E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Modificación de procesos de fusión de información en redes convolucionales aplicadas a la segmentación de radiografías

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Eduardo Gómez Girón

Director: Javier Fernández Fernández

Pamplona, 22/05/2022

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Índice:

1. Resumen:

Nuestra sociedad ha vivido recientemente una de las peores epidemias en décadas, la COVID 19. Esta enfermedad ha modificado los hábitos de vida que hasta ahora conocíamos, poniendo en jaque al sistema sanitario y afectando gravemente a la calidad de vida de la gente.

Uno de los síntomas más notables en un estadio avanzado de la enfermedad es la neumonía bilateral, que consiste en la inflamación de naturaleza infecciosa de los espacios alveolares de los pulmones [1].

El progreso tecnológico en medicina está siendo impulsado en gran medida por la inteligencia artificial y el procesamiento de imágen. El procesamiento de imágenes es una parte vital de la radiología, ya que permite realzar los bordes, realzar el contraste, reducir el ruido, mejorar el rango dinámico y en resumen, ofrecer una imagen más clara a los profesionales médicos que las analizan [2].

Este proyecto trata sobre el uso de técnicas de inteligencia artificial para conseguir hallar el método óptimo de segmentación de imágenes aplicado a la segmentación de radiografías de tórax con el fin de poder diferenciar en una radiografía qué pixeles pertenecen a los pulmones y qué pixeles no. Para ello se ha empleado una red neuronal convolucional conocida como U-Net y hemos realizado un trabajo de investigación de diferentes configuraciones del modelo para encontrar alternativas a la configuración clásica al mecanismo de fusión de características que utiliza esta red.

1.1 Palabras clave:

- U-Net
- Redes Neuronales
- Función de pooling
- Aprendizaje supervisado
- Segmentación de imagen

2. Introducción:

2.1 Introducción a la segmentación de imágenes:

Dentro de la visión artificial, la segmentación de imágenes es un problema clásico que consiste en dividir una imagen digital en varias regiones denominadas segmentos. La segmentación de imágenes no es más que un problema de clasificación de píxeles en el cual se asigna una categoría a cada píxel de la imagen analizada. En nuestro caso, hemos realizado una segmentación binaria. En otras palabras, solo tenemos dos clases en las que dividir los píxeles de la imagen: “objeto” y “fondo”.

En la segmentación de imágenes, la clasificación de los pixeles utiliza las propias características de la imagen y en sus valores de intensidad.

Existen distintas maneras de extraer estas características como la umbralización de la intensidad de los píxeles, el estudio de las discontinuidades en los valores de intensidad de píxeles contiguos, que pueden indicar la presencia de bordes, entre otras [2].

En este trabajo, hemos empleado una arquitectura U-Net, que es un modelo de red neuronal convolucional pensada específicamente para la segmentación de imágenes médicas.

2.2 Aplicaciones:

La segmentación de imágenes médicas juega un papel importante en el diagnóstico y tratamiento médico. Estos algoritmos ya se usan a día de hoy en el análisis de tejidos en radiografías, escáneres de resonancia magnética y ecografías. Su aplicación incluye aplicaciones médicas como la detección de tumores cerebrales [3], extracción de la zona afectada por tuberculosis extra pulmonar [4], visualización de patologías de corazón [5], medición y detección de tumores [6], estudio de partes de cuerpo [7], entre otras aplicaciones.

2.3 Origen de la arquitectura U-Net:

Las redes neuronales llevan planteadas desde mediados del siglo pasado. Los primeros perceptrones fueron diseñados por McCulloch et al. En 1943, nombraron a su modelo neuronal: Threshhold Logic Unit [8]. Más adelante surgirían las primeras CNNs de la mano del necognitron

de Fukishima, en 1979 [9]. Durante muchos años se “abandonó” la idea al no disponer de la suficiente capacidad de cómputo necesaria para entrenar redes complejas. Otro de los motivos por los cuales no florecieron en el pasado era la falta de datos necesarios para entrenar dichos modelos. Alrededor del cambio de siglo ya se teorizaba que de poder entrenar redes más complejas, estas darían un buen resultado como algoritmos de aprendizaje automático. Gracias a los grandes avances en tecnología en estas últimas décadas, a día de hoy podemos crear redes neuronales profundas y complejas de manera mucho más sencilla y entrenar los modelos en un tiempo razonable.

El eje central del estudio de la segmentación de imágenes en este proyecto será la arquitectura U-Net. La arquitectura U-Net es una red neuronal convolucional desarrollada, específicamente para segmentación de imágenes biomédicas. Fue creada por Ronneberger et al. en 2015 [10] como una mejora al uso de redes totalmente convolucionales para la segmentación semántica [11].

Al estar enfocado el problema a segmentar un conjunto de radiografías, es justo lo que necesitamos. La U-net ha sido muy contrastada y aprobada por la comunidad como una herramienta de segmentación de imágenes que ofrece buenos resultados.

2.4 Problema y resolución:

En la segmentación de imágenes y en concreto en el uso de la U-Net, las funciones de pooling son un eje central del modelo, interviniendo directamente en su funcionamiento. Nuestro principal problema será estudiar distintos métodos de pooling y evaluarlos para poder llegar a alguna conclusión sobre su rendimiento en segmentación de radiografías de pulmón.

Para obtener datos de entrenamiento, empezamos buscando datasets de radiografías de tórax ya preparados en la web. Debido a la pandemia, estamos en un momento en el que abundan las radiografías de tórax, así que es difícil encontrar ejemplos.

Tras comprobar que la segmentación de imágenes es correcta, hemos probado a optimizar al máximo la U-Net estudiando distintos hiperparámetros y aplicando distintas funciones de pooling. Nuestro objetivo es hallar qué ventajas nos ofrece cada una y cual sería óptimo usar.

2.5 Objetivos:

El objetivo final de este trabajo de investigación consiste en generar un clasificador de radiografías capaz de segmentar una radiografía de pulmón de un paciente.

Una vez planteado el objetivo principal a conseguir, este objetivo puede descomponerse en objetivos más sencillos a modo de hitos menos complejos a conseguir a lo largo del proyecto.

El primer objetivo de todos será documentarse y aprender a desenvolverse con las herramientas que vamos a utilizar. Este objetivo trata de conseguir cierto grado de conocimiento de Python y de sus librerías, sobretodo de aquellas que suponen una curva de aprendizaje mayor como PyTorch y el trabajo con tensores. PyTorch es una herramienta potente para trabajar con modelos de aprendizaje automático. Esto se debe a que tiene implementadas muchas funcionalidades útiles de uso general en el aprendizaje automático. Además, permite la ejecución de los cálculos matemáticos de los tensores a través de la GPU (Tarjeta gráfica), haciendo que la ejecución sea mucho más rápida [12]. El tensor es una estructura de datos que relaciona distintos espacios vectoriales y que puede representarse como un espacio matricial multidimensional.

El siguiente objetivo será documentarse de manera extensa sobre el funcionamiento de la U-Net y estudiar datasets ya generados por la comunidad que puedan servir para este proyecto.

Posteriormente, se realizará un estudio de los parámetros utilizados durante el entrenamiento y las funciones de pooling que usa la U-Net, para así poder realizar un ajuste de hiperparámetros, buscando el mejor desempeño para nuestra red. El pooling es una técnica utilizada en las redes neuronales convolucionales. Consiste en reducir el tamaño de la imagen agrupando píxeles según el criterio que describe una función. De esta manera se busca retener la mayor parte de la información pero reduciendo la dimensionalidad de la imagen y por tanto, el coste de procesarla. La función de pooling “MaxPool” es el estándar debido a que da buenos resultados, aunque hay muchas otras como la media, el estocástico, el mixed pooling, entre otras.

2.6 Estructura:

Esta memoria se dividirá en las siguientes secciones:

- Preliminares: Se explicará de manera más técnica y precisa las técnicas utilizadas y los retos que estas suponen.
- Implementación del modelo: Se expone el trabajo realizado de procesamiento de datos, creación de los modelos y puesta a punto de la U-Net para segmentación de imágenes.
- Experimentación: Se atacará el problema de investigación con los resultados obtenidos tras experimentar con distintas configuraciones de hiperparámetros del modelo.
- Conclusiones: Finalmente, se expondrá las conclusiones a las que hemos llegado a través de los resultados obtenidos durante la experimentación y las líneas de investigación futuras que nos abre el trabajo realizado.

3. Preliminares:

En este apartado, se exponen diferentes conceptos relacionados con las redes neuronales y en concreto, con la U-Net, que es la red aplicada. También se presentará el concepto de función de pooling y las distintas funciones de pooling utilizadas.

3.1 Aprendizaje supervisado:

En aprendizaje automático, el aprendizaje supervisado es una técnica para deducir y simular el comportamiento de una función desconocida. Para hallar dicha función se dispone de una serie de valores de entrada con su salida correcta. Esta función tiene que generalizar a partir de estos datos conocidos para poder predecir correctamente una salida con cualquier valor válido de entrada.

Los datos de entrenamiento se componen de pares de objetos. Por una parte tenemos los datos de entrada y por la otra tenemos las etiquetas, que son los valores que buscamos que el modelo aprenda a predecir. La salida de la función puede ser un valor numérico, pero también puede ser una imagen, una recta, un plano… Dependerá del problema a resolver.

El objetivo de un algoritmo de aprendizaje supervisado consiste en entrenar un modelo que simule el funcionamiento de una función, usando los datos de entrenamiento para contrastar cómo de correctos son los resultados predichos frente a los resultados que tenemos y modificando la función en base a estos resultados. De esta manera se busca generar una función capaz de predecir la salida de un valor de entrada nuevo, que no haya sido usado durante el entrenamiento, es decir, se busca que el modelo sea capaz de generalizar correctamente a partir de una serie de datos provistos.

En nuestro caso, el valor de entrada será una imagen en blanco y negro correspondiente a una radiografía y la salida será una imagen umbralizada que nos indica qué pixeles de la imagen original pertenecen a los pulmones. A esta imagen se le conoce como máscara.

3.2 Introducción a Redes Neuronales:

En el campo de la Inteligencia Artificial, las redes neuronales funcionan como aproximadores paramétricos de funciones no lineales. Utilizando el teorema de proyección se demostró en 1989 que estas funciones forman un aproximador universal. [13]

Las redes neuronales artificiales emulan el funcionamiento de una red neuronal biológica, donde a partir de unos datos de entrada se busca que la red neuronal produzca unos resultados esperados de salida.

La unidad básica de una red neuronal se conoce como perceptrón, los cuales se combinan para crear la red. Cada uno de estos perceptrones, actúa como una función que genera una salida en base a una entrada. Podemos ver un perceptrón en la Figura 1.

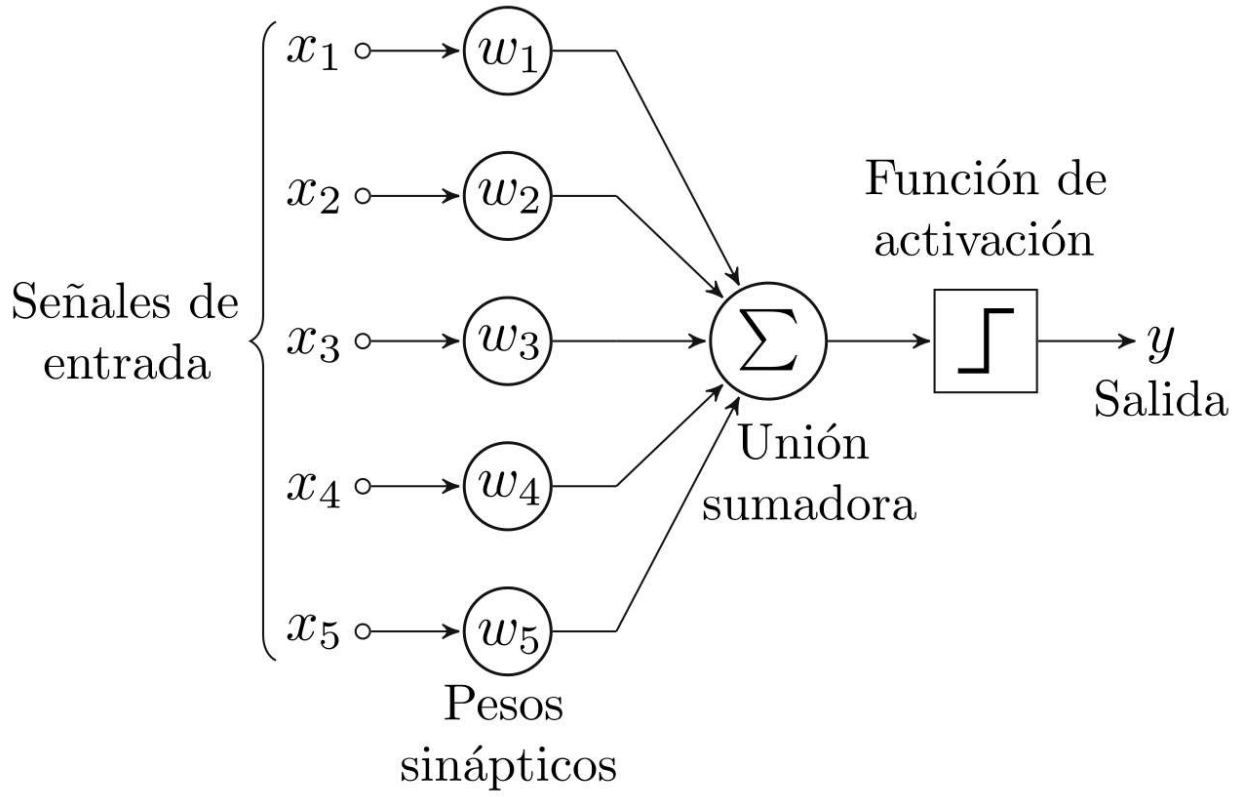

Figura 1. Perceptrón simple con cinco señales de entrada.

