



UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática



Segmentación de aorta en imágenes médicas 2D y 3D utilizando redes neuronales

Autor: Elio García González

Tutores: Agustín Trujillo Pino, Carmelo Cuenca Hernández

Grado en Ingeniería Informática, Julio de 2018

Universidad de Las Palmas de Gran Canaria, España

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/D^a Elio García González, autor del Trabajo de Fin de Título *Segmentación de aorta en imágenes médicas 2D y 3D utilizando redes neuronales*, correspondiente a la titulación de Grado en Ingeniería Informática, en colaboración con la empresa/proyecto (indicar en su caso) _____

S O L I C I T A

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[] Se ha iniciado o hay intención de iniciarlo (defensa no pública).
[X] No está previsto.

Y para que así conste firma la presente.

Las Palmas de Gran Canaria, a 25 de Junio de 2018.

El estudiante

Fdo.: _____

A rellenar y firmar **obligatoriamente** por el/los tutor/es

En relación a la presente solicitud, se informa:

[] Positivamente

[] Negativamente

(la justificación en caso de informe negativo deberá incluirse en el TFT05)

Fdo.: _____

DIRECTOR DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Resumen

En los últimos años el uso de redes neuronales se ha disparado en todo tipo de aplicaciones posibles. Concretamente, en el campo de la segmentación se han producido grandes avances, consiguiendo superar las técnicas tradicionales de procesamiento de imágenes en numerosas ocasiones.

A pesar de estas mejoras, la clasificación de cada píxel en una imagen sigue siendo una tarea complicada dentro del mundo de la medicina debido, principalmente, a la falta de imágenes presegmentadas por expertos. Otros problemas, como las diferencias que hay entre imágenes tomadas entre dos escáneres, dificultan dicha tarea.

En este trabajo de investigación utilizaremos varias arquitecturas populares, y no tan populares, para la segmentación de la arteria aorta en imágenes TC (Tomografía axial computarizada). Específicamente hablaremos de la viabilidad entre los modelos 2D y 3D, además de entender el efecto que tienen las diferentes técnicas y operaciones implementadas.

Abstract

Over the last years the use of neuronal networks has skyrocketed in every posible application. Specifically, great breakthroughs has been achieved in the field of segmentation, surpassing traditional imaging processing techniques in numerous occasions.

In spite of this advances, single pixel identification within an image is still a difficult task in medical fields, mainly due to the lack of pre-segmented images by experts. Other problems include the variation between images of two different scanners.

This research paper uses a variety of well-known, and lesser-known, architectures for aorta images segmentation in CAT (Computed Axial Tomography) scans. We will talk, specifically, about the viability of 2 and 3D , as well as, the effect that different techniques and operations have.

1. Introducción	6
1.1 Estado actual y Motivación	6
1.2 Objetivos y Metodología	6
1.3 Competencias	7
1.4 Organización del Documento	7
2. Tecnologías y herramientas	8
2.1 MRICron e ImageJ	8
2.2 Python y Tensorflow	8
2.3 Tensorboard	9
2.4 Hardware y FloydHub	9
2.5 Aplicación de despliegue	10
3. Redes Neuronales	11
3.1 Introducción a las redes neuronales	11
3.1.1 Funcionamiento de una neurona	11
3.1.2 El Gradiente y la propagación hacia atrás	15
3.1.3 Funciones de activación	18
3.1.4 Conexiones Residuales	21
3.1.5 Evaluación de la red	22
3.1.6 Elementos regularizadores	23
3.2 Redes neuronales convolutivas	24
3.2.1 La convolución	24
3.2.2 Campo de visión	26
3.2.3 Segmentación y Convolución	28
3.3 Redes Neuronales Recurrentes	29
4. Pruebas y Resultados	30
4.1 Preparación	30
4.1.1 Datos de entrada.	30
4.1.2 Salida y calidad de la red.	32
4.1.3 Tipo de error y método de optimización	33
4.1.4 Dropout y normalización por lotes	34
4.2 Segmentación 2D	35
4.2.1 U-net	35
4.2.2 DeepLabv3	38
4.2.3 Observaciones 2D	42
4.3 Segmentación 3D	45
4.3.1 3D U-net y V-net	45
4.3.2 CRNNs	47
4.3.3 Observaciones 3D	48

5. Conclusiones	48
6. Bibliografía	51
5. Enlaces de interés	53

1. Introducción

1.1 Estado actual y Motivación

La facultad de Ingeniería Informática de la ULPGC se encuentra trabajando con imágenes de múltiples pacientes con el objetivo de segmentar la arteria aorta. Las imágenes han sido obtenidas en el hospital de Santiago de Compostela y pre-segmentadas por un experto.

Las técnicas que se usan en segmentación son muy variadas y suelen depender en gran medida del problema y conjunto de datos con el que se trabaja. La mayoría de los métodos que se utilizan actualmente, incluido el de la facultad, obtienen resultados del 90% o más en precisión[25][26]. Por otro lado, el uso de aprendizaje máquina, para la segmentación de imágenes médicas, está teniendo bastante éxito en otras aplicaciones[27][28].

Debido a la facilidad de obtención de las imágenes y a que la solución actual del problema utiliza un procesamiento tradicional de imágenes, resulta atractivo comprobar el desempeño de las redes neuronales para esta tarea.

1.2 Objetivos y Metodología

En este trabajo se comparará el uso de arquitecturas 2D y 3D para la segmentación de la aorta en imágenes tomadas por escáner CT. Las redes convolucionales en dos dimensiones trabajan cada imagen por separado perdiendo información espacial sobre su profundidad y por tanto no pueden reconocer que la aorta es un elemento tubular y alargado. Esperamos tener una mejora de los resultados en tres dimensiones, a pesar de que la cantidad de parámetros puede jugar un papel relevante durante su implementación y entrenamiento.

Otros tipos de arquitecturas utilizadas usan redes convolucionales recurrentes (CRNN) para mantener información sobre la secuencia de imágenes en el volumen. Este tipo de red trabaja en dos dimensiones y por tanto consumen menos memoria. En el documento la incluiremos junto con los modelos 3D.

Se tendrá presente el uso de técnicas regularizadoras, como *dropout*[1] y la normalización por lotes o *batchnormalization*[2], que han adquirido popularidad y que suelen mejorar los resultados cuando se utilizan.

También analizaremos los resultados entre métodos para el aumento y la reducción de las resolución, funciones de error y activación típicos de la segmentación.

1.3 Competencias

IS04. Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales.

Se cumple al fusionar en una aplicación herramientas y funciones útiles para el desarrollo de la investigación. (apartado 2.5)

CP03. Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.

CP04. Capacidad para conocer los fundamentos, paradigmas y técnicas propias de los sistemas inteligentes y analizar, diseñar y construir sistemas, servicios y aplicaciones informáticas que utilicen dichas técnicas en cualquier ámbito de aplicación.

Las competencias CP03 y CP04 se cumplen durante el estudio y la aplicación de las redes neuronales y sus técnicas para la segmentación.

1.4 Organización del Documento

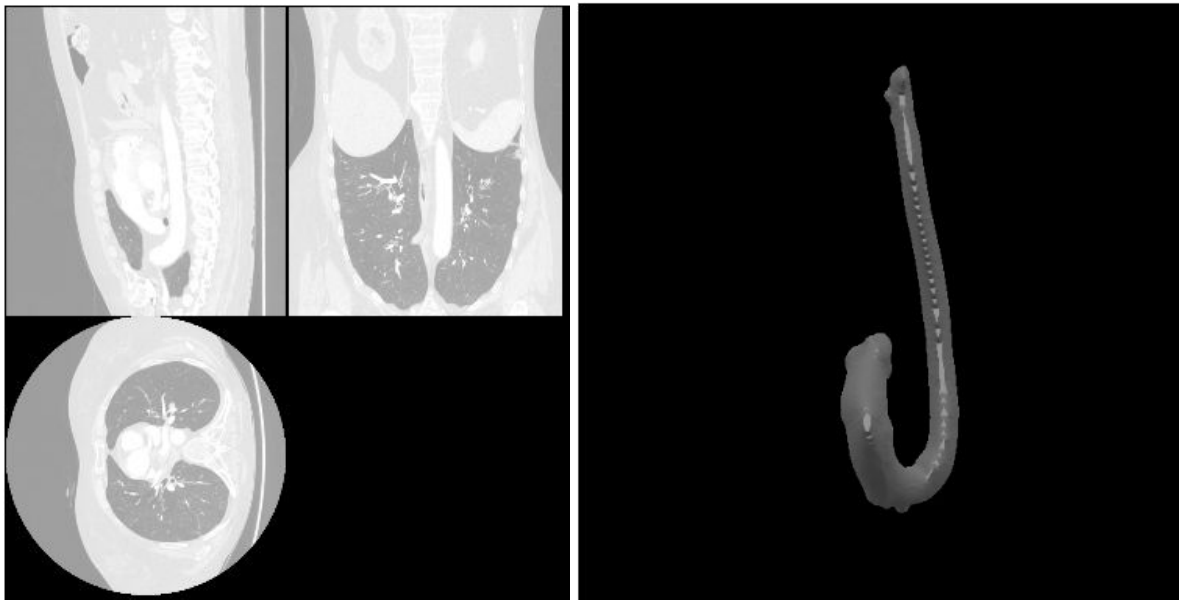
Tras este apartado, el documento se encuentra organizado en cuatro bloques. En el primero expondremos las herramientas y tecnologías usadas. En el segundo se hará una introducción al funcionamiento de las redes neuronales y, en concreto, al proceso de segmentación con arquitecturas convolutivas.

Los dos últimos apartados de este documento hacen referencia a las pruebas realizadas y conclusiones obtenidas.

2. Tecnologías y herramientas

2.1 MRICron e ImageJ

MRICron[\[a\]](#) es una herramienta para visualizar imágenes en diferentes formatos médicos. Es simple y fácil de usar comparada con ImageJ[\[b\]](#), pero esta última nos permite aplicar diferentes procesos sobre los volúmenes de datos.



(Izquierda) Vistas de cada eje para una imagen 3D (Derecha) Renderización 3D de la aorta.

2.2 Python y Tensorflow

Python es uno de los lenguajes más usados para hacer Deep Learning actualmente. Muchas librerías y frameworks para el tratamiento de datos y aprendizaje máquina tienen interfaces en Python, convirtiéndose en la opción más popular para este tipo de aplicaciones.

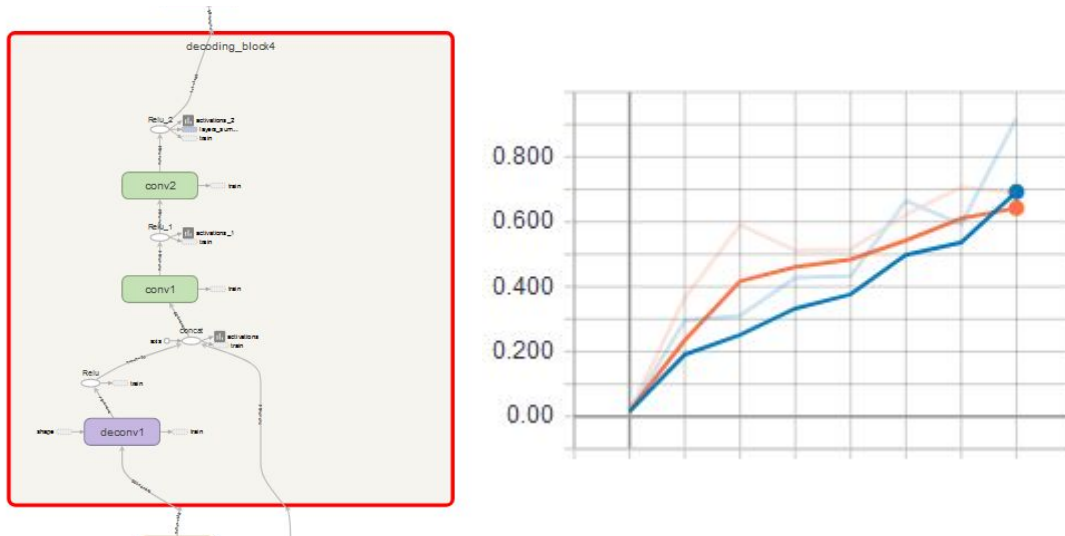
Las librerías por excelencia para Deep Learning son Theano[\[c\]](#), CNTK[\[d\]](#) y Tensorflow[\[e\]](#), luego existen diferentes APIs escritas en Python que trabajan encima de estas librerías como Keras[\[f\]](#) y TFLearn[\[g\]](#).

En este proyecto no usaremos ninguna de las APIs de alto nivel y nos centraremos en la versión 1.5 de Tensorflow por sus herramientas y comodidad frente a otras opciones. Se trata de una librería abierta que permite una sencilla implementación en GPU además de un fácil guardado, recuperación y despliegue de los modelos entrenados.

Tensorflow crea un grafo internamente para modelar el flujo de datos y utiliza sesiones para ejecutar partes de este grafo, permitiendo ciertas ventajas en paralelismo, ejecuciones distribuidas, compilación y portabilidad.

2.3 Tensorboard

Para visualizar y hacer debug de la redes mientras entrenan usamos Tensorboard. Se trata de un aplicación web para inspeccionar y entender los ejecuciones en Tensorflow y sus grafos. Viene incluido en la distribución oficial de Tensorflow.



(Izquierda) Ejemplo de visualización del grafo del modelo. (Derecha) Datos recogidos durante el entrenamiento

2.4 Hardware y FloydHub

Las ejecuciones y recogida de datos se lanzan en Floydhub[\[h\]](#). Se trata de un servicio de computación en la nube donde utilizamos una gráfica Tesla K80 de 12GB. El entorno virtual incluye 64 GB de ram permitiéndonos precargar todas las imágenes y evitar así cuellos de botella con los discos duros.

