer el Clustering, primero se definió un conjunto de entrenamiento X con 299^2 observaciones, es decir, una observación por píxel en la imagen de control transformada. Cada observación de X consta de 3 componentes, donde la primera y segunda son espaciales (posición del píxel en la imagen) y la última es el valor de intensidad asociado al

píxel en la imagen de control en escala de grises.

El método de Clustering elegido fue *K-Means* y se escogió la cantidad de Clusters como la cantidad de súper-píxeles en la imagen de control segmentada de la Figura 3. Así, la cantidad de Clusters fue de 38. Al visualizar los súper-píxeles inducidos por estos se obtuvo una imagen segmentada que se puede visualizar en la Figura 9.

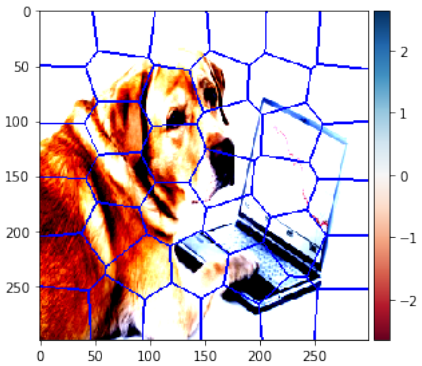
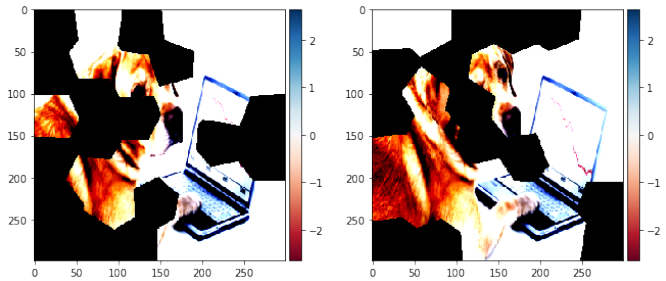


Figura 9: Imagen de Control Segmentada por Clustering

En seguida se aplicaron los siguientes pasos del procedimiento LIME (IV-A2 a IV-A5). Los resultados se muestran en las Figuras 10 y 11.



(a) Imagen Perturbada 1 (b) Imagen Perturbada 2

Figura 10: Ejemplos de imágenes perturbadas para el método LIME con Clustering.

Los resultados obtenidos en la Figura 11 no son tan precisos como los de la Figura 8 (método clásico), dado que se muestran marcadas partes de la imagen que no tienen coherencia con cómo debería tomar una decisión la red. Aun así, persiste la importancia de la cabeza , sin embargo esta vez no se marcó por completo.

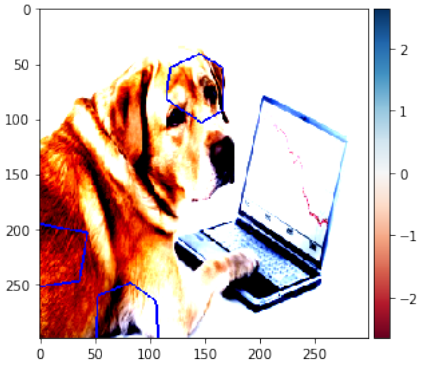


Figura 11: Imagen de Control con los súper-píxeles de mayor importancia marcados en azul

IV-C. LIME sobre una predicción de la red VGG16DWSep.

La imagen de control del conjunto de Prueba elegida para aplicar el procedimiento LIME es la siguiente:

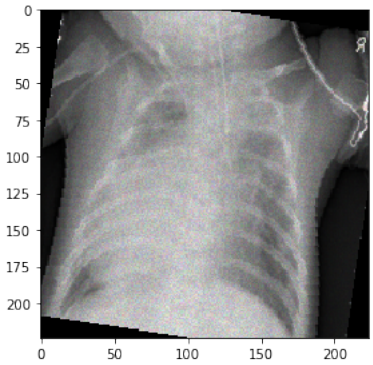


Figura 12: Imagen de Control Transformada

Al procesarla por la red VGG16DWSep, esta arrojó que corresponde a la clase 1, es decir, un paciente con neumonía. Luego, al aplicar los pasos detallados en la sección IV-A, es posible ver los resultados obtenidos en las Figuras 13, 14 y 15.

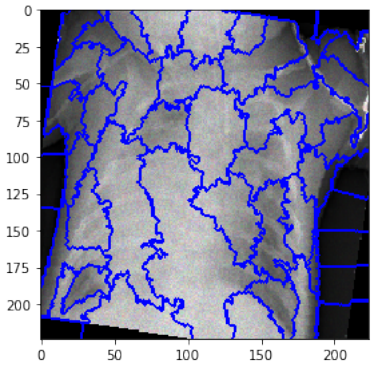
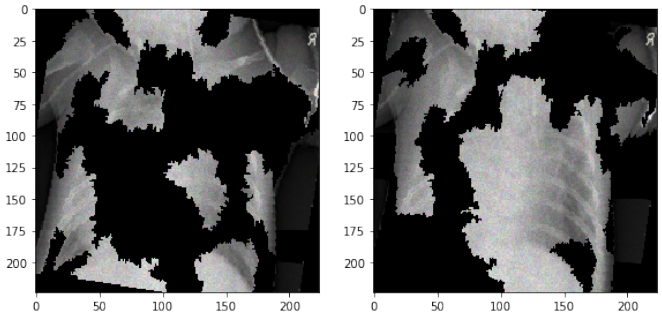


Figura 13: Imagen de Control Segmentada

Finalmente, es posible observar los súper-píxeles más importantes encontrados en la imagen de control en la Figura



(a) Imagen Perturbada 1 (b) Imagen Perturbada 2

Figura 14: Ejemplos de imágenes perturbadas

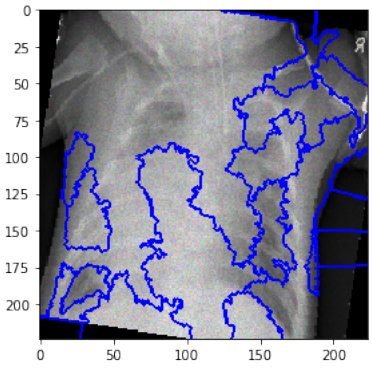


Figura 15: Imagen de Control con los súper-píxeles de mayor importancia marcados en azul

15, gracias a los cuales un experto podría dirimir si las secciones obtenidas se justifican con la clase asignada.

Cabe mencionar que para obtener los súper-píxeles importantes en esta sección se hizo necesario aumentar el umbral de decisión comparado con las secciones IV-A y IV-B. Por lo que se concluye que es posible que haya un error en la forma de elegir los súper-píxeles importantes, dado que no se cumple con la premisa de LIME, la cual es que es agnóstico al modelo.

V. Conclusiones

Se cumplen los objetivos parcialmente, dado que, por un lado la red se entrenó correctamente logrando una precisión aceptable para el conjunto de prueba ($86\,\%$). Sin embargo, por otro lado, no se obtuvo del todo un procedimiento LIME Agnóstico.

Se pudo comprobar la utilidad de las capas Dephtwise Separable Convolution, debido a que ayudan a reducir el tiempo de ejecución sin perder rendimiento.

En cuanto a la aumentación de datos, se comprueba que es un método muy efectivo para entrenar modelos con una gran cantidad de parámetros cuando se posee una base de datos pequeña.