eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA COMPUTACIONAL Y SISTEMAS INTELIGENTES

Trabajo fin de Máster

Segmentación automática de procesos neuronales en microscopía electrónica mediante técnicas de aprendizaje profundo

---

Tiago Manuel Esteves Sieiro

Dirigido por
Ignacio Arganda Carreras

30 de septiembre de 2019

Resumen

En este trabajo se han utilizado redes neuronales convolucionales para la segmentación de imágenes biomédicas obtenidas mediante microscopia electrónica.

El trabajo se ha desarrollado usando la librería de Keras y ayudándonos de la herramienta Google Colaboratory para la ejecución de los modelos más pesados computacionalmente. Se ha comenzado entrenando redes neuronales artificiales para ir adentrándonos en el funcionamiento de la librería. Después se ha entrenado una red preentrenada, concretamente la VGG16, bloqueando todas sus capas convolucionales y dejando alguna desbloqueada. Y finalmente se ha modelado una red neuronal convolucional siguiendo la estructura de la red U-Net. Esta red ha dado buenos resultados en la segmentación de imágenes y se utiliza sobre todo en la segmentación de imágenes biomédicas.

La base de datos del caso principal, se ha obtenido de la competición lanzada en el International Symposium on Biomedical Imaging (ISBI) de 2012 y está compuesta por un conjunto de cortes de microscopia electrónica para entrenar algoritmos de aprendizaje automáticos y así poder realizar la segmentación automática de neuritas.

Agradecimientos

En especial a Ignacio Arganda, director de este trabajo, que siempre me ha ayudado con todas las dudas y problemas que han ido surgiendo y me ha guiado y animado en todo momento desde el principio.

También quiero agradecer a mis compañeros y profesores de máster que, aún no teniendo un contacto cara a cara con ellos ya que la realización del máster ha sido online, siempre han estado ahí para resolver dudas tanto por correo electrónico como por el foro de las asignaturas.

Índice

Índice de figuras

Índice de Tablas

1. Introducción

1.1. Motivación

La microscopia electrónica (EM) es una técnica que utiliza un haz de electrones acelerados para iluminar y producir imágenes de objetos diminutos. Usando EM, se pueden lograr niveles de magnificación y resolución mucho mayores que con un microscopio óptico porque la longitud de onda de los electrones es bastante menor que la de los fotones.

Esta técnica ha revelado datos novedosos sobre las sinapsis y otras estructuras subcelulares en el sistema nervioso de los mamíferos [5]. La EM seriada se ha utilizado para reconstruir la conectividad del sistema nervioso de *Caenorhabditis elegans* [6, 7]. Esta es una especie de nematodo de la familia Rhabditidae que mide aproximadamente 1 mm de longitud y vive en ambientes templados. Ha sido un importante modelo de estudio para la biología, muy especialmente la genética del desarrollo, desde los años 70. Esta técnica también está presente en la conectomía y está creando una línea de investigación ambiciosa cuyo objetivo es estudiar mapas integrales de conectividad cerebral mediante el uso de microscopia de nanoescala de alto rendimiento.

Las mejoras más recientes en la técnica EM han llevado a la obtención de imágenes de volúmenes mucho más grandes de tejido cerebral y conocimientos interesantes sobre los sistemas nerviosos de invertebrados [8–10], y circuitos neuronales de los mamíferos [11–14]. Sin embargo, estos estudios recientes también apuntan a una importante necesidad del desarrollo de una nueva tecnología computacional para ayudar al análisis de las imágenes de EM del tejido cerebral.

En el estudio [13], se reconstruyeron aproximadamente 1000 neuronas a partir de una retina de ratón utilizando 20,000 h de trabajo humano. A pesar de este gran esfuerzo, el volumen retinal reconstruido era de solo 0,1 mm en cada lado, lo suficientemente grande como para abarcar los tipos más pequeños de neuronas retinales. Este estudio empleó métodos semiautomatizados, utilizando avances en el aprendizaje automático para automatizar la mayor parte de la reconstrucción [15]. Sin la automatización, la reconstrucción habría requerido 10–100x más esfuerzo humano. Para reconstruir volúmenes más grandes, es fundamental mejorar la precisión de los algoritmos informáticos y, por lo tanto, reducir la cantidad de trabajo humano que requieren los sistemas semiautomáticos. Idealmente, la necesidad de interacción humana se eliminará progresivamente, permitiendo gradualmente la reconstrucción automática completa con la eventual corrección de sus resultados.

1.2. Objetivos

Este trabajo de fin de máster consiste en estudiar, implementar y evaluar redes neuronales convolucionales para la segmentación de imágenes de cerebro con EM. Los objetivos concretos son los siguientes:

- **Introducción al uso de técnicas y herramientas modernas de *deep learning* para el análisis de imágenes**. Se utilizará el ejemplo sencillo de clasificación de dígitos del conjunto de datos MNIST.
- **Introducción a las redes neuronales convolucionales para la clasificación de imágenes**. Se utilizará el mismo ejemplo anterior realizando una comparación con este, y también se aplicará sobre otra base de datos para clasificar imágenes de perros y gatos empleando redes preentrenadas.
- **Introducción a las redes neuronales convolucionales para la segmentación de imágenes**. Se utilizará la red U-Net para segmentar imágenes de núcleos.
- **Estudio práctico y reproducción del estado del arte en redes neuronales convolucionales para la segmentación de neuritas en microscopia electrónica**. Este estudio es favorable para reducir el tiempo en la reconstrucción de imágenes biomédicas y, de esta manera, tener un proceso completamente automatizado reduciendo la cantidad de trabajo humano.

2. Estado del arte

2.1. Aprendizaje profundo o *Deep learning*

En los últimos años, el aprendizaje profundo ha tenido un gran auge, sobre todo porque ha superado el nivel humano en resolver muchos problemas complejos. En concreto, trata de emular el aprendizaje humando haciendo uso de redes neuronales. Estas redes están formadas por capas y estas a su vez por neuronas, si incluyen 3 o más capas ocultas se les denominan redes neuronales profundas. El principal problema es que estas redes necesitan una gran cantidad de datos para ser entrenadas. Antes del uso del aprendizaje profundo, se usaban otras técnicas de clasificación como los árboles de decisión, modelos de regresión o técnicas de clusterización, junto con métodos deterministas de extracción de características de la imagen.

Muchos investigadores están recopilando datos para poder solventar este problema, como es el caso de la base de datos ImageNet [16]. Gracias a esto, se han creado nuevas arquitecturas como la AlexNet [1] (ver Figura 2.1), que fue la primera red profunda entrenada de manera eficiente y que ganó la competición ImageNet en 2012 con mucho margen sobre los otros competidores. Viendo los resultados que se están obteniendo con el tratamiento de imágenes, los investigadores están trabajando para poder crear métodos de aprendizaje profundo para imágenes médicas y que esta técnica sea la principal opción para poder realizar la segmentación, tracking o detección de objetos en dichas imágenes. Pero volvemos a lo mismo, la dificultad de obtener las imágenes porque se tienen que adquirir a través de un microscopio y sólo los expertos están capacitados para clasificarlas. Además, estas imágenes contienen muchas más instancias de otros objetos que las imágenes naturales.

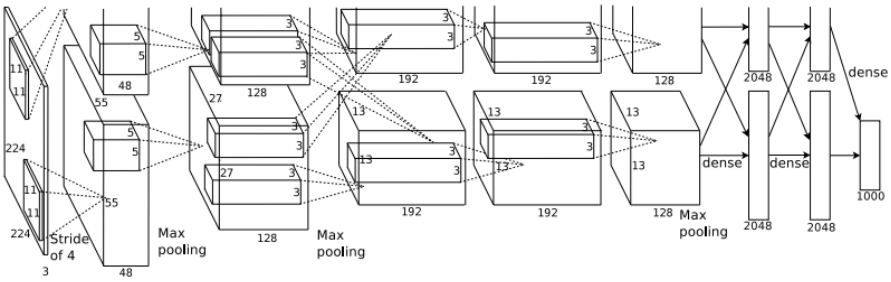

Figura 2.1: Ilustración de la arquitectura usada en la red AlexNet y obtenida del artículo oficial [1]. La arquitectura está compuesta por 8 capas con pesos. Las primeras 5 capas son convolucionales y las 3 restantes son totalmente conectadas. La primera capa recibe cómo entrada una imagen de tamaño 224x224x3 con 96 *kernels* de tamaño 11x11x3 y un salto de 4 píxeles. La última capa se alimenta de un *softmax* de tamaño 1.000 que realiza la distribución sobre las 1.000 clases.

2.2. Segmentación de imágenes

La segmentación de imágenes consiste en dividir una imagen en varias partes o segmentos de tal manera que se simplifique y/o cambie la representación de la imagen en otra más significativa y más fácil de analizar. Básicamente, se asigna a cada píxel de la imagen una etiqueta de tal manera que se puedan localizar objetos o regiones de interés en dicha imagen. En la Figura 2.2 tenemos un ejemplo de segmentación de dos imágenes, una de un gato y la otra de un perro.

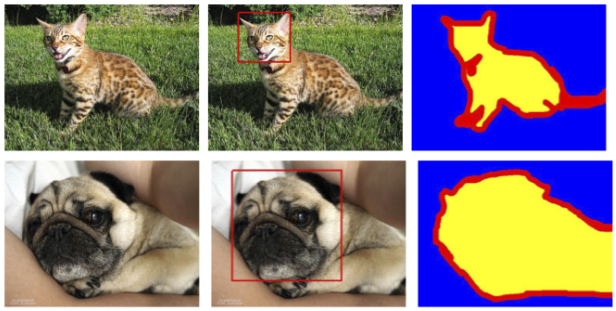

Figura 2.2: Ejemplo de segmentación de la imagen de un gato y un perro. Imagen obtenida de `https://www.robots.ox.ac.uk/~vgg/data/pets/`. En las imágenes de la izquierda podemos ver las imágenes originales, en el centro las detecciones de los rostros de los animales mediante un cuadrado, y en la de la derecha se han segmentado las imágenes para diferenciar el animal (amarillo), el contorno (rojo) y el fondo (azul).

2.3. Segmentación de neuritas en imágenes de microscopia electrónica

A pesar de lo descrito anteriormente, el aprendizaje profundo se ha adoptado rápidamente para la segmentación automática de imágenes en EM. En 2012 se organizó la competición ISBI 2012 [17] con el objetivo de comparar y clasificar los diferentes métodos competitivos en función de su precisión para clasificar píxeles y objetos con un mismo conjunto de datos. Para ello había que entrenar algoritmos de aprendizaje automático con el fin de segmentar automáticamente las estructuras neurales.

Uno de los métodos fue MLL-ETH, propuesto por Laptev et al. [18]. Comienza construyendo una correspondencia densa entre secciones vecinas. Esto es, para cualquier sección dada, las secciones anteriores y siguientes se deforman de forma no lineal utilizando el flujo de transformación de características invariantes de escala (SIFT) [19]. Comienzan con un total de 626 características para cada píxel combinando 18 filtros en 5 escalas diferentes. Se incluyen características de radón y rayos, filtros lineales y todos los componentes del histograma SIFT. Al incorporar las características de las secciones vecinas, el número final de características aumenta a 1878. Utilizan un clasificador de bosques aleatorio o *random forest* y se calcula la probabilidad de que un píxel pertenezca a una

membrana de manera independiente para cada píxel. Después, la segmentación de corte de gráfico se utiliza para tener en cuenta el hecho de que las etiquetas de píxeles vecinos tienen más probabilidades de estar conectadas. Por último y como procedimiento de procesamiento posterior, se realizan dos pasos de forma iterativa: la eliminación de la región, que se realiza mediante una serie de operaciones de umbral basadas en las propiedades de la región, como el área, la solidez, el número de Euler y la excentricidad (los parámetros de umbral se eligen a mano), y la transformación lineal que suaviza los resultados de segmentación y llena los espacios entre los segmentos de membrana.

Otro método fue LIC, que utiliza un enfoque de dos pasos. Se propone un marco de segmentación que explota un clasificador de píxeles [20] seguido de un árbol de fusión de cuenca [21] para la segmentación de neuronas 2D y la detección de membranas.

Otro de los métodos participantes consistía en un protocolo o *pipeline* de varios pasos implementados en la herramienta de código abierto CellProfiler [22].

El método que introdujeron Uher y Burget [23] fue el IMMI. Esta técnica comienza igualando los histogramas de todas las imágenes de entrenamiento y test, y de esta manera reduce la variabilidad de intensidad entre los cortes. Después se seleccionan manualmente los puntos de entrenamiento dando mayor prioridad a las áreas dónde existe el riesgo de no reconocer la membrana o fusionar la membrana con las mitocondrias. Por último se entrena una máquina de vectores de soporte (SVM) con *kernel* de puntos utilizando las diferentes características extraídas en función de un conjunto de filtros clásicos de cada píxel.

El algoritmo de Tan et al. [24] comienza con una clasificación inicial de píxeles que incluye la clasificación de umbral adaptativo utilizando la fuerza del borde (ADTES) y la ubicación de las regiones más oscuras de la imagen. Luego, se revisan las manchas oscuras que pueden clasificarse incorrectamente en la clase de membrana y se eliminan verificando su ocurrencia conjunta en las imágenes vecinas. Después de eso, se extrae un conjunto de características, que incluyen filtros a diferentes escalas, el filtro de apertura de atributos, es un filtro que elimina objetos según un atributo determinado como tamaño o circularidad, y la transformación de distancia del resultado obtenido de ADTES, y se utilizan para entrenar un SVM para una clasificación de píxeles adicional.

Otro método, introducido por Bas y Uzunbas [25], fue CLP, que caracteriza primero las regiones tanto estructural como contextualmente. Consiste en un mecanismo de decisión de múltiples etapas que utiliza las propiedades geométricas diferenciales subyacentes de los objetos en un marco biológicamente heredado. Un procedimiento de selección de características para seleccionar las características más relevantes caracteriza distintas regiones, como membrana, citoplasma y valores atípicos. Utilizan un clasificador de bosques aleatorio o *random forest* en inglés, que, si lo comparamos con un mapa topográfico, este clasificador re-

salta las estructuras de cresta de montaña, por ejemplo, membranas, así como mesetas, por ejemplo, citoplasma.

Pero la que ganó relmente fue IDSIA, la aproximación realizada por Ciresan et al. [26] que utilizó una red neuronal convolucional profunda como clasificador de píxeles. Para el entrenamiento aumentaron el conjunto de datos al comienzo de cada época reflejando aleatoriamente cada instancia de entrenamiento y/o girándola en $\pm 90°$. Para la salida, se realiza una calibración empleando un polinomio cúbico monótono cuyos coeficientes se calculan por ajuste de mínimos cuadrados. Después de esto se aplicó un filtro de mediana de radio de 2 píxeles para suavizar la imagen y así regularizar los límites de la membrana. Además de lo anterior, también realizaron unos experimentos manipulando las imágenes de entrada aplicando las técnicas de *foveation* y muestreo no uniforme. A partir de esto, todo el mundo empezó a usar *deep learning* con estos datos.