En la Figura 1 se puede observar cómo este perceptrón tiene 5 valores numéricos de entrada y cada uno de estos valores tiene asignado un peso. Las entradas se agregan a través de una

función sumadora que suma las señales de entrada multiplicadas por su peso correspondiente. Tras la agregación, una función de activación se encarga de transformar el valor obtenido en un valor válido de salida. Para este proceso pueden usarse diferentes funciones no lineales como un simple umbral numérico, una función sigmoide o una ReLU. Un perceptrón puede representarse como un vector de pesos. Además, este puede contar con un sesgo, un valor numérico ajustable que se añade a la función sumadora. El perceptrón define un hiperplano que separa el espacio de características en dos regiones, el bias sirve para poder desplazar dicho hiperplano y que no necesariamente tenga que pasar por el origen, pudiendo mejorar así los resultados.

El proceso de entrenamiento de un perceptrón consiste en ajustar los pesos asociados a cada entrada para conseguir la salida esperada.

3.3 Diseño de una red neuronal:

Las redes neuronales se diseñan conectando perceptrones a lo largo de distintas capas, se llaman perceptrón multicapa. Tienen la capacidad de resolver problemas que no son linealmente separables. Los perceptrones multicapa están conectados haciendo que la salida de cada uno de los perceptrones en la capa “i”, es entrada de todos los perceptrones de la capa “i+1”. Hay tres tipos de capa:

- Capa de entrada: Es la capa donde introducimos los datos de entrada. Tendrá tantos nodos como entradas tengamos.
- Capas oculta: Son el conjunto de capas que se sitúan entre la capa de entrada y la de salida. Se encargan de procesar los datos y propagar los resultados hacia la capa de salida.
- Capa de salida: Es la capa que se encarga de recibir los valores de la última capa oculta y devolver los valores de salida de la red.

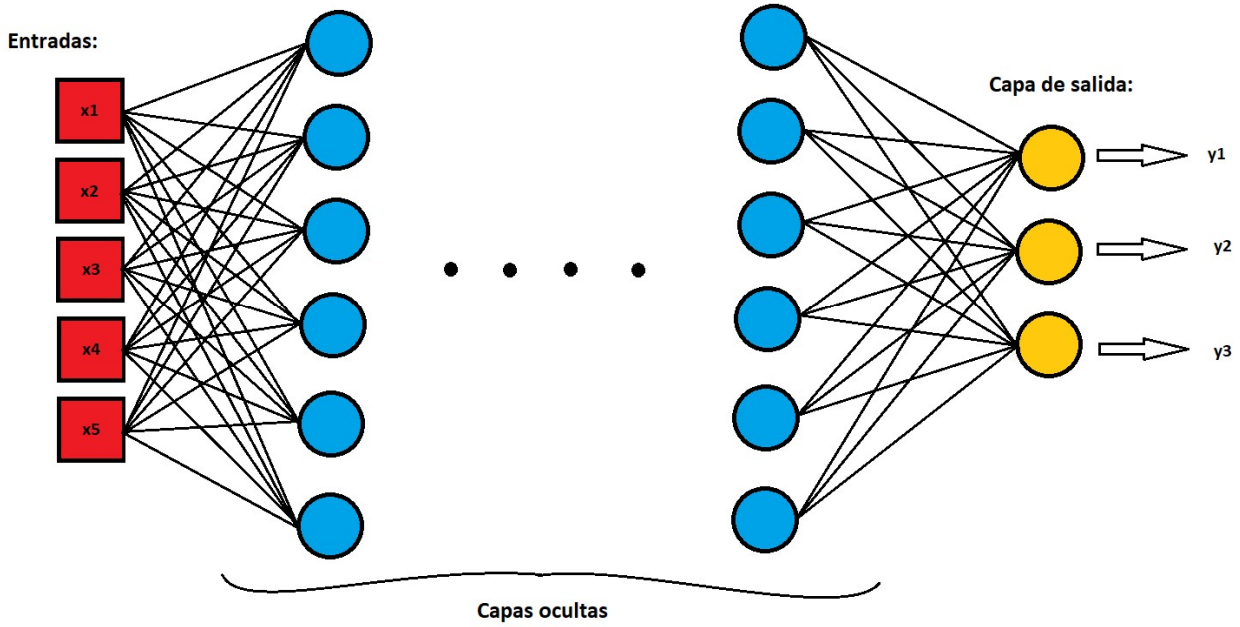

Figura 2. Arquitectura de una red neuronal.

Las redes neuronales pueden llegar a tener arquitecturas muy complejas. Muchos factores son los que intervienen en el nivel de complejidad y forma que requiere una red neuronal y no existe una manera "correcta" de diseñar una red, por lo que la obtención de una arquitectura satisfactoria u óptima se basa en la prueba y error de diferentes configuraciones de capas y neuronas por capa.

Al igual que un perceptrón puede visualizarse como un vector de pesos, al estar una red neuronal compuesta por capas de perceptrones, se puede visualizar una red neuronal como un conjunto de matrices de pesos, donde la matriz i-ésima corresponde a la capa i-ésima de la red y la fila j-ésima de una matriz corresponde al vector de pesos del perceptrón j-ésimo de la capa.

3.4 Entrenamiento y Backpropagation:

Una red neuronal entrena buscando la minimización del error de las salidas predichas respecto a las salidas reales. No es una tarea trivial ya que la red puede llegar a ser muy compleja y hay una gran cantidad de parámetros a ajustar.

Para poder entrenar una red neuronal se necesita primero definir una función de coste. La función de coste juega un papel crucial en los modelos de aprendizaje automático. Dicha función

nos indica lo bien que rinde un modelo para un dataset y una configuración específica del modelo dados. Tiene que devolvernos valores más altos cuanto peor rinda el modelo y viceversa. Ninguna solución tiene un costo menor que el costo de la solución óptima. A pesar de que en aprendizaje supervisado tenemos otras métricas que nos indican lo bien que predice nuestro modelo, como el accuracy, no sirven para indicar al modelo cómo mejorar.

El algoritmo de gradiente es un algoritmo de optimización, que busca minimizar la función de coste de nuestro modelo. Nos permite calcular cómo debemos cambiar los parámetros del modelo en la dirección del gradiente de la función de coste, reduciendo así su valor. Este es un proceso iterativo en el cual el modelo va convergiendo hacia un valor mínimo, hasta que ya no se producen cambios o la mejora es casi inapreciable. A este punto se le llama punto de convergencia, el error es mínimo y la función de coste ha sido optimizada.

En aprendizaje supervisado es muy utilizado el error cuadrático medio, que calcula la media de las diferencias entre las salidas reales y las deseadas elevadas al cuadrado tal y como se muestra en la Ecuación 1.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - y'_i)^2$$

Ecuación 1. Fórmula del error cuadrático medio.

También es muy usado la entropía cruzada en problemas de clasificación donde no se evalúan valores numéricos si no clases. Podemos observar en la Ecuación 2, la fórmula del cálculo de la entropía cruzada para un problema de clasificación binaria. Donde *y* es el valor correcto de clasificación y *p* es la probabilidad de pertenecer a la clase positiva.

$$CE = -(y * \log(p) + (1 - y) * \log(1 - \mathrm{p}))$$

Ecuación 2. Fórmula de la entropía cruzada.

El algoritmo de "backpropagation" es un método de cálculo de gradiente utilizado en redes neuronales de aprendizaje supervisado para la actualización de pesos. Se llama método de retropropagación debido a que se calcula la derivada del error del modelo con respecto de los

pesos de las distintas capas partiendo de la capa de salida, hacia todas las neuronas de la capa oculta en dirección a la capa de entrada. Este es un proceso iterativo que se propaga a través de cada una de las capas de la red hasta que todas las neuronas hayan sido actualizadas. El algoritmo es el siguiente:

1. Calcular la salida de la red a partir del conjunto de pruebas *X*.
2. Comparar la salida predicha con la correcta y calcular el error en base a la función de coste.
3. Calcular las derivadas parciales del error con respecto a los pesos que unen la última capa oculta con la de salida.
4. Retroceder capa por capa calculando las derivadas parciales del error con respecto a cada uno de los parámetros de la capa hasta llegar a la primera capa.
5. Ajustar los pesos de cada neurona para reducir el error a partir del valor de gradiente calculado.
6. Volver al paso 1 hasta llegar al punto de convergencia.

$$\Delta W = -\gamma \frac{\partial E(W)}{\partial W}$$

Ecuación 3. Fórmula de modificación de pesos.

En la Ecuación 3 podemos observar la fórmula para la modificación de W. Para calcular cuánto tenemos que modificar los pesos, se tiene en cuenta el gradiente del error respecto a W. Si hemos definido bien nuestra función de coste, cuanto menor sea, más precisas serán las predicciones realizadas por nuestro modelo. El algoritmo de gradiente funciona derivando la función de coste, de esta forma hallamos la "pendiente" de la función. Una vez hallada, se modifican los parámetros respecto a sus gradientes minimizando así el error.

En la Ecuación 3 el parámetro "γ" es conocido como factor de aprendizaje, y modula cuánto se modifican los parámetros respecto a sus gradientes en cada iteración. Se le asigna un valor entre 0 y 1. Por tanto, si le asignamos un valor muy grande, el algoritmo aprenderá rápido, pero corremos el riesgo de que nunca llegue a converger del todo. Esto se debe a que, al hacer grandes cambios de los parámetros en cada iteración, puede suceder que se modifiquen demasiado en la

dirección del gradiente estando ya cerca del mínimo. Si avanzamos más distancia de la necesaria para llegar al mínimo, nos estamos acercando menos de lo que deberíamos o incluso nos podemos alejar de él en el sentido opuesto. Esto puede dificultar encontrar el mínimo, incluso puede llegar a causar que el modelo diverja. Un modelo diverge cuando en cada iteración se modifican tanto los parámetros que dejamos el mínimo atrás en cada avance y se incrementa el coste en vez de disminuirse. En cambio, si le asignamos un valor demasiado pequeño, las modificaciones a los parámetros en cada iteración serán pequeñas, por tanto la velocidad de convergencia del modelo puede ser muy lenta. Debido a la naturaleza compleja de este problema, cada modelo tendrá un factor de aprendizaje óptimo distinto. Para encontrar un valor óptimo de este factor hay ciertos principios generales que ayudan a escogerlo. Suele elegirse un valor inferior a 0.1. Pueden seguirse varios métodos como probar con distintos valores, reducir su valor en cada iteración, hacer que sea una variable que cambie en función del error devuelto, entre otras.

En la Figura 3 se puede observar una representación en tres dimensiones de una función de coste y de cómo funciona el algoritmo de gradiente. Se puede observar en cada flecha como en cada etapa se avanza hacia el punto de pendiente máxima. Las flechas en la figura representan la dirección y el módulo del avance. Cada uno de los puntos del gráfico, representan el valor de la función de coste para una configuración de parámetros concreta del modelo.

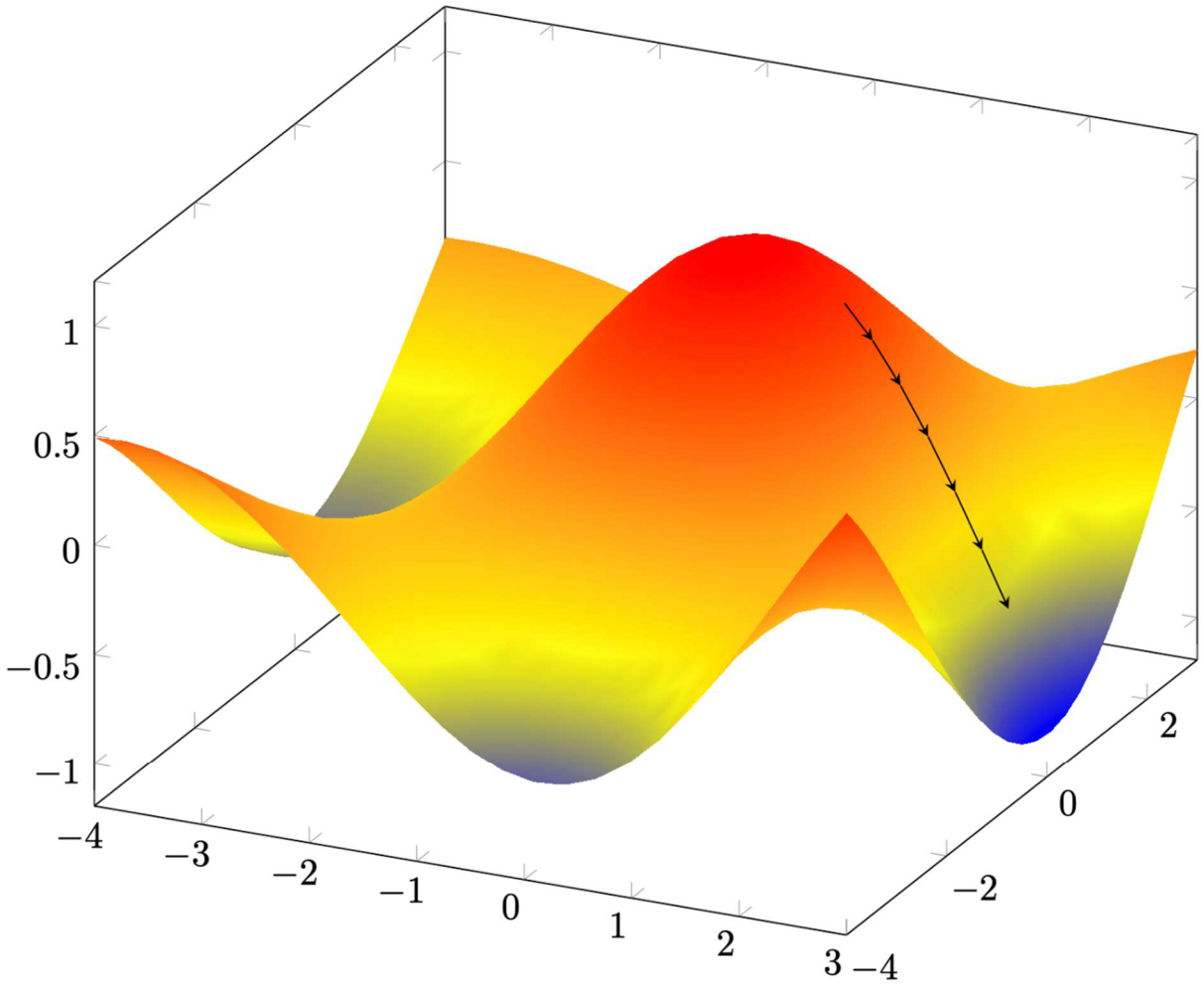

Figura 3: Algoritmo de gradiente.