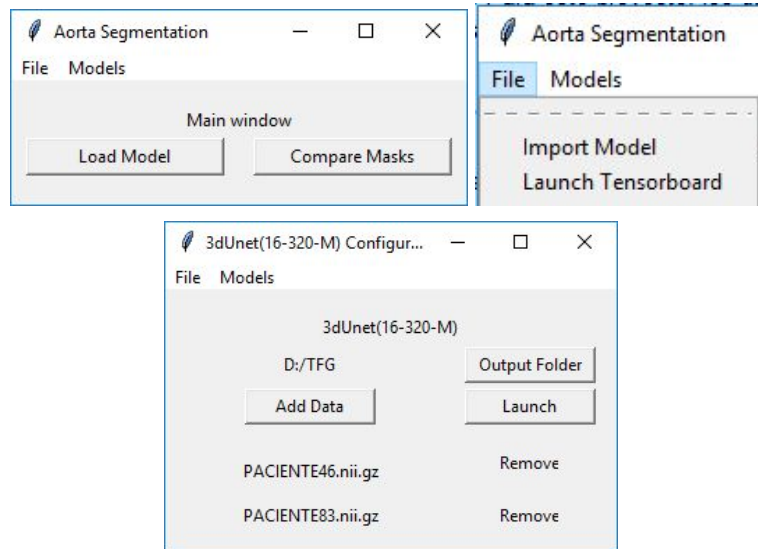
Running		zathay/projects/tfg/267	zathay submitted 25 minutes ago			
Shutdown		UnetDice(D-M)	Loss	mIoU	validation_loss	validation_mIoU
		Manage tags	0.042	0.92	0.181	0.708
Success		zathay/projects/tfg/265	zathay submitted 1 week ago			
		Unet RCRNN77(D-M)	Loss	mIoU	validation_loss	validation_mIoU
		Manage tags	0.013	0	0.034	0.29

Ejecuciones en floydhub

Floydhub guarda un historial de las ejecuciones, su código y los datos recogidos dentro de sus servidores. Para este proyecto, los datos superan los 2TB por lo que se agradece su espacio de almacenamiento.

2.5 Aplicación de despliegue

Para el despliegue de los diferentes modelos, el lanzamiento de Tensorboard y la comparación entre máscaras de segmentación se ha desarrollado un ejecutable con interfaz gráfica y multiplataforma en Python. Este ejecutable permite iniciar Tensorfboard, ejecutar la inferencia de los modelos con un conjunto de imágenes y comparar la calidad de los resultados obtenidos.



(Arriba) Ventana inicial del programa. (Abajo) Ventana de configuración para la ejecución de un modelo.

3. Redes Neuronales

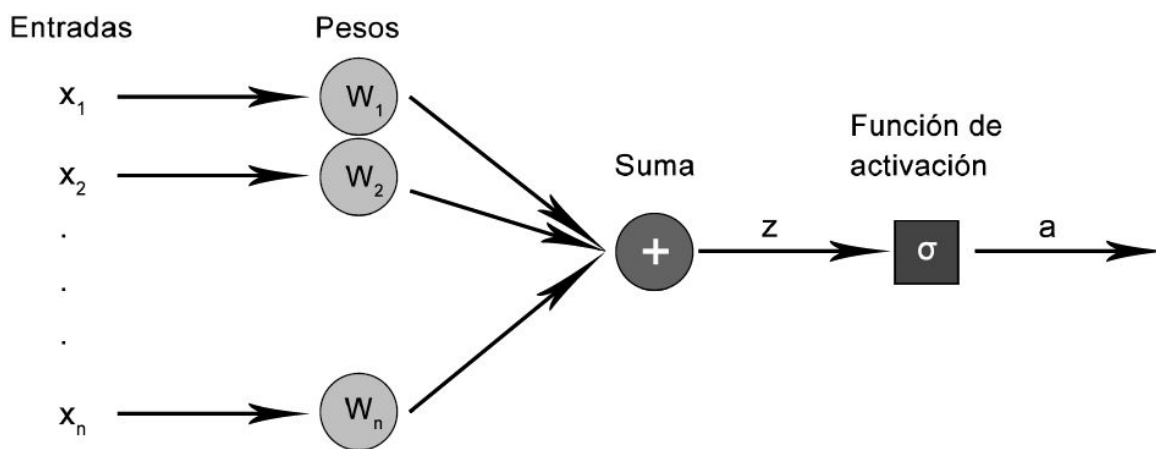
En este apartado se incluyen los conocimientos necesarios para entender el funcionamiento de las redes neuronales y su uso para problemas de segmentación.

3.1 Introducción a las redes neuronales

Una red neuronal es un sistema de procesamiento de la información compuesto por nodos o neuronas interconectadas. Estas neuronas asignan un valor numérico o peso a sus entradas y producen una salida que permite resolver problemas de regresión (predecir una cantidad) y clasificación (predecir una clase).

3.1.1 Funcionamiento de una neurona

Cuando las entradas a una neurona superan un umbral, esta se activa propagando su salida. Las conexiones de entrada que son fuertes aportan más que las débiles a esta activación. A continuación se describe el modelo matemático más simple que implementa el funcionamiento de una neurona biológica.



Modelo de una neurona.

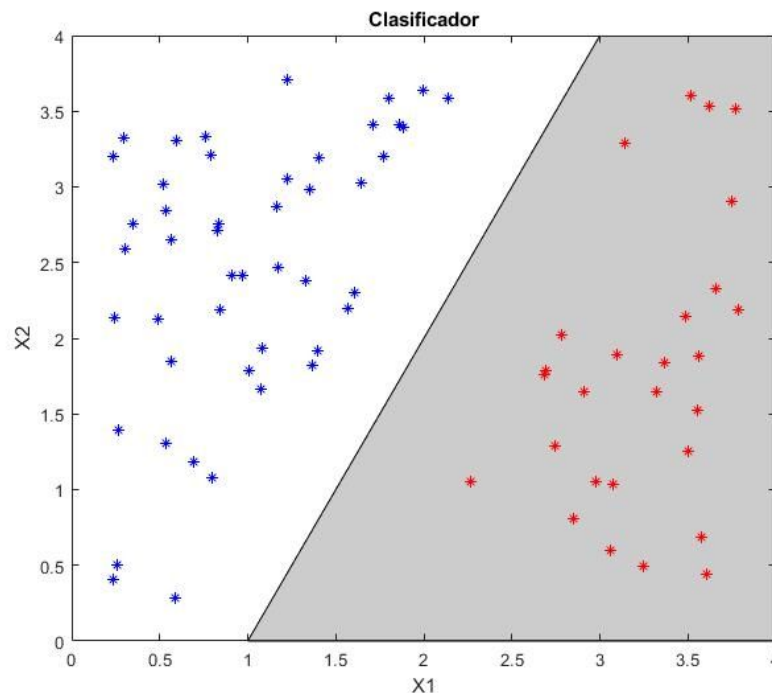
Siendo x_i la entrada y los parámetros ajustables w_i y b , tenemos que la función para la regresión lineal es:

$$z = \sum_i (w_i \times x_i) + b$$

Si utilizamos una función para la activación de la neurona que aplique un umbral sobre z , $a = \sigma(z)$, obtenemos la ecuación de un clasificador lineal:

$$a = \sigma(\sum_i (w_i \times x_i) + b)$$

En el caso de que la función de activación sea $\sigma(z) = 1$ para $z > 0$ y $\sigma(z) = 0$ para $z \leq 0$ la neurona puede resolver problemas de clasificación binaria. Utilizando dos entradas (x_1, x_2), dos tipos de datos (Rojo, Azul) y el siguiente hiperplano:



Siendo m la pendiente y c el punto de corte con el eje x_2 , tenemos que el espacio de activación (sombreado) viene descrito por:

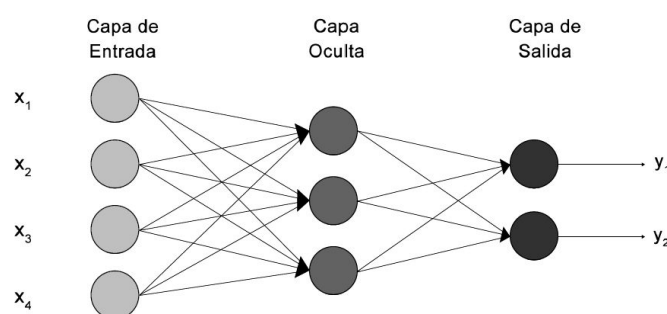
$$x_2 < m \cdot x_1 + c \Rightarrow m \cdot x_1 - x_2 + c > 0$$

Al compararlo con la ecuación de la neurona para la clasificación binaria

$w_1 \cdot x_1 + w_2 \cdot x_2 + b > 0$ obtenemos los valores $w_1 = m$, $w_2 = -1$ y $b = c$.

De esta forma la neurona se activará siempre que la entrada se encuentre en la región descrita por sus parámetros internos.

La colocación de neuronas en forma de capas se denomina arquitectura multicapa y permite producir regiones de decisión de diferente complejidad y forma, siempre y cuando no se utilicen funciones de activación lineales. Al número de capas se le conoce por profundidad.



Arquitectura multicapa con dos salidas.

Siendo la salida de la neurona j en la capa L igual a:

$$a_j^L = \sigma_j^L(z_j^L), z_j^L = \sum_i (w_{i \rightarrow j}^L \times a_i^{L-1}) + b_j^L, y_i^0 = x_i$$

Se puede convertir en operaciones con matrices:

$$A^L = \sigma^L(Z^L), Z^L = W^L \times A^{L-1} + B^L, Y^0 = X$$

Cuando usamos la función de activación identidad ($\sigma(z) = z$) en arquitecturas multicapa, al igual que el resto de funciones de activación lineales, se produce una salida lineal. Esto quiere decir que equivale a implementar una monocapa como se muestra en la siguiente transformación:

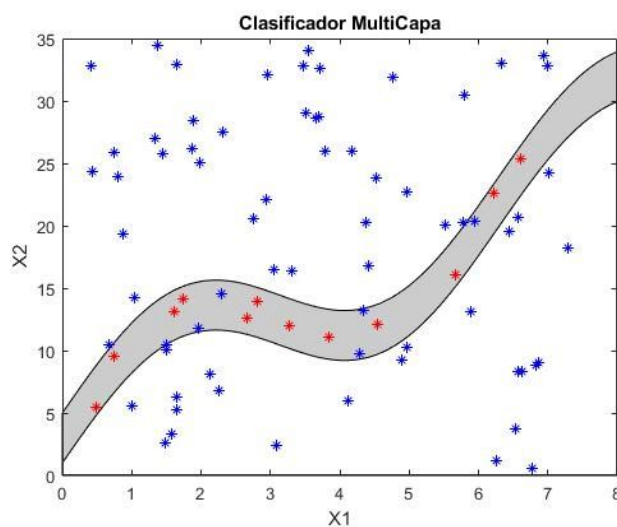
Utilizando una red de dos capas con una neurona en cada capa y una sola entrada:

$$y = w_2 \cdot (w_1 \cdot x_1 + b_1) + b_2$$

$$w_{monocapa} = w_1 \cdot w_2 \text{ y } b_{monocapa} = w_2 \cdot b_1 + b_2$$

Se comprueba que el resultado sigue siendo una función lineal, pudiendo reducirse a una sola neurona y perdiendo la capacidad de solucionar problemas complejos.

La introducción de funciones no lineales permiten a la redes neuronales, con los suficientes parámetros y tiempo, modelar cualquier tipo de función en arquitecturas multicapa. Es común utilizar un tipo de activación para construir la red y otro diferente justo antes de la salida para cambiar el tipo de resultado buscado (clasificación binaria, regresión...).



Ejemplo de espacio de decisión complejo

Un problema complejo es capaz de necesitar miles de neuronas y el ajuste de sus parámetros se vuelve una práctica difícil de hacer manualmente. Según el tipo de problema y la información conocida previamente, existen diferentes técnicas de aprendizaje que se pueden agrupar en 3 tipos.

Aprendizaje por refuerzo

Los datos de entrada no son conocidos directamente. Existe un agente (la red neuronal) que interacciona con un entorno o medio realizando acciones y cambiando de estado. Las acciones en cada paso que ayuden a llegar al estado deseado son reforzadas de manera que, cuando el agente se encuentre en la misma situación, seleccione la acción con mayor recompensa.

Aprendizaje no supervisado

Los datos de entrada son conocidos y utilizamos una función de coste dependiente de los valores de entrada y la salida de la red que tratamos de minimizar. La selección de esta función depende del problema a solucionar.

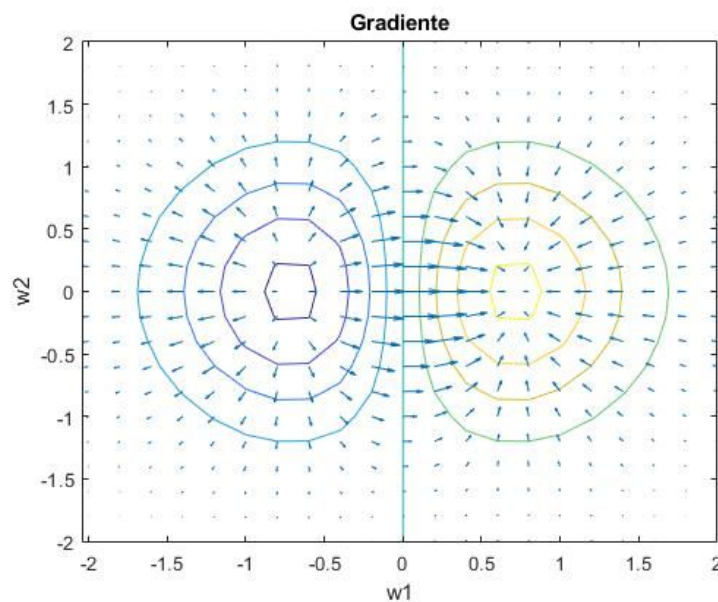
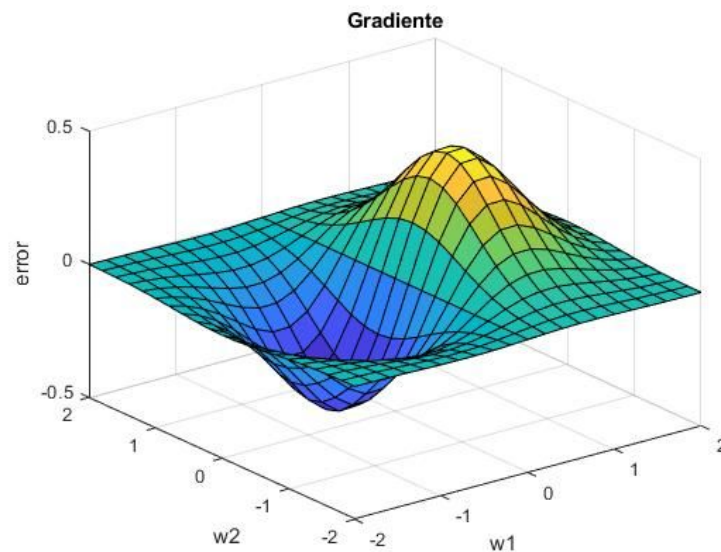
Aprendizaje supervisado

En el aprendizaje supervisado se conocen los pares de datos de entrada y la salida. Se utiliza una función de error, como puede ser el error cuadrático medio, entre la salida de la red y la salida deseada para ajustar los parámetros de la red.