Otro avance en la segmentación automática de imágenes EM, fue cuando se introdujo una red neuronal totalmente convolucional (FCN) [27]. La idea principal fue la construir una FCN a partir de otras como AlexNet [1], VGG [28] y GoogLeNet [29] de tal manera que reciba datos de tamaño arbitrario y produzca resultados de tamaño correspondiente con inferencia y aprendizaje eficientes. En [30] se extendieron las redes FCN agregándole conexiones de salto cortas, que son similares a las introducidas en las redes residuales (ResNet) [31], de tal manera que se construyó una FCN muy profunda. Hasta entonces las FCN sólo utilizaban conexiones de salto largo para saltar características de la ruta de contracción a la ruta de expansión con el fin de recuperar información espacial perdida durante la disminución de muestreo. Posteriormente en [32] combinaron las FCN con las redes residuales totalmente convolucionales (FC-ResNet) y cuyo enfoque se centra en la importancia de un preprocesamiento entrenable cuando se utilizan FC-ResNets y mostraron que un modelo FCN de baja capacidad puede servir como un preprocesador para normalizar los datos de entrada médica.

En [4] se presentó la red U-Net (ver Figura 2.3). Esta red utiliza un aumento masivo de datos para utilizar las muestras anotadas disponibles de manera más eficiente. La arquitectura consiste en una ruta de contracción para capturar el contexto y otra ruta de expansión simétrica que permite una localización precisa.

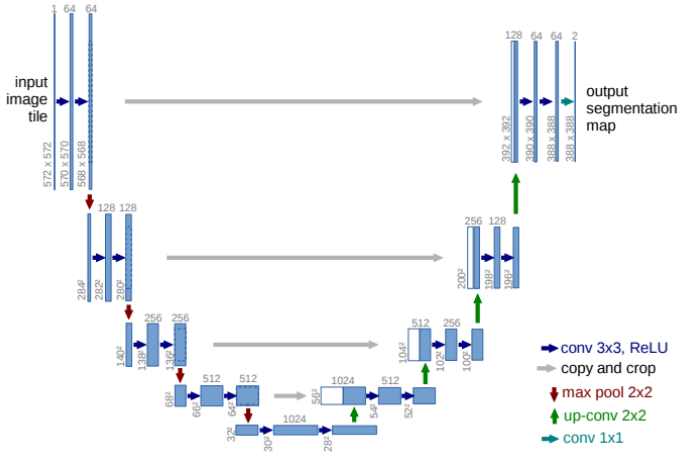

Figura 2.3: Arquitectura de la red U-Net. Una explicación más detallada se da en la Sección 3.4.

Otra arquitectura de red neuronal profunda fue la FusionNet [2] (ver Figura 2.4). Esta red se usó para la segmentación automática de estructuras neuronales de datos de conectomas aprovechando los últimos avances en aprendizaje automático, como la segmentación semántica y las redes neuronales residuales, y añadiéndole conexiones de salto basadas en suma para permitir una arquitectura de red mucho más profunda para una segmentación más precisa. Para demostrar el rendimiento, fue comparada con los métodos de segmentación de EM de la competición ISBI 2012.

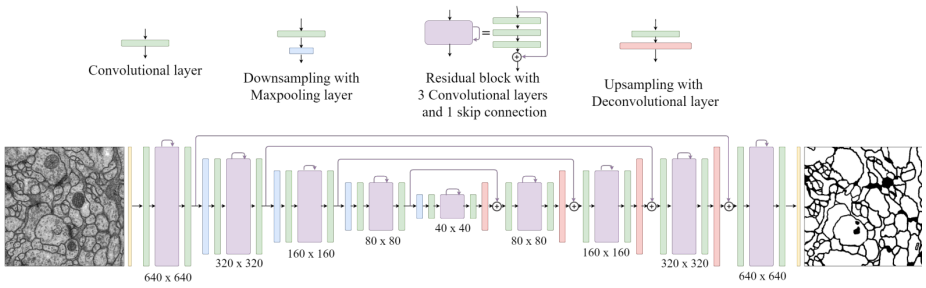

Figura 2.4: Ilustración de la arquitectura usada en la red FusionNet y obtenida y modificada del artículo oficial [2].

Otra algoritmo que también utilizó los datos de la competición ISBI 2012 fue [33], que redujo a la mitad el error del ganador de dicha competición en 2012 y actualmente son los líderes del ranking. Este algoritmo está dividido en tres pasos. Primero se aplica un bosque aleatorio en cascada (esto necesita menos datos de entrenamiento) o una CNN (que proporciona una mejor precisión) y se

predicen las probabilidades de membrana. Se percataron que en la red neuronal las capas de salto, el aumento de datos elásticos durante el entrenamiento y la predicción, y los módulos de inicio son críticos para el rendimiento. Después se agregaron píxeles en "superpíxeles"para resolver el problema y extraer información de la región de orden superior de una manera dependiente de los datos. Los superpíxeles obtenidos mediante el método *watershed* sobre la imagen de distancias ofrecieron la mejor compensación, produciendo grandes superpíxeles que son robustos frente a brechas menores en los mapas de probabilidad de límites. Finalmente fusionaron los superpíxeles con las posibles neuritas mientras respetaban las restricciones de consistencia en distancias que son más grandes que el campo de visión de una red neuronal. De esta manera resolvieron el problema Multicut [34], que introduce potenciales atractivos o repulsivos entre superpíxeles (no adyacentes), y encontraron la partición gráfica que equilibra de manera óptima estas señales.

En [35] propusieron redes totalemte convolucionales de entrada multi-recursiva de múltiples etapas para abordar el problema de la detección de los límites neuronales. Las múltiples entradas recursivas para una etapa, es decir, las salidas laterales múltiples con diferentes tamaños de campo receptivo aprendidos en la etapa inferior, proporcionan información de límites contextuales de múltiples escalas para el aprendizaje consecutivo.

Una característica deseable de un modelo de *deep learning* es que se transforme de manera equivalente bajo las transformaciones de su entrada. Las CNN implementan equivalencia traslacional por construcción; para otras transformaciones, sin embargo, se ven obligados a aprender el mapeo adecuado. Es por ello que en [36] se desarrolló una CNN de filtro orientable (SFCNN) que lograba equivalencia conjunta bajo traslaciones y rotaciones por diseño. Emplean filtros orientables para calcular eficientemente las respuestas dependientes de la orientación para muchas orientaciones sin sufrir artefactos de interpolación por la rotación del filtro.

En la conferencia ISBI 2018, se presentó un nuevo enfoque [37] efectivo de *deep learning* dónde se utiliza una red residual espacialmente eficiente y representaciones multinivel de señales contextuales para lograr un rendimiento de segmentación preciso.

Recientemente se ha publicado la red ACE-Net [3] (ver Figura 2.5) como variante a la red U-Net, que sigue siendo una red predominante en varias aplicaciones de segmentación de imágenes biomédicas. El objetivo de esta nueva red es mejorar la representación y utilización de las funciones al aumentar las rutas de contracción y expansión. Se aumentaron los caminos mediante las técnicas avanzadas recientemente propuestas que incluyen la agrupación de pirámides especiales atroces o *atrous pool piramide espacial* en inglés (ASPP) [38], conexión densa y mecanismos de supervisión profunda, y conexiones novedosas como la conexión directa de la imagen en bruto al lado expansivo. Con estos aumentos,

ACE-Net puede utilizar funciones de múltiples fuentes, escalas y campos de recepción para segmentar mientras se mantiene una arquitectura relativamente simple.

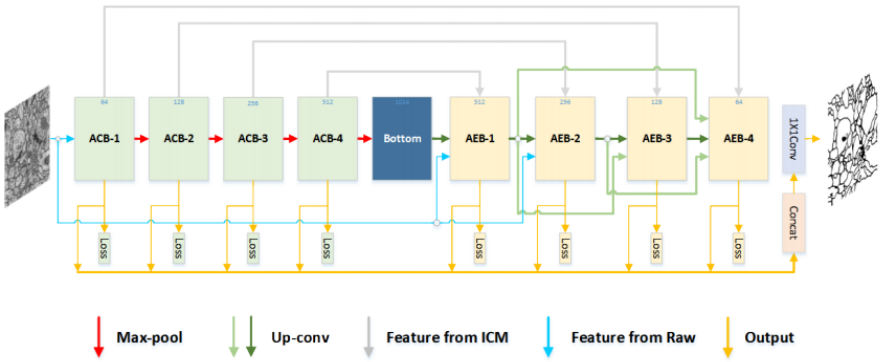

Figura 2.5: Ilustración de la arquitectura usada en la red ACE-Net y obtenida del artículo oficial [3].

Tras la competición ISBI 2012, el interés en la segmentación de neuritas continuó, surgiendo otras 2 nuevas competiciones, la ISBI 2013[1] y el MICCAI[2]. En ambas competiciones el objetivo era entrenar algoritmos para la segmentación de neuritas en 3D, pero en la segunda también se evaluaba la precisión de la detección de sinapsis y la identificación de parejas sinápticas, además contaba con datos más grandes.

[1] http://brainiac2.mit.edu/SNEMI3D/home
[2] https://cremi.org/

3. Metodología

3.1. Modelado de una neurona

Las redes neuronales son un modelo computacional que se inspiró principalmente con el objetivo de emular el comportamiento de los sistemas neuronales biológicos. Estas redes están compuestas por neuronas que reciben unos valores de entrada y tras aplicar unas fórmulas matemáticas, generan un valor de salida.

La unidad computacional básica del cerebro es la neurona. Cada neurona recibe señales de entrada de sus **dendritas** y produce señales de salida a lo largo de su **axón**. Eventualmente el axón se ramifica y se conecta a través de **sinapsis** a las dendritas de otras neuronas.

En el modelo computacional, las señales que viajan a través de los axones ($\boldsymbol{x_0}$) interactúan con la neurona multiplicándose por el valor de la fuerza sináptica ($\boldsymbol{w_0}$) en esa sinapsis ($\boldsymbol{w_0 x_0}$). Estas fuerzas sinápticas o pesos ($\boldsymbol{w}$) son aprendibles y controlan la fuerza de influencia de una neurona en otra. Esta información atraviesa la neurona y produce un valor de salida. Tras esta salida, puede existir una función limitadora o umbral, que modifica el valor generado o impone un límite que se debe sobrepasar para poder propagarse a la siguiente neurona. Esta función se conoce como **función de activación**. Un ejemplo de función de activación es la sigmoide σ que toma cómo entrada un valor real y lo modifica para que oscile entre 0 y 1.

En la Figura 3.1 podemos ver una neurona biológica (izquierda) y un modelo matemático común (derecha).

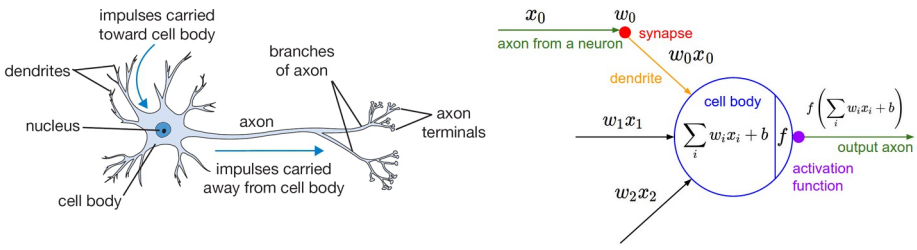

Figura 3.1: Dibujo de una neurona biológica (izquierda) y su modelo matemático (derecha). Imágenes obtenidas de `http://cs231n.github.io/neural-networks-1/`

3.2. Redes neuronales artificiales

Las redes neuronales artificiales son unos algoritmos disponibles dentro del campo del aprendizaje automático o *Machine Learning* (en inglés). El *Machine Learning* es una rama de la Inteligencia Artificial cuyo objetivo es desarrollar algoritmos para crear modelos estadísticos que se ajusten a un conjunto de datos.

Aunque parece que el concepto de redes neuronales es actual, en realidad llevan existiendo desde mediados del siglo XX. Pero es ahora, gracias a los avances tecnológicos y mejoras en las técnicas, cuando han revivido con más fuerza.

Estas redes neuronales se componen principalmente de capas y estas a su vez de neuronas.

3.2.1. La neurona (artificial)

Las neuronas, son las unidades básicas de procesamiento dentro de una red neuronal y están basadas en el perceptrón de Rosenblatt [39] aunque su funcionamiento ha cambiado ligeramente desde entonces.

Básicamente, una neurona recibe unos valores de entrada x_i y tras aplicar un cálculo interno, genera un valor de salida. Los componentes de la neurona son los siguientes:

- **Valores de entrada:** vector con los N valores de entrada a la neurona, se representa de la siguiente manera, $x = [x_1, x_2, x_3, ..., x_N]$.
- **Pesos:** es un vector con el peso correspondiente a cada valor de entrada y determina la importancia de este. Tiene que tener el mismo tamaño que los valores de entrada y se representa de la siguiente manera, $w = [w_1, w_2, w_3, ..., w_N]$.
- **Parámetro sesgo o *b*:** indica la facilidad que tiene una neurona para activarse.
- **Función de propagación:** actúa sobre los pesos y el sesgo. Consiste en una suma ponderada de los valores de entrada junto con su correspondiente peso. Tiene la siguiente forma matemática:

$$y = \sum_{i=1}^{N} w_i x_i + b \tag{3.1}$$

- **Función de activación:** determina si se activa la salida de la neurona en función al valor obtenido tras realizar la función de propagación. Para eliminar la linealidad de la función de propagación distorsiona el valor obtenido.

En la Figura 3.2 podemos visualizar una representación gráfica de la neurona con todos sus componentes.

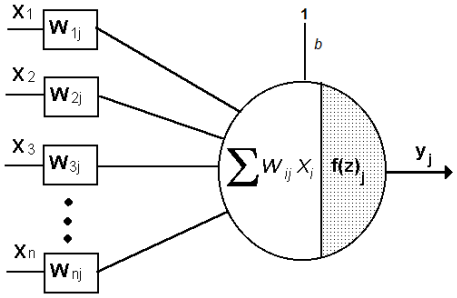

Figura 3.2: Representación gráfica de una neurona artificial Imagen adaptada de `http://joel-redesneuronalesartificiales.blogspot.com/2008/07/redes-neuronales-artificiales.html`.

Existen diversas funciones de activación, pero las más comunes son las siguientes:

- **Sigmoidea:** es de la forma $f(y) = (1 + e^{-y})^{-1}$. Esta función lo que hace es 'aplastar' el valor de entrada en un rango entre 0 y 1. En particular, los valores negativos grandes se convierten en 0 y los positivos grandes en 1. Esta función se ha utilizado bastante, pero en la actualidad rara vez se utiliza ya que tiene dos inconvenientes principales:
  - Los sigmoides se saturan y matan el gradiente: esto quiere decir que cuando la activación de la neurona se satura en la cola de 0 o 1, el gradiente es casi cero. El concepto de gradiente se explicará con más detalle en capítulos posteriores.
  - Las salidas sigmoideas no están centradas en 0: esto es algo no deseado ya que las neuronas en capas posteriores van a recibir valores no centrados en cero. Como consecuencia, en el cálculo del descenso del gradiente todos los valores serán positivos o negativos.
- **Tanh:** es de la forma $f(y) = tanh(y)$. Modifica el valor de entrada para que esté en un rango entre -1 y 1. Esta función sí que genera los valores centrados en 0, por lo que es preferible usar esta función que la sigmoidea.
- **ReLU:** es de la forma $f(y) = max(0, y)$ y se ha vuelto muy popular en los últimos años.

Las 3 funciones descritas anteriormente, están representadas en la Figura 3.3.