3.5 Convolución:

En matemáticas, una convolución es un operador matemático que transforma dos funciones f y ω en una tercera función que representa la superposición de f y una versión trasladada e invertida de ω. En procesamiento de imágenes las funciones f y ω son imágenes que se superponen, f es la imagen original y ω es una matriz llamada kernel. El kernel es una matriz de menor tamaño que la imagen original. Para calcular la imagen resultante tiene que calcularse esta superoposición para todos los píxeles de la imagen original con el kernel. La expresión general para la convolución la podemos observar en la Ecuación 4:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x-dx, y-dy)$$

Ecuación 4. Expresión de convolución entre una imagen *f(x,y)* y un kernel ω.

En procesamiento de imágenes la convolución es una herramienta que tiene muchas aplicaciones como el realce de bordes, el enfoque, el desenfoque entre otros. En los modelos que hemos implementado, los filtros de convolución son por defecto matrices de números aleatorios y se aplica el uso de neuronas que aprenden mediante backtracking para modificar los valores de estas máscaras. El fin es lograr que el modelo aprenda a generar sus propios filtros de convolución con los que se obtenga la extracción de características.

3.6 Pooling:

Los operadores de agregación son operadores que agregan una serie de valores en uno solo. Se pueden definir como una función $A: [0,1]^n \rightarrow [0,1]$ que satisface las siguientes condiciones:

- A es creciente: $\forall\ i\ \in \{1, \dots, n\}: x_i \leq y$ entonces $A(x_1, \dots, x_n) \leq A(x_1, \dots, x_{i-1}, y, \dots, x_n)$
- $A(0, \dots, 0) = 0$
- $A(1, \dots, 1) = 1$

Las funciones de grouping son operadores de agregación $G: [0,1]^n \rightarrow [0,1]$ que satisfacen las siguientes condiciones [14]:

1. Ges simétrica.
2. $G(X) = 0$ si solo si $x_i = 0\ \forall\ i\ \in \{1, \dots, n\}$
3. $G(X) = 1$ si solo si $x_i = 1\ \forall\ i\ \in \{1, \dots, n\}$
4. G es no decreciente
5. G es continua

Una función de pooling es una operación de reducción de imagen, la reducción más común es la ventana cuadrada de tamaño KxK. El rol de una capa de pooling es reducir la resolución de la imagen, pero intentando retener las características de mayor relevancia y eliminando aquellas que tienen menor relevancia. Esto consigue una representación de la imagen más eficiente.

Se utiliza en distintos algoritmos de tratamiento de imágenes debido a que las CNN y otros algoritmos de clasificación funcionan mejor sobre entradas con menor dimensionalidad. Aplicando funciones de pooling se consigue una representación más comprimida de la información más relevante de la imagen tratada.

También al reducir las imágenes se reduce el costo computacional de forma considerable. Con una única operación de pooling KxK somos capaces de reducir en un factor de k tanto el número de columnas como el número de filas de las imágenes de características extraídas. Por tanto, el número de cálculos necesarios para procesar una imagen se reduce en un factor de 2k.

A cada una de las ventanas de valores que se agrupan en uno sólo tras la operación de pooling se le conoce como kernel.

En la U-Net que vamos a utilizar, nuestro kernel siempre será de 2x2. La función de pooling por defecto en la U-net y por tanto la que usaremos en el modelo base es la de pooling máximo, que consiste en devolver el máximo valor de cada ventana a agregar. El pooling máximo es comúnmente utilizado frente a otras funciones debido a que extrae las características más intensas de la imagen y estas forman una buena representación de la imagen. El máximo es un caso particular de grouping, por ello consideramos otros groupings para usar en nuestro modelo y observar las diferencias que producen en el entrenamiento.

3.7 CNN:

En aprendizaje profundo, las CNN (Redes Neuronales Convolucionales) son un tipo de redes neuronales comúnmente aplicadas en problemas de procesamiento de imágenes. Son un tipo de redes neuronales especializadas que utilizan múltiples capas de filtros convolucionales de una o más dimensiones.

Las CNN necesitan de poco preprocesamiento de imágenes en comparación a otros algoritmos de aprendizaje. Esto es porque la red neuronal optimiza sus propios filtros a través del aprendizaje automático, mientras que en algoritmos más tradicionales estos filtros tienen que ser previamente diseñados. Esto las hace mucho más independientes a nuestra intervención y conocimiento previo de la extracción de características. Esto es ventajoso porque no están

condicionadas por lo que nosotros podamos pensar que deberíamos hacer, que no siempre es correcto, y además nos ahorran una gran parte del trabajo de preprocesamiento.

Podemos dividir su arquitectura en 3 fases. En la fase de extracción de características se usan unas neuronas llamadas neuronas convolucionales. Estas realizan operaciones en 2 dimensiones sobre un área concreta de las imágenes, normalmente de tamaño KxK. Estas neuronas se entrenan para generar una máscara, que se utiliza para filtrar la imagen. Con esta operación se busca realzar las características de la imagen, dichas máscaras pueden extraer características más abstractas y no tan triviales como las que nosotros pudiéramos diseñar.

En la fase de reducción de muestreo se aplican funciones de pooling para reducir la dimensionalidad de las imágenes.

Después de una combinación de las fases anteriores, los datos llegan a la fase de clasificación. Al llegar a esta fase, los datos han sido depurados hasta una serie de características únicas y es trabajo de las neuronas de clasificación el predecir la salida final. El funcionamiento de esta capa es idéntico al de un perceptrón multicapa.

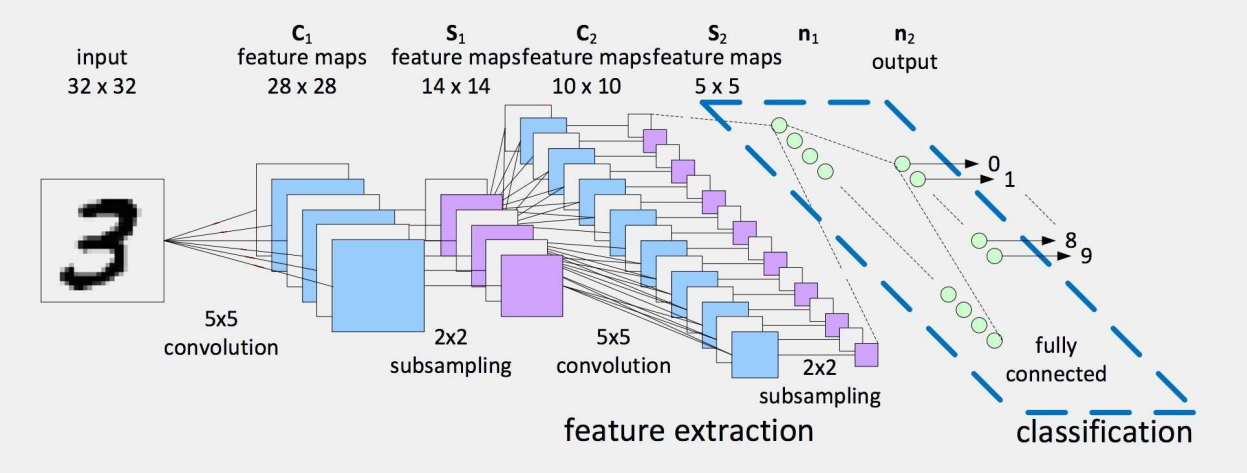

Figura 4. Arquitectura CNN.

En la Figura 4 podemos observar un ejemplo de arquitectura CNN de un clasificador de imágenes con números. Podemos observar como primero se realizan varias funciones de convolución (extracción de características) y funciones de subsampling (reducción de muestreo) y en la fase

final se usa un perceptrón multicapa completamente conexo para finalmente clasificar las muestras.

3.8 U-Net:

La U-Net es una arquitectura de red neuronal convolucional diseñada para una segmentación rápida y precisa de imágenes.

La idea principal es generar una red de contracción de imagen por capas sucesivas de convolución y pooling para posteriormente, reemplazar las operaciones de pooling por operaciones de upsampling, aumentando así la resolución de la imagen de salida. Podemos dividir la red en dos partes, la parte que contrae la imagen y la parte que la expande.

Una operación de pooling no es más que una operación de reducción de imagen, que agrupa segmentos de la imagen, generalmente de pequeño tamaño, en un único píxel. Cada función de pooling usa un operador de agrupación distinto.

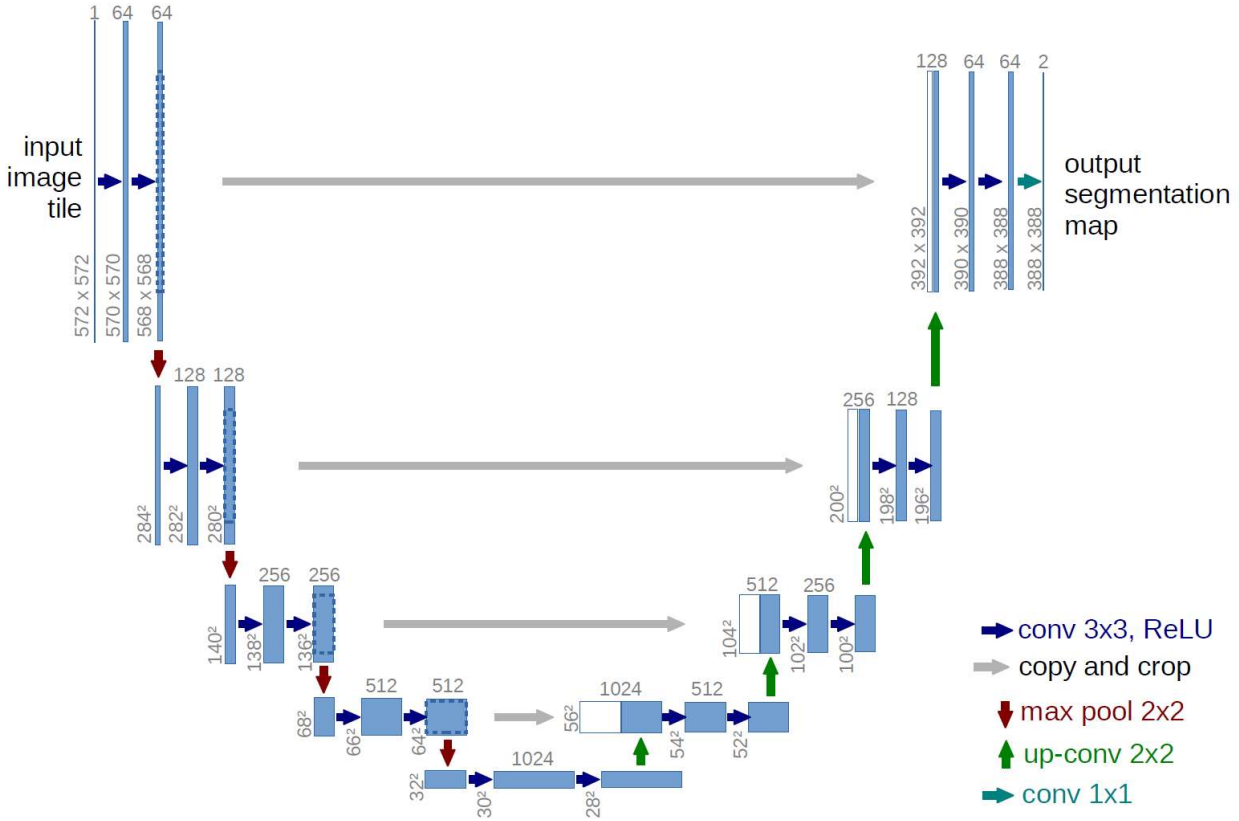

Figura 5. Arquitectura de una U-Net.

En la Figura 5 se observa de manera gráfica la arquitectura de una U-Net. Se parte de la imagen original y podemos distinguir claramente la parte izquierda de reducción y la parte derecha de expansión. La forma que tiene es lo que le da el nombre de U-Net al tener forma de U. Los recuadros azules representan un tensor que contiene la información de una imagen y como va cambiando su dimensionalidad a través de la red.

La fase de reducción está compuesta de varias capas. En cada una de las capas, se realizan varias operaciones de convolución, cada una seguida de una unidad lineal rectificada (ReLU) y una operación de pooling que reduce la imagen. Durante la contracción la información espacial se reduce mientras que aumenta la cantidad de filtros de características usados. Esta fase es comparable a una secuencia de fases de extracción de características y reducción de muestreo de una CNN.

En una U-Net, al realizar el pooling reducimos la dimensionalidad de la imagen por dos y seguidamente utilizamos una capa de convolución que a su vez duplica el número de características. Esto sucede por dos motivos principales. El primero es que la capacidad de aprendizaje de la red crece de manera logarítmica, así que tenemos que doblar el número de canales de características para incrementar la capacidad de aprendizaje de manera significativa. El segundo motivo es que reducir la dimensionalidad de una imagen a la par que se incrementan sus canales de características hace que la representación de la información sea cada vez más “semántica” y menos “espacial”. La U-Net necesita tener la información distribuida de manera semántica o abstracta, debido a que las redes convolucionales funcionan mejor con una dimensionalidad reducida y con un gran número de imágenes de características. La U-Net surge precisamente a través de la evolución de redes totalmente convolucionales para la segmentación semántica diseñadas por Shelhamer et al. en 2015 [7].