3.1.2 El Gradiente y la propagación hacia atrás

El gradiente es un vector de n-dimensiones que proporciona la dirección de crecimiento y la variación de una función con respecto a los cambios de sus variables independientes. En nuestro caso el gradiente del error tendrá tantas dimensiones como variables entrenables y, al conocer hacia dónde decrece la función de error, podremos encontrar sus valores óptimos.

$$\nabla E = \left(\frac{\partial E}{\partial w^1}, \frac{\partial E}{\partial b^1}, \dots, \frac{\partial E}{\partial w^L}, \frac{\partial E}{\partial b^L} \right)$$



Recordar que cuando tenemos una función dentro de otra, $f(g(x))$, su derivada es la variación de f con respecto a g por la variación de g respecto a la variable independiente x . Esto se puede aplicar tantas veces como funciones intermedias existan por lo que es conocida como la regla de la cadena.

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial c} \frac{\partial c}{\partial y} \dots \frac{\partial b}{\partial x}$$

En el caso de una arquitectura con una neurona por capa definimos su función de error como:

$$E(w^1, b^1, \dots, w^L, b^L) = (y_{predicción} - y_{esperada})^2 = (a^L - y_{esperada})^2$$

$$a^L = \sigma(z^L) \text{ y } z^L = w^L a^{L-1} + b^L$$

El cálculo de las componentes del gradiente en cada capa viene dado por:

Capa de Salida L:

$$\frac{\partial E}{\partial w^L} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial w^L} \Rightarrow \frac{\partial a^L}{\partial w^L} = \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L}$$

$$\frac{\partial E}{\partial w^L} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^L} \Rightarrow \frac{\partial z^L}{\partial w^L} = a^{L-1}, \frac{\partial E}{\partial a^L} = 2(a^L - y), \frac{\partial a^L}{\partial z^L} = \sigma'(z^L)$$

En el caso del bias hay que sustituir $\frac{\partial z^L}{\partial w^L} \Rightarrow \frac{\partial z^L}{\partial b^L} = 1$

Capa L-1:

$$\frac{\partial E}{\partial w^{L-1}} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial w^{L-1}} \Rightarrow \frac{\partial a^L}{\partial w^{L-1}} = \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial w^{L-1}} \Rightarrow \frac{\partial z^L}{\partial w^{L-1}} = \frac{\partial a^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial w^{L-1}} \Rightarrow \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial w^{L-1}}$$

$$\frac{\partial E}{\partial w^{L-1}} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial w^{L-1}} \Rightarrow \frac{\partial z^{L-1}}{\partial w^{L-1}} = a^{L-2}, \frac{\partial z^L}{\partial a^{L-1}} = w^L, \frac{\partial a^{L-1}}{\partial z^{L-1}} = \sigma'(z^{L-1})$$

Capa L-2:

$$\frac{\partial E}{\partial w^{L-2}} = \frac{\partial E}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \frac{\partial a^{L-1}}{\partial z^{L-1}} \frac{\partial z^{L-1}}{\partial a^{L-2}} \frac{\partial a^{L-2}}{\partial z^{L-2}} \frac{\partial z^{L-2}}{\partial w^{L-2}} \Rightarrow$$

$$\Rightarrow \frac{\partial z^{L-2}}{\partial w^{L-2}} = a^{L-3}, \frac{\partial z^{L-1}}{\partial a^{L-2}} = w^{L-1}, \frac{\partial a^{L-2}}{\partial z^{L-2}} = \sigma'(z^{L-2})$$

Capa X:

$$\frac{\partial E}{\partial w^X} = 2(a^L - y) \times a^{X-1} \times \prod_{i=X}^{i=L} (\sigma'(z^i)) \times \prod_{i=X+1}^{i=L} w^i$$

Como se observa en las ecuaciones, todas las operaciones deben ser diferenciables y, para saber cuánto aportan los parámetros de una capa al error total, es necesario conocer el resultado de su variación por las capas siguientes (elementos coloreados). Para ello comenzamos el cálculo desde la salida dándosele el nombre de propagación hacia atrás del gradiente.

Existen dos problemas con el cálculo del gradiente a medida que aumenta la profundidad de la red. Supongamos el siguiente caso.

Uno de los términos de los que depende $\frac{\partial E}{\partial w^x}$ son los pesos de las capas consecuentes $\prod_{i=x+1}^{i=L} w^i$.

Utilizando el mismo peso en cada capa, cuando $|w| > 1$ obtenemos un aumento o explosión del gradiente pero si $0 < |w| < 1$ tendemos a 0, lo que se conoce como desvanecimiento del gradiente. En caso de que sature, las capas más cercanas a la entrada de la red tendrán mayores modificaciones que las últimas. Si por el contrario ocurre el desvanecimiento, las modificaciones de los pesos son insignificantes haciendo que la red deje de aprender en las primeras capas y repercutiendo en el resto.

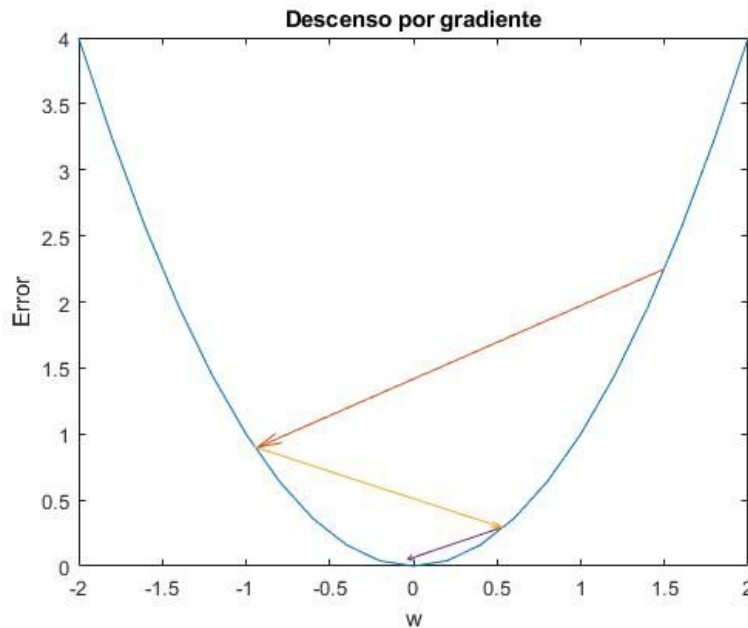
La modificación de los pesos también depende de la función de activación elegida $\prod_{i=x}^{i=L} (\sigma'(z))$ permitiendo mitigar o agravar los problemas de gradiente como veremos en el siguiente apartado. Otra opción bastante popular es añadir conexiones residuales que sumen la señal de entrada de una capa con su salida de forma que cada una afecta directamente al error de la red. $a_L = \sigma(x + a_1 + a_2 + \dots + a_{L-1})$

Otras soluciones añaden un componente dentro del cálculo del error como las funciones L1 y L2 de regularización. Este tipo de funciones son dependientes de los pesos, penalizando los valores muy grandes o muy pequeños.

Conocidos todos los componentes del gradiente es hora de minimizar la función de error. Un método de optimización común se conoce como descenso por gradiente que mediante saltos, proporcionales a la aportación del error, en la dirección de decrecimiento, resulta en un proceso iterativo para la búsqueda de mínimos. La cantidad de salto se denomina ratio de aprendizaje y valores muy grandes pueden provocar que no llegue a converger en el mínimo (zig-zag) o incluso divergir mientras que valores muy pequeños hace que el aprendizaje sea más lento.

$$w = w_{anterior} - \alpha \times \frac{\partial E}{\partial w}$$

No suele ser necesario decrecer el ratio de aprendizaje con el número de iteraciones ya que los saltos son proporcionales al módulo del gradiente y tiende a 0 a medida que nos acercamos a un mínimo.



Ejemplo de descenso por gradiente sobre la función de error para una variable.

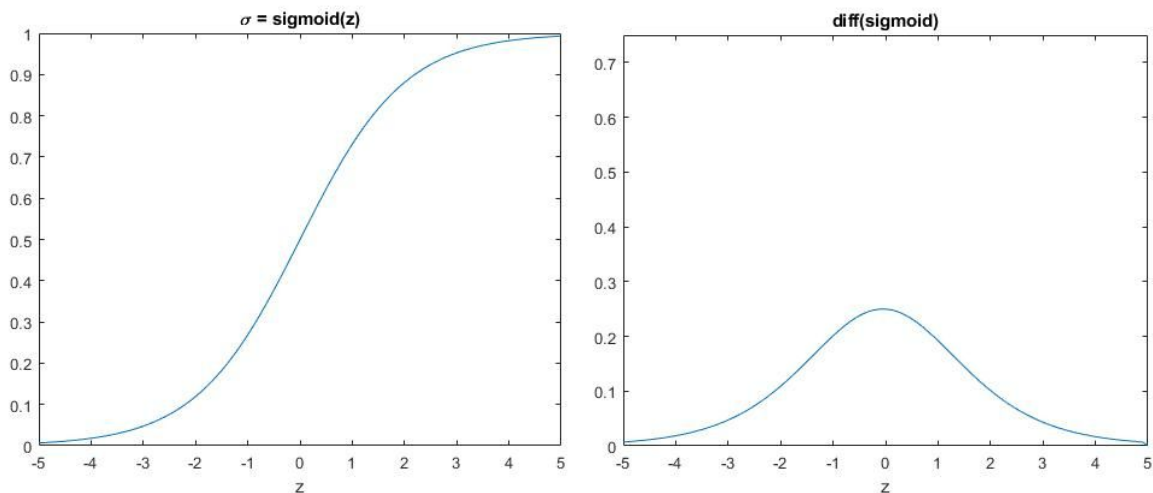
En caso de que el tipo de datos de entrada tenga grandes diferencias entre sí puede ocurrir que la red deshaga los cambios en los pesos al tratar de minimizar sus errores por separado. Por lo tanto, se recomienda calcular la media del error sobre un conjunto o lote de datos que preferiblemente contenga un caso de cada tipo. En numerosas ocasiones no es posible separar en diferentes clases los datos de entrada, por lo que suele recomendarse su reordenación de manera aleatoria antes de extraer el mayor número posible para el lote.

3.1.3 Funciones de activación

Como vimos anteriormente las funciones de activación juegan un papel importante en el cálculo del gradiente. Es común encontrar en una arquitectura multicapa dos tipos de funciones de activación, el que utiliza las capas internas para aprender y el que utiliza la capa de salida para producir el resultado del problema tratado.

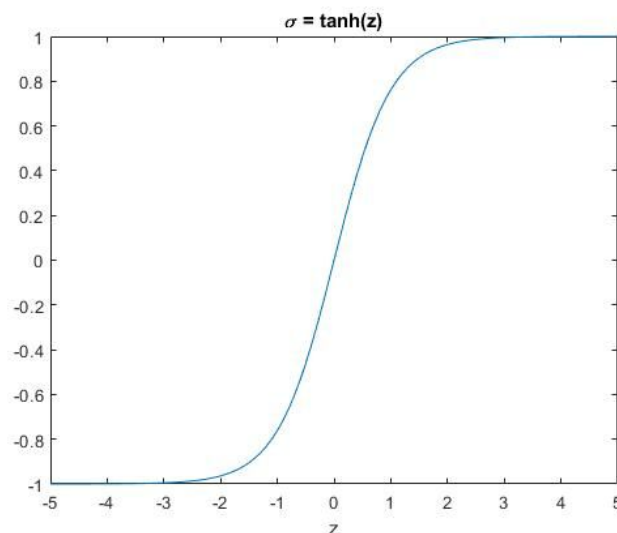
A continuación nombramos tres de las funciones más usadas:

Sigmoid. La función sigmoideal o logística se basa en la activación de una neurona biológica que devuelve valores comprendidos entre 0 y 1. Observando su derivada, comprobamos que agrava el problema de desvanecimiento del gradiente en los extremos, donde tiende a cero.



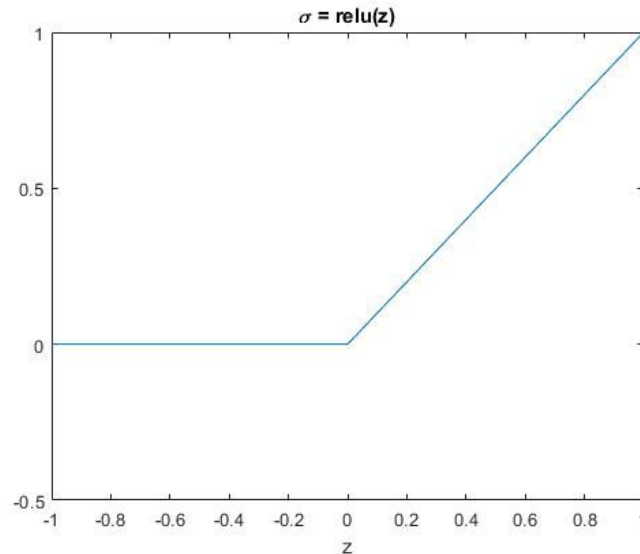
Su valor a la salida de la red se interpreta como la probabilidad de pertenecer a una clase. Cuando se tienen múltiples salidas no tienen por qué sumar uno entre sí (80% gato y 95% animal).

Tanh. Parecida a la sigmoideal pero entre -1 y 1. Tiene la ventaja de que al estar centrada en 0 tiene gradientes de mayor tamaño pero termina sufriendo los mismos problemas en sus extremos.



Softmax. Generalización de la función sigmoid para resolver problemas de clasificación multiclase. Produce una probabilidad entre todas las salidas igual a 1 (80% perro, 20% gato).

RELU. La Unidad Rectificadora Lineal es una función cuyo uso se vuelve casi obligatorio en redes multicapas dejando el uso de otras funciones únicamente para la salida de la red.



Como se observa, esta función es parecida a la identidad para valores positivos de entrada, pero añade no-linealidad al devolver cero para valores negativos de entrada. Su derivada sólo puede tomar los valores constantes 1 o 0 y por tanto esta función no sufre de desvanecimiento ni explosión de gradiente.

Las activaciones a 0 mejoran la convergencia de la red debido a que fuerza a un conjunto más pequeño de neuronas a aprender de manera óptima, ya que en ellas recae la aportación del gradiente.

El reducido número de activaciones producido por las salidas a 0 introduce velocidad y un menor uso de memoria en la red. También mejoran la convergencia forzando a las neuronas activadas a aprender de manera óptima, ya que en ellas recae la aportación del gradiente. Por tanto RELU simplifica el modelo y lo vuelve más ligero en comparación con sigmoid o tanh, donde la cantidad de activaciones es más densa y el cálculo del gradiente se vuelve más complejo.

Existen otras funciones parecidas como LRELU y PRELU que tratan de evitar las activaciones a 0 en caso de que se vuelva un problema para el aprendizaje de la red (RELU moribundo), pero no suelen resultar en mejoras significativas de los resultados[3].