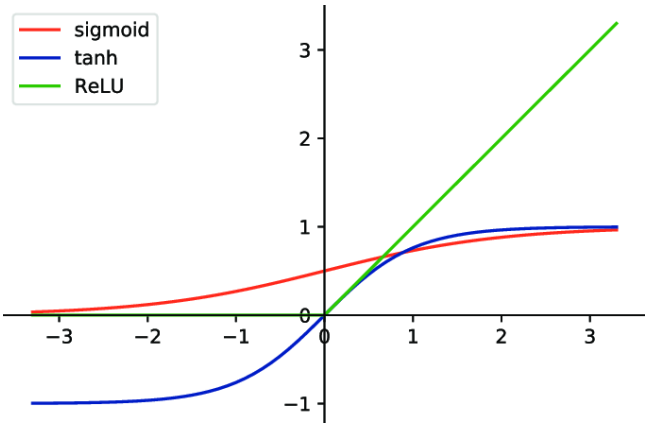

Figura 3.3: Gráfica de las tres funciones de activación. Imagen obtenida de `https://www.researchgate.net/figure/Function-curves-of-sigmoid-Tanh-and-ReLU_fig1_330467251`

3.2.2. Redes neuronales

Como se ha comentado anteriormente, las neuronas se agrupan en capas y estas a su vez en una red neuronal formando un gráfico acíclico. La primera capa es la capa de entrada y la última es la capa de salida. El resto de capas que puede contener la red, son capas ocultas. La salida de cada neurona está conectada con todas las neuronas de la capa siguiente pero no con las neuronas de su misma capa ya que esto provocaría un bucle infinito.

En el ejemplo donde sólo teníamos una neurona, el componente de pesos w era un vector, pero en el caso de múltiples capas y neuronas pasaría a ser una matriz. La matriz de pesos empleada en cada capa tendría el número de filas igual al número de neuronas de la capa actual y el número de columnas igual al número de neuronas de la capa anterior. En la Figura 3.4 tenemos un ejemplo de una red neuronal de tres capas con tres entradas, dos capas ocultas de cuatro neuronas cada una y una capa de salida. Este ejemplo tiene tres capas ya que la capa de entrada no se cuenta porque son los valores de entrada.

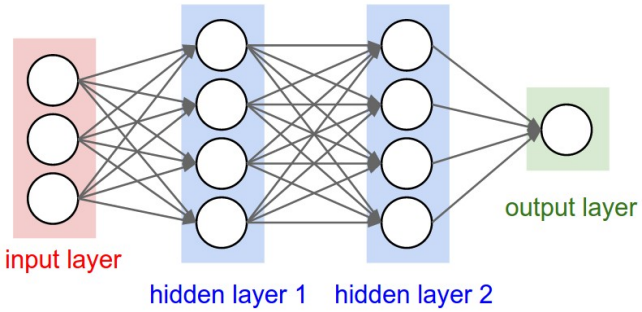

Figura 3.4: Red neuronal con dos capas ocultas. Imagen obtenida de `http://cs231n.github.io/neural-networks-1/`

En la capa de salida se suele utilizar una función de activación diferente a las capas ocultas. Esta es la función *softmax* que genera una distribución de probabilidad sobre las distintas clases, es decir, los valores generados por las neuronas oscilan entre los valores 0 y 1, ambos incluidos, y la suma de los valores es igual a 1.

$$softmax(z) = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_k}} \qquad i = 1, 2, ...K, \tag{3.2}$$

dónde K es el número de clases.

La red neuronal descrita anteriormente, es una red *Feedforward* ya que no tiene ciclos en el grafo. Una vez explicada la estructura de la red neuronal, hay que saber cómo obtener el valor de los pesos w y los sesgos b. Estos valores se obtienen a través del entrenamiento.

3.2.3. Entrenamiento de la red neuronal

Para el entrenamiento de la red partimos de un conjunto de entrenamiento. Como estamos en un proceso de aprendizaje supervisado, estos datos están compuestos por lo valores x y la clase y' a la que pertenece. El objetivo es calcular los valores de los pesos w y los sesgos b tal que la clase y predicha se aproxime a la clase correcta y'. Podemos decir que este es un proceso de 'ir y venir' continuo.

Recordemos que la red neuronal está compuesta por capas, estas a su vez por neuronas y cada neurona está totalmente conectada con las neuronas de la capa contigua. Las neuronas de la primera capa oculta reciben los datos de entrada, a estos se les aplica la función de propagación junto con los pesos y los sesgos, y al resultado se le aplica la función de activación. Todo esto genera unos datos de salida, que pasarían a convertirse como datos de entrada de la capa siguiente y se aplicaría el mismo procedimiento. Todo esto se iría ejecutando a través de todas las capas hasta llegar a la capa de salida dónde se obtendrían las predicciones o *labels* y a los datos de entrada. Este proceso se denomina

fordward propagation, y es el 'ir' anteriormente mencionado. A continuación tenemos que saber si la predicción es buena o mala, y para ello se utiliza la función de coste. Esta función compara el resultado predicho con el resultado real y su objetivo es reducir esta diferencia y aproximarse a cero. Una vez calculado este coste, propagamos hacia atrás esta información a través de todas las neuronas que contribuyen directamente a la salida, pero cada neurona sólo recibe una fracción de este valor dependiendo de la contribución relativa que haya aportado a la salida. A este proceso se le denomina *backward propagation*, y es el 'venir' mencionado anteriormente. El número de veces que tenemos que ejecutar estas dos acciones sobre todo el conjunto de datos está determinado por el número de épocas o *epoch*. Por cada época podemos determinar con el parámetro *batch size* el número de iteraciones para completar una época. El *batch size* determina la cantidad de datos a coger por cada época. Es decir, si tenemos 10 épocas, 1000 datos y establecemos el tamaño de *batch size* a 100, realizaríamos 10 iteraciones para completar una época. En la Figura 3.5 hay un esquema de este proceso.

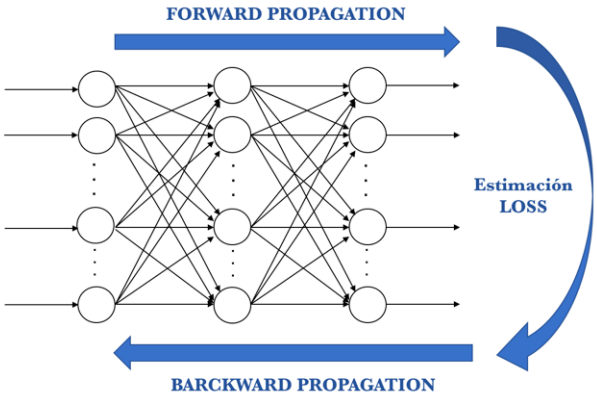

Figura 3.5: Etapas del entrenamiento. Imagen obtenida de `https://torres.ai/deep-learning-inteligencia-artificial-keras/`

Una vez propagado el valor de pérdida, hay que ajustar los pesos y los sesgos para que la próxima vez que se ejecute el *fordward propagation* el valor de coste tienda a cero. Esto se traduce como un problema de optimización. Existen diferentes optimizadores como el *stochastic gradient descent* (SGD) [40], RMSprop [41], Adagrad [42], Adadelta [43] o Adam [44]. En función de los datos y del problema se suelen usar unos u otros.

3.2.4. Condiciones de parada del entrenamiento

Uno de los temas a tratar cuando estamos entrenando el modelo es cuándo debería parar de entrenar. De primeras, tenemos el valor de las épocas, que nos indica cuántas veces tenemos que ejecutar el *fordward propagation* y el *backward propagation* con todo el conjunto de datos. Pero tenemos otra opción que

se llama parada temprana o *early stopping* en inglés. Consiste en controlar el número de épocas que se ejecutan sin que el modelo mejore la métrica de evaluación en el conjunto de validación. Alcanzado ese valor, se procede a parar el entrenamiento. Este número de épocas se conoce normalmente con el nombre de *paciencia*.

La métrica de evaluación se obtiene sobre un conjunto de validación mientras se hace el entrenamiento. Es decir, una vez finalizada la época se evalúa la red con los datos de validación para comprobar que tal va el entrenamiento empleando unos datos que no influyen en la configuración de los pesos. Estos datos suelen ser un subconjunto de los de entrenamiento por lo que también contamos con los valores de las etiquetas.

En Keras tenemos la opción de *callbacks*[3], que son funciones que se ejecutan mientras se está entrenando el modelo. Dentro de estas funciones se encuentra *EarlyStopping*[4] que sirve exactamente para realizar la conocida como *parada temprana*.

3.2.5. Optimización

Recordemos que la función de coste nos permite comprobar cómo de bueno ha sido el resultado utilizando un conjunto de pesos w y sesgos b. Llegados a este punto tenemos que mejorar el resultado obtenido, por lo que hay que encontrar los valores de los pesos w y sesgos b que optimicen o minimicen el valor de la función de coste. A continuación se explican tres estrategias para ello.

- **Búsqueda aleatoria:** la primera estrategia consiste en ir generando valores aleatorios de los parámetros y hacer un seguimiento para saber qué conjunto de datos es mejor. Esta es una mala idea ya que requiere un tiempo de computación elevado y con resultados muy malos.
- **Búsqueda local aleatoria:** en esta estrategia generamos un valor aleatorio, después generamos una perturbación en dicho valor, y si la pérdida con la perturbación en menor, aplicamos la actualización de los datos.
- **Siguiendo el gradiente:** la última estrategia sería emplear el descenso del gradiente ya que permite encontrar mínimos locales de una función. Este optimizador es la base de muchos otros y uno de los algoritmos de optimización más comunes. Determina de manera iterativa los valores de w y de b para minimizar el valor de la función de coste y para ello calcula la primera derivada parcial de esta función con respecto a los parámetros en cada una de las neuronas de cada capa, este dato es el gradiente. Al gradiente se le multiplica por una constante que afecta sobre cuánto hay

[3] `https://keras.io/callbacks/`
[4] `https://keras.io/callbacks/#earlystopping`

que moverse con respecto a la posición en la que estás. Este valor se denomina factor de aprendizaje o *learning rate*. Es decir, si el gradiente nos devuelve 1,5 y el *learning rate* es de 0,01, entonces el siguiente punto estaría a 0,015 del punto anterior. Cómo el gradiente nos da la dirección en sentido ascendente de la recta, nos tendríamos que mover en sentido contrario.

3.2.6. Regularización

La regularización es el proceso por el cual se intenta que una red neuronal evite el sobreajuste u *overfitting*. El sobreajuste es el efecto de sobreentrenar una red neuronal con unos datos de tal manera que el modelo se ajusta excesivamente a los datos de entrenamiento. Como consecuencia, si introducimos nuevos datos, obtendríamos un mayor error ya que el modelo ha memorizado y no aprendido. Algunas de las técnicas para evitar el *overfitting* son las siguientes:

- **Regularización L2:** quizá es la forma más común de regularización. Consiste en añadir un término adicional a la función de coste. Este término sería $\lambda \sum(w^2)$ dónde λ es la fuerza de regularización.
- **Regularización L1:** es similar a la anterior, pero en este caso a la función de coste se le sumaria el término $\lambda \sum |w|$.
- ***Dropout*:** es una técnica de regularización extremadamente efectiva, simple y recientemente introducida [45]. Consiste en seleccionar aleatoriamente, pero con una determinada probabilidad, unas neuronas de cierta capa y poner a 0 su salida en la fase de entrenamiento. Esta probabilidad se define mediante un parámetro que toma los valores de 0 a 1. El valor por defecto es 0.5 lo que indica que la mitad de las neuronas quedaran activadas, según se vaya reduciendo o aumentando este valor, se desactivarán menos o más neuronas. Cada vez que se actualizan los pesos, las neuronas desactivadas son distintas.
- **Aumento de datos:** o *data augmentation*, consiste en modificar los datos de entrenamiento de manera controlada. Esto es, realizar pequeñas modificaciones como rotar, escalar, trasladar, relleno... sobre la imagen de tal manera que la red sería más robusta ante estos cambios. En la Figura 3.6 tenemos un ejemplo de *data augmentation*.

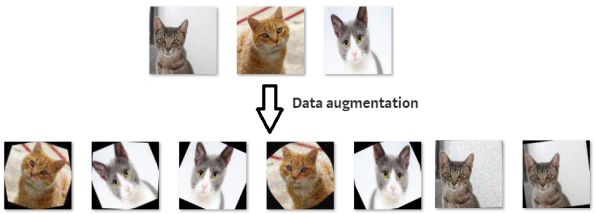

Figura 3.6: Ejemplos de *data augmentation*. Imagen obtenida de `https://medium.com/@thimblot/data-augmentation-boost-your-image-dataset-with-few-lines-of-python-155c2dc1baec`

3.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN por sus siglas en inglés) son muy similares a las redes del apartado anterior, tienen pesos, sesgos, entradas, función de activación, función de coste, etc. La única diferencia es que estas redes están pensadas para el trabajo con imágenes ya que permiten codificar ciertas propiedades de la imagen en la arquitectura. Con las redes neuronales artificiales básicas se puede trabajar con imágenes, pero estas presentan dos grandes problemas. El primero es que el número de neuronas depende del tamaño de la imagen, es decir, una imagen de tamaño 32x32x3 (32 de ancho, 32 de alto, 3 canales de color) tendría un total de 3072 neuronas en la capa de entrada. El otro problema es que un píxel solo dentro de la imagen no te da mucha información, ya que dependiendo de los pixeles vecinos, tiene un sentido u otro. Esto podría provocar un sobreajuste y un entrenamiento lento y costoso.

Las neuronas de una CNN están dispuestas en 3 dimensiones (ancho, alto y profundo) y las neuronas de cada capa sólo se conectarán a una pequeña región de la capa anterior. Están formadas por una secuencia de capas y estas pueden ser la capa convolucional, la capa de *pooling* y la capa totalmente conectada. Las dos primeras componen el extractor de características y la última clasifica dichas características mediante una red neuronal básica.

Grosso modo, el proceso sería el siguiente: la convolución genera unos mapas de características tras aplicar diferentes filtros a la imagen obteniendo así una imagen transformada y de dimensiones reducidas. Después se aplica la función de activación como *ReLU*, *ELU*, etc. Tras esto, se extraen datos más significativos con la operación de *pooling*. Esta combinación de operaciones se puede implementar tantas veces cómo sea necesaria. Por último, la salida de la última capa de *pooling* se conecta a una red neuronal normal para clasificar el resultado. En la Figura 3.7 podemos ver un ejemplo de una red neuronal convolucional.

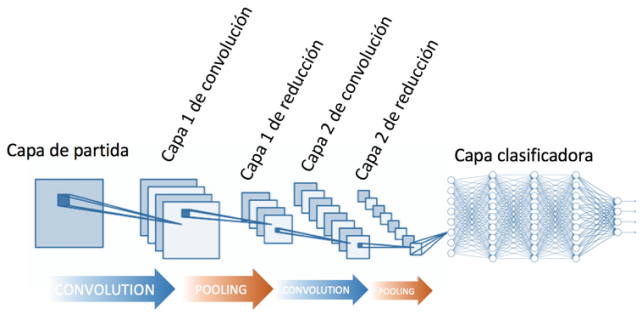

Figura 3.7: Red neuronal convolucional. Imagen obtenida de `http://www.diegocalvo.es/red-neuronal-convolucional/`

3.3.1. Capa de convolución

Esta capa es la que realiza la mayor parte del trabajo pesado computacional. Los parámetros de esta capa, consisten en un conjunto de filtros que se pueden aprender. Este filtro o *kernel* es un tensor de un tamaño más pequeño que la imagen original (3x3, 5x5, 11x11 ...) pero con la misma profundidad que la imagen original (1 para la escala de grises o 3 para una imagen RGB). Este filtro es la matriz de pesos w que se ha empleado en las redes neuronales artificiales básicas. Cómo se ha comentado, no todas las neuronas de la capa de entrada se conectan con todas las neuronas de la primera capa oculta, se conectan por pequeñas zonas cuyo tamaño lo define el filtro. Esto es, cada neurona de la primera capa oculta se conecta a una pequeña región de 3x3 neuronas (9 neuronas).