Por ejemplo, si en la primera capa de la U-Net cada imagen tiene una forma de 6x512x512, cada píxel de la imagen contiene 6 características. En la quinta capa tendrán la forma de 192x16x16. Es decir, las imágenes tendrán un tamaño de 16x16, pero en cambio tendrán un total de 192 características, lo que se puede traducir en que una imagen está compuesta por 192 imágenes de características. Al principio tenemos la información repartida de manera espacial y conforme

vamos aplicando capas de pooling conseguimos aumentar el valor semántico de cada píxel y reducir la información espacial.

La fase expansiva de la red está compuesta del mismo número de capas que en la fase de reducción. El objetivo de esta parte de la red es transformar la representación abstracta de los datos y generar una máscara de segmentación.

Para reconstruir la imagen y volver a aumentar su dimensionalidad, se realiza una operación de upsampling, en nuestro caso es una convolución transpuesta 2x2, que aumenta la dimensionalidad de la imagen. Posteriormente se aplican dos filtros de convolución cada uno seguido de una unidad lineal rectificada para transformar la información abstracta en información espacial de nuevo.

En la Figura 5 se observa como en cada capa de la ruta expansiva se toma información de características de la capa con misma profundidad de la ruta reductora. Cuando realizamos los filtros de convolución de la fase expansiva, tomamos como input no solo la imagen recientemente aumentada, si no también el output de la capa equivalente de la fase reductora. De esta manera, se consigue conservar gran parte de la información espacial de la imagen ayudando así a no perder el contexto de la imagen.

4. Dataset:

En este apartado se expone el cómo y por qué de cada una de las fases de la obtención del dataset y sus respectivas modificaciones para acomodarlo a nuestro problema.

4.1 Obtención del dataset:

En esta fase necesitamos encontrar un dataset que contenga las suficientes radiografías de pulmón con sus debidas máscaras. Para ello, usamos el dataset Kaggle[15] pensado trabajar en la segmentación de imágenes. El dataset contiene radiografías de tórax de los hospitales de Montgomery y Shenzhen[16] con sus respectivas máscaras realizadas por médicos de dichos hospitales.

Una visualización rápida de los datos nos permite observar varias cualidades de este dataset.

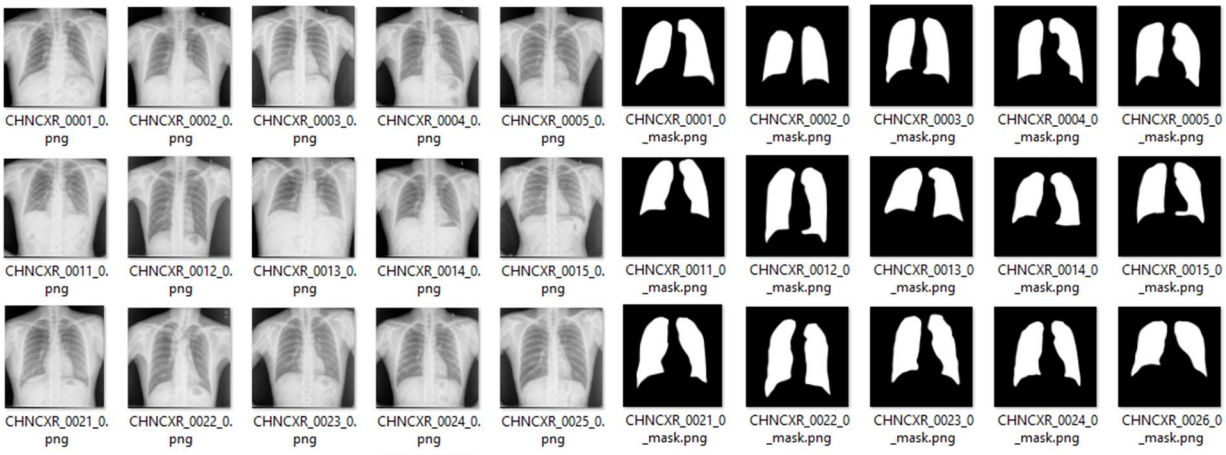

Figura 6. Ejemplos del dataset descargado.

La primera es la nomenclatura, se distinguen entre “MCU” y “CHN” en función de si pertenecen al hospital de Montgomery o de China. Seguido de eso hay un índice de imagen lo cual nos servirá para llevar una asociación entre las imágenes y su etiqueta. El dataset tiene un total de 800 radiografías y 706 máscaras. Una máscara es una imagen que contiene los píxeles etiquetados en su clase que correspondiente a una imagen de entrada. No todas las imágenes tienen máscara. Esto es debido a que hay radiografías que se usarán como validación, no se usarán para el entrenamiento, si no para realizar tests una vez entrenado el modelo. Lo que se busca es que el modelo generalice correctamente, es decir, que sepa predecir datos con los que no ha entrenado.

De esta forma veremos si el modelo es capaz de predecir datos que previamente no ha visto o si por el contrario ha sobre-aprendido demasiado y no es capaz de hacerlo.

La siguiente es que todas las imágenes tienen una resolución similar de unos 2000x2000 pixeles y tienen todas una relación de aspecto cercana a 1:1, dado que no son perfectas y no tienen todas el mismo tamaño. Que sean todas de dimensiones similares nos permite no preocuparnos en exceso por la dimensionalidad de las imágenes. Podemos reducir todas las imágenes a un tamaño más pequeño sin preocuparnos porque puedan deformarse en exceso o porque tengan un ratio de compresión distinto.

Por último, si realizamos un visualizado más detenidamente, nos damos cuenta de que hay pequeñas diferencias a la hora de la creación de las máscaras. Esto se debe probablemente a que varios médicos han trabajado en la creación de las máscaras y por tanto cada uno tiene un criterio propio que difiere mínimamente del de otros. Esto a priori puede resultar desventajoso, pero ayudará a que el modelo aprenda a generalizar mejor y no sobre-aprenda demasiado sobre los bordes de los pulmones. Es decir, esto contribuye a aportar un efecto regularizador a nuestro modelo.

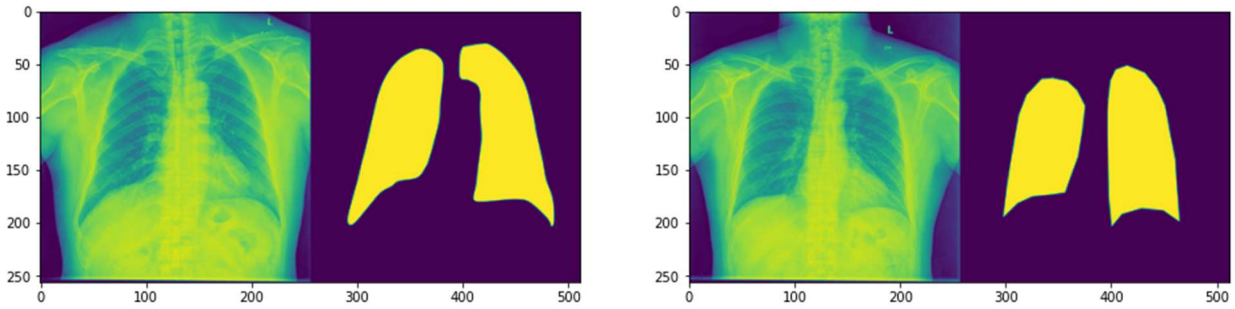

Figura 7: Radiografías del dataset y sus diferencias en las máscaras.

Como podemos observar en la Figura 7, en estas dos radiografías aparentemente similares se han tomado dos criterios de segmentación diferentes. En la primera radiografía se ha segmentado los pulmones dejando fuera de la máscara otros órganos que se superponen en la imagen. En cambio en la segunda radiografía el criterio ha sido el de introducir en la máscara la zona donde se encuentra el pulmón sin importar si también hay otros elementos en la imagen.

4.2 Preprocesamiento de datos:

En esta fase, pasamos a explicar todo el pre-procesamiento de datos que ha sido necesario para acomodar el dataset a las necesidades de nuestro proyecto.

Una vez hemos estudiado el dataset y hemos comprobado que hay suficientes muestras y que estas tienen las características que necesitamos, tenemos que trabajar en el preprocesamiento de imágenes.

La primera tarea a realizar ha sido el transformar todas las imágenes a un formato en escala de grises. Aunque todas las radiografías y sus máscaras estén en blanco y negro, no todas tienen dicho formato, estando algunas en RGB, haciendo que ocupen un espacio extra innecesario y teniendo un tratamiento distinto a la hora de cargarlas. El problema de tener imágenes codificadas de distinta forma reside en que las imágenes RGB tienen 3 canales de información, una para cada color, y las imágenes en escala de grises solo tienen un canal. Por tanto todas las imágenes las hemos cargado, transformado en imágenes de un solo canal de intensidad por píxel y guardadas de nuevo.

La siguiente tarea consiste en igualar el tamaño de las imágenes para poder trabajar con ellas. El modelo que hemos entrenado tiene unas características específicas que nos han hecho seguir el siguiente criterio: Nuestro modelo consiste de varias capas en las cuales reduce la altura y la anchura de la imagen en un factor de 2 y en cambio multiplica los filtros de características por 2. Esto hace que el procesamiento de cada imagen sea muy costoso, por tanto no podemos empezar con un tamaño demasiado grande. Además, al elegir el tamaño de imagen que usamos es conveniente que pueda ser fácilmente divisible entre 2, de no ser así podríamos tener problemas de dimensionalidad en una o varias de las capas con los cuales habría que lidiar.

Es por ello que hemos decidido que el tamaño de imagen sea de 256x256. Es un tamaño lo suficientemente pequeño para que el costo computacional de procesar una imagen no sea muy alto y además las dimensiones son potencia de 2.

En la siguiente tarea del pre-procesamiento hemos trabajado en hacer coincidir las imágenes con sus respectivas máscaras. Debido a que no hay el mismo número de imágenes que de máscaras,

tenemos que mover las imágenes de validación a otra carpeta separada para que haya una correspondencia 1 a 1 de imagen y máscara. Para esta tarea hemos programado un script que compara el índice único de cada una de las imágenes con el índice de las máscaras y las introduce en una carpeta de train y test en caso de haber encontrado su máscara o en una carpeta de validación en caso de que esta no exista. Una vez hecho esto, podemos observar en la Figura 8 como las imágenes y las máscaras se corresponden en un pequeño test realizado.

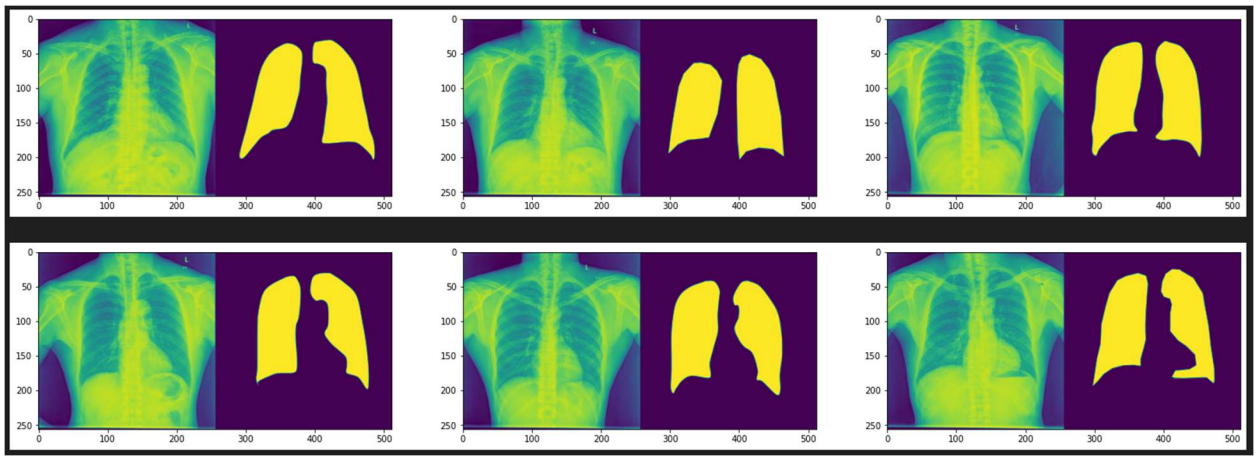

Figura 8. Comprobación de la asociación imagen-máscara.

Como última tarea a realizar para tener el dataset listo, hemos creado un Dataset propio en PyTorch, que nos permite cargar los datos por lotes de manera eficiente. De esta manera podemos recorrer el dataset sin necesidad de que esté cargado entero en la memoria, realizando esta carga en segundo plano y de manera casi automática. También hemos configurado nuestro Dataset para que en cada iteración, devuelva un conjunto aleatorio de train y test, de manera que cada vez que entrenamos el modelo, tenemos datos de train y test distintos. La división de los datos originales en los conjuntos de train y test que hemos escogido ha sido 80% train, 20% test.

Tras todos estos pasos, ya tenemos el dataset preparado para empezar a trabajar con estos datos e implementar el modelo.

5. Implementación:

En este apartado se expone la implementación del modelo, así como los problemas y decisiones que hemos tomado.

5.1 Por qué PyTorch:

Para trabajar con modelos de machine learning hemos usado la librería PyTorch de Python. PyTorch es una librería diseñada para realizar cálculos numéricos haciendo uso de la programación de tensores. Un tensor es un objeto algebraico que describe una relación lineal entre conjuntos de objetos algebraicos relacionados en un espacio vectorial. Nosotros los usamos como matrices multidimensionales que nos permiten manejar los datos en bloque de manera rápida y eficiente.

Además, PyTorch dispone de soporte para realizar los cálculos matemáticos en los tensores en GPU (Tarjeta Gráfica). Poder trabajar con la GPU es conveniente debido a que una CPU, por potente que sea, tiene un número bastante limitado de núcleos. Los núcleos de una CPU son altamente complejos y pueden desempeñar tareas muy costosas en un solo ciclo.

En cambio, en machine learning los cálculos matemáticos a realizar son muy abundantes pero fácilmente paralelizables. Es por esto por lo que usar una CPU es ineficiente, realizamos pocos cálculos por ciclo (tantos como núcleos tenga nuestra CPU), y aunque ocupemos la CPU en su totalidad para esta tarea, se desaprovecha toda la potencia que tiene.

Las tarjetas gráficas en contrapartida tienen miles de nucleos centrados en el cálculo de operaciones en coma flotante. Cada núcleo es muy sencillo, pero en un solo ciclo podemos realizar miles de operaciones de manera simultánea. Este tipo de arquitectura es necesaria debido a la gran cantidad de operaciones que tienen que realizar para renderizar entornos en tres dimensiones y mostrarlos por pantalla.

Para ello, Nvidia desarrolló la plataforma CUDA (Compute Unified Device Architecture), una plataforma de computación paralela y un modelo de programación que permite la ejecución de estas operaciones matemáticas en los cientos de núcleos que tienen sus tarjetas gráficas.[17]

Dicha tecnología está implementada en PyTorch, todas las operaciones de PyTorch tienen una versión que hace uso de CUDA. Gracias a ello obtenemos un rendimiento muy superior reduciendo considerablemente los tiempos de ejecución que obtendríamos si usásemos otras librerías que trabajasen en CPU como numpy.

5.2 Implementación de U-Net:

La U-Net que hemos decidido implementar tiene la siguiente arquitectura: En la fase reductora tiene 4 bloques de reducción y en la fase expansiva tiene 4 bloques de expansión, hemos escogido un número de filtros de característica inicial de 6.

Esto último es debido a que el tamaño del modelo aumenta exponencialmente con el número de filtros de características y números más grandes causaban que el modelo fuese imposible de cargarse en la memoria de la máquina de la que disponemos para el entrenamiento.

En cada una de estos bloques de la fase reductora se realiza una convolución, que multiplica el número de filtros de características por 2 y se reduce la dimensionalidad aplicando funciones de "pooling". Las imágenes, que originalmente tienen un tamaño de 256x256, pasan a tener la siguiente forma a lo largo de las capas: [número de filtros, altura, anchura]

- Capa 1: [6, 128, 128]
- Capa 2: [12, 64, 64]
- Capa 3: [24, 32, 32]
- Capa 4: [48, 16, 16]
- Capa Final: [92, 8, 8]

En la fase expansiva se revierte este proceso, realizando operaciones de convolución inversa y ampliación, volviendo a aumentar el tamaño de la imagen y reduciendo el número de filtros de características.

La arquitectura U-Net combina la información de decodificación de cada bloque de la fase expansiva con información de codificación del respectivo bloque de profundidad de la fase reductora. Al realizar tantas operaciones de filtros de convolución, reducción y ampliación de imagen es fácil pensar que mantener la información espacial va a ser complicado. Pero como

podemos observar en la Figura 9, al procesar una radiografía a través de la U-Net sin entrenar, es decir, con todos los filtros configurados de manera aleatoria, en vez de obtener una imagen llena de ruido, hemos obtenido una imagen que se parece bastante a una radiografía y que guarda cierta relación espacial con la imagen original.

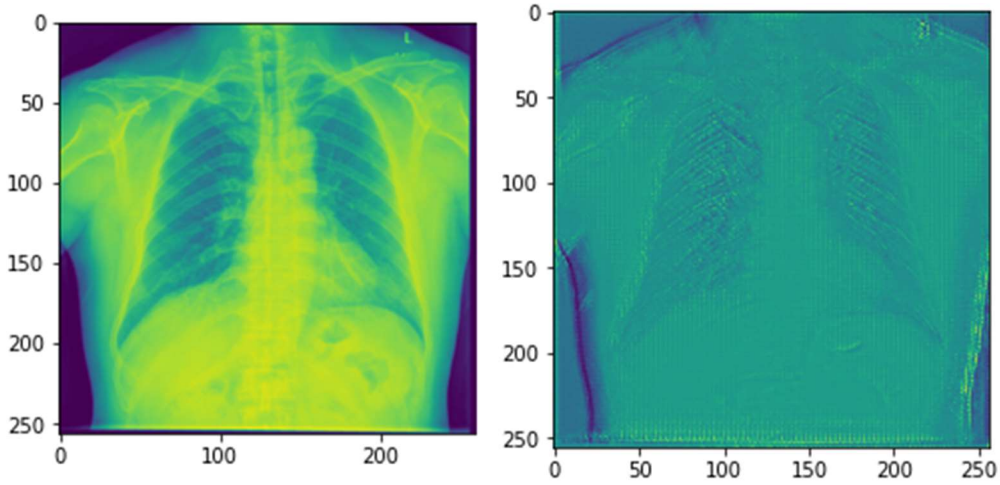

Figura 9. Izquierda: Imagen original. Derecha: Salida de la U-Net sin entrenar.

Esto demuestra que desde un principio la U-Net tiene en cuenta la información espacial de la imagen original, haciendo que la obtención de una máscara tenga un buen punto de partida.

5.3 Función de optimización:

Para entrenar nuestro modelo necesitamos un método de optimización. En PyTorch tenemos distintos optimizadores implementados. Todos estos optimizadores se basan en el algoritmo de descenso por gradiente, implementando mejoras que dan como resultado una mejor velocidad de convergencia del modelo. Nosotros decidimos usar el optimizador Adam. Funciona de manera similar al descenso por gradiente, calculando el gradiente de la función de coste para actualizar los parámetros, con la diferencia de que su tasa de aprendizaje es adaptativa.

Mientras que el descenso de gradiente estocástico mantiene una única tasa de aprendizaje para todas las actualizaciones de peso y se mantiene constante durante todo el entrenamiento. El optimizador Adam adapta la tasa de aprendizaje para todos los parámetros. El algoritmo calcula

un promedio del gradiente y del gradiente al cuadrado y utiliza dos parámetros ajustables para controlar la tasa de descenso de estos promedios. En función de esta tasa de descenso el algoritmo calcula la tasa de aprendizaje actualizada.

El algoritmo usa los siguientes elementos:

- α: Tasa de aprendizaje.
- β1, β2 ∈ [0, 1): Tasa de descenso, se recomienda usar valores cercanos a 1.
- f(W): Función objetivo a minimizar.
- W: Vector de parámetros.
- m: vector de inercia.
- v: vector de inercia con gradiente al cuadrado.
- ϵ: épsilon, valor pequeño para eliminar la posibilidad de dividir por 0.
- t: instante de tiempo.

Calcula los nuevos parámetros de esta manera:

Para un instante de tiempo t:

1. $g_t \leftarrow \nabla_w f_t(W_{t-1})$: Obtenemos los gradientes.
2. $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$: Actualizamos el primer vector de inercia.
3. $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$: Actualizamos el segundo vector de inercia.
4. $m_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$: Suavizamos el primer vector de inercia con el bias.
5. $v_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$: Suavizamos el segundo vector de inercia con el bias.
6. $W_t \leftarrow W_{t-1} - \alpha \cdot \frac{m_t}{\sqrt{v_t+\epsilon}}$: Actualizamos los parámetros.

5.4 Función de Coste:

Las funciones de coste son utilizadas por el optimizado para calcular el gradiente, esta será la función a minimizar para encontrar la mejor solución. La función de coste es esencial para la resolución del problema, debido a que la propia función de coste es lo que define el entrenamiento del modelo.

PyTorch nos proporciona una gran diversidad de funciones de coste, pero nosotros vamos a implementar una función de coste que no viene implementada por defecto, llamada DiceLoss.

La función DiceLoss está basada en el coeficiente de similitud Dice [18, 19], el cual evalúa la superposición existente entre dos muestras segmentadas. El valor del coeficiente de similitud está comprendido entre 0 y 1. Un valor 0 significa que las muestras son completamente opuestas. Cuanto más cercano a 1 sea el coeficiente, más se parecerán las muestras y 1 significa que las muestras son completamente iguales.

Podemos observar en la Ecuación 5, la fórmula empleada para obtener dicho coeficiente:

$$DICE = \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Ecuación 5. Cálculo del coeficiente Dice.

Se calcula el doble de la intersección entre ambos conjuntos y se divide por la suma de elementos de cada conjunto. Una vez obtenido este coeficiente, debemos usar una función que transforme el valor en un coste válido. Esto se debe a que el coste es una función a minimizar y el coeficiente Dice es una función que devuelve un mayor valor cuanto mejor es la solución. Además, calcularemos la media del coste para el lote de muestras que tengamos.

Por tanto, se realiza la operación descrita en la Ecuación 6, para obtener la función de coste:

$$DiceLoss = \frac{\sum_{i=1}^{N}(1 - DICE)}{N}$$

Ecuación 6. Función de coste DiceLoss.

5.5 Funciones de Pooling:

En este apartado, se explica la implementación de una variedad de funciones de pooling para encontrar sus diferencias en los resultados de entrenamiento.

5.5.1 MaxPool:

MaxPool es la función por defecto usada en la U-Net, viene implementada ya en la librería PyTorch.

$$MaxPool(x_1, \dots, x_n) = \max\{x_1, \dots, x_n\}$$

Ecuación 7. Función de agregación MaxPool.

Como podemos observar en la Ecuación 7, MaxPool es una función sencilla que funciona agregando el máximo de los valores de cada ventana.

5.5.2 Normalización y desnormalización:

A la hora de implementar nuestras propias funciones de pooling tenemos un problema, estas funciones están hechas para trabajar sobre un dominio de valores: [0, 1]. Así que antes de aplicar la función de agregación tenemos que aplicar una normalización para pasar del intervalo original [a, b] al intervalo [0, 1]. Después de aplicar la función de agregación tenemos que devolver el valor generado del intervalo [0, 1] al intervalo [a, b].

Dada una función de agregación $g : [0,1]^n \rightarrow [0,1]$ y un isomorfismo $f : [a,b] \rightarrow [0,1]$, se puede emplear la Ecuación 8 para operar con un conjunto de valores $x \in [a,b]^n$.

$$g^*(x) = f^{-1}(g(f(x_1), f(x_2), \dots, f(x_n)))$$

Ecuación 8. Funcionamiento de normalización, agregación y des normalización.

Nosotros hemos decidido aplicar una transformación lineal: $f(x) = \frac{x-a}{b-a}$

Debido a que el tensor no está acotado en ningún intervalo específico, hemos tomando como intervalo [a, b] el mínimo y el máximo elemento, respectivamente, del tensor de entrada.

5.5.3 Funciones de agregación implementadas:

Una vez implementado el modelo usando el máximo, hemos decidido implementar las funciones de grouping descritas en la Tabla 1. Como podemos observar, algunas de las funciones tienen un parámetro adicional "ϵ". Como el algoritmo de “backpropagation” requiere calcular las derivadas de estas funciones y la derivada de $\frac{\partial}{\partial x}\sqrt{x} = \frac{1}{2\sqrt{x}}$, añadimos "$\epsilon$", que es un valor muy pequeño para evitar divisiones por cero en la derivada cuando x = 0.

$$GMA(x_1, \dots, x_n) = (\max\{x_1, \dots, x_n\})^p$$
$$GS(x_1, \dots, x_n) = 1 - \prod_{i=1}^{n} (1 - x_i)^p$$
$$GMP(x_1, \dots, x_n) = 1 - \sqrt[n]{\prod_{i=1}^{n} (1 - x_i)^p + \epsilon}$$
$$GB(x_1, \dots, x_n) = 1 - \sqrt[n]{(\prod_{i=1}^{n} (1 - x_i)) \cdot \min\{1 - x_1, \dots, 1 - x_n\} \cdot + \epsilon}$$
$$GU(x_1, \dots, x_n) = \frac{\max\{x_1, \dots, x_n\}}{\max\{x_1, \dots, x_n\} + \sqrt[n]{\prod_{i=1}^{n} (1 - x_i) + \epsilon}}$$

Tabla 1. Funciones de grouping implementadas.

6. Experimentación y resultados:

En esta sección exponemos las diferentes fases de experimentación, las pruebas que hemos realizado, los resultados que hemos obtenido y las conclusiones a las que hemos llegado.

6.1 Métricas de entrenamiento y rendimiento:

En este apartado expondremos los resultados tras entrenar varios modelos usando distintas funciones de agregación. Hemos implementado una "pipeline" para ejecutar cada entrenamiento con diferentes particiones de train y test, las diferentes funciones de agregación y la hemos ejecutado repetidas veces para obtener una media de los resultados. Hemos entrenado todos los modelos 12 veces con un total de 30 épocas. Una época consiste en una pasada sobre todos los ejemplos del conjunto de entrenamiento.

Hemos valorado los resultados obtenidos en base a las siguientes métricas:

- Observando su DiceLoss, para ver como converge el modelo a lo largo de las iteraciones.
- Observando su Accuracy Score, que se calcula como el porcentaje de los píxeles acertados de la imagen.

6.1.1 MaxPool:

En la Figura 10 podemos observar la gráfica de convergencia del modelo para la función de agregación MaxPool.

Como podemos apreciar, el modelo converge rápidamente, llega a converger en unas 7 iteraciones y a partir de ahí el entrenamiento se estanca.

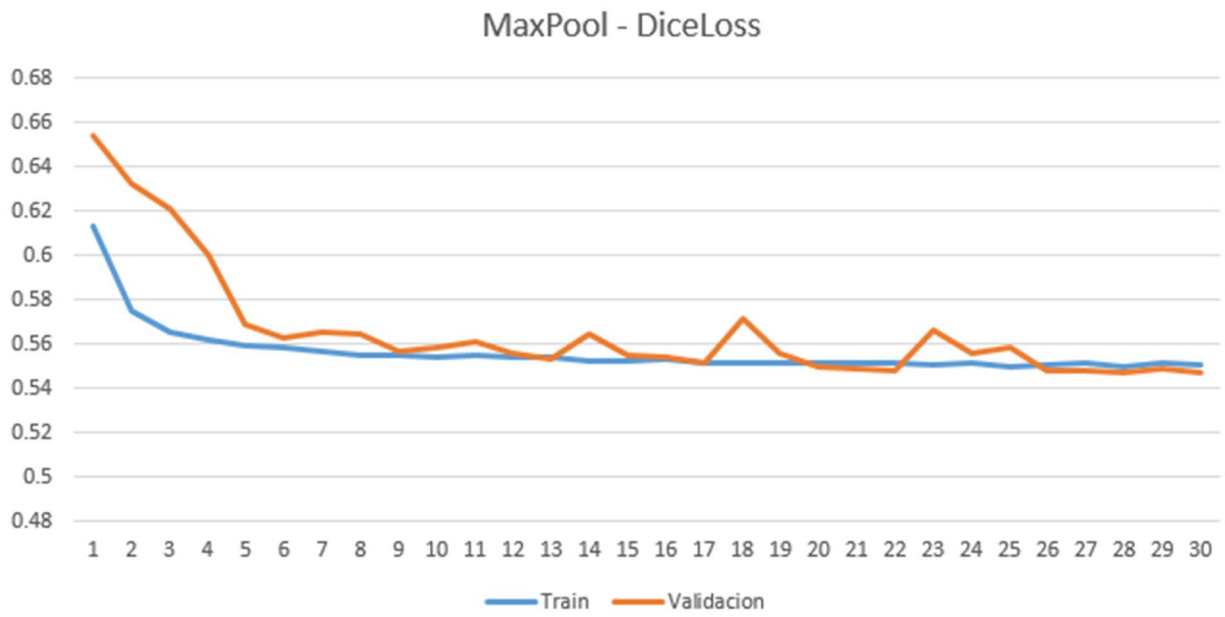

Figura 10. Gráfica de evolución de DiceLoss para la función MaxPool.

Accuracy de los resultados obtenidos:

- Train Accuracy: 97,68%
- Validation Accuracy: 97,41%

6.1.2 GMA:

En la Figura 11 podemos observar la gráfica de convergencia del modelo para la función de agregación GMA.

Este modelo tarda en converger algo más, podemos observar que también llega rápidamente al estancamiento en unas 8 iteraciones.

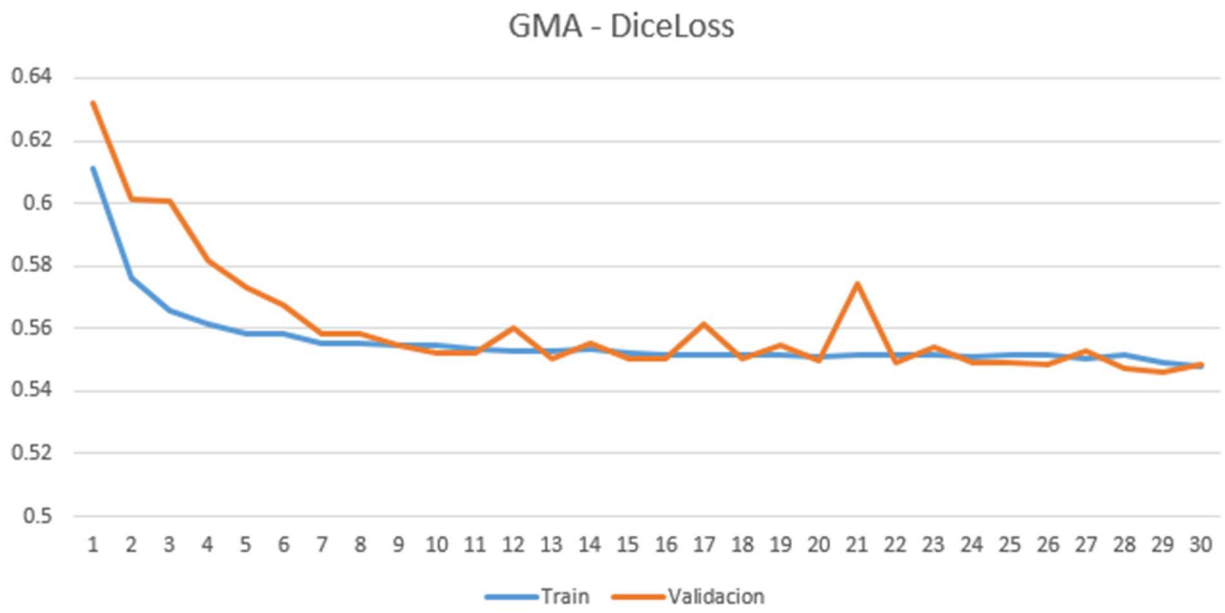

Figura 11. Gráfica de evolución de DiceLoss para la función GMA.

Accuracy de los resultados obtenidos:

- Train Accuracy: 97,37%
- Validation Accuracy: 97.01%

6.1.3 GS:

En la Figura 12 podemos observar la gráfica de convergencia del modelo para la función de agregación GS.

Este modelo parece converger rápido en train, en las primeras 10 iteraciones, sin embargo, podemos observar que le lleva unas 20 iteraciones llegar a los valores de los modelos MaxPool o GMA. Podemos observar que su DiceLoss en Validación no desciende de la misma manera, sino que lo hace de forma menos pronunciada y más errática. Lo cual podría indicar que este modelo tiende más al over-fitting que los mencionados anteriormente.

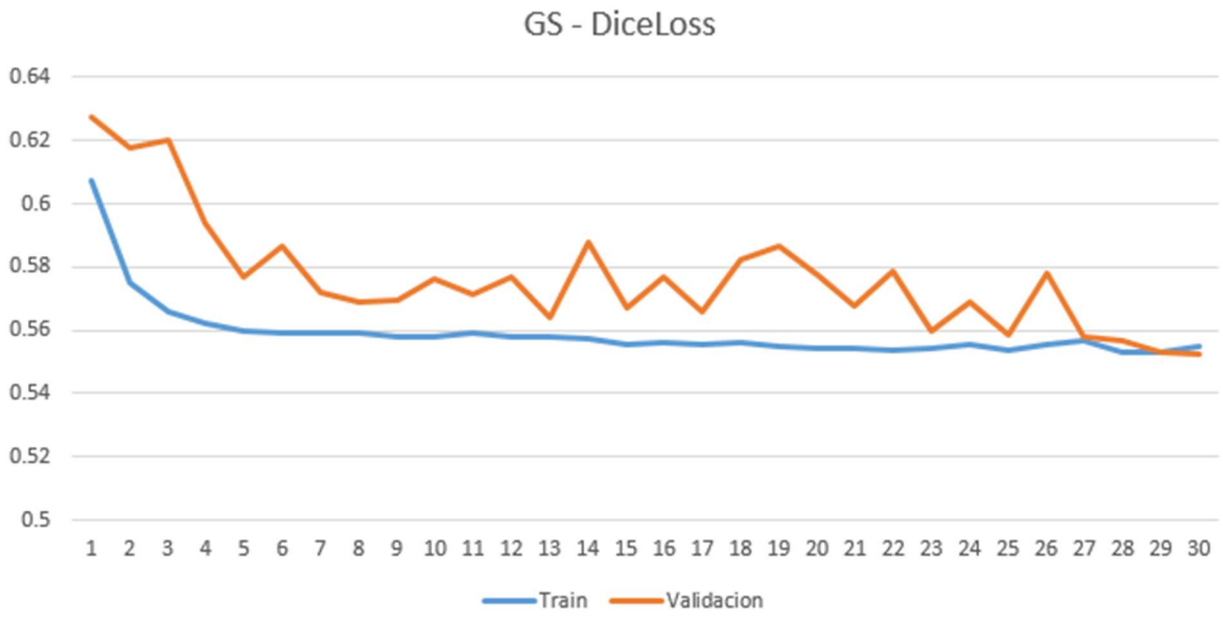

Figura 12. Gráfica de evolución de DiceLoss para la función GS.

Accuracy de los resultados obtenidos:

- Train Accuracy: 95,94%
- Validation Accuracy: 95,86%

6.1.4 GMP:

En la Figura 13 podemos observar la gráfica de convergencia del modelo para la función de agregación GMP.

Este modelo al igual que GS, converge rápido en Train pero le cuesta alcanzar los valores de DiceLoss de GMA o MaxPool. Podemos observar que, al igual que en GS, el descenso en Validación tiene una varianza todavía mayor y es menos pronunciado que en GS.

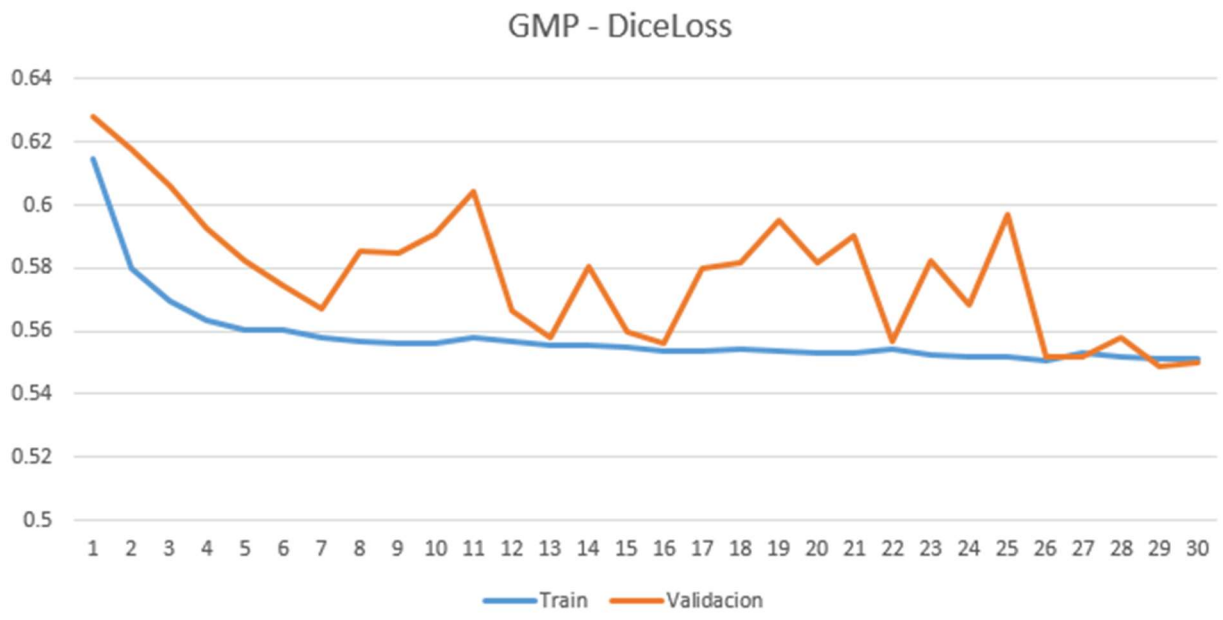

Figura 13. Gráfica de evolución de DiceLoss para la función GMP.

Accuracy de los resultados obtenidos:

- Train Accuracy: 95,34%
- Validation Accuracy: 94,92%

6.1.5 GB:

En la Figura 14 podemos observar la gráfica de convergencia del modelo para la función de agregación GB.

Este modelo al igual que GS y GMP, converge rápido en Train pero le cuesta alcanzar los valores de DiceLoss de GMA o MaxPool, en este caso podríamos hacer notar que el entrenamiento se estanca antes, pero tarda más que los anteriores en alcanzar los mismos valores. Respecto a los valores de DiceLoss de validación, podemos observar que su descenso es algo más estable que GMP y se acerca más a la métrica de train.

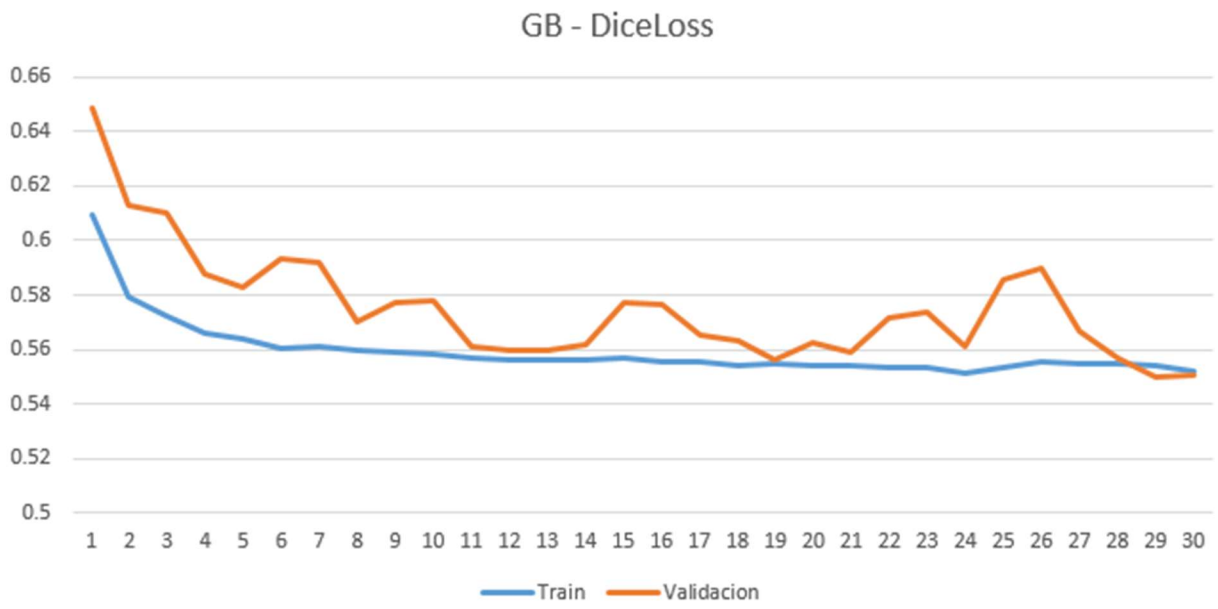

Figura 14. Gráfica de evolución de DiceLoss para la función GB.