Otra función que está obteniendo éxito es la unidad exponencial lineal o ELU que ha mostrado en algunos problemas ser mejor que la unidad rectificadora[4]

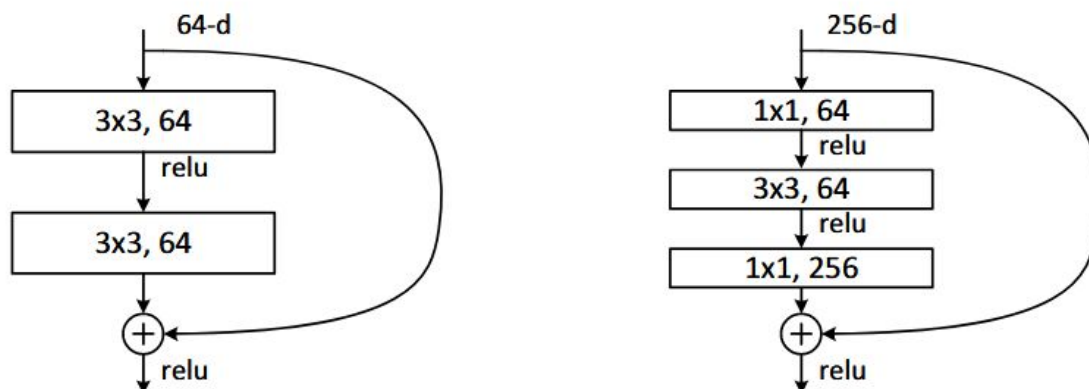
3.1.4 Conexiones Residuales

Las conexiones residuales son una solución a los problemas de desvanecimiento del gradiente. Antes de su uso, las redes que eran muy profundas daban mayor error que su contrapartida menos profunda. Esto se debe a la desaparición de la información a través de sus capas. Las conexiones residuales propagan la entrada de una capa hacia su salida permitiendo mantener la fuerza de la señal a través de la red. Esto conduce a una relación directa entre el gradiente de una capa y el error de salida.

Al usar la arquitectura clásica, suponemos que una capa solo necesita conocer la información producida por la anterior, con el problema de que a mayor profundidad mayor abstracción de la información. Las conexiones residuales permiten el acceso a la información producida por todas las capas anteriores, inclusive la entrada.

Este tipo de conexiones transforman la profundidad en multiplicidad.

Otros beneficios de estas conexiones residuales son la reducción de uso de memoria y velocidad a la hora de entrenar por lo que permiten un mayor número de capas como se explica en el papel *Deep Residual Learning for Image Recognition*[\[5\]](#) donde proponen diferentes combinaciones de bloques.



Bloques residuales. *Deep Residual Learning for Image Recognition*[\[5\]](#)

Además, en este papel se proponen varias implementaciones según el número de capas que se quieran usar, conocidas como ResNet.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Arquitecturas residuales. *Deep Residual Learning for Image Recognition*[5]

3.1.5 Evaluación de la red

Terminada la etapa de entrenamiento, es el momento de comprobar cómo se desenvuelve la red con información completamente nueva. Si la red obtiene resultados parecidos al entrenamiento, significa que ha sido capaz de aprender relaciones globales o generales para llegar a ese resultado. Si los resultados son peores hasta el punto de que no parece haber aprendido nada, significa que la red se ha sobre-especializado. Por eso, mientras la red se encuentra entrenando suele usarse un conjunto apartado del de entrenamiento para validar el desempeño de la red. A este conjunto se le conoce como conjunto de validación y no se usa para entrenar los parámetros de la red sino para una comprobación de su estado. Si la precisión/calidad de los resultados aumenta durante el entrenamiento pero el del conjunto de validación disminuye, sabremos que ha ocurrido la sobre-especialización.

En caso de que la red no consiga buenos resultados con el conjunto de entrenamiento es posible que necesite más complejidad como aumentar el número de neuronas y la profundidad de la red. Si, por el contrario, ocurre sobre-aprendizaje, reducir el número de parámetros aprendibles o aumentar la cantidad de datos y su variedad suele resultar en su solución. Cuando no se dispone de más datos para entrenar, es posible aumentar los que ya tenemos aplicando diferentes tipos de transformaciones a la entrada. Este tipo de operaciones, como añadir ruido o variar las intensidades, son dependientes de los datos y del problema a solucionar.

A pesar de las precauciones, tras múltiples modificaciones de la red puede ocurrir que se obtengan buenos resultados durante el entrenamiento y la validación, pero no con datos nuevos. Esto se debe a que hemos ajustado nuestra red basándonos en la calidad obtenida durante la validación y, por tanto, el set de validación deja de servirnos para comprobar el desempeño de la red con nueva información. La solución vuelve a consistir en tener otro

conjunto (test) de pares de entrada y salida que no debe ser usado más que para comprobar el desempeño de la red antes de su despliegue. No se debe reutilizar en casos en los que la red deba ser modificada nuevamente ya que dejaríamos de ser objetivos.

3.1.6 Elementos regularizadores

La introducción de regularización en la red aumenta la capacidad de generalización lo que produce una red más robusta frente a datos nuevos de entrada.

Las operaciones más comunes son:

Dropout[1]. Esta operación añade una probabilidad de apagar valores individuales de su entrada volviéndolas a 0. Por tanto, mejora la velocidad de ejecución y la convergencia al entrenar en cada iteración un menor número de neuronas como hace RELU. Es similar a una operación que añade ruido dentro de la red.

Normalización por lotes[2]. Esta operación permite normalizar las activaciones entre capas de la red asegurándose de que no produzcan valores muy altos o muy bajos. Esta operación se aplica sobre el lote de datos de entrada También añade un efecto regularizador al igual que hace dropout.

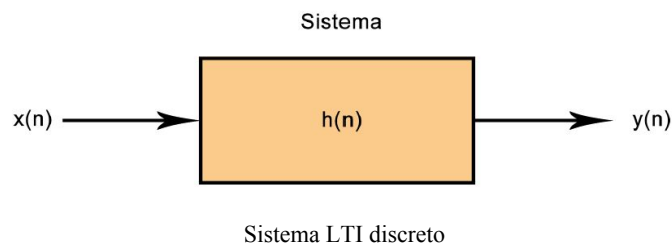
Se trata de una operación que mueve y escala los valores de entrada utilizando parámetros aprendibles por la red y reduciendo los efectos de las inicializaciones del resto de parámetros de la red. Permitiéndonos por ejemplo utilizar valores del ratio de aprendizaje mayores.

Cuanto mayor sea el número y variedad de datos por lote mejor será el aprendizaje de la normalización. No obstante, existe una modificación conocida como re-normalización por lotes[6] que produce mejores resultado cuando el número de datos es pequeño.

3.2 Redes neuronales convolutivas

3.2.1 La convolución

La convolución es una operación matemática popular dentro de la rama de sistemas y teoría de la señal. Se utiliza para describir el funcionamiento de sistemas lineales invariantes en el tiempo (LTI).



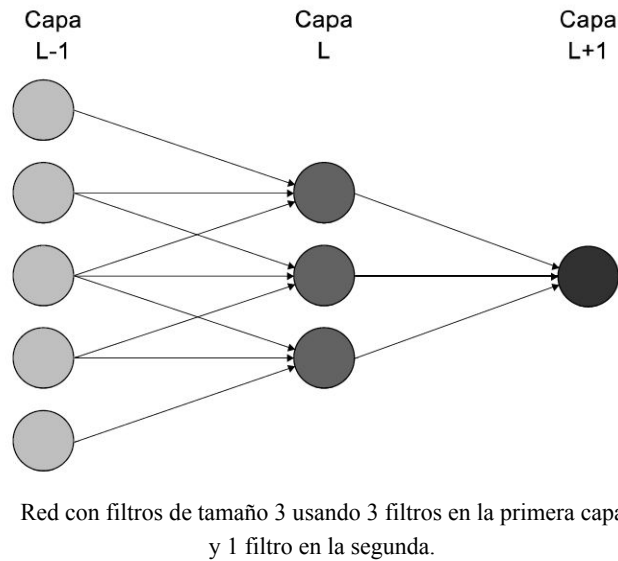
Su notación matemática se define como $f(x) * h(x)$ pudiendo ser x continuo o discreto. Para sistemas discretos se define como la suma del producto de dos funciones invirtiendo y desplazando una de ellas.

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k) * h(k - n)$$

La relación entre convolución y redes neuronales viene dada por el funcionamiento de la corteza visual del ser humano. Se encuentra compuesta de células sensibles a una parte de la entrada y “apiladas” para cubrir el total del campo de visión, permitiendo una fuerte correlación espacial que suele estar presente en las imágenes o audio.

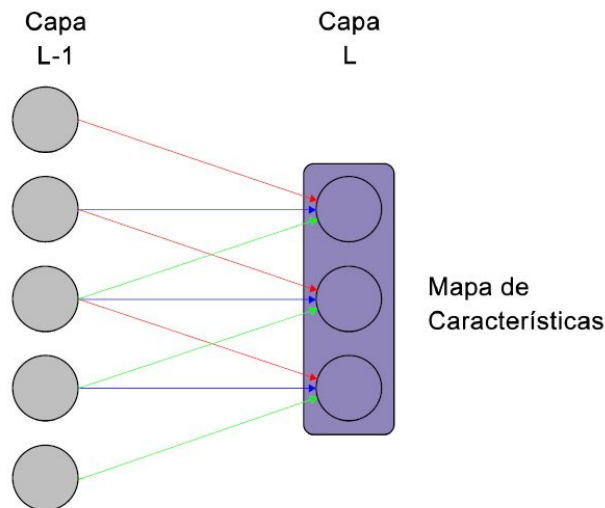
Queremos entonces que la respuesta al impulso $h(n)$ sean los pesos aprendidos por una capa de nuestra red. La convolución por tanto aplica $h(n)$ a cada instante de entrada permitiendo una salida independiente de la posición de dicho valor.

Si aplicamos estas ideas a la arquitectura básica de una red obtenemos que las neuronas de la siguiente capa reciben información de un conjunto más pequeño de entrada. Al conjunto de pesos encargadas de esas subregiones lo denominamos filtros y a la cantidad de pesos por filtro (tamaño del filtro) se le denomina campo receptivo.



Cada filtro determina una característica diferente al resto y por tanto es dependiente de su posición con respecto a su entrada. La solución consiste en desplazar cada filtro por el campo total de visión produciendo múltiples salidas.

Al conjunto de neuronas que comparten un filtro se le denomina mapa de características y permite a la red aprender un patrón sin importar su localización en la entrada. Para que cada capa reconozca más de un patrón se utilizan varios mapas de características o canales.

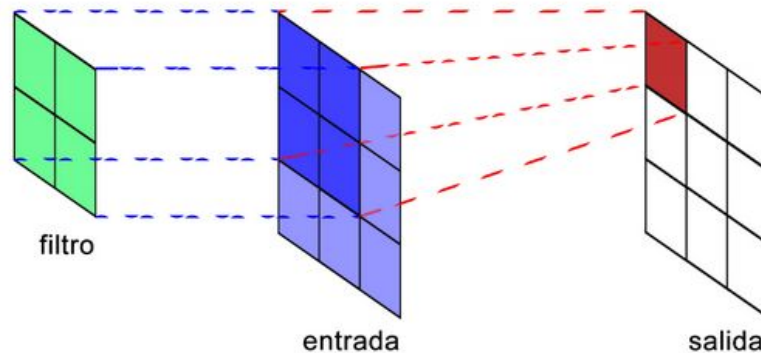


Capa convolutiva con pesos compartidos (coloreados) entre los filtros.

Como podemos comprobar, esta operación de mover una función o filtro y multiplicarlo en cada paso por la entrada es igual que la convolución. Aunque en ningún momento se produce la inversión del filtro, la red se encarga de aprenderlo. Por tanto el mapa de características se escribe como:

$$a(n) = \sigma(w(n) * x(n) + b)$$

Es importante destacar que la convolución aplica el filtro independientemente del tamaño de la entrada, afectando únicamente al tamaño de su salida. Otra propiedad importante es que la convolución es fácilmente adaptable a entradas con N-dimensiones, permitiendo mantener las relaciones espaciales como su uso con imágenes.



Convolución 2D

Mientras que el número de capas o profundidad de una red es necesario según la complejidad del problema, en las redes convolutivas el número de mapas de características se vuelve una parte igual de importante.

A pesar de todos los beneficios destacados, un problema que traen las capas convolutivas es la cantidad de parámetros que utilizan y el consecuente incremento del tiempo de computación.

3.2.2 Campo de visión

Aunque el mapa de características cubre toda la entrada, cada filtro tienen acceso a un sub-espacio o campo de visión igual a su tamaño. Si existe un patrón o característica en la imagen que sea de mayor tamaño que el filtro no podrá ser aprendido por este. Aumentar el tamaño de los filtros repercute en un aumento en el tamaño de la red y, por tanto, muchas redes convolutivas utilizan otras técnicas para aumentar el campo de visión. A continuación expongo los tres tipos que utilizaremos.

Max-pooling

Esta operación utiliza una ventana, que se desplaza por la entrada, y produce una salida igual al valor máximo encontrado. Una ventana de tamaño 2 y moviéndose de dos en dos reduce la resolución de su entrada a la mitad. Esto resulta en un aumento del campo de visión de los filtros en las capas siguientes, además de una reducción en el número de parámetros.

Aunque es una operación de compresión con pérdidas, al quedarse con el máximo no suele repercutir mucho a la hora de aprender. Suele usarse principalmente porque reduce la cantidad de parámetros que requiere la red e introduce invariancia espacial al quedarse siempre con el valor máximo.

Algunas arquitecturas utilizan otras operaciones de pooling según el tipo de problema y datos.

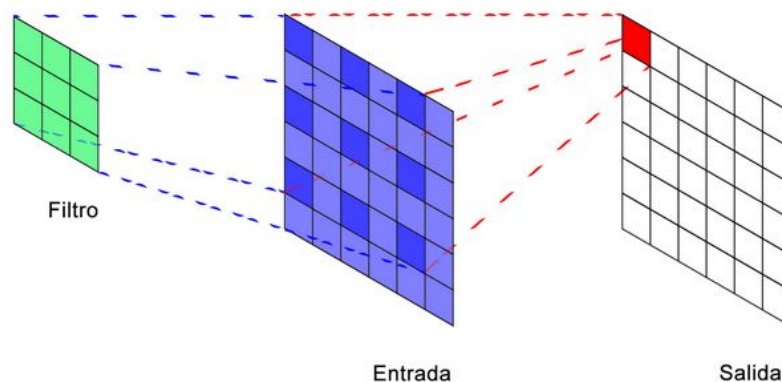
Convolución con paso

Si aplicamos un paso diferente de uno sobre el movimiento del filtro obtenemos una reducción de la salida al igual que con las operaciones de pooling. Su principal característica es que esta operación es aprendible por la red, con el consecuente aumento de parámetros.

En el papel de Striving For Simplicity[7] demuestran como se puede sustituir la operación de pooling sin ocasionar ninguna pérdida de precisión en problemas típicos de la segmentación.

Convolución dilatada

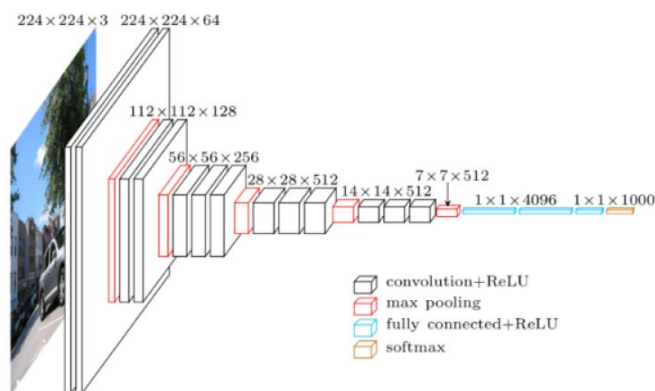
La convolución dilatada[8] permite mantener el tamaño del filtro en la convolución a la vez que aumenta su campo de visión. Esta operación introduce espacios entre los elementos del filtro antes de proceder a su operación con la entrada. Al simplemente dilatar/ensanchar el filtro, la resolución se mantiene y el campo de visión aumenta. Destacar que no disminuye los parámetros de las subsecuentes capas y puede volverse una operación costosa de usar.



Convolución dilatada. Filtro expandido en la entrada.

3.2.3 Segmentación y Convolución

Una de las arquitecturas convolutivas más conocida es VGG[9] y ha sido utilizada como base para todo tipo de arquitecturas más complejas.



Arquitectura VGG-16[1].

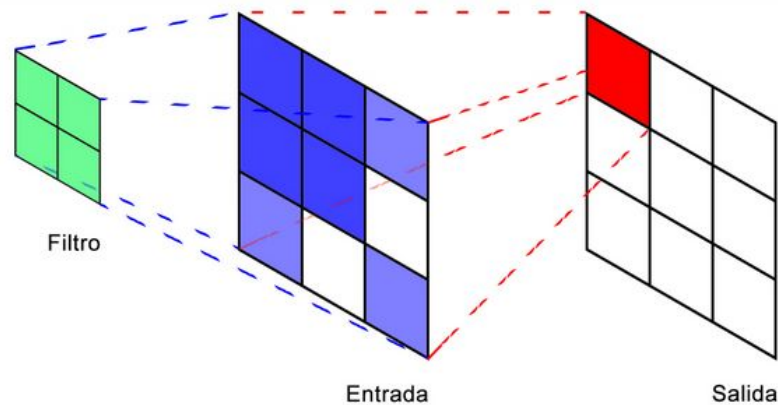
Al reemplazar las capas “completamente conectadas” de salida por la convolución, resulta en una arquitectura capaz de clasificar los píxeles en una imagen. Esto lo demuestran en el papel *Fully Convolutional Networks for Semantic Segmentation*[10] utilizando la arquitectura de VGG-16. A la redes que solo usan capas convolutivas se las conoce por FCN (Fully Convolutional Network) y una de las ventajas es que es independiente de la resolución de entrada.

Aunque esta arquitectura es capaz de aprender la segmentación, su salida es de menor tamaño que la entrada debido a las operaciones de max-pooling. Aunque es común utilizar interpolación para aumentar la resolución, otras arquitecturas implementan técnicas más complejas.

Arquitectura codificador-decodificador. Estas arquitecturas tienen una etapa de reducción de la resolución a medida que aumenta la profundidad, con el incremento del campo de visión consecuente, y otra etapa a la salida que se encarga de recuperar gradualmente el tamaño original.

Arquitecturas dilatadas. Estas arquitecturas utilizan convolución dilatada para aumentar el campo de visión y mantener el tamaño de la entrada. No obstante, se suelen reducir las dimensiones inicialmente para reducir los cantidad de parámetros que puede llegar a tener la red.

Para el aumento de resolución además de la interpolación multivariable se puede utilizar la propia operación de convolución. Se le da el nombre de convolución transpuesta porque el papel de la entrada y salida se invierten. Esto quiere decir que produce varios valores de salida por cada valor de entrada.



Convolución transpuesta. Entrada dilatada.

3.3 Redes Neuronales Recurrentes

Se trata de un tipo especial de arquitectura con neuronas que utilizan la salida anterior como parte de la entrada siguiente. Por tanto, la red mantiene un estado que depende de todas las salidas producidas anteriormente y funciona como una especie de memoria.

El uso de estas redes suele encontrarse en problemas donde existe una secuencia de datos y cuya salida en un instante dependa del orden de esta secuencia. Ejemplos de uso suelen ser problemas con texto o sonido. El estado es necesario reiniciarlo entre secuencias para que no crezca indefinidamente, aunque existen unidades recurrentes que añaden “puertas” que aprenden a hacerlo.

Existen dos tipos de celdas o unidades recurrentes cuyo uso se encuentra bastante generalizado, *long short term memory* (LSTM) y *gated recurrent unit* (GRU). Una evaluación de los dos tipos de celdas la podemos encontrar dentro del papel *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*[\[11\]](#).

4. Pruebas y Resultados

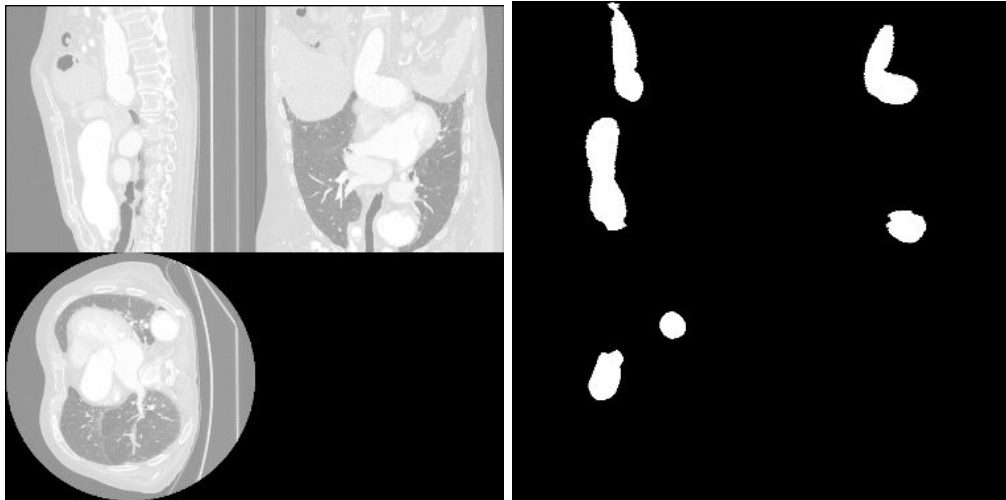
En este apartado veremos la configuración inicial, la implementación de los modelos, la recogida de datos y las conclusiones finales a las que llegamos. Debido a la cantidad de pruebas, hacemos una división entre implementación 2D y 3D con sus propias observaciones.

4.1 Preparación

La siguiente información describe la configuración base para el entrenamiento y ejecución de las arquitecturas implementadas.

4.1.1 Datos de entrada.

Para esta investigación se disponen de volúmenes de imágenes TC de 12 pacientes y sus respectivas máscaras de la aorta segmentadas por un experto médico. A cada paciente se le ha aplicado un contraste para resaltar la sangre durante el escáner.



(Izquierda) Vistas del volumen de un paciente. (Derecha) Vistas de la máscara 3D segmentada por el experto.

Las imágenes 3D tienen un canal de información (blanco/negro) y su rango de intensidades es variable. Las imágenes 2D contenidas en cada volumen son de 512x512 píxeles y su cantidad es variable. En total tenemos unas 7000 imágenes para el entrenamiento 2D.

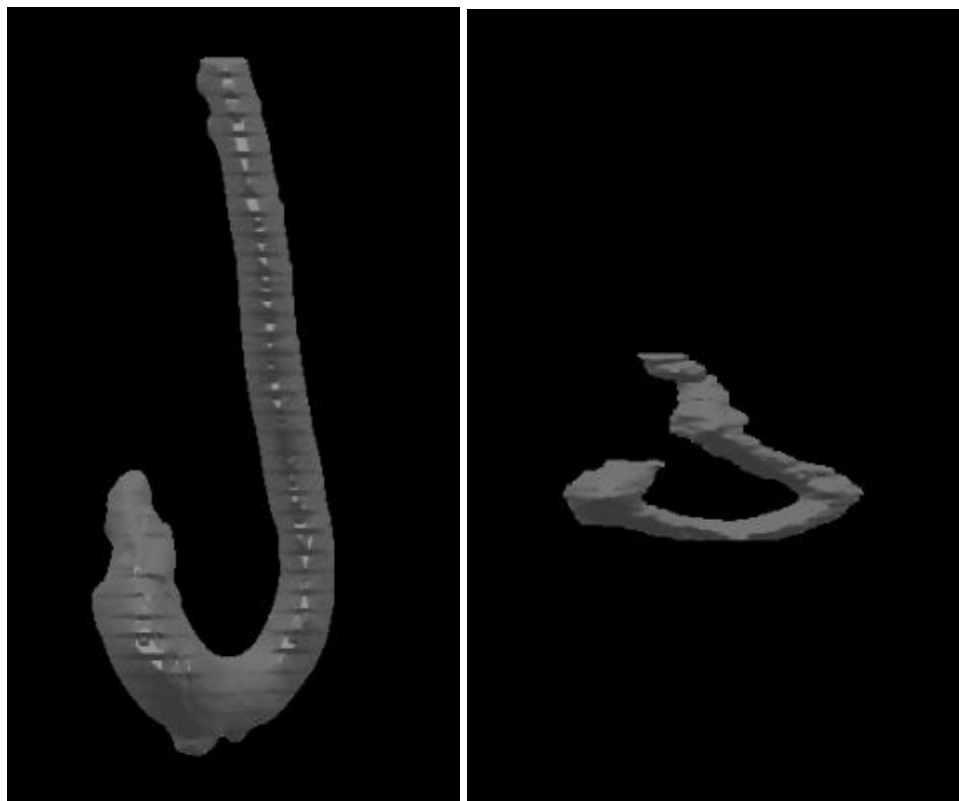
La normalización de las imágenes la hacemos al sustraer de cada píxel el valor medio de intensidad y limitar los valores de entrada entre -1 y 1. De esta forma centramos los datos y conseguimos facilitar el aprendizaje de la red. Siendo la entrada *data*, la desviación estándar *std* y el cálculo del valor medio *mean* el proceso de normalización utilizado es:

$$\frac{data - mean(data)}{std(data)}$$

Debido a que la aorta es un elemento cuya colocación es céntrica en la imagen y su tamaño aproximado es de $\frac{1}{8}$ del total, se procede a recortar sus bordes. Al quedarnos con un tamaño de 320x320 reducimos el consumo de memoria. Una mayor reducción puede producir que la aorta no esté completa, perdiéndose así información importante.

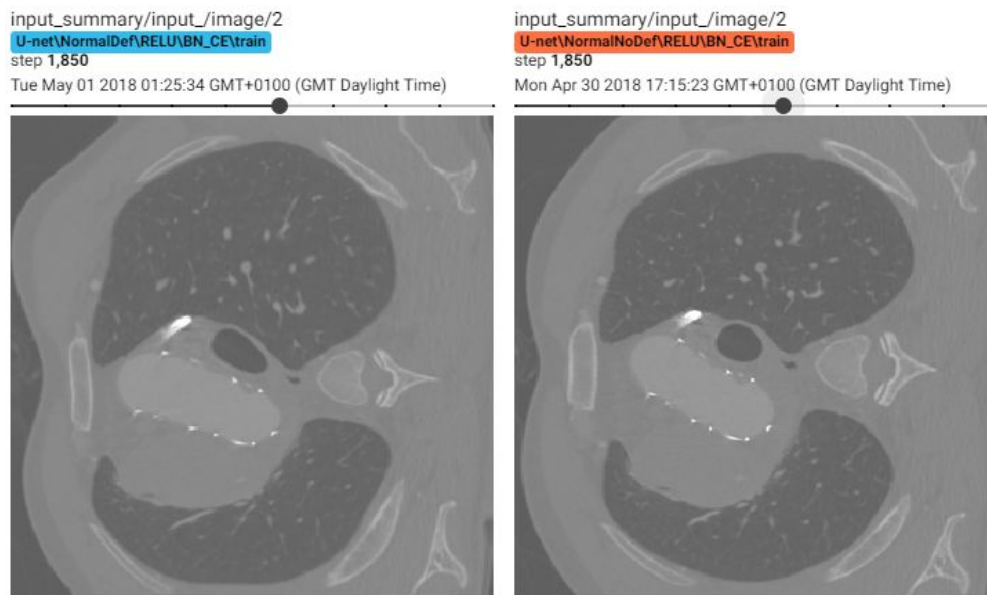
Dentro de un volumen no existe mucha variación entre imágenes consecutivas lo que provoca que el número de datos efectivos para entrenar nuestra red se reduzca. La división de los conjuntos se realizará por paciente para tratar de mantener la objetividad durante la validación y el testeo.

Repartimos los volúmenes de los pacientes quedándonos con 6/3/3 para el entrenamiento, validación y test respectivamente. La selección de cada volumen en cada conjunto la hacemos por su dificultad/forma incluyendo, como mínimo, uno de cada tipo.



(Izquierda) Caso fácil. (Derecha) Caso difícil.

El entrenamiento de la red se hará por lotes a los que aplicaremos varias transformaciones adaptables tanto a volúmenes de datos como a imágenes individuales. Por cada imagen pasada a la red, existe una probabilidad de aplicar la transformación elástica además de la inversión y rotación, permitiéndonos una mayor variedad de datos. La transformación elástica es una operación que deforma la imagen al aplicar una fuerza imaginaria sobre esta.



(Izquierda) Transformación elástica. (Derecha) Imagen original

4.1.2 Salida y calidad de la red.

Para segmentar la aorta utilizaremos la clasificación binaria de cada píxel (1 para la aorta y 0 para el fondo) y para ello emplearemos la función sigmoid a la salida de cada arquitectura implementada. Después se aplicará un umbral a la salida de la red ($>0,5$) para separar casos positivos (aorta) de los negativos (fondo) y denominaremos como mIOU al valor medio de calidad.

Como medida de la calidad de la red tenemos el índice de Jaccard conocido también como la intersección sobre la unión (IOU). Se aplica sobre la salida de la red y la máscara deseada y su resultado se interpreta como el porcentaje o ratio de casos positivos acertados entre los casos positivos totales. Esta medida es independiente del número de casos negativos acertados haciendo que la clasificación correcta de píxeles sin aorta no aumente la calidad de la red.

A no ser que se especifique lo contrario, los resultados de calidad mostrados son obtenidos a partir del conjunto de validación.

4.1.3 Tipo de error y método de optimización

La función de error base que utilizaremos es la entropía cruzada binaria (BNCE) que es típica en problemas de clasificación binaria. Para la implementación y muestra de resultados se utilizará esta función siempre que no se especifique su cambio.

Además, probaremos otras dos funciones que pueden ser relevantes para nuestro problema:

Entropía cruzada binaria con peso (WBCE). Utilizamos esta función debido a que, en comparación con el tamaño de la máscara de segmentación, la aorta es un elemento pequeño que provoca una desproporción entre casos positivos y casos negativos en la imagen. Esta desproporción ocasiona que la red aprenda a devolver solo ceros para minimizar su error inicialmente.

Para resolver este problema la función WBCE añade un parámetro que amplifica el error de los casos positivos acertados en relación con los negativos. Por tanto, valores >1 decrece el número de falsos negativos (Píxeles con salidas a 0 que deberían ser 1).

Intersección sobre unión (IOU). Como sabemos, esta operación no es diferenciable y por tanto no puede ser usada directamente por la red. En el papel *Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation*[\[12\]](#) proponen una aproximación permitiéndonos utilizar esta medida.

Como operación para minimizar el error usaremos el método ADAM[\[13\]](#), es un tipo de descenso de gradiente estocástico que, como muestran los resultados de su papel, produce mejores resultados que sus similares. Los métodos estocásticos calculan el gradiente del lote a partir de un conjunto más reducido. En especial, el método ADAM modifica los parámetros de la red aplicando una técnica de “momento” basada en la actualización anterior. En vez de tener un solo ratio de aprendizaje, este método calcula diferentes ratios adaptables a cada parámetro entrenable.

4.1.4 Dropout y normalización por lotes

Como ya se explicó en apartados anteriores, el uso de estas dos técnicas tienen efectos que pueden ser beneficiosos para el aumento de la generalización de la red. Su uso se hará antes de la operación de activación de las neuronas a excepción de la capa de salida de la red.

Además de dropout usaremos su variante *spatial dropout*. Esta operación desactiva mapas de características enteros en vez de píxeles individuales, como bien se explica en el apartado 3.2 de *Efficient Object Localization Using Convolutional Networks*[\[14\]](#)

También aplicaremos re-normalización por lotes por producir mejores resultados cuando se utilizan pequeños valores de lote.

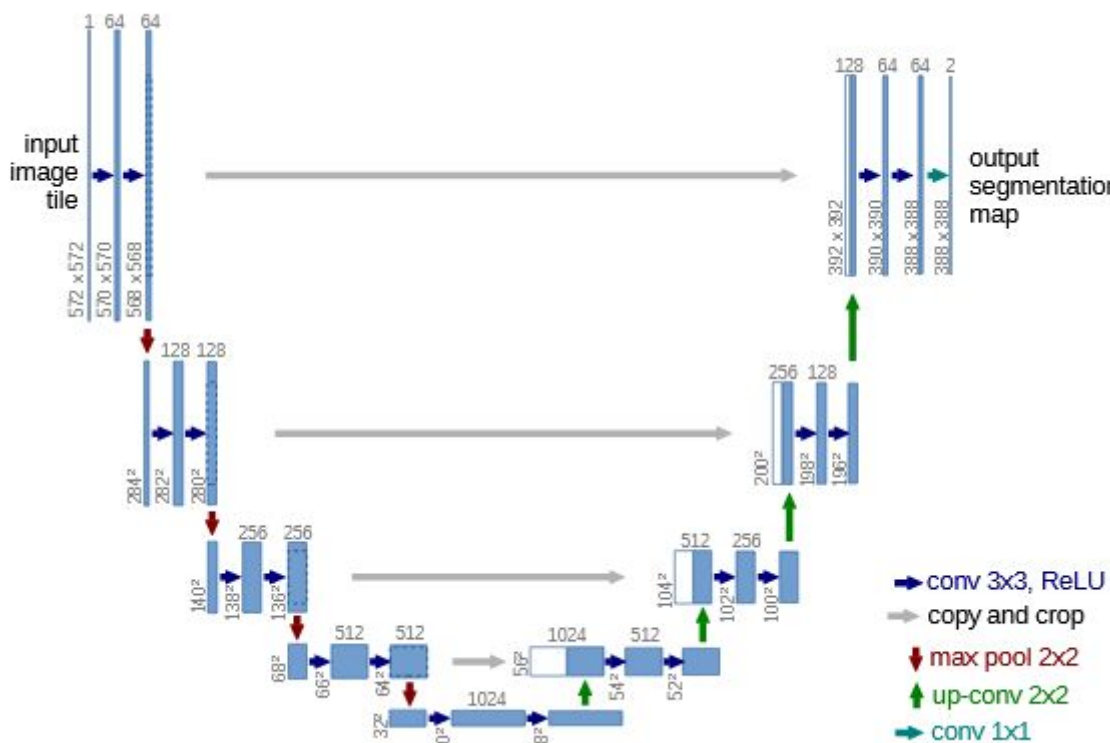
4.2 Segmentación 2D

Para la segmentación de la Aorta en dos dimensiones se decidió optar por dos arquitecturas basándonos en la forma que aumentan el campo de visión efectivo de sus filtros y su profundidad.

4.2.1 U-net[15]

Esta arquitectura hace su aparición en 2015 produciendo muy buenos resultados en diferentes problemas de segmentación con imágenes médicas. Es del tipo codificador-decodificador utilizando en total 23 capas convolutivas. Su uso no está restringido solo a la medicina porque es capaz de resolver todo tipo de problemas de segmentación.

La etapa codificadora está formada por bloques con max-pooling que reducen la resolución a la mitad y duplican los canales. La etapa decodificadora utiliza bloques con convolución transpuesta que invierten el proceso. Cuanta más profundidad, mayor es la reducción, siendo 1/16 de la entrada en la arquitectura propuesta. Lo especial de esta arquitectura es que concatena la entrada, antes de la reducción, con la salida después de la operación de aumento, consiguiendo mantener la información espacial, que de otra manera se habría perdido.



Arquitectura de U-net: Convolutional Networks for Biomedical Image Segmentation[15]

Ratio de aprendizaje y número de imágenes por lote

Ratio de aprendizaje: 1×10^{-4} .

Número de imágenes por lote: 10.

Función de error

Tipo de error	Calidad Media Validación (mIOU)
BNCE	0.819
WBNCE Peso = 4	0.747
IOU	0.777

Función de activación

Tipo de activación	Calidad Media Validación (mIOU)
RELU	0.819
PRELU	0.8
ELU	0.82

Elementos regularizadores

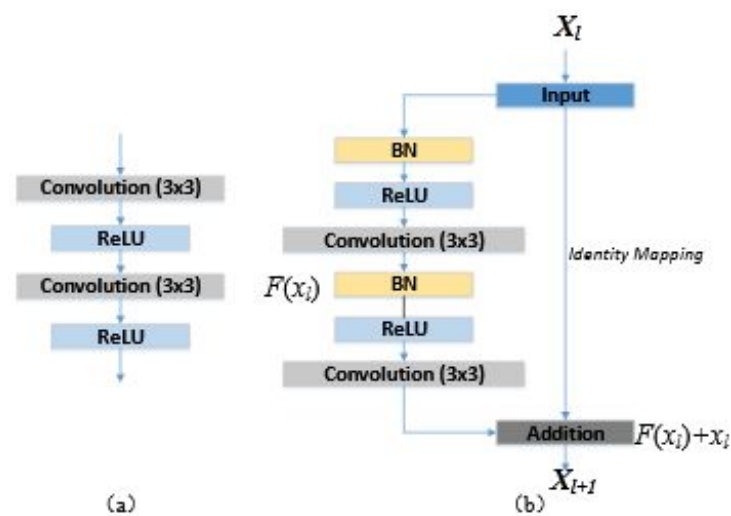
Tipo de capa	Calidad Media Validación (mIOU)
Normalización por lotes	0.762
Re-Normalización por lotes	0.776
Dropout p=0.1	0.773
Dropout p=0.3	0.749
Spatial Dropout p=0.2	0.772

Max-Pooling y convolución con paso

Probamos a sustituir la operación de agrupamiento por una convolución de filtros 2x2 y paso 2. Los resultados son similares siendo max-pooling ligeramente mejor (+0.09 mIoU). Podemos decir que la invarianza espacial y el menor número de parámetros entrenables hace que sea la operación elegida para nuestro modelo.

Conexiones Residuales

Decidimos implementar el uso de conexiones residuales a la arquitectura de U-net basándonos en el papel de Road Extraction by Deep Residual U-Net[16]



a) Bloque normal b) Bloque residual.

Road Extraction by Deep Residual U-Net[16]

Tampoco mejora nuestra arquitectura base, pero consume $\frac{1}{4}$ de la memoria original lo que permite usar imágenes de mayor tamaño y aumentar la profundidad de la red en problemas que lo requieran.

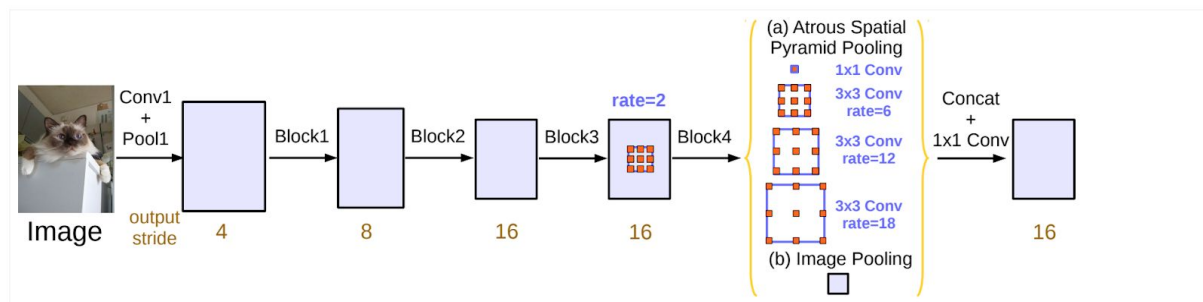
Conexiones Residuales	Calidad Media Validación (mIOU)
No	0.801
Si	0.819

4.2.2 DeepLabv3 [17]

La arquitectura de DeepLab ha sido actualizada en sucesivas versiones añadiendo nuevas técnicas para la segmentación. Su versión dos[18] añade convoluciones dilatadas a la arquitectura residual ResNet101[5].

También introducen el módulo Spatial Pyramid Pooling [19] cambiándolo por convoluciones dilatadas y dándole el nombre de Atrous Spatial Pyramid Pooling o ASPP. Este bloque utiliza convoluciones en paralelo para capturar características de diferentes resoluciones y fusionarlas en su salida.

La implementación se realiza sobre su versión tres, en donde añaden características a nivel de imagen dentro del módulo ASPP. Además DeepLab utiliza un campo aleatorio condicional o CRF, como post-procesamiento de la salida de la red, para suavizar los resultados. En esta investigación solo nos centraremos en la red neuronal encargada de la segmentación.



Arquitectura de Deeplab v3[17] sin el aumento de la resolución.

Ratio de aprendizaje y número de imágenes por lote

Ratio de aprendizaje: 1×10^{-4} .

Número de imágenes por lote: 10.

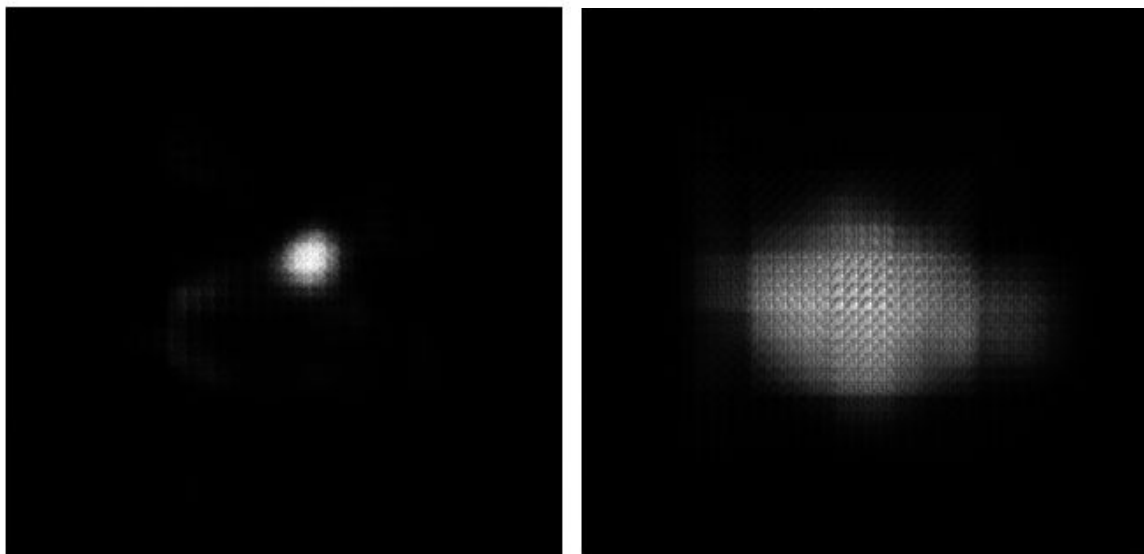
Aumento de Resolución

Para el aumento de la resolución probamos con varios tipos de bloques y combinaciones.

Inicialmente se utilizaron dos bloques con la composición *Convolución-Traspuesta* + *Convolución* antes de la capa de salida. Los filtros son de tamaño 2x2 y en el caso de la convolución traspuesta el paso es de 4 para el primer bloque y 4 para el segundo.

Bloques usados	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
2xConv-Traspuesta	0.806	0.667	0.715

Mirando las salidas de la red y de estos bloques nos encontramos con artefactos producidos por la convolución traspuesta.



(Izquierda) Salida de la red con artefactos

(Derecha) Salida de la primera convolución traspuesta

Por otro lado el uso del aumento de resolución bilineal, cuando es seguida de convoluciones normales, se considera un aumento aprendible y que no produce artefactos. Utilizando los dos mismos bloques de antes, sustituimos la convolución traspuesta por la interpolación bilineal con aumentos de 4 y 4 respectivamente. Aunque resulta en la desaparición de los artefactos, la red no aumenta mucho en precisión.

Bloques usados	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
2x Bilineal	0.774	0.741	0.739

Por último una combinación de los dos tipos en el orden, Bloque Conv-Traspuesta + Bloque Bilineal, produce los mejores resultados obtenidos con una reducción significativa de los artefactos.

Bloques usados	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
Conv-Traspuesta + Bilineal	0.794	0.775	0.751
Bilineal + Conv-Traspuesta	0.827	0.733	0.737

Creemos que el uso del bloque traspuesto antes que el bilineal obtiene mejor calidad cuando su entrada es más abstracta y se tienen múltiples canales, como ocurre tras varias capas dentro de una red. Esto se debe a que es capaz de aprender formas de aumentar la resolución más complejas que la interpolación.

En el papel de *Checkerboard artifact free sub-pixel convolution*[\[20\]](#) tratan diferentes formas de reducir los artefactos provocados por las capas convolutivas. Entre ellas proponen una inicialización de los filtros a un tamaño menor (ICNR) con el que conseguimos mitigar los efectos iniciales y mejorar los resultados de la red.

Bloques usados	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
Conv-Traspuesta (ICNR) + Bilineal	0.831	0.760	0.771

El resto de pruebas se harán con esta configuración.

Funciones de error

Tipo de error	Calidad Media Validación (mIOU)
BNCE	0.771
WBNCE Peso = 4	0.692
IOU	0.636

Función de activación

Tipo de activación	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
RELU	0.831	0.760	0.771
PRELU	0.695	0.617	0.671
ELU	0.463	0.752	0.644

Elementos regularizadores

Aplicando la regularización antes de RELU obtenemos:

Tipo de capa	Calidad Media Validación (mIOU)
Normalización por lotes	0.715
Re-Normalización por lotes	0.699
Dropout $p=0.2$	0.518
Spatial Dropout $p=0.2$	0.495

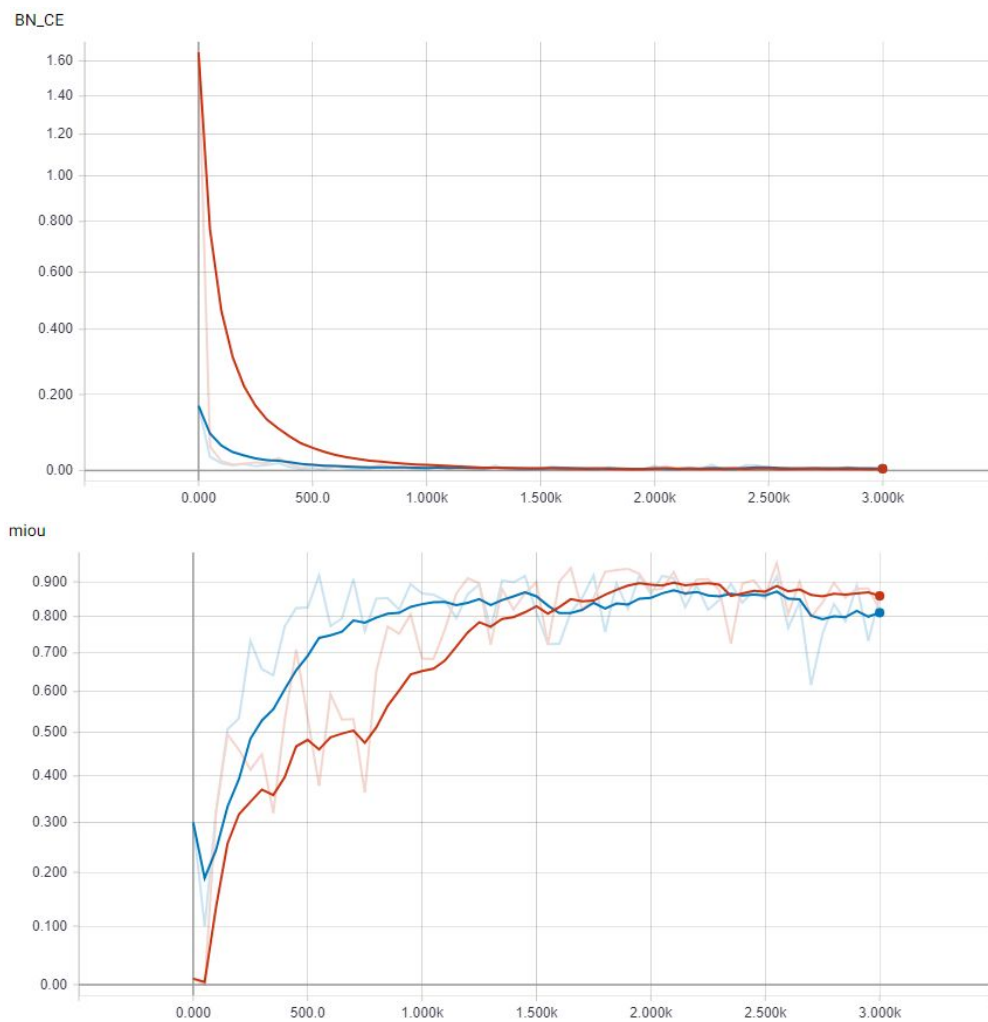
4.2.3 Observaciones 2D

Deformaciones

Antes de hacer ningún cambio comprobamos que el uso de transformaciones elásticas mejora la generalización de la red. A continuación mostramos su efecto en el error (BNCE) y la calidad de la red (mIOU) durante el entrenamiento de U-net.

----- Con transformación elástica

----- Sin Transformación elástica



Las deformaciones crean un empeoramiento en el aprendizaje inicial debido a la variedad de imágenes que introduce, pero tras unas cuantas iteraciones es capaz de aprender características que suavizan los resultados entre casos fáciles y difíciles. Destacar que el aumento medio en calidad no es significativo.

Transformación elástica	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
No	0.828	0.743	0.815
Si	0.835	0.815	0.819

Funciones de error

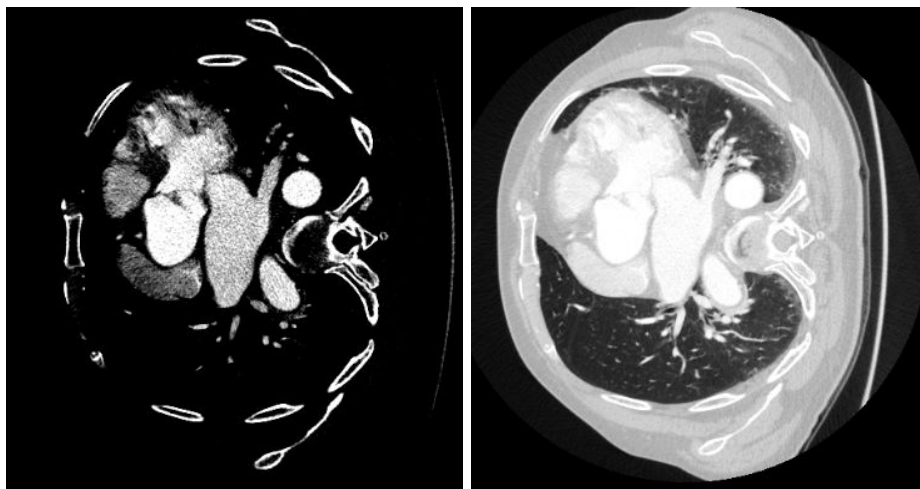
Entre las funciones de error probadas obtenemos los mejores resultados con la función de error BNCE. Para la función WBNCE, a medida que aumentamos el valor del peso aumenta la velocidad de convergencia pero disminuye su calidad. Por otro lado, aunque la intersección sobre la unión es una medida bastante fiable de la calidad de la red, tampoco resulta en una mejor función de error para minimizar.

Funciones de activación

El resultado y la velocidad de entrenamiento al sustituir RELU por ELU son muy similares en U-net, mientras que PRELU no mejora los resultados y ralentiza considerablemente el tiempo de entrenamiento. En cambio, DeepLab no parece funcionar muy bien con ninguna de estas dos funciones alternativas a RELU.

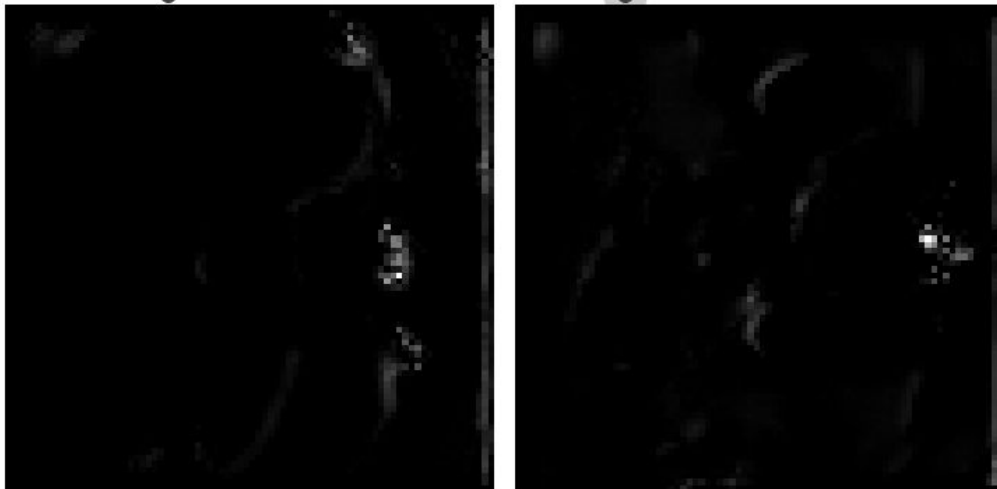
Datos de entrada

Ambas redes tardan bastante en identificar qué otros elementos de la imagen no son aorta. Incluso, una vez entrenadas, si se les presenta un caso difícil es posible que terminen segmentado lo que no es. Decidimos entonces aplicar un filtro sobre los datos de entrada para eliminar valores de intensidad fuera del rango de la sangre.



(Izquierda) Imagen con filtro de intensidad. (Derecha) Imagen sin filtro de intensidad

Los resultados producidos por esta nueva entrada no tienen una clara convergencia. Analizando las capas intermedias, con los datos originales, comprobamos como la red utiliza la información de la columna vertebral, entre otros elementos, para inferir donde se encuentra la aorta.



Aprendizaje de la columna vertebral en capas intermedias.

Elementos regularizadores

Al parecer, estas operaciones sólo consiguen empeorar los resultados de la red a pesar de que deberían mejorar su capacidad de generalización. Para estar seguros, probamos a reducir el número de imágenes del conjunto de entrenamiento a $\frac{1}{10}$ del total y dejamos de aplicar transformaciones sobre los datos de entrada.

U-net (Pocos datos)

Tipo de capa	Caso Fácil (mIOU)	Caso Medio (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
Normal	0.805	0.692	0.374	0.624
Normalización de lotes	0.795	0.672	0.610	0.692
Dropout $p=0.3$	0.674	0.802	0.668	0.714

Al reducir el número de imágenes conseguimos efectos positivos con las operaciones regularizadoras. Por una parte disminuye la calidad en los casos más sencillos pero se consigue un aumento considerable en los casos difíciles.

La reducción de calidad con grandes cantidades de datos se debe a que estas operaciones introducen una mayor variación dentro de la red cuando no se tienen los parámetros suficientes para su aprendizaje (Apartado 1.1 del papel de Groupout[21]).

U-net vs DeepLab

A pesar de que la arquitectura de Deeplab es más reciente y contiene técnicas novedosas, no consigue mejores resultados a la hora de segmentar la aorta. Lo que podemos resaltar sobre Deeplab es que consigue una reducción del 40% en los tiempos de entrenamiento y ejecución.

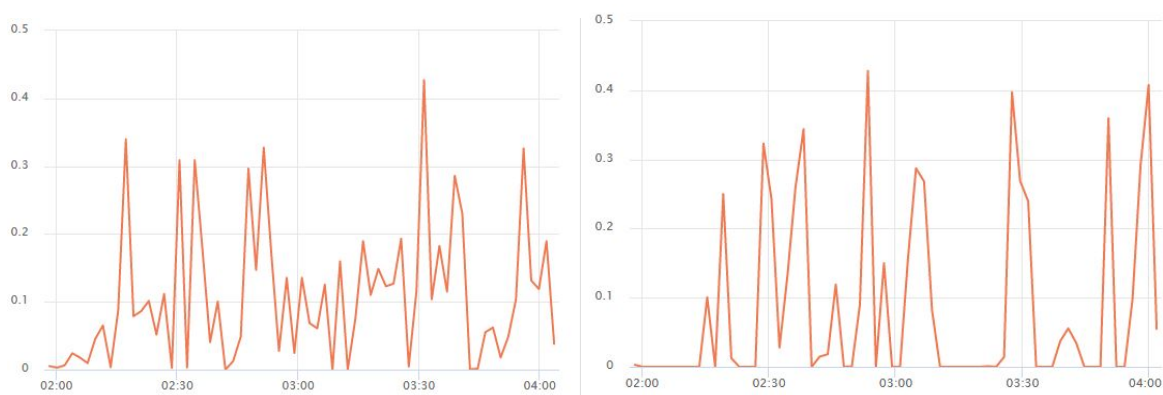
4.3 Segmentación 3D

Para la segmentación de volúmenes utilizaremos la arquitectura original de U-net modificada para trabajar con datos en tres dimensiones. Utilizamos esta arquitectura por haber obtenido un mejor rendimiento en comparación con DeepLab.

4.3.1 3D U-net y V-net

Se actualizan todas las operaciones de U-net para que trabajen con la dimensión extra (eje z). Debido al alto número de parámetros necesarios para la convolución 3D, el número de volúmenes por lote es uno y reducimos el número de canales en cada capa a la mitad. Para extraer los volúmenes, y tener suficiente variedad, utilizamos la mitad del volumen anterior y la mitad del siguiente en su formación.

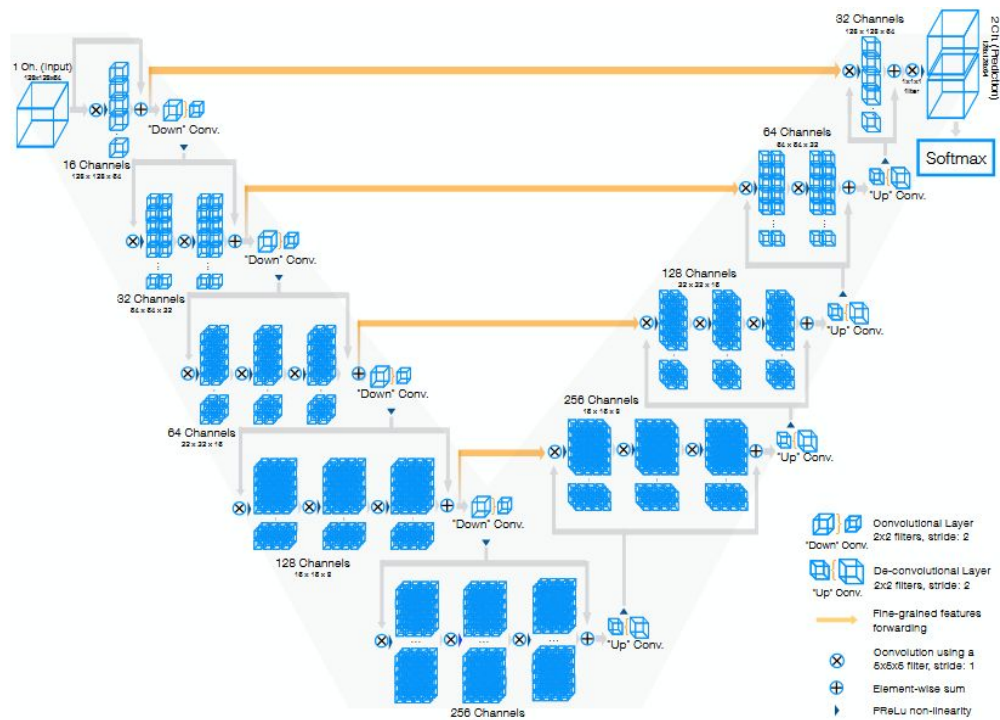
Durante las pruebas utilizamos diferentes tamaños de volumen, varias operaciones de reducción de la resolución y múltiples tamaños de filtros convolutivos. La red fue incapaz de converger hasta que dimos con una configuración estable.



(Izquierda) mIOU de entrenamiento. (Derecha) mIOU de validación

El modelo que finalmente nos funcionó utiliza un tamaño de volumen o eje z de 16 y con un ratio de aprendizaje de 1×10^{-5} . La operación de max-pooling reduce este eje a la mitad en el primer bloque codificador y mantiene su tamaño en el resto. Las operaciones de convolución utilizan un tamaño del filtro en el eje z de uno exceptuando el bloque de salida, el de entrada y el bloque intermedio que utilizan tamaño dos.

Estos cambios producen que la mayor parte de la red no aprenda relaciones sobre este eje pareciéndose a una arquitectura 2D. Por tanto decidimos aplicar la arquitectura residual de V-net[22], pero obtenemos los mismos resultados sin convergencia. Solo con la modificación propuesta llega a aprender.



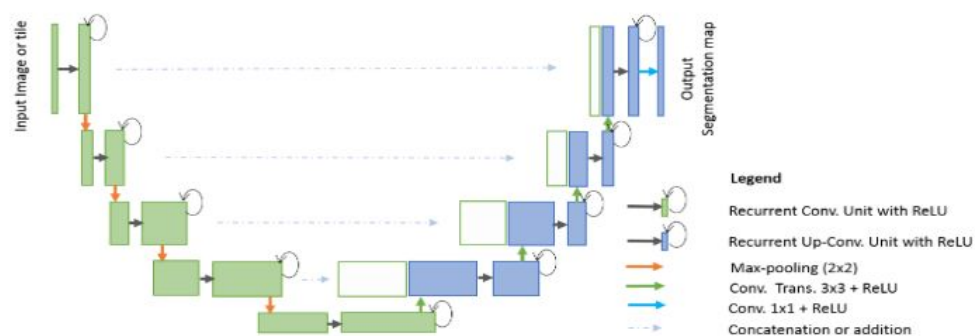
Arquitectura de V-net[22]

Arquitectura	Caso Fácil (mIOU)	Caso Medio (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
3D U-net (modificada)	0.796	0.694	0.433	0.641
V-net (modificada*)	0.602	0.481	0.540	0.541

*Aprovechamos la reducción de memoria de las conexiones residuales para duplicar el número de canales y dejarlos como en la arquitectura original U-net.

4.3.2 CRNNs

Dada la baja calidad que obtenemos con la arquitectura 3D decidimos probar con el uso de redes neuronales convolutivas recurrentes (CRNN). Al tratar el volumen de entrada como una secuencia de imágenes podemos utilizar la información espacial necesaria para la segmentación 3D sin un aumento tan grave de los parámetros. Aplicamos la arquitectura propuesta por R2U-Net[23] en el mes de febrero de 2018. En su papel proponen el uso de unidades convolucionales recurrentes para formar los bloques de U-net.



Arquitectura propuesta por R2U-Net[23]

Nosotros utilizaremos unidades convolucionales del tipo GRU[24] por su sencillez y, debido a que las redes recurrentes suelen tener problemas con secuencias muy largas, partiremos los volúmenes de cada paciente en secuencias de tamaño 16.

Arquitectura	Caso Fácil (mIOU)	Caso Medio (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
R2U-Net	0.014	0.376	0.503	0.298

También optamos por utilizar celdas recurrentes al final de la arquitectura U-net (antes de la convolución final). Esto nos permite utilizar la información de la secuencia a modo de corrector de la segmentación 2D. El número utilizado de celdas son dos, el tamaño de los filtros 2x2 y se mantienen los canales del último bloque.

Arquitectura	Caso Fácil (mIOU)	Caso Medio (mIOU)	Caso Difícil (mIOU)	Calidad Media Validación (mIOU)
U-net + parte recurrente	0.410	0.417	0.436	0.421

4.3.3 Observaciones 3D

Los modelos 3D basados en U-net parecen tener grandes dificultades para la segmentación. Al reducir la complejidad del aprendizaje conseguimos resultados más estables pero no más precisos.

Por otro lado hace, falta más experimentación para averiguar por qué las redes recurrentes tienen problemas para aprender. Por un lado, parece ser que el aprendizaje del estado repercute cuando ocurre una variación importante en la entrada. Por otro lado, los errores en la segmentación se magnifican a lo largo de la secuencia y la red es incapaz de corregirlos. Aún así, en secuencias sin variaciones muy grandes, o con la variación cerca del final de la secuencia, la red es capaz de llegar a una precisión del 90-99% con el conjunto de validación.

Respecto al uso de operaciones regularizadoras, al igual que en las arquitecturas 2D, no resulta en ninguna mejora.

5. Conclusiones

La segmentación de aorta con redes neuronales convolutivas sufre bastante debido a la dificultad intrínseca de los datos de entrada. El no poder diferenciar la aorta por la forma o intensidad, tanto en la imágenes 2D como en los pequeños volúmenes 3D, hace que la red necesite de relaciones mucho más complejas como la posición relativa de diferentes elementos. El aprendizaje de estos patrones complejos requiere de un aumento en parámetros que los modelos 3D implementados no han sido capaces de solventar.

Por otro lado, el uso de las operaciones regularizadoras, dropout y normalización por lotes, resultan en un empeoramiento cuando las redes se encuentran en su límite de aprendizaje. En casos en los que se pueda aumentar el número de parámetros de la red o se tengan pocos datos para entrenar, su uso sí ayuda a la generalización.

Para el cálculo del error, demostramos que BNCE es efectivo y preciso, con el suficiente tiempo de entrenamiento, y que las medidas de calidad no tienen por qué producir mejores resultados cuando se utilizan para el aprendizaje.

Entre las funciones de activación tratadas, vemos que sus resultados varían dependiendo de la arquitectura en la que se implementen. En cambio, no parece haber mucha diferencia de resultados entre las mismas.

Es posible mejorar los resultados al aplicar CRF a las imágenes de salida, como hace DeepLab, pero creemos que el uso de recurrencia, a la salida de un arquitectura 2D, puede llegar a dar mejores resultados. Por tanto, dejamos su refinamiento para futuras investigaciones en la materia.

Por último, una comparación entre el método Snake3D creado por la facultad y la arquitectura 2D de U-net producen resultados similares de media, pero la precisión de U-net varía menos entre casos complicados y sencillos.

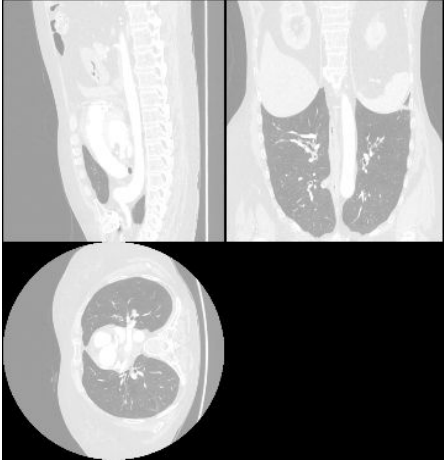




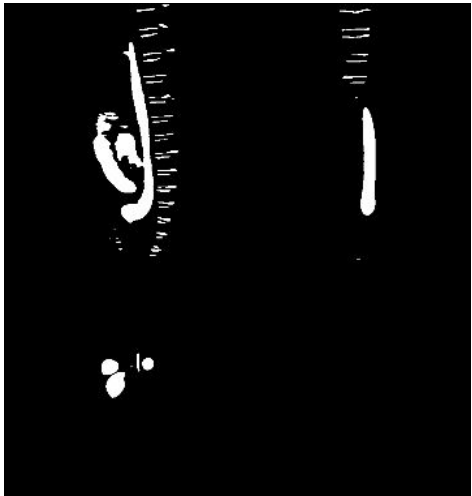
Comparación del método de la facultad con U-net

Método	Caso Fácil (mIOU)	Caso Difícil (mIOU)	Calidad Media Test (mIOU)
SNAKE3D (EII)	0.904	0.742	0.842
U-net	0.813	0.821	0.811

Resultados y tiempos de cada arquitectura

Red implementada	Tiempo de entrenamiento por lote	Calidad Media Test (mIOU)
U-net	2.82 sec	0.811
DeepLab v3	1.7 sec	0.761
3D U-net (modificada)	10.1 sec	0.616
U-net + parte recurrente	38.2 sec (por secuencia)	0.428

Comparación de la salida de cada arquitectura. Caso Medio

Entrada	Salida Deseada
	
Salida U-net	Salida DeepLabv3
	
Salida 3D U-net (modificada)	Salida U-net + parte recurrente
	

6. Bibliografia

- [1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov.: 2014.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
- [2] Sergey Ioffe, Christian Szegedy.: 2015.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.
- [3] Andrew L. Maas, Awni Y. Hannun, Andrew Y. Ng.: 2013.: Rectifier Nonlinearities Improve Neural Network Acoustic Models.
- [4] Djork-Arné Clevert, Thomas Unterthiner, Sepp Hochreiter.: 2015.: Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).
- [5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.: 2015.: Deep Residual Learning for Image Recognition.
- [6] Sergey Ioffe.: 2017.: Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models.
- [7] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller.: 2015.: Striving For Simplicity: The All Convolutional Net.
- [8] Fisher. Fisher Yu, Vladlen Koltun.: 2016.: Multi-Scale Context Aggregation By Dilated Convolutions.
- [9] Karen Simonyan, Andrew Zisserman.: 2014.: Very Deep Convolutional Networks for Large-Scale Image Recognition.
- [10] Evan Shelhamer, Jonathan Long, and Trevor Darrell.: 2016.: Fully Convolutional Networks for Semantic Segmentation.
- [11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio.: 2014.: Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.
- [12] Md Atiqur Rahman and Yang Wang.: 2016.: Optimizing Intersection-Over-Union in Deep Neural Networks for Image Segmentation.
- [13] Diederik P. Kingma, Jimmy Ba.: 2017.: Adam: A Method for Stochastic Optimization.
- [14] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, Christoph Bregler.:2015.: Efficient Object Localization Using Convolutional Networks.
- [15] Olaf Ronneberger, Philipp Fischer, Thomas Brox.: 2015.: U-Net: Convolutional Networks for Biomedical Image Segmentation.

- [16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun.: 2015.: Road Extraction by Deep Residual U-Net.
- [17] Liang-Chieh Chen, George Papandreou, Florian Schroff, Hartwig Adam.: 2017.: Rethinking Atrous Convolution for Semantic Image Segmentation.
- [18] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, Alan L. Yuille.: 2017.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.
- [19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun.: 2015.: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition.
- [20] Andrew Aitken, Christian Ledig, Lucas Theis, Jose Caballero, Zehan Wang, Wenzhe Shi.: 2017.: Checkerboard artifact free sub-pixel convolution: A note on sub-pixel convolution, resize convolution and convolution resize.
- [21] Eunbyung Park.: Groupout: A Way to Regularize Deep Convolutional Neural Network.
- [22] Fausto Milletari, Nassir Navab, Seyed-Ahmad Ahmadi.: 2016.: V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation.
- [23] Md Zahangir Alom, Mahmudul Hasan, Chris Yakopcic, Tarek M. Taha, Vijayan K. Asari.: 2018.: Recurrent Residual Convolutional Neural Network based on U-Net (R2U-Net) for Medical Image Segmentation.
- [24] Nicolas Ballas, Li Yao, Chris Pal, Aaron Courville.: 2017.: Delving Deeper into Convolutional Networks for Learning Video Representations.
- [25] Kurugol S1, San Jose Estepar R, Ross J, Washko GR.: 2013.: Aorta segmentation with a 3D level set approach and quantification of aortic calcifications in non-contrast chest CT.
- [26] Jan Egger, Bernd Freisleben, Randolph Setser, Rahul Renapuraar, Christina Biermann, Thomas O'Donnell.: 2009.: Aorta Segmentation for Stent Simulation.
- [27] Zhiwei Fan, Chen Wang, Yuqing Zhu, Tianrun Li, Zhen Zhang.: Automate Aorta Segmentation in Thoracic Aortic Aneurysm CT Images Via Convolutional Neural Network.
- [28] Karen López-Linares, Nerea Aranjuelo, Luis Kabongo, Gregory Maclair, Nerea Lete, Mario Ceresa, Ainhoa García-Familiar, Iván Macía, Miguel A. González Ballester.: 2018.: Fully automatic detection and segmentation of abdominal aortic thrombus in post-operative CTA images using deep convolutional neural networks.

5. Enlaces de interés

- [a] MRICron - <http://people.cas.sc.edu/rorden/mricron/index.html>
- [b] ImageJ - <https://imagej.nih.gov/ij/>
- [c] Theano - <http://deeplearning.net/software/theano/>
- [d] CNTK - <https://www.microsoft.com/en-us/cognitive-toolkit/>
- [e] Tensorflow - <https://www.tensorflow.org/>
- [f] Keras - <https://keras.io/>
- [g] TFLearn - <http://tflearn.org/>
- [h] FloydHub - <https://www.floydhub.com/>
- [i] [2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5](#)