Intuitivamente, este filtro se va deslizando sobre el ancho y alto de la imagen de entrada obteniendo así el producto escalar entre los valores del filtro y de la región de la imagen. Esto va a generar un mapa bidimensional denominado mapa de activación. El tamaño de este mapa depende del tamaño del filtro, el salto o *stride* empleado para mover el filtro a través de la imagen y el modo en que tratemos el borde la imagen.

Por ejemplo, si tenemos una imagen de dimensiones 7x7x1, un filtro de tamaño 3x3, el valor del *stride* a 1 y sin tratar los bordes, obtendríamos un mapa de activación de tamaño 5x5 ya que sólo podríamos mover el filtro arriba y abajo 4 posiciones empezando desde la esquina superior izquierda. En la Figura 3.8 tenemos de manera visual este proceso.

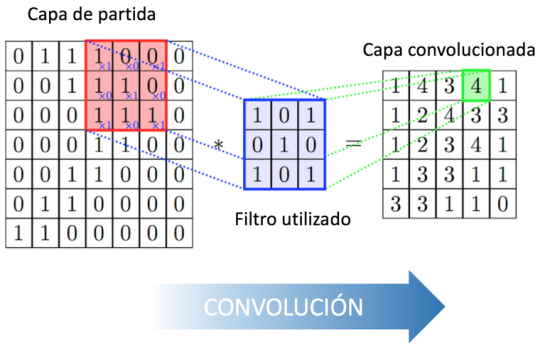

Figura 3.8: Ejemplo de convolución usando un filtro 3x3. Imagen obtenida de `http://www.diegocalvo.es/red-neuronal-convolucional/`

Una vez obtenido el mapa de activación, procederíamos a sumar el valor del sesgo b a cada elemento y aplicar la función de activación no lineal (*ReLU*, *Leaky ReLU*, *Maxout* ...).

3.3.2. Capa de *pooling*

Estas capas se suelen aplicar después de una serie de capas de convolución. Su principal objetivo es simplificar la información recogida en la capa convolucional y crear una versión reducida de esta. Es decir, teniendo un mapa de activación de tamaño 4x4, podemos usar regiones de tamaño 2x2 para reducirlo creando un mapa de tamaño 2x2. En la Figura 3.9 tenemos de manera visual este procedimiento.

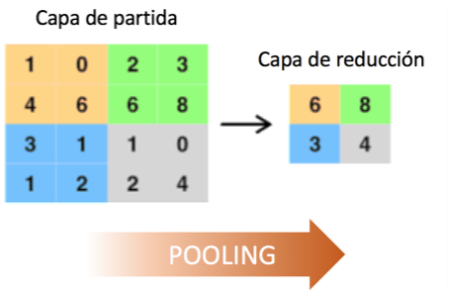

Figura 3.9: Ejemplo de *pooling* usando un filtro 2x2. Imagen obtenida de `http://www.diegocalvo.es/red-neuronal-convolucional/`

Para condensar esta información hay diferentes operaciones, la más habitual es el *max-pooling*, que se queda con el valor más alto de los que había aplicando el filtro. También existe el *average-pooling*, que obtendría el promedio de los valores.

3.4. U-Net

La red U-Net, es una red totalmente convolucional que realiza la segmentación de imágenes. Su objetivo es predecir la clase de cada pixel y así solventar el problema de segmentación de imágenes biomédicas. Las CNN, tiene para cada imagen de entrada una única clase de salida, pero para la segmentación biomédica necesitamos clasificar cada píxel de la imagen ya que necesitamos incluir una localización.

Fue presentada por Ronneberger et al. [4] mejorando la estrategia tomada por Ciresan et al. [26] para solventar los problemas que presentaba. Primero, era lento porque la red se tenía que ejecutar de manera independiente por cada región seleccionada alrededor de cada pixel. Segundo, es que hay una compensación entre la precisión de localización y el uso del contexto. Grandes regiones necesitan más capas de *max-pooling* que reducen la precisión en la localización, mientras que las pequeñas regiones permiten a la red ver sólo un pequeño contexto.

Esta red sigue la arquitectura de reloj de arena y tiene forma de U, que es lo que le da el nombre. Primero aplica convoluciones a la imagen de entrada, comprimiendo la información y detectando las diferentes características en la imagen, a esta parte también se la conoce como ruta de contracción. Después genera la nueva imágen empleando las características aprendidas durante la contracción, es la ruta de expansión. Las deconvoluciones son las operaciones contrarias a las convoluciones y utilizan los filtros aprendidos para restaurar la imagen comprimida. En la Figura 3.10, podemos ver la estructura de esta red.

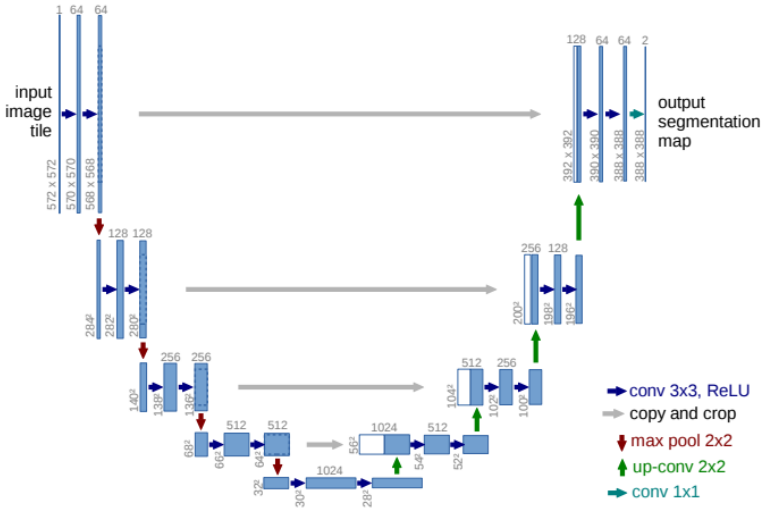

Figura 3.10: Arquitectura de la red U-Net. Imagen obtenida del artículo oficial [4]

La parte de contracción, tiene una arquitectura típica de una red convolucional. Consiste en la aplicación repetida de dos convoluciones de 3x3 seguidas de la función de activación *ReLU*, y de una capa de *pooling* de 2x2 empleando el *max-pooling* con el *stride* a dos y así reducir la resolución. Cada vez que reducimos la resolución, duplicamos el número de mapas de funciones, de tal manera que empezamos con 64, después 128 y así sucesivamente hasta llegar a 1024. La finalidad de esta parte es la de capturar el contexto de la imagen para poder realizar la segmentación.

La siguiente parte es la de expansión. Consiste en la aplicación repetida de una capa de deconvolución con un *stride* de 2 y un tamaño de 2x2, seguido de la concatenación con el mapa de entidades recortado correspondiente desde la parte de contracción y por último dos capas de convolución de 3x3 seguidas de la función *ReLU*. En la capa final, se utiliza una convolución 1x1 para asignar cada vector de características de 64 componentes al número deseado de clases. En total, la red tiene 23 capas convolucionales.

Cómo principal ventaja, destaca que la U-Net combina la información de ubicación de la parte de contracción con la información contextual en la parte de expansión para obtener finalmente una información general que combine localización y contexto, que es necesaria para predecir un buen mapa de segmentación. Habitualmente también hace uso del *data augmentation* obteniendo buenos resultados. Esto es importante en la segmentación biomédica, ya que el número de muestras anotadas suele ser limitado.

Dada su sencillez y gran rendimiento en este tipo de problemas, hemos seleccionado la arquitectura U-Net como la base de nuestro modelo para el estudio práctico de segmentación de neuritas en imágenes de EM (Sección 4.4).

4. Experimentos

En esta sección se van a presentar los experimentos realizados. Se han utilizado 4 conjuntos de datos diferentes. La ejecución de los 2 primeros conjuntos se ha realizado de manera local en un ordenador con CPU Intel Core i5-430M, 8GB de memoria RAM y un disco HDD sobre un sistema operativo Windows 7 Home Premium. En estas ejecuciones no ha sido necesario el uso de la GPU ya que consistían en entrenamientos y predicciones computacionalmente sencillas. En las dos se ha utilizado la versión de Python 3.6 y la de Keras 2.2.4 [46]. Estos datos han servido para introducirnos en el API de Keras, comenzando por unos conceptos básicos de las redes neuronales y terminando con una red convolucional.

En cambio, para los otros dos conjuntos de datos, uno de los cuales está compuesto por los datos principales de este proyecto, se ha utilizado la herramienta online disponible y proporcionada por Google, *Google Colaboratory* o *Google Colab* [47]. Esta herramienta está basada en los *Notebooks de Jupyter*, y permite el uso de GPUs y TPUs de manera gratuita junto con unas librerías como *PyTorch*, *TensorFlow* o *Keras*, por lo que es una buena opción si no se dispone de un ordenador con una GPU potente para hacer los cálculos. Todo ello pudiendo usar las versiones 2.7 y 3.6 de Python.

4.1. Reconocimiento de dígitos

4.1.1. Conjunto de datos

Este primer conjunto de datos, corresponde al conjunto de datos MNIST. Está compuesto por imágenes de dígitos, del 0 al 9, escritos a mano y escaneados. Contiene 60.000 imágenes para entrenar el modelo y otras 10.000 para testearlo y se pueden descargar desde su página web [48]. Aparte de las imágenes, también está disponible la etiqueta a la que pertenece cada dígito, de esta manera tenemos un aprendizaje supervisado. Cómo las imágenes son dígitos, estas etiquetas van del 0 al 9 indicando el dígito al que pertenecen,

Cada imagen original ha sido preprocesada, obteniendo el resultado de la Figura 4.1, y centradas en una imagen de tamaño 28x28. Este conjunto de datos es muy práctico para comenzar a probar las técnicas modernas de aprendizaje y reconocimiento de patrones.

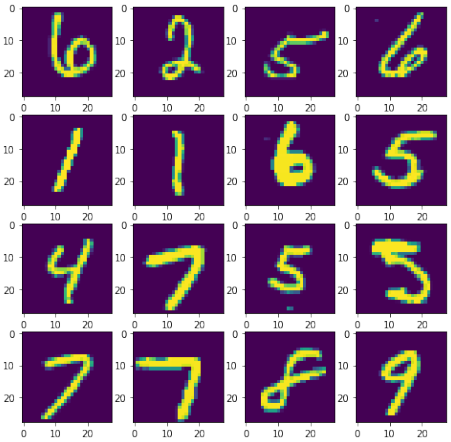

Figura 4.1: Ejemplos de algunos dígitos de la base de datos MNIST.

4.1.2. Arquitectura

Para esta base de datos, se han realizado pruebas con dos tipos de redes neuronales, las artificiales básicas y las convolucionales. Para las primeras se ha ejecutado 8 veces la misma arquitectura pero variando los diferentes parámetros como las funciones de activación, el optimizador, el número de épocas y el tamaño del *batch*. Esta arquitectura consiste en dos capas totalmente conectadas de 10 neuronas y de entrada tiene un vector de dimensión 784, que viene de convertir la imagen de tamaño 28x28 a un vector unidimensional. El objetivo es ir conociendo los diferentes parámetros que componen la red y cómo estos afectan a dicha red.

Para la CNN, se han creado dos arquitecturas diferentes. Para la primera arquitectura y en la parte de extracción de características, hemos juntado dos bloques compuestos de una capa de convolución de 32 y 64 neuronas, con la función de activación *ReLU* y un filtro de tamaño 5x5. Y otra capa de *pooling* empleando el *max-pooling* con el filtro 2x2. Tras esto, se le junta la parte de clasificación compuesta por una capa densamente conectada de 10 neuronas y haciendo uso de la función de activación *softmax*. En la Figura 4.2 podemos ver el esquema de dicha arquitectura.

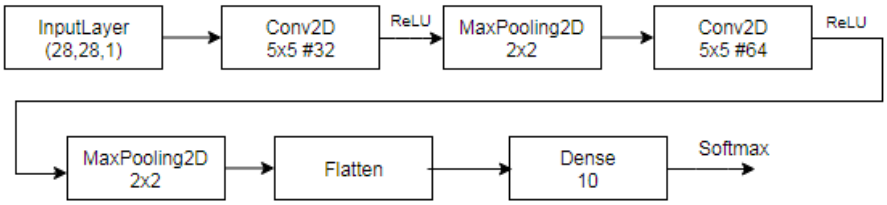

Figura 4.2: CNN sencilla para clasificar dígitos.

Para la segunda arquitectura se han añadido más capas. Para la parte de extracción de características, se han juntado dos bloques compuestos de dos capas de convolución de 32 y 64 neuronas, con la función de activación *ReLU* y un filtro de tamaño 5x5. Y otra capa de *pooling* empleando el *max-pooling* con el filtro 2x2. Tras esto, se le aplica la parte de clasificación compuesta por una capa densamente conectada de 100 neuronas y la función de activación *ReLU*, seguido de un *Droput* de 0,5 y, por último, otra capa densamente conectada de 10 neuronas y la función de activación *softmax*. En la Figura 4.3 podemos ver el esquema de dicha arquitectura.

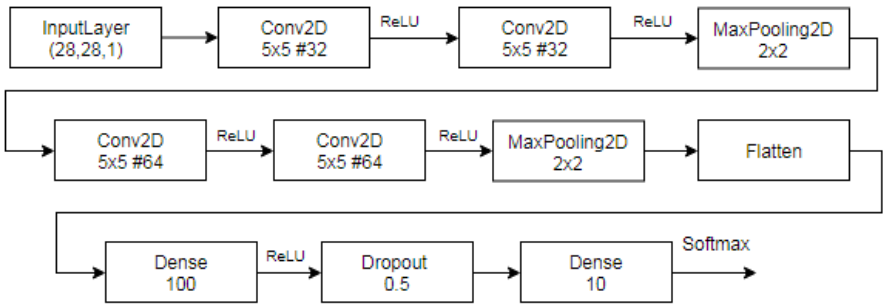

Figura 4.3: CNN ampliada para clasificar dígitos.

4.1.3. Experimentos y Resultados

Como ya hemos comentado, podemos dividir estos experimentos en 2 partes, en una primera parte usando redes neuronales artificiales básicas (perceptrones multi-capa), y en la otra empleando CNN.

Para la primera, podemos ver en la Tabla 4.1 la configuración de la red empleada en cada caso. Sólo se muestran los datos diferentes. Recordemos que esta arquitectura utiliza dos capas densamente conectadas de 10 neuronas, con un vector de entrada de tamaño 784 y cuya función de activación en la última capa, va a ser siempre *softmax*, por lo que la función de activación reflejada en la tabla corresponde a la de la primera capa. Para entrenar, se ha ido variando

el optimizador, las épocas y el tamaño del *batch*, mientras que la función de coste y la métrica de evaluación han sido siempre la entropía cruzada categórica o *categorical_crossentropy*, y la precisión o *accuracy* respectivamente. Para el optimizador del descenso del gradiente estocástico (SGD) se ha utilizado un *learning rate* de 0,01 y para el *RMSprop* de 0,001.