Accuracy de los resultados obtenidos:

- Train Accuracy: 96,36%
- Validation Accuracy: 96,16%

6.1.6 GU:

En la figura 15 podemos observar la gráfica de convergencia del modelo para la función de agregación GU.

Este modelo converge de una manera muy similar a GB, se estanca rápido y los valores de la DiceLoss en validación se acercan más a los valores de train que en otros modelos. Podemos intuir que el modelo tiende menos al over-fitting que GMP, pero a niveles similares de GB o GS.

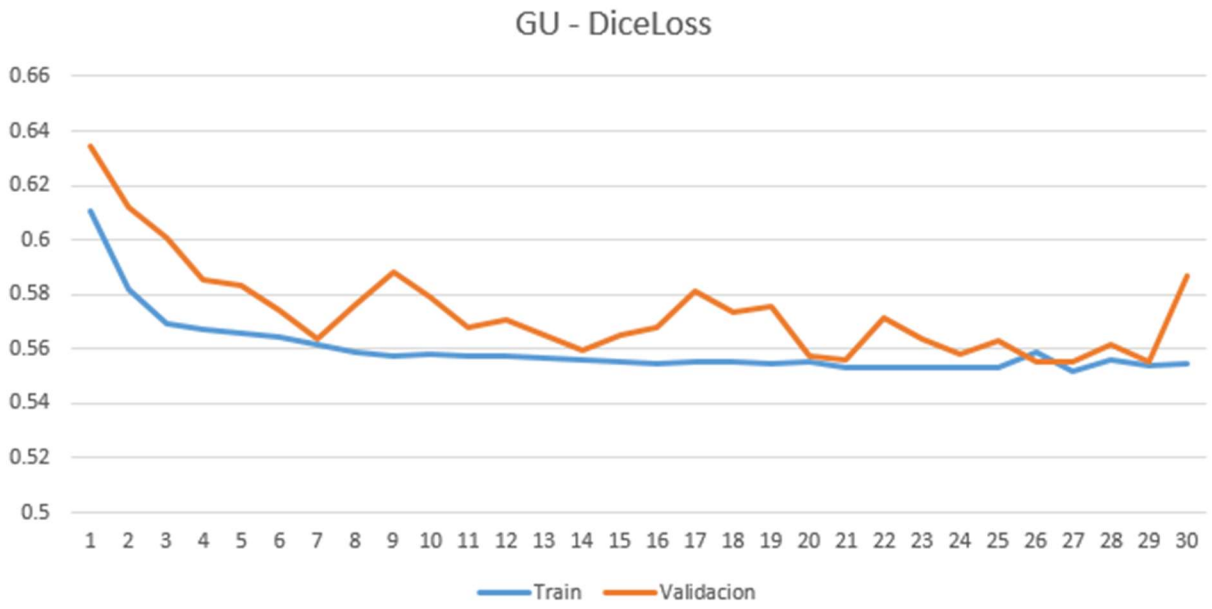

Figura 15. Gráfica de evolución de DiceLoss para la función GU.

Accuracy de los resultados obtenidos:

- Train Accuracy: 95,92%
- Validation Accuracy: 94.93%

6.1.7 Resumen de resultados:

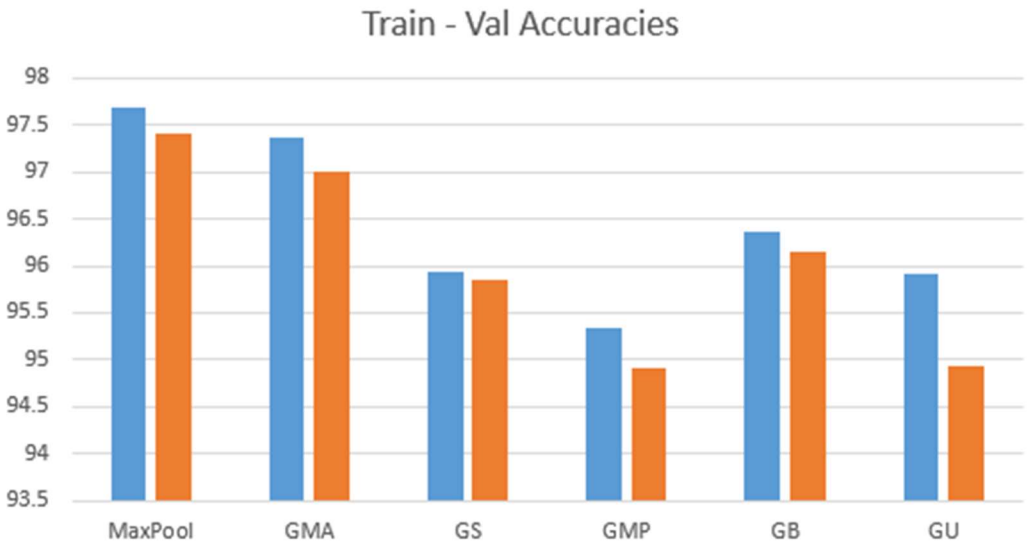

Figura 16. Comparativa de accuracies de train y validación de los modelos.

Todos los modelos convergen de manera similar, no parece que a partir de las 10 iteraciones haya demasiada diferencia en los valores de DiceLoss. Hemos dividido los resultados en los 3 siguientes grupos:

MaxPool y GMA son las dos funciones que mejores resultados dan a priori, tiene sentido que estos resultados se parezcan tanto ya que ambas funciones de agregación se parecen en su funcionamiento. Tienen una convergencia tanto en train como validación estable y sin mucha separación ni varianza.

Las funciones de agregación GS, GB y GU dan unos resultados similares entre sí, no llegan a los mismos valores de Accuracy que el grupo comentado anteriormente. Su convergencia en train es estable, pero el descenso en validación tiene una varianza más alta y hay una diferencia mayor respecto al descenso de train.

La función de agregación GMP es la que peor resultados nos da. Su Accuracy es menor en comparación con todas las demás funciones y su descenso en el Loss de validación tiene una varianza muy alta, pudiendo ser este un signo de que generaliza peor que el resto.

6.2 Resultados de entrenamiento:

Para obtener una mejor visualización de estos resultados y observar las diferencias en la práctica hemos predicho varias radiografías de la carpeta de validación con cada uno de los modelos entrenados. Al ser imágenes con las que los modelos no han entrenado podemos comprobar lo bien o mal que generaliza cada modelo.

La ilustración 8 muestra una comparativa de las predicciones de validación que hemos realizado. Cada columna pertenece a un ejemplo distinto, la primera fila corresponde a las imágenes de entrada que los modelos han tenido que predecir y las siguientes filas corresponden a los diferentes modelos entrenados etiquetados por su función de agregación.

En las máscaras obtenidas, las zonas en verde representan los pulmones y las zonas de la imagen en azul oscuro representan el fondo.

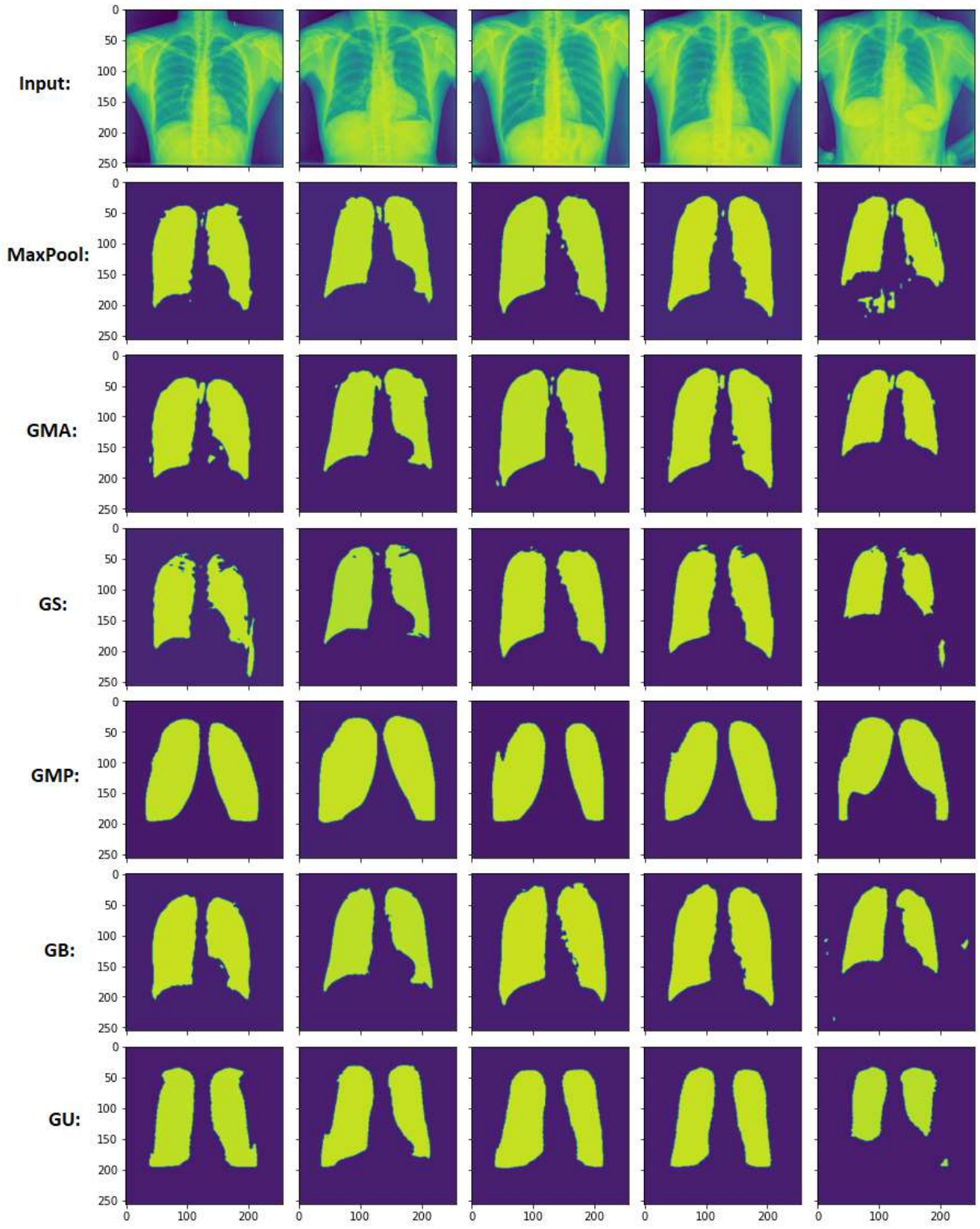

Figura 17. Prueba de predicción de validación. En la fila 1 las radiografías de input, en el resto las predicciones de los distintos modelos.

Las observaciones basadas en nuestra experimentación y la Figura 17 nos hacen sacar el siguiente análisis respecto al entrenamiento de los clasificadores:

- Como podemos observar en la Figura 17, todos los modelos han dado unos resultados lo suficientemente correctos para segmentar las imágenes y guardar una buena correlación entre los pulmones de las imágenes de entrada y de las máscaras. En todos los casos podemos ver el fondo y los pulmones claramente diferenciados en las máscaras. Esto respalda las métricas obtenidas en el apartado anterior.

Podemos ver como los diferentes modelos realizan distintas segmentaciones:

- **MaxPool:** Fijándonos en Figura 17, podemos llegar a la conclusión de que la función de agregación MaxPool realiza una segmentación con las siguientes características: Las máscaras obtenidas tienen los bordes irregulares, por lo que podemos intuir que intenta sobre-aprender los bordes. Podemos observar también que hay pequeñas manchas en el fondo clasificadas como pulmón, ya sea entre los pulmones y cerca de ellos o incluso alejadas de ellos como es el caso de la última muestra.
- **GMA:** Fijándonos en la Figura 17, podemos llegar a la conclusión de que la función de agregación GMA realiza una segmentación con las siguientes características: Tal y como esperábamos, los resultados obtenidos se parecen bastante a los devueltos cuando usamos la función MaxPool. Las máscaras obtenidas tienen los bordes irregulares. Podemos observar que al igual que en el caso anterior, hay pequeñas manchas en el fondo clasificadas como pulmón, pero en este caso dichas manchas están más cerca, o incluso pegadas al pulmón mientras que no parece haber manchas clasificadas como pulmón lejos de este, como se observa en la última muestra.
- **GS:** Fijándonos en la Figura 17, podemos llegar a la conclusión de que la función de agregación GS realiza una segmentación con las siguientes características: Como esperábamos, este clasificador nos ha dado resultados peores que los dos anteriores. Podemos observar como hay pequeñas zonas de los pulmones que están clasificadas como fondo en la parte superior de los pulmones en casi todas las muestras. También podemos observar como los bordes son más irregulares que en los casos anteriores y

como hay alguna zona del fondo clasificada como pulmón. Estas carencias son más aparentes en la primera y la última imagen.

- **GMP:** Fijándonos en la Figura 17, podemos llegar a la conclusión de que la función de agregación GMP realiza una segmentación con las siguientes características: Esperábamos que este modelo devolviera los peores resultados basándonos en las métricas devueltas, con un porcentaje de acierto en validación bastante bajo. Sin embargo, hemos obtenido resultados igualmente interesantes. Es quizá el modelo que menos ha aprendido, sus bordes son poco orgánicos, es decir, son están redondeados. Podemos observar como este modelo ha clasificado zonas más allá de los pulmones como pulmón, es interesante ver como los anteriores modelos han intentado clasificar la intersección entre corazón y pulmón como fondo y en cambio este modelo ha optado por considerarla parte de los pulmones. No hay grandes manchas clasificadas como pulmón alejadas de estos, pero en cambio podemos observar como en la quinta muestra la zona de los pulmones se alarga excesivamente hacia abajo.
- **GB:** Fijándonos en la Figura 17, podemos llegar a la conclusión de que la función de agregación GU realiza una segmentación con las siguientes características: Según las métricas esperábamos obtener unos resultados similares al máximo, pero algo peores. Como podemos ver en las imágenes, los bordes son irregulares, llegando a verse pequeñas manchas que conforman los bordes de los pulmones. No hay grandes fallos en la clasificación aunque si podemos observar pequeñas manchas alejadas de nuevo en la última muestra evaluada.
- **GU:** Fijándonos en la Figura 17, podemos llegar a la conclusión de que la función de agregación GU realiza una segmentación con las siguientes características: No esperábamos los mejores resultados debido a sus valores de Loss en validación. Sin embargo, al igual que en GMP, se ha realizado una segmentación interesante. De la misma forma que ocurría en GMP los bordes son bastante redondeados y poco precisos, con la diferencia de que GMP tiende a "expandir" la zona de pulmón más allá de los pulmones y en cambio GU tiende a tener una zona de pulmón más reducida. Como se

puede ver, no hay muchas zonas fuera de los pulmones que hayan sido clasificadas como tal. También hay una pequeña mancha alejada de los pulmones en la muestra 5.

6.3 Generalización del modelo:

Basándonos en los resultados obtenidos, podemos concluir que el modelo entrenado con la función máximo segmenta las imágenes de mejor forma que cualquiera de las demás funciones. Para comprobar si la implementación de nuestro modelo generaliza bien, hemos probado a predecir muestras de otro dataset completamente distinto. El dataset "covid_gr" es un dataset que contiene radiografías de tórax de personas con coronavirus y personas sanas. Las imágenes de este dataset tienen una relación de aspecto y resolución distintos al dataset utilizado para entrenamiento. Además de que las imágenes tienen un encuadre diferente, no estando tan igualmente centradas en los pulmones.

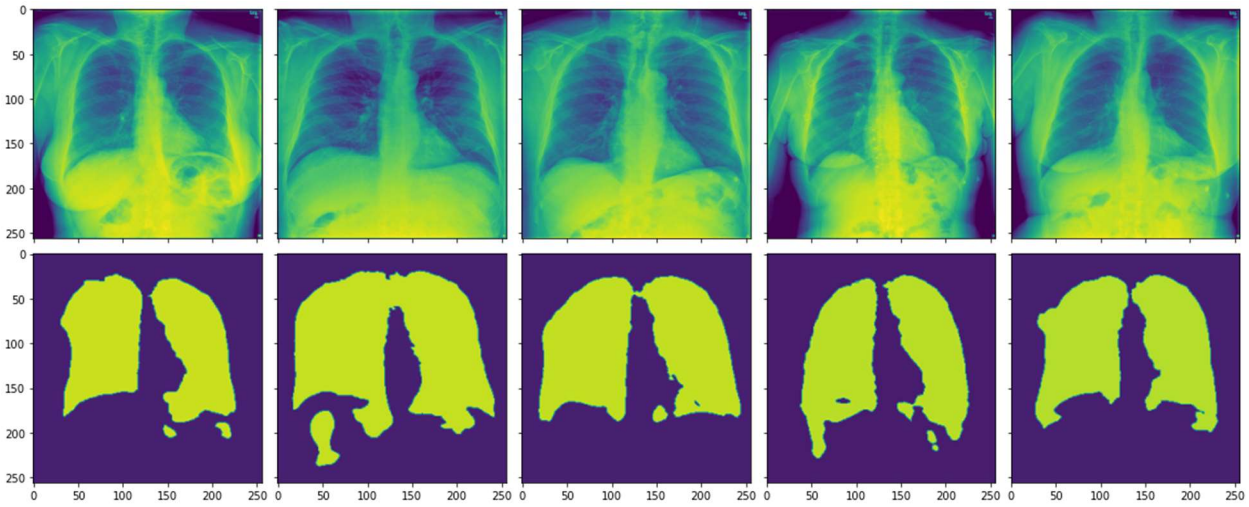

Figura 18. Prueba de validación con dataset covid_gr. Pacientes covid negativo.

En la Figura 18 podemos observar los resultados de predecir una serie de muestras con nuestro modelo. Podemos apreciar como la segmentación ha empeorado considerablemente, se sigue manteniendo una forma bastante clara de los pulmones. Sin embargo los bordes ya no son tan precisos, clasificando erróneamente las áreas cercanas a estos. También apreciamos un claro incremento de las manchas en el fondo, alejadas de los pulmones, consideradas como pulmones.

Una vez hemos visto los resultados que arroja en pacientes sanos, vamos a probar con pacientes con una sintomatología de covid-19 presente. Esta clasificación supone un reto extra ya que la neumonía aparece en forma de manchas de alta intensidad en las radiografías, similares a las de otros tejidos del cuerpo, por lo que intuimos que esto puede confundir al modelo y hacer que clasifique dicha parte de los pulmones como fondo.

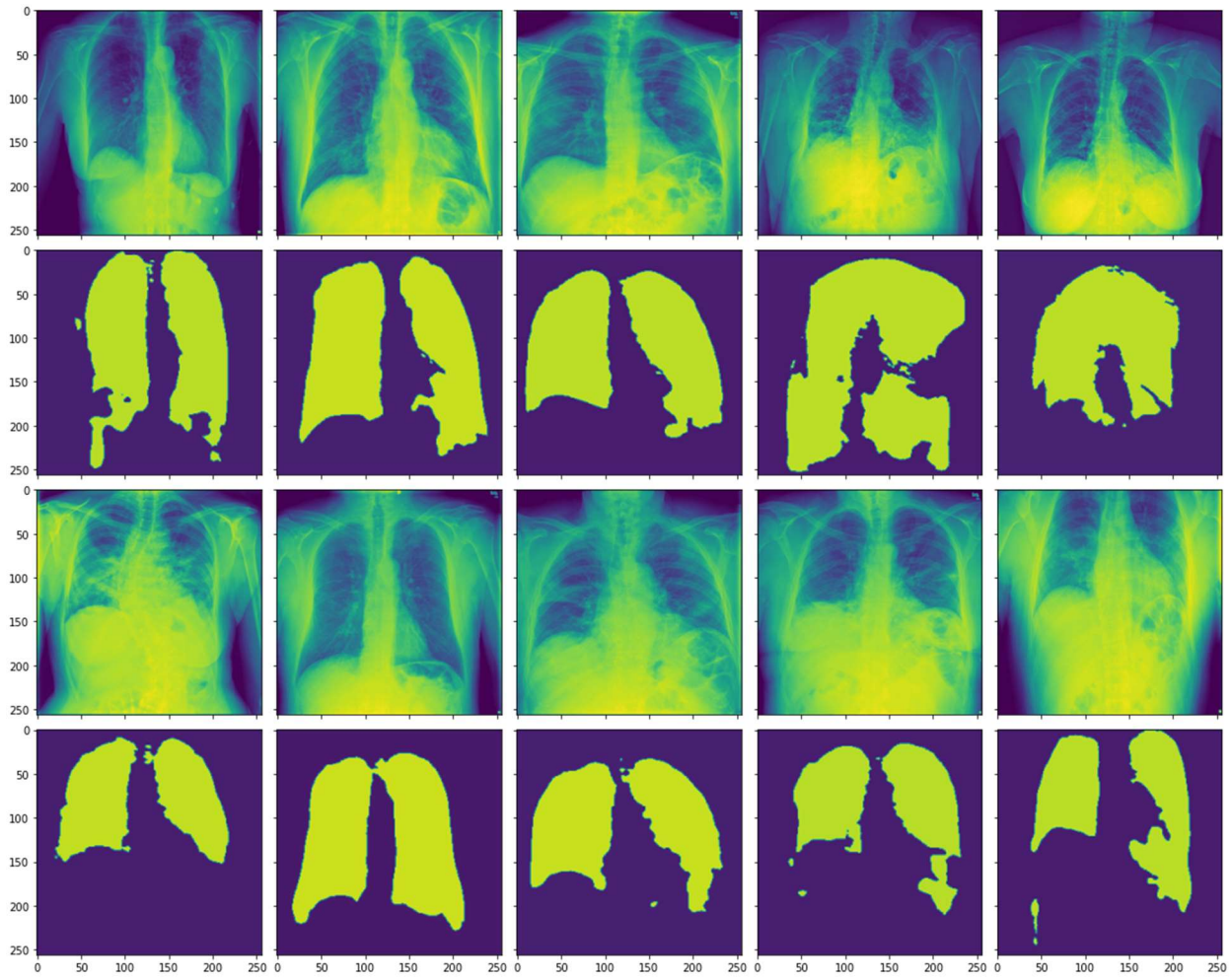

Figura 19. Prueba de validación con dataset covid_gr. Pacientes covid positivo.

En la Figura 19 podemos observar las predicciones realizadas por el modelo en radiografías de pacientes con covid-19. A simple vista podemos ver como la segmentación es algo peor. Hay imágenes en las que el modelo se ajusta mejor a los pulmones, sin embargo, las radiografías en las que la neumonía es muy grave, puede verse como deja zonas de estos sin clasificar. Podemos destacar también que en la última radiografía los pulmones no están completamente dentro de

la imagen, pero el modelo redondea el borde superior como si ahí acabasen. Esto se debe claramente a que el modelo tiene sobre aprendidas ciertas zonas donde deberían estar los pulmones y por tanto tiende a clasificarlas más como tal. Lo cual nos indica que el encuadre y la forma que tengan las radiografías sí que es algo a tener en cuenta a la hora de configurar y entrenar nuestro modelo.

7. Conclusión:

Tras los resultados obtenidos en nuestra experimentación podemos llegar a varias conclusiones:

- La primera conclusión es que la arquitectura U-Net funciona realmente bien segmentando radiografías de pulmón. También comprobamos que la mejor función de agregación a la hora de implementar el grouping es el máximo. Aun así, cabe destacar que otras funciones de grouping aportan resultados interesantes.
- La siguiente conclusión a la que podemos llegar es que al verse reflejada la neumonía en las radiografías, en forma de manchas blancas, ésta es fácilmente confundible con otros tejidos del cuerpo. Resulta complicado para un modelo, que ha entrenado con muestras sanas, segmentar las imágenes de un paciente enfermo.

Cabe destacar que los objetivos del trabajo de fin de grado han sido alcanzados con éxito, ya que hemos conseguido segmentar las imágenes del dataset original con un porcentaje de acierto notablemente alto con todas las funciones de agregación implementadas, demostrando que pueden usarse otras funciones más allá del máximo.

En cuanto a líneas futuras, podríamos intentar que el modelo generalizase mejor utilizando un dataset más variado, con radiografías en distintos encuadres, tamaños y orientaciones. También podríamos intentar mejorar la segmentación de radiografías en pacientes positivos utilizando un dataset que incluya tanto radiografías como máscaras con pacientes positivos y probar con más funciones de pooling que quizá ofrezcan mejores resultados.

También nos hubiese gustado utilizar esta segmentación de radiografías para, más adelante, entrenar un modelo que clasifique las imágenes de pacientes enfermos de covid-19 en diferentes grados de gravedad de neumonía.

8. Bibliografía:

[1]: Sattar, S. B. A., Sharma, S., & Headley, A. (2021). Bacterial Pneumonia (Nursing). In StatPearls [Internet]. StatPearls Publishing.

[2]: Williams, M. B., Krupinski, E. A., Strauss, K. J., Breeden III, W. K., Rzeszotarski, M. S., Applegate, K., ... & Seibert, J. A. (2007). Digital radiography image quality: image acquisition. Journal of the American College of Radiology, 4(6), 371-388.

[3]: Chuang, C. L., & Chen, C. M. (2007). A novel region-based approach for extracting brain tumor in CT images with precision. In World Congress on Medical Physics and Biomedical Engineering 2006 (pp. 2488-2492). Springer, Berlin, Heidelberg.

[4]: Avazpour, I., Saripan, M. I., Nordin, A. J., & Abdullah, R. S. A. I. (2009). Segmentation of extrapulmonary tuberculosis infection using modified automatic seeded region growing. Biological procedures online, 11(1), 241-252.

[5]: Ciofolo, C., & Fradkin, M. (2008, September). Segmentation of pathologic hearts in long-axis late-enhancement MRI. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 186-193). Springer, Berlin, Heidelberg.

[6]: Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., ... & Larochelle, H. (2017). Brain tumor segmentation with deep neural networks. Medical image analysis, 35, 18-31.

[7]: Kayalibay, B., Jensen, G., & van der Smagt, P. (2017). CNN-based segmentation of medical imaging data. arXiv preprint arXiv:1701.03056.

[8]: McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

[9]: Fukushima, K., & Miyake, S. (1982). Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Competition and cooperation in neural nets (pp. 267-285). Springer, Berlin, Heidelberg.

[10]: Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

[11]: Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

[12]: Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32.

[13]: Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural networks, 2(5), 359-366.

[14]: Gómez, D., Rodriguez, J. T., Montero, J., Bustince, H., & Barrenechea, E. (2016). n-Dimensional overlap functions. Fuzzy Sets and Systems, 287, 57-75.

[15]: E. (2018, 22 octubre). U-Net lung segmentation (Montgomery + Shenzhen). Kaggle. https://www.kaggle.com/code/eduardomineo/u-net-lung-segmentation-montgomery-shenzhen/notebook

[16]: Lung Masks for Shenzhen Hospital Chest X-ray Set. (2018a, marzo 3). Kaggle. https://www.kaggle.com/datasets/yoctoman/shcxr-lung-mask

[17]: CUDA Zone. (2022, 18 febrero). NVIDIA Developer. https://developer.nvidia.com/cuda-zone

[18]: Sørensen, T. (1948). "A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons". Kongelige Danske Videnskabernes Selskab. 5 (4): 1–34.

[19]: Dice, L. R. (1945). Measures of the amount of ecologic association between species. Ecology, 26(3), 297-302