Experim.	Función activ.	Optim.	Tamaño batch	Épocas	Precisión test
1	Sigmoide	SGD	100	5	0,7532
2	ReLU	SGD	100	5	0,8986
3	Sigmoide	RMSprop	100	5	0,9049
4	ReLU	RMSprop	100	5	0,9238
5	Sigmoide	SGD	64	20	0,8943
6	ReLU	SGD	64	20	0,9268
7	Sigmoide	RMSprop	64	20	0,9294
8	**ReLU**	**RMSprop**	**64**	**20**	**0,9374**

Tabla 4.1: Experimentos y resultados con red neuronal artificial básica en la base de datos MNIST. En negrita el mejor resultado.

Por un lado, si miramos los resultados anteriores y comparamos por las funciones de activación, observamos que la función de activación *ReLU* tiene mejores resultados que la *sigmoide*. Por otro lado, el optimizador *RMSprop* es más preciso que el *SGD*, y si a esto le reducimos el tamaño del *batch* y aumentando el número de épocas, obtenemos el mejor resultado con los datos de test con un **93,74 %** de acierto.

En la segunda parte ya se usan las CNN. Estas redes utilizan las dos arquitecturas explicadas en la Sección 4.1.2 y se han ejecutado una vez cada una. Para el entrenamiento se ha utilizado la función de coste *categorical_crossentropy*, la métrica de evaluación *accuracy*, el optimizador *RMSprop*, el tamaño del *batch* es de 64 y el número de épocas 20. Esta configuración ha sido aplicada a ambas redes. Para la primera configuración (detallada en la Figura 4.2) obtenemos una tasa de acierto en el conjunto de test del **99,16 %**, mientras que en la segunda configuración (detallada en la Figura 4.3) obtenemos una tasa de acierto inferior a la anterior, un **98,97 %**, ambos resultados están cerca, pero la primera es mejor.

En la Tabla 4.2 visualizamos los tres algoritmos usados para resolver MNIST con la tasa de aciertos obtenida en el conjunto de test.

	NN de 2 capas	CNN sencilla	CNN ampliada
Tasa de aciertos en test	93,74 %	**99,16 %**	98,97 %

Tabla 4.2: Experimentos y resultados con los modelos empleados en el conjunto de datos MNIST. En negrita el mejor resultado.

4.2. Clasificación de imágenes de perros y gatos

4.2.1. Conjunto de datos

Este conjunto se corresponde a una base de datos con imágenes de perros y gatos a color y con diferentes resoluciones, disponible en Kaggle[5]. Estas imágenes son imágenes naturales, es decir, son diferentes en relación a las poses de los animales, la iluminación, los fondos, etc. En la Figura 4.4 tenemos unos ejemplos de estas imágenes. El conjunto de datos que usamos está compuesto por 2.000 imágenes (1.000 de perros y 1.000 de gatos) de entrenamiento y 800 de validación (400 de perros y 400 de gatos), aunque el conjunto de datos original consiste en 12.500 de gatos y otras tantas de perros. No tenemos conjunto de testeo ya que nuestra finalidad es aprender a usar las CNN. El objetivo del clasificador es distinguir entre los dos tipos de animales *dog* y *cat*, así que esas son las clases a predecir. Para la resolución de este problema, se ha contado con la ayuda del artículo oficial de Keras [6].

Figura 4.4: Ejemplos de imágenes la base de datos de perros y gatos de Kaggle.

4.2.2. Arquitectura

Para esta base de datos, se han realizado dos tipos de pruebas diferentes con CNN. En ambos casos se han utilizado dos técnicas nuevas, el *data augmentation*, comentado en el apartado de regularización, y la transferencia de aprendizaje o *transfer learning*.

El *transfer learning* consiste en emplear una red preentrenada con una base de datos con más datos. Normalmente no se suele entrenar una CNN desde cero ya que es muy costoso obtener el número suficiente de imágenes para entrenarla

[5] `https://www.kaggle.com/c/dogs-vs-cats`

[6] `https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html`

correctamente. Lo habitual es usar una CNN que ya ha sido entrenada y, usarla como un extractor de características de las imágenes o para reentrenarla y adaptarla a otra base de datos.

En nuestro caso, hemos usado la red VGG16 [28] que ha sido preentrenada con la base de datos de la competición de ImageNet (compuesta por 1.000 clases y aproximadamente 1,2 millones de imágenes), como extractor de características de las imágenes y después sólo entrenamos sobre el clasificador con estas características.

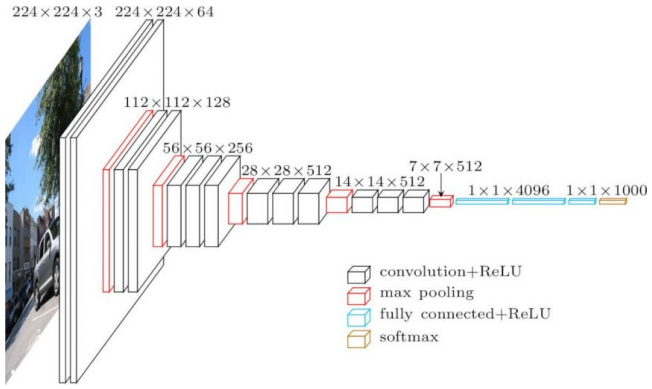

Figura 4.5: Arquitectura de la red VGG16. Imagen obtenida de `https://neurohive.io/en/popular-networks/vgg16/`

Como podemos ver en la Figura 4.5, la red tiene un tensor de 224x224x3 como entrada y la salida es un vector de longitud 1000. La parte convolucional está compuesta por 5 bloques. Cada bloque tiene 2 o 3 capas convolucionales con el filtro de tamaño 3x3, y termina con la capa de *pooling* usando *max-pooling* de tamaño 2x2. El número de filtros de cada capa es el siguiente: 64-128-256-512-512. El bloque de clasificación o *fully-connected*, clasifica las características obtenidas de los bloques de convolución en las 1000 clases diferentes. Todas las funciones de activación son *ReLU*, excepto en la última capa que usa *softmax* para obtener probabilidades.

Como hemos dicho antes, sólo vamos a usar la parte convolucional de red VGG16 para extraer las características de alto nivel de las imágenes empleando el *data augmentation*. A esta extracción de características también se la conoce como *bottleneck features*. El objetivo de este experimento es utilizar CNN entrenadas y ajustarlas a nuestros datos.

Para la primera prueba, hemos obtenido estas características una vez y se han creado diferentes modelos para la parte *fully-connected*. Básicamente la arquitectura usada es similar (ver Figura 4.6) salvo que se ha ido cambiando la capa de la regularización.

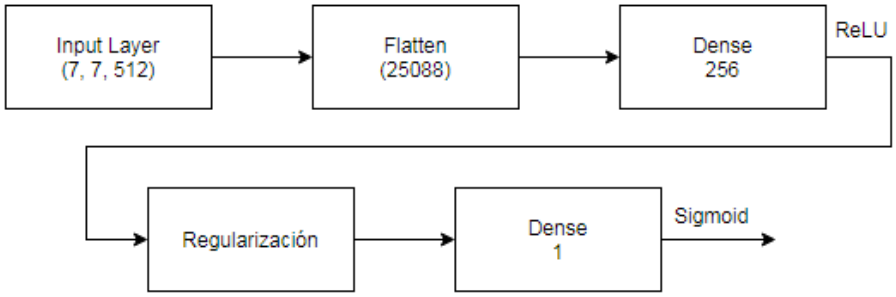

Figura 4.6: Parte *fully-connected* empleada para clasificar las características de la parte convolucional de la red VGG16 (base de datos de gatos y perros).

Para la segunda prueba, se ha realizado el ajuste de los pesos en algunas de las capas de la parte convolucional, esto es lo que se denomina como *fine-tune*. Para nuestro caso, hemos entrenado las capas del último bloque de la parte convolucional (Figura 4.7), y para la parte *fully-connected*, hemos usado la de la prueba anterior (Figura 4.6) con la regularización *dropout* de 0,5. Hemos escogido sólo el último bloque convolucional para evitar el *overfitting*, y se han empleado unas velocidades de aprendizaje lentas, diferentes por cada experimento, y reflejadas en la Tabla 4.4.

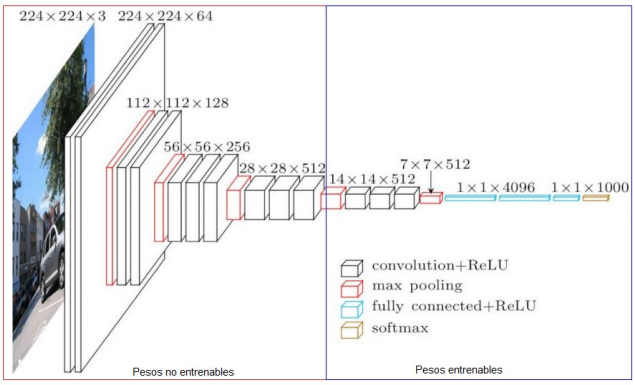

Figura 4.7: Nueva CNN basada en VGG16 dónde se actualizan los pesos del último bloque convolucional. Imagen adaptada de `https://neurohive.io/en/popular-networks/vgg16/`

4.2.3. Experimentos y Resultados

Se han realizado 5 experimentos correspondientes a la primera parte siguiendo la arquitectura reflejada en la Figura 4.6. En la Tabla 4.3 se muestra la regularización empleada en cada caso, junto con el tamaño del *batch*, las épocas y los resultados obtenidos tras el entrenamiento una vez ejecutadas todas las épocas ya que no teníamos condición de parada. Estos experimentos se han realizado

usando el optimizador *RMSprop* con *learning rate* de 0,001, la función de coste *binary_crossentropy* y el *accuracy* como métrica de evaluación.

Exp.	Regul.	Tamaño batch	Épocas	Train loss	Train acc.	Val loss	Val acc
1	Drop. = 0,5	16	50	0,0136	0,9960	1,3357	0,8700
2	L1 = 0,001	16	50	0,0771	0,9920	0,6272	0,8762
3	L2 = 0,001	16	50	0,1483	0,9700	0,6759	0,8470
4	L1 = 0,001; L2 = 0,001	16	50	0,1454	0,9845	**0,5986**	0,8387
5	**Drop. = 0,5**	**10**	**50**	**0,0133**	**0,9970**	0,9689	**0,9037**

Tabla 4.3: Experimentos y resultados con la base de datos de perros y gatos modificando los valores de la regularización, el tamaño del *batch* y el número de épocas. El optimizador es *rmsprop*, la función de coste *binary_crossentropy* y la métrica de evaluación el *accuracy*. En negrita los mejores resultados.

Mirando los resultados obtenidos, comprobamos que aplicando la regularización *dropout* y bajando el tamaño del *batch*, obtenemos una precisión del **99,70 %** en los datos de entrenamiento y un **90,37 %** en los datos de validación, por lo que este último experimento parece una buena opción.

Para la segunda parte, se han ejecutado 6 experimentos modificando el optimizador a la hora de entrenar el modelo. Recordemos que para este caso se ha utilizado la red VGG16 preentrenada pero pudiendo modificar los pesos de la última capa convolucional. En la Tabla 4.4 se refleja el optimizador empleado en cada experimento junto con los valores obtenidos durante el entrenamiento. La función de coste es la *binary_crossentropy* y la métrica *accuracy*. Para la parte de clasificación, se ha utilizado una capa densamente conectada de 256 neuronas y con la función de activación *ReLU*, seguido de la regularización *dropout* con valor 0,5, y por último otra capa densamente conectada de 1 neurona y cuya función de activación es *sigmoid*. Para todos los casos, se ha fijado el valor de las épocas a 50 y el tamaño del *batch* a 16. En todos los experimentos se han ejecutado todas las épocas porque no había condición de parada.

Exp.	Optimizador	Train loss	Train acc.	Val loss	Val acc
1	SGD lr=0,0001; momentum=0,9	0,5980	**0,9375**	0,9686	**0,9038**
2	RMSprop lr=0,001	**0,5257**	**0,9375**	0,9686	**0,9038**
3	SGD lr=0,0001; momentum=0,7	0,5729	0,9260	0,9686	**0,9038**
4	SGD lr=0,00001; momentum=0,9	0,5717	0,9305	0,9686	**0,9038**
5	SGD lr=0,00001; momentum=0,7	0,5784	0,9320	**0,9689**	**0,9038**
6	SGD lr=0,00001; momentum=0,5	0,5289	**0,9375**	**0,9689**	**0,9038**

Tabla 4.4: Experimentos y resultados con la base de datos de perros y gatos modificando los valores del optimizador. La función de coste es la *binary_crossentropy*, la métrica *accuracy*, el valor de las épocas 50 y el tamaño del *batch* a 16. En negrita los mejores resultados.

Podemos observar cómo los resultados son todos muy similares, pero parece ser que el mejor es el experimento 2 que, aún estando empatado con los experimentos 1 y 6 en los valores de la métrica (**0,9375** en entrenamiento y **0,9038** en validación), tiene unos valores de pérdida menores. En los datos de entrenamiento obtiene un **0,5257** frente a **0,5980** del experimento 1 y **0,5289** del experimento 6, y en los datos de validación obtiene un **0,9686** frente a **0,9686** del experimento 1 y **0,9689** del experimento 6.

4.3. Segmentación de núcleos con U-Net

4.3.1. Conjunto de datos

Con estos datos, ya nos introducimos a implementar una red U-Net. Pertenecen a la *2018 Data Science Bowl* de Kaggle[7]. Básicamente están compuestos por una gran cantidad de imágenes de núcleos segmentados. Contiene un conjunto de entrenamiento y otro de test. Cada imagen tiene un identificador asociado y la carpeta que contiene dicha imagen tiene el mismo identificador como nombre. El conjunto de entrenamiento está estructurado en dos carpetas, la carpeta *images* dónde se encuentran las imágenes de los núcleos, y la carpeta *masks* dónde se encuentras las máscaras segmentadas de cada núcleo. En cambio, el conjunto de test sólo contiene la carpeta *images* y no la carpeta *masks* ya que la competición sigue abierta. En la Figura 4.8 podemos visualizar un ejemplo de imagen junto con algunas de sus máscaras.

[7] `https://www.kaggle.com/c/data-science-bowl-2018`

Figura 4.8: Ejemplo de imagen de núcleos (izquierda) y sus máscaras en imágenes individuales (base de datos *2018 Data Science Bowl*).

4.3.2. Arquitectura

Como ya se ha comentado, en este experimento hemos usado una arquitectura U-Net. No vamos a entrar a detallar cómo funciona esta red, ya que ya lo hemos hecho en la Sección 3.4. El objetivo es ir aprendiendo a modelarla y ver cómo funciona. La arquitectura de este experimento varía en algunos aspectos con respecto a la original [4], se detallan a continuación:

- Se ha reducido el número de funciones por capa, comenzando en 16 y terminando en 256 (pasando por 32, 64 y 128).
- La función de activación empleada es la Unidad Lineal Exponencial, *Exponential Linear Units* en inglés (ELU) [49].
- Se ha añadido la regularización con el *dropout* de valor 0,2 para la parte de las convoluciones y los dos primeros bloques de las capas de deconvoluciones, y el valor de 0,1 para las otras dos capas de deconvoluciones. Esta regularización se coloca entre las dos capas seguidas de convolución.
- La imagen de entrada y de salida, a diferencia de la red original, tienen el mismo tamaño, 256x256.

4.3.3. Experimentos y Resultados

Se han realizado dos tipos de experimentos con la arquitectura descrita. El primero está en el tutorial de Kaggle[8]. El segundo también se encuentra en dicho tutorial pero es la implementación de un participante y mencionada en la parte de los comentarios [9]. En ambos casos se ha utilizado el 10 % de los datos de entrenamiento para realizar la validación y se ha usado el *callback EarlyStopping*[10] con el valor de paciencia a 5 para que no se ejecuten todas las épocas.

Para el primero se ha ejecutado la red empleando el optimizador *Adam* con un *learning rate* de 0,001, un tamaño de *batch* de 16, épocas a 50, la función de coste *binary_crossentropy* y una métrica de evaluación definida por el usuario, *mean_iou*. Esta métrica representa la media de los promedios de precisión

[8] `https://www.kaggle.com/keegil/keras-u-net-starter-lb-0-277`
[9] `https://github.com/kamalkraj/DATA-SCIENCE-BOWL-2018`
[10] `https://keras.io/callbacks/#earlystopping`

de las diferentes intersecciones sobre umbrales de unión (IoU) de Keras. IoU (*intersection over the union* en inglés), es una medida del solapamiento entre las máscaras originales y las predichas. En este caso se calcula su media para diferentes umbrales aplicados a las probabilidades que da la red, lo que es una práctica muy común. Esta métrica viene dada por la fórmula:

$$IoU = \frac{AS}{AU} \tag{4.3}$$

donde el área de solapamiento (AU) es el área que tienen ambas máscaras en común, mientras que el área de unión (AU) es el área de las dos máscaras juntas. En la Figura 4.9 tenemos unos ejemplos de buenas y malas intersecciones.

Figura 4.9: Ejemplo de 3 resultados de la métrica IoU: pobre (izquierda), bueno (centro) y excelente (derecha). El recuadro rojo corresponde a la clase predicha y la verde a la verdadera. Imagen obtenida de `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`

Tras la ejecución de 19 de las 50 épocas configuradas, se han obtenido los resultados presentados en la Tabla 4.5.

Train loss	Train mean IoU	Val loss	Val mean IoU.
0,0822	0,8356	0,0715	0,8367

Tabla 4.5: Resultados del entrenamiento de la primera prueba con U-Net (base de datos *2018 Data Science Bowl*). Se ha usado el optimizador *Adam* con un *learning rate* de 0,001, un tamaño de *batch* de 16, máximo número de épocas a 50, la función de coste *binary_crossentropy* y la métrica de evaluación definida por el usuario, *mean_iou*.

Para el segundo experimento hemos modificado sólo la métrica y se ha usado el 'Coeficiente de Sorensen-Dice' [50, 51]. Este coeficiente es un estadístico utilizado para comparar la similitud de dos muestras. Supongamos que VP es el número de verdaderos positivos de la segmentación, FP son los falsos positivos y FN el número de falsos negativos, entonces el coeficiente (DSC) viene representado por:

$$DSC = \frac{2VP}{2VP + FP + FN} \tag{4.4}$$

El resultado obtenido para los datos de entrenamiento y validación han sido mejores que en el caso anterior (ver Tabla 4.6).

Train loss	Train DSC	Val loss	Val DSC
0,0654	**0,8797**	**0,0666**	**0,8731**

Tabla 4.6: Resultados del entrenamiento de la segunda prueba con U-Net (base de datos *2018 Data Science Bowl*). Se ha usado el optimizador *Adam* con un *learning rate* de 0,001, un tamaño de *batch* de 16, épocas a 50, la función de coste *binary_crossentropy* y la métrica ha sido el 'Coeficiente de Sorensen-Dice'.

4.4. Segmentación de neuritas con U-Net

4.4.1. Conjunto de datos

Este conjunto de datos pertenece a la competición lanzada en la conferencia ISBI de 2012 en Barcelona. Está compuesto por un conjunto de imágenes de EM, y su objetivo es ser usado para entrenar algoritmos de aprendizaje automático y así poder segmentar automáticamente las neuritas. Estas imágenes también contienen ruido y pequeños errores de alineación.

Se compone de 30 imágenes de entrenamiento junto con las etiquetas correspondientes y otro conjunto de 30 imágenes para evaluar el modelo, en este caso no conocemos las etiquetas porque la competición sigue abierta. En concreto, se tratan de secciones de EM de transmisión del cordón del nervio ventral de la primera instancia de la larva de *Drosophila*. El microcubo mide 2x2x1,5 micras aproximadamente con una resolución de 4x4x50 nm/píxel. En la Figura 4.10 tenemos una ejemplo de una imagen de entrenamiento junto con su imagen de etiquetas.

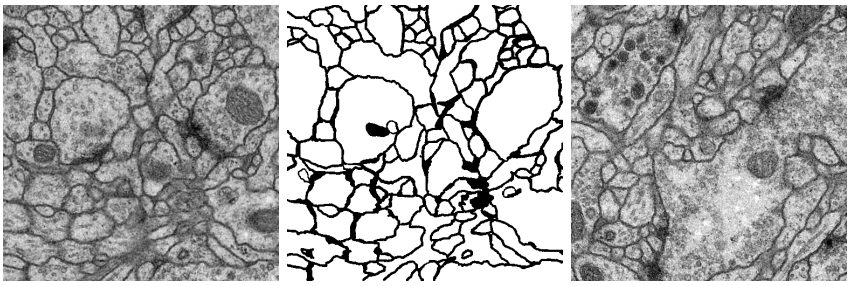

Figura 4.10: Ejemplo de un corte EM (izquierda) junto con su imagen de etiquetas (centro) y una imagen de test (derecha) del conjunto de datos de la competición del ISBI 2012.

Estas etiquetas son binarias y los píxeles de los objetos segmentados están en blanco mientras que el resto está en negro (principalmente son las membranas).

4.4.2. Métricas de evaluación

Para evaluar el modelo, y puesto que el conjunto de test no tiene las etiquetas públicas, tenemos que subir las imágenes generadas a la competición, lo cual también ayuda a compararse con el estado del arte automáticamente ya que existe un ranking con el resultado de todos los métodos participantes. Estas

imágenes tienen que estar todas juntas en un único fichero .TIFF.

La métrica principal es una variante del error de Rand llamada V^{Rand}. El error de Rand fue propuesto como una métrica en el rendimiento de segmentación [52,53] y también se ha utilizado como una función objetiva para optimizar directamente el rendimiento de los algoritmos de aprendizaje [54]. A su vez, el error de Rand es una variante del *Rand index* [55] (RI) que es un criterio objetivo para valorar los resultados que se obtienen con los métodos de agrupación (*clustering*) de datos. Aísla aspectos específicos del rendimiento de un método, como su recuperación de la estructura inherente, su sensibilidad al remuestreo y la estabilidad de sus resultados con nuevos datos. Supongamos que VP es el número de verdaderos positivos de la segmentación, FP son los falsos positivos, VN el número de verdaderos negativos y FN el número de falsos negativos, tenemos que:

$$RI = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.5}$$

La otra métrica es la puntuación de información teórica V^{Info} [56] que fue propuesta como alternativa al error de Rand. Es una variante del *Variation of Information* (VI) que fue propuesta por [57] también cómo una métrica de agrupación de datos, por [53] para segmentación de imágenes y por [58] como una función objetivo para la segmentación en aprendizaje profundo. Suponiendo que tenemos dos *cluster* c y c', sus entropías son $H(c)$ y $H(c')$ respectivamente, e $I(c, c')$ es la información mutua entre las asociaciones de variables aleatorias, podemos formular la VI de la siguiente manera:

$$VI(c, c') = H(c) + H(c') - 2I(c, c') \tag{4.6}$$

4.4.3. Arquitectura(s)

La arquitectura usada se ha basado en la red U-Net pero se han ido realizando ciertos cambios en los distintos experimentos para comprobar los efectos de estos en la red. También se ha añadido la regularización de *dropout* con valor 0,5 para prevenir el *overfitting*. Esta regularización se ha puesto al finalizar la capa de convolución y también antes de empezar la de deconvolución. La configuración base o *baseline* de la red es la siguiente:

- **Optimizador:** se ha utilizado *Adam* con el factor de aprendizaje a 0,0001.
- **Función de coste:** es la cross-entropía binaria o *binary crossentropy*.
- **Métrica de evaluación:** es la precisión o *accuracy*.
- **Función de activación:** es la función *ReLU*.
- **Pooling:** es el *max-pooling* con filtro 2x2.
- **Épocas:** 5.

- **Tamaño del *batch*:** 1.
- **Estructura del número de filtros:** 64-128-256-512-1024-512-256-128-64

Para todos los experimentos, excepto para el primero y el último, se ha utilizado un subconjunto de los datos de entrenamiento para realizar la validación, este subconjunto consiste en el 10 % de los datos totales. Puesto que sólo tenemos a disposición 30 imágenes para realizar el entrenamiento y la predicción, se ha optado para hacer una separación en 4 partes de cada imagen, de tal manera que de 30 imágenes de tamaño 512x512 se ha pasado a 120 de tamaño 256x256. Además, también se ha realizado la técnica del *data augmentation*. Tras realizar la predicción, cómo se han generado 120 imágenes, se han juntado de 4 en 4 para obtener las 30 imágenes definitivas. El *data augmentation* utilizado es el siguiente:

- Rango de grados para las rotaciones aleatorias a 0,2, *rotation_range*.
- Desplazamiento sobre el ancho a 0,05.
- Desplazamiento sobre el alto a 0,05.
- Rango para el cambio de corte a 0,05.
- Rango para el zoom aleatorio a 0,05.
- Volteo horizontal.
- Modo de relleno se ha usado el más cercano.

El modo de relleno indica cómo se tienen que rellenar los puntos fuera de los límites de la entrada tras realizar por ejemplo una rotación. Las opciones disponibles son las siguientes:

- Constante: aplica a los nuevo píxeles el mismo valor k, kkkkkkk—abcd—kkkkkkkk. Si k es 0 se rellena con negro los píxeles mientras con el valor a 1 los rellena con blanco.

Figura 4.11: Ejemplo donde se aplica el modo de relleno constante con el valor 0. Imagen obtenida de `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/image-augmentation`

- Más cercano: añade el valor del píxel vecino más cercano, aaaaaaaa—abcd—dddddddd.

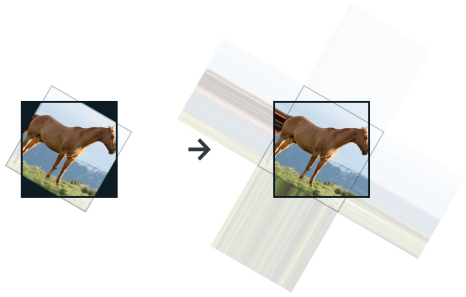

Figura 4.12: Ejemplo donde se aplica el modo de relleno más cercano. Imagen obtenida de `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/image-augmentation`

- Reflejo: aplica el efecto espejo a los nuevos píxeles, abcddcba—abcd—dcbaabcd.

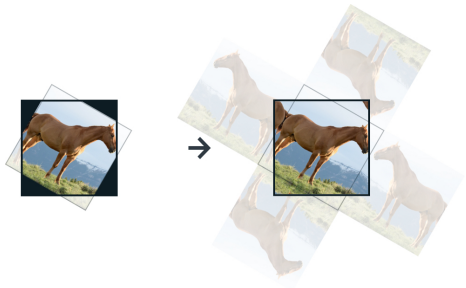

Figura 4.13: Ejemplo donde se aplica el modo de relleno reflejo. Imagen obtenida de `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/image-augmentation`

- Envoltura: la imagen original se repite en los nuevos píxeles, abcdabcd—abcd—abcdabcd.

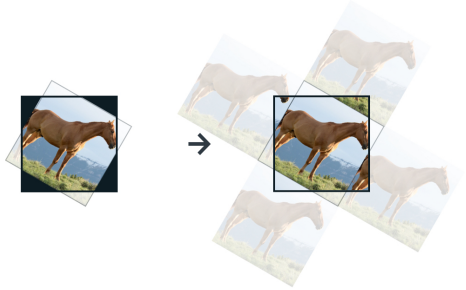

Figura 4.14: Ejemplo donde se aplica el modo de relleno envoltura. Imagen obtenida de `https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/image-augmentation`

4.4.4. Experimentos y Resultados

Se han realizado un total de 22 experimentos. Cada experimento deja todos los parámetros fijos menos uno para estudiar su efecto en los resultados. Los resultados obtenidos se detallan en la Tabla 4.8 y las siguientes explicaciones hacen referencia a dicha tabla. Estas explicaciones están agrupadas por el tipo de parámetro modificado pero puede que un experimento esté relacionado con otro que esté en otra agrupación. El orden de ejecución lo dicta el número del experimento. Todos estos experimentos se han realizado con los *notebook* en *Colab* y gracias a esto, son fácil y totalmente reproducibles en el siguiente enlace[11].

Baseline

- **Experimento 1:** este es el experimento base o *baseline* por lo que el modelo y la configuración es la definida en la Sección 4.4.3. En la Figura 4.15 se muestran las gráficas de *loss* y *accuracy* obtenidas en este experimento para el conjunto de entrenamiento. Comprobamos que la red aprende rápido el conjunto de entrenamiento ya que la precisión sube a valores altos rápidamente. El *loss* baja rápido y parece que sigue bajando, por lo que sería necesario continuar el entrenamiento.

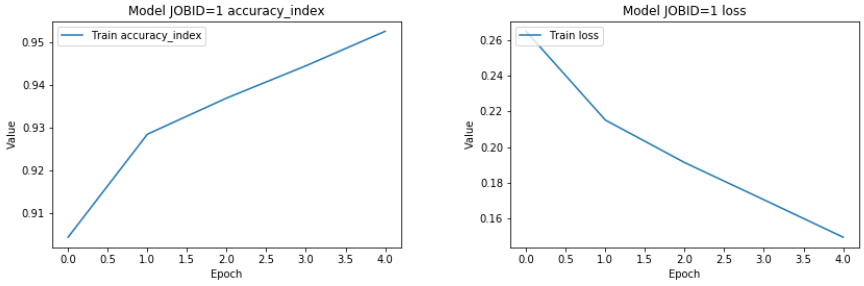

Figura 4.15: Gráficas de *accuracy* (izquierda) y *loss* (derecha) del experimento base o *baseline* en el conjunto de entrenamiento (sin validación).

Efecto del *learning rate*

- **Experimento 5:** para este caso, hemos cogido la configuración del experimento 4 con el tamaño de *batch* a 2 pero hemos reducido el factor de aprendizaje del optimizador, hemos pasado de 0,0001 a 0,00001. Hemos obtenido peores resultados con relación al anterior. En la Figura 4.16 tenemos una comparación de de las gráficas de *loss* y *accuracy* obtenidas para estos dos experimentos. Observamos como con un *learning rate* bajo, tenemos una curva de aprendizaje más lenta para el mismo número de épocas.

[11] `https://drive.google.com/drive/folders/1DtwDTt-uZuZ7ZlOghr5HfzOSZDwvYXYx?usp=sharing`

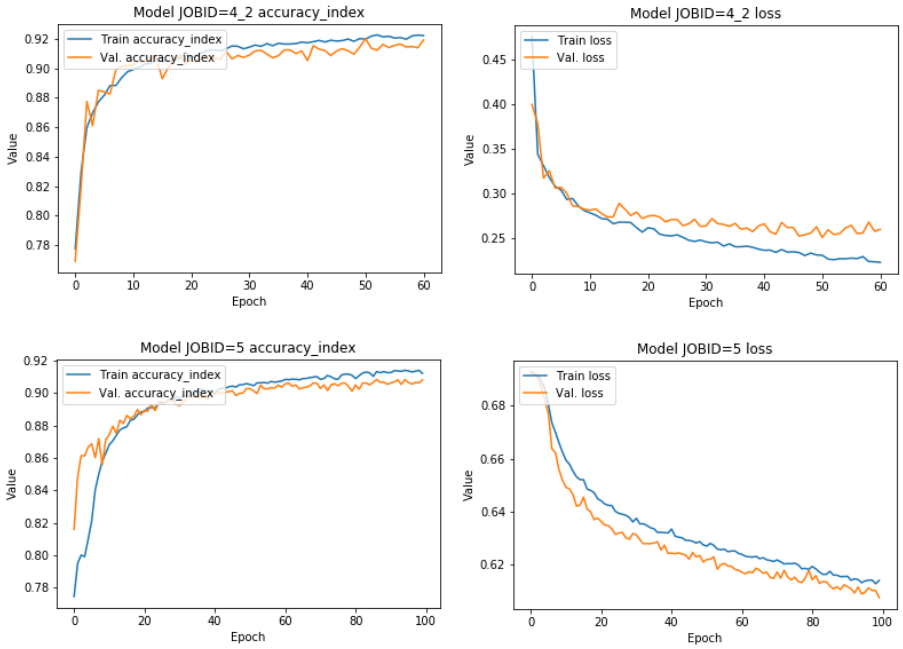

Figura 4.16: Efecto del *learning rate* en el *accuracy* (imágenes de la izquierda) y *loss* (imágenes de la derecha) de los experimentos 4 con el tamaño de *batch* a 2 y *learning rate* a 0,0001 (imágenes de arriba) y 5 con el *learning rate* a 0,00001 (imágenes de abajo).

- **Experimento 6:** este experimento nos ha dado resultados muy malos, ya que las imágenes predichas del conjunto de test han sido imágenes completamente en gris. La configuración usada ha sido la del experimento 4 con el tamaño de *batch* a 2 pero pero se ha aumentado el factor de aprendizaje del optimizador, hemos pasado de 0,0001 a 0,001. Para este resultado no hemos obtenido las puntuaciones de V^{Info} ni de V^{Rand}.

Efecto del *batch size*

- **Experimento 4:** en este experimento hemos cogido la configuración del 3 aumentando el número de épocas a 100 y ejecutando varias veces modificando el tamaño de *batch*. Se han usado los tamaños 1, 2, 4 y 6, y hemos podido comprobar que un valor bajo de *batch* (1 o 2) nos da mejores resultados que uno alto (4 o 6) (Ver Tabla 4.7).

Batch	Épocas	Train loss	Train accuracy	Val loss	Val accuracy	Rand score	Info score
1	58	**0,1709**	**0,9250**	0,2051	0,9148	0,9568	0,9813
2	61	0,2227	0,9223	0,2597	**0,9193**	**0,9642**	**0,9845**
4	53	0,1799	0,9215	**0,2014**	0,9148	0,9292	0,9790
6	52	0,1824	0,9205	0,2075	0,9124	0,9429	0,9779

Tabla 4.7: Resultados del experimento 4 con los diferentes tamaños de *batch*. En negrita los mejores resultados.

- **Experimento 9:** hemos optado por coger la configuración del experimento 8 y modificar el tamaño de *batch* a 1 y el número de épocas a 300. Observamos que se han mejorado los resultados anteriores pero el número de épocas realizadas ha subido de 61 a 107.
- **Experimento 10:** se ha vuelto a coger la configuración del experimento 8 pero cambiando sólo el número de épocas para saber si aumentaban tanto como en el experimento 9. Este no ha sido el caso ya que se ha parado en la época 49. Los resultados son similares a los del 9 pero siguen siendo peores, por lo que nos quedamos con la configuración del 9.

Efecto del data augmentation

- **Experimento 2:** para este caso, ya hemos realizado algunas modificaciones sobre el anterior. En el *data augmentation* se ha cambiado el modo de relleno a reflejar, se ha aumentado el número de épocas a 50 y se ha añadido *callback EarlyStopping* para que finalice sin tener que ejecutar todas las épocas. La puntuación de V^{Rand} ha sido ligeramente mejor que la del experimento anterior pero por el contrario V^{Info} ha sido peor.
- **Experimento 3:** se ha añadido el volteo vertical en el *data augmentation* para ver si así podríamos mejorar los datos anteriores, pero esto sólo ha sido efectivo para el caso del valor de V^{Info} ya que el V^{Rand} ha sido ligeramente inferior al 2 pero mejor que el 1.
- **Experimento 11:** en este experimento hemos optado por cambiar la configuración del *data augmentation*. Concretamente hemos aumentado el valor de rotaciones aleatorias a 0,3 y el del cambio de corte a 0,1 partiendo de la configuración del experimento 9. Los resultados obtenidos han sido muy similares a dicho experimento por lo que se ha optado por hacer más cambios en estos campos.
- **Experimento 12:** hemos vuelto a cambiar los mismos datos que los del experimento 9 pero esta vez hemos reducido a 0,1 las rotaciones aleatorias y a 0,01 el cambio de corte. En este caso los resultados han sido inferiores al anterior, por lo que se va a optar por dejar estos valores con su configuración inicial.

- **Experimento 13:** partiendo de la configuración del 9, hemos cambiado el modo de relleno al de envoltura. Los resultados empeoran, por lo que realizamos otro experimento cambiando este valor.
- **Experimento 14:** ahora hemos usado el modo de relleno al valor más cercano. En este caso si hemos mejorado los resultados en relación al 13.
- **Experimento 15:** en este caso hemos usado el último valor disponible para el modo de relleno. Este es el constante, usando para ello el valor a 0. Hemos obtenido buenos resultados por lo que vamos a seguir con esta configuración.

Efecto de la función de coste

- **Experimento 8:** para este caso, hemos optado por cambiar la función de coste y se ha escogido el error cuadrático medio o *mean squared error*. El resto de parámetros son los del experimento 4 con el tamaño de *batch* a 2. Comprobamos que el valor de la función de coste se ha visto reducido considerablemente (ver Figura 4.17), por debajo del 0,1 tanto para el conjunto de entrenamiento como para el de validación, por lo que vamos a seguir usando esta función. En la Figura 4.17 observamos como el cambio de la función de coste ha afectado sobre todo a la gráfica de *loss* y parece que esta se estabiliza bastante y de manera muy rápida para el experimento 8. También en este experimento se observa que el *loss* de validación y de entrenamiento siguen bajando juntos, mientras que en el del experimento 4 con el tamaño de *batch* a 2 parece que el validación se estanca.

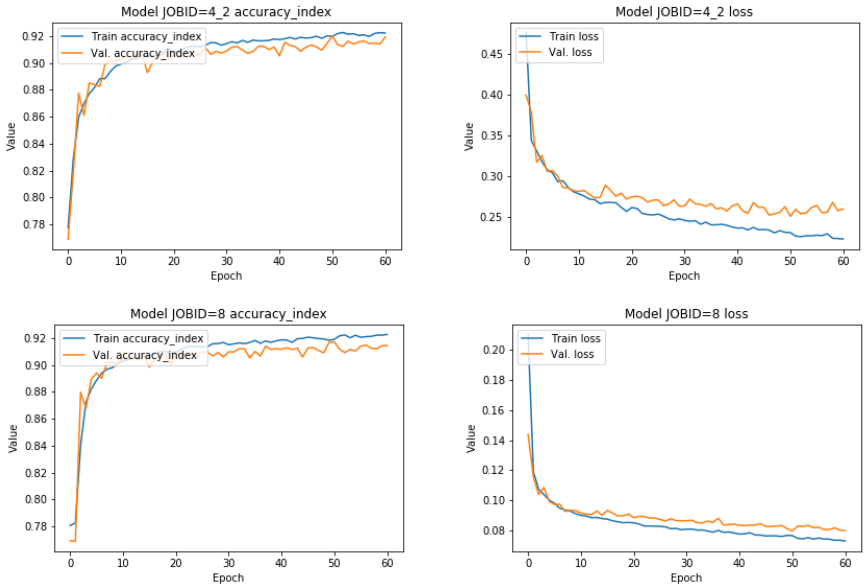

Figura 4.17: Comparación de las gráficas de *accuracy* (imágenes de la izquierda) y *loss* (imágenes de la derecha) de los experimentos 4 con el tamaño de *batch* a 2 (imágenes de arriba) y 8 (imágenes de abajo).

- **Experimento 16:** decidimos comprobar otra vez el cambio en la función de coste, volviendo a la de la configuración inicial (*binary crossentropy*). El resultado ha sido peor, por lo que desechamos dicho cambio.

Efecto de la función de activación

- **Experimento 18:** en este punto hemos decidido coger la configuración del 15 y cambiamos la función de activación. Se ha cambiado de $ReLU$ a ELU. El cambio ha hecho que empeoren los resultados drásticamente, tanto que los valores de V^{Info} y V^{Rand} se han visto reducidos por debajo del 0,9. En la Figura 4.22 podemos ver el reflejo de este resultado.

- **Experimento 20:** hemos vuelto a probar a cambiar la función de activación, en este caso hemos usado $SELU$. Observamos que los valores de V^{Info} y V^{Rand} vuelven a empeorar, más que los del experimento 18. Esto reafirmaría que la función de activación $ReLU$ da buenos resultados.

Efecto del optimizador

- **Experimento 7:** aquí hemos cambiado el optimizador *Adam* por el *SGD* y con el mismo factor de aprendizaje. El resultado ha sido similar al del 6 ya que nos ha generado unas imágenes completamente en gris y tampoco

hemos obtenido las puntuaciones de V^{Info} ni de V^{Rand}.

Efecto de la métrica de evaluación

- **Experimento 17:** volvemos a la función de coste *mean squared error* y cambiamos también la métrica de *accuracy* a *binary accuracy*. Los resultados son ligeramente peores que los del 15, por lo que no aplicamos el cambio.

Efecto del tipo de *pooling*

- **Experimento 19:** puesto que el cambio de la función de activación no ha funcionado, hemos optado por cambiar la capa de *pooling*. Se ha cambiado de *max pooling* a *average pooling*. Se ve que los resultados remontan con relación al anterior pero aún así están por debajo que usando la configuración inicial, por lo que desechamos este cambio.

Efecto del número de filtros inicial

- **Experimento 21:** en este experimento hemos optado por cambiar la configuración del número de filtros de la red. Se ha pasado de 64-128-256-512-1024-512-256-128-64 a 32-64-128-256-512-256-128-64-32. Volvemos a remontar en relación a los últimos resultados obtenidos en las puntuaciones V^{Info} y V^{Rand}.

Efecto conjunto o *ensemble*

- **Experimento 22:** por último, este experimento es diferente al resto. Hemos escogido los 6 mejores resultados obtenidos anteriormente (estos son: el experimento 9, 11, 14, 15, 17 y 21) y hemos aplicado las media a las imágenes generadas. Es una especie de solución tipo *ensemble* manual, con la que se consigue eliminar errores puntuales de algunos modelos. Para seleccionar los mejores resultados, hemos realizado la media de las puntuaciones V^{Info} y V^{Rand}, y nos hemos quedado con las 6 que mejor puntuación tenían. Si nos fijamos en los valores obtenidos, observamos que el valor de V^{Rand} es de los más bajos dentro del conjunto de los 6 mejores, pero el valor de V^{Info} es de los más altos.

ID	Épocas	Train loss	Train accuracy	Val loss	Val accuracy	Test Rand score	Test Info score
1	5	0,1475	**0,9513**	-	-	0,9136	0,9723
2	26	0,1475	0,9352	0,2172	0,9185	0,9199	0,9690
3	11	0,2209	0,9230	0,2581	0,9164	0,9247	0,9717
4_1	58	0,1709	0,9250	0,2051	0,9148	0,9568	0,9813
4_2	61	0,2227	0,9223	0,2597	**0,9193**	0,9642	0,9845
4_4	53	0,1799	0,9215	0,2014	0,9148	0,9292	0,9790
4_6	52	0,1824	0,9205	0,2075	0,9124	0,9429	0,9779
5	53	0,3459	0,9042	0,3707	0,8983	0,8881	0,9566
6	52	0,5248	0,7817	0,5408	0,7690	-	-
7	100	0,5978	0,7817	0,6098	0,7690	-	-
8	61	0,0730	0,9228	0,0800	0,9145	0,9639	0,9825
9	107	0,0542	0,9334	0,0700	0,9092	0,9716	0,9867
10	49	0,0561	0,9214	0,0622	0,9130	0,9687	0,9842
11	107	0,0541	0,9338	0,0710	0,9081	0,9698	0,9851
12	34	0,0580	0,9191	0,0620	0,9064	0,9601	0,9834
13	48	0,0622	0,9127	**0,0598**	0,9169	0,9682	0,9841
14	63	**0,0532**	0,9260	0,0640	0,9111	0,9717	**0,9875**
15	103	0,0538	0,9390	0,0702	0,9088	**0,9740**	0,9863
16	48	0,2309	0,9263	0,2513	0,9132	0,9679	0,9846
17*	82	0,0622	0,9323	0,0694	0,9144	0,9702	0,9855
18	31	0,0577	0,9250	0,0678	0,9096	0,8040	0,8820
19	34	0,0540	0,9243	0,0640	0,9107	0,9406	0,9767
20	34	0,0588	0,9185	0,0672	0,9053	0,7306	0,8612
21	82	0,0639	0,9296	0,0698	0,9140	0,9680	0,9853
22	-	-	-	-	-	0,9696	0,9872

Tabla 4.8: Experimentos y resultados de la segmentación de neuritas con U-Net. En negrita el mejor resultado. El experimento marcado con *, indica que para ese caso la métrica usada fue *binary accuracy*.

Aunque el experimento 15 se ha considerado como el mejor resultado ya que ha obtenido el valor más alto de Rand score en test, en la Figura 4.18 podemos observar cómo se aprecian algunos errores en la segmentación. En ese sentido, parece que queda margen para la mejora. En la Figura 4.19 se muestran las gráficas de *loss* y *accuracy* obtenidas para este experimento. Observamos cómo el valor de *loss* del entrenamiento parece que sigue bajando por lo que podría indicar que necesite más entrenamiento. Además el *loss* y el *accuracy* de validación se empiezan a estancar.

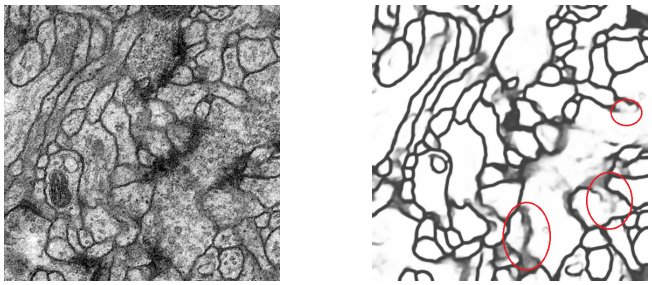

Figura 4.18: Ejemplo de una imagen de test (izquierda) junto con su segmentación (derecha) tras aplicar el mejor modelo (experimento 15). Se aprecian errores como que no se cierran bien las membranas o no están bien definidos los bordes (resaltados con elipses rojas).

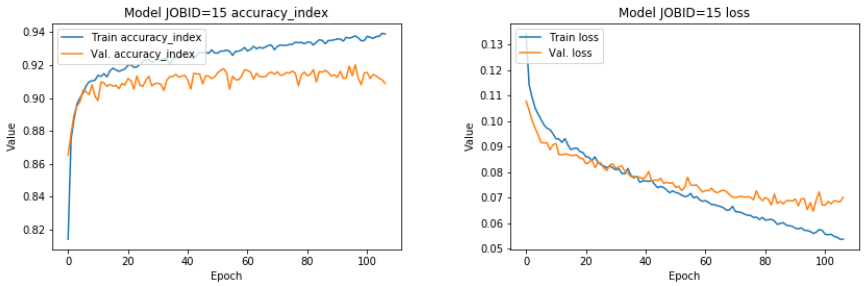

Figura 4.19: Gráficas de *accuracy* (izquierda) y *loss* (derecha) del experimento 15 considerado como el mejor.

A continuación se muestran una serie de gráficas con los valores de *loss*, la métrica de evaluación o *metric*, y las puntuaciones V^{Info} y V^{Rand} de la Tabla 4.8, que describen de una manera visual nuestra búsqueda de los mejores parámetros del modelo.

En la Figura 4.20 podemos observar como los valores de pérdida son bastante altos al principio (el objetivo es minimizar estos datos), alcanzando un pico máximo en el experimento 7. Este es uno de los experimentos que nos dieron como resultado unas imágenes completamente de color gris. Hasta que no aplicamos el cambio de la función de coste a la *mean squared error* en el experimento 8, estos valores no se redujeron. Si recordamos, en el experimento 16 se volvió a cambiar la función de coste a la *binary crossentropy* y esto se refleja cómo un máximo local en la gráfica.

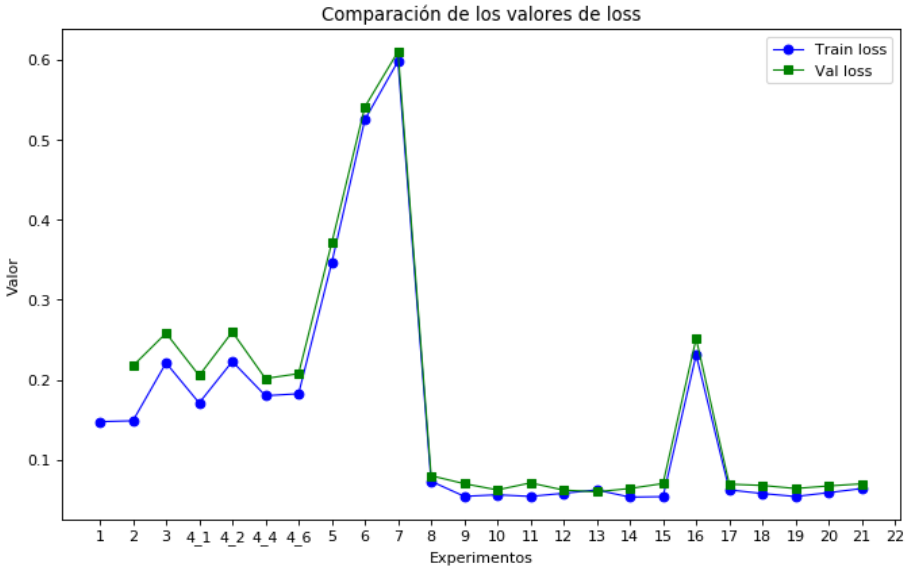

Figura 4.20: Gráfica con los valores de *loss* obtenidos por cada experimento, tanto los de entrenamiento (azul) como los de evaluación (verde).

En la Figura 4.21 visualizamos que la mayoría de los valores de la métrica de evaluación son similares excepto en los experimento 6 y 7 (son los que nos dieron como resultado unas imágenes completamente de color gris). En el experimento 15 se produce un máximo local debido al cambio del modo de relleno del *data augmentation*.

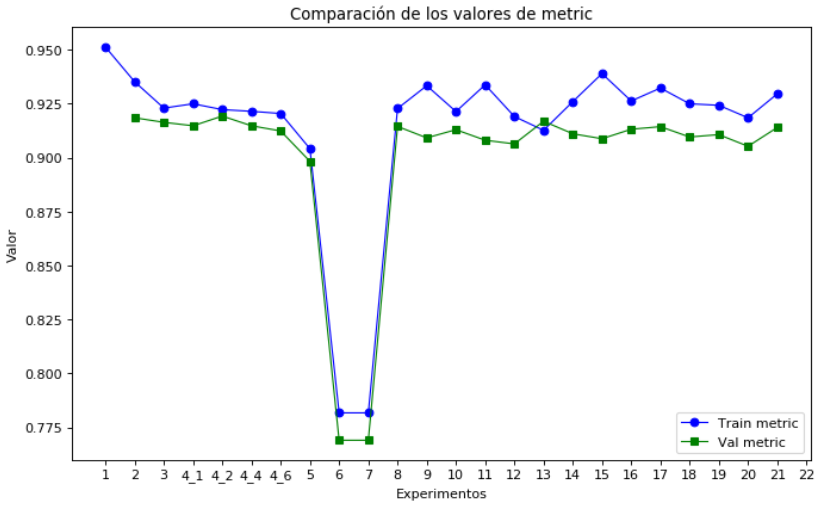

Figura 4.21: Gráfica con los valores de *metric* obtenidos por cada experimento, tanto los de entrenamiento (azul) como los de evaluación (verde). El valor de *metric* para todos los experimentos es de *accuracy*, excepto el del experimento 17 que ha usado *binary_accuracy*.

Por último, En la Figura 4.22 vemos que los parámetros de la evaluación se mantienen altos tras cambiar la función de coste a la *mean squared error* en el experimento 8. También observamos los picos mínimos de los experimentos 18 y 20, que son los que hicimos el cambio de la función de activación. Para finalizar, hay un vacío en los experimentos 6 y 7 que son los de las imágenes completamente grises y no fueron enviadas para su evaluación por el sistema de la competición.

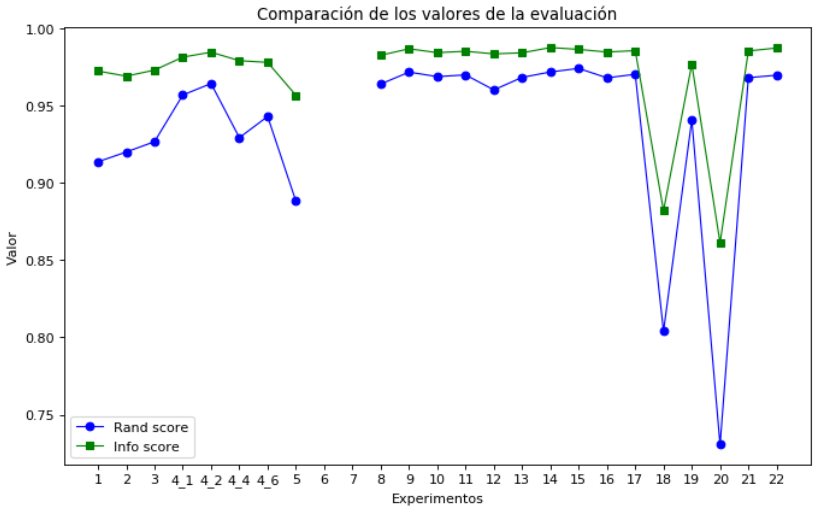

Figura 4.22: Gráfica con los valores de los parámetros V^{Info} (verde) y V^{Rand} (azul) de cada experimento.

4.4.5. Post-procesamiento

Tras la ejecución de los primeros experimentos, se observa que, al separar la imagen en 4 trozos para así tener más datos y luego juntar los resultados, se produce un efecto de borde. Por ello se opta por aplicar una técnica de post-procesamiento. Esta técnica consiste en coger cada subimagen y pasarla más veces por la red tras aplicarle transformaciones simples (rotaciones de múltiplos de 90 grados y/o volteos), aplicar la transformación inversa al resultado y sacar la media[12]. En la Figura 4.23 podemos observar una comparación de dos imágenes de salida, una con el efecto de los bordes y la otra no. Esta técnica se aplicó a partir del experimento 4.

[12] `https://github.com/Vooban/Smoothly-Blend-Image-Patches`

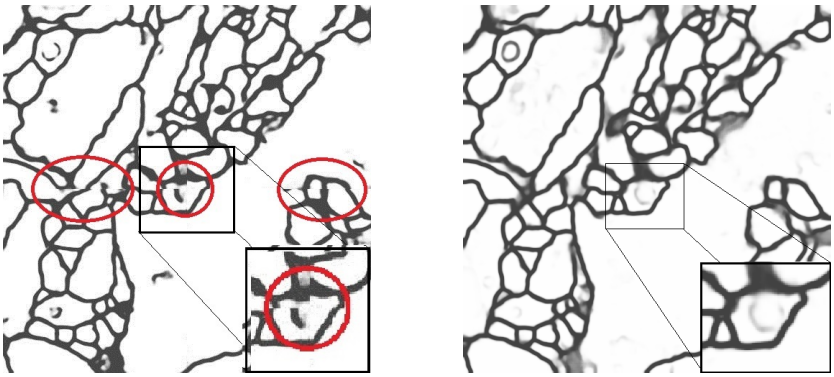

Figura 4.23: Corrección del efecto borde: se muestran 2 imágenes de salida, la de la izquierda no tiene técnica de post-procesamiento y pertenece al experimento base, y a la de la derecha se le aplica la técnica de post-procesamiento y pertenece al experimento 15. En rojo se marcan los errores más prominentes.

5. Conclusiones y trabajo futuro

En este trabajo se ha abordado el tema de la segmentación de imágenes biomédicas utilizando redes convolucionales, concretamente partiendo de la arquitectura U-Net que es la base de mayoría de trabajos del estado del arte. En primer lugar se ha explicado la teoría de las redes neuronales, cuáles son los elementos que la componen y cómo estas aprenden a través del entrenamiento. Después, nos hemos adentrado en un tipo de redes neuronales, concretamente las redes neuronales convolucionales. La principal finalidad de la creación de estas redes fue la de poder trabajar con imágenes ya que permiten codificar ciertas propiedades de la imagen en la arquitectura. Tras esto, se explicó la arquitectura de una red totalmente convolucional, concretamente la red U-Net, para solventar los problemas de la segmentación en imágenes biomédicas ya que predicen la clase de cada píxel.

Los experimentos que se han realizado han sido varios. Comenzamos con un ejemplo sencillo para adentrarnos en el funcionamiento de las redes neuronales, concretamente fue la clasificación de dígitos escritos a mano de la base de datos de MNIST. Tras esto, se realizaron pruebas con una CNN previamente entrenada para adentrarnos en los conceptos de *data augmentation* y de *transfer learning*, y en el funcionamiento de este tipo de redes. La arquitectura de red seleccionada fue la VGG16, previamente entrenada con la base de datos de ImageNet, y los datos utilizados fueron un conjunto de imágenes de perros y gatos. Después nos adentramos de manera superficial en la red U-Net para realizar la segmentación de núcleos celulares con la base de datos de la *2018 Data Science Bowl*. Finalmente, utilizamos las imágenes proporcionadas por la competición lanzada en la conferencia ISBI de 2012 para realizar la segmentación automática de procesos neuronales en imágenes de EM. En un estudio más detallado, ejecutamos diferentes experimentos con una arquitectura tipo U-Net y observamos:

- Un tamaño de *batch* bajo ofrece mejores resultados que uno alto.
- La mejor función de activación para estos casos es la función $ReLU$.
- Cambiar una función de coste por otra puede mejorar bastante el resultado de la red. En nuestro caso fue cambiar de *binary crossentropy* por *mean squared error*.
- Una buena configuración del *data augmentation* nos ofrece buenos resultados.

El mejor resultado para la métrica de evaluación V^{Rand} fue de **0.9740** y de la métrica V^{Info} fue de **0.9875**. Si nos fijamos en los resultados obtenidos por otros participantes[13] y, teniendo en cuenta la cantidad de experimentos realizados y el post-procesamiento básico empleado, se ha alcanzado una posición

[13] `http://brainiac2.mit.edu/isbi_challenge/leaders-board-new`

bastante aceptable.

En el futuro se podría seguir experimentando con la red U-Net, utilizada en este proyecto, ya que predomina en varias aplicaciones de segmentación de imágenes biomédicas y alcanzan un rendimiento prometedor. Esto en gran parte es debido a sus arquitecturas elegantes, por ejemplo, contracciones simétricas y caminos expansivos, así como conexiones de salto laterales. También se podría usar su variante para segmentaciones volumétricas la red U-Net 3D [59], ya que la segmentación de una sección tiene que estar relacionada necesariamente con las secciones vecinas. Se pueden realizar cambios en los caminos que usa o en el *data augmentation*. Además también se pueden utilizar otro tipo de redes como la ACE-Net [3] o FusionNet [2] y empleando una técnica de post-procesamiento más elaborada ya que la utilizada en este proyecto ha sido muy básica.

Referencias

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[2] Tran Minh Quan, David G. C. Hildebrand, and Won-Ki Jeong. Fusionnet: A deep fully residual convolutional neural network for image segmentation in connectomics, 2016.

[3] Yanhao Zhu, Zhineng Chen, Shuai Zhao, Hongtao Xie, Wenming Guo, and Yongdong Zhang. Ace-net: Biomedical image segmentation with augmented contracting and expansive paths, 2019.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[5] Jennifer N Bourne and Kristen M Harris. Nanoscale analysis of structural synaptic plasticity. *Current Opinion in Neurobiology*, 22(3):372 – 382, 2012. Synaptic structure and function.

[6] Southgate E. Thomson J. N. White, J. G. and S. Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos. Trans. R. Soc. Lond. B Biol. Sci*, 314:1 – 340, 1986.

[7] Travis A. Jarrell, Yi Wang, Adam E. Bloniarz, Christopher A. Brittin, Meng Xu, J. Nichol Thomson, Donna G. Albertson, David H. Hall, and Scott W. Emmons. The connectome of a decision-making neural network. *Science*, 337(6093):437–444, 2012.

[8] Christian Rödelsperger Daniel J Bumbarger, Metta Riebesell and Ralf J Sommer. System-wide rewiring underlies behavioral differences in predatory and bacterial-feeding nematodes. *Cell*, 152(1):109–119, Jan 2013.

[9] Zhiyuan Lu Aljoscha Nern Shiv Vitaladevuni Patricia K. Rivlin William T. Katz Donald J. Olbris Stephen M. Plaza Philip Winston Ting Zhao Jane Anne Horne Richard D. Fetter Satoko Takemura Katerina Blazek Lei-Ann Chang Omotara Ogundeyi Mathew A. Saunders Victor Shapiro Christopher Sigmund Gerald M. Rubin Louis K. Scheffer Ian A. Meinertzhagen Dmitri B. Chklovskii Shin-ya Takemura, Arjun Bharioke. A visual motion detection circuit suggested by drosophila connectomics. *Nature*, 500:175 EP –, Aug 2013. Article.

[10] Daniel Raimund Berger Richard Lee Schalek José Angel Conchello Seymour Knowles-Barley Dongil Lee Amelio Vázquez-Reina Verena Kaynig Thouis Raymond Jones Mike Roberts Josh Lyskowski Morgan Juan Carlos Tapia H. ?Sebastian Seung-William Gray Roncal Joshua Tzvi Vogelstein Randal Burns Daniel Lewis Sussman Carey Eldin Priebe Hanspeter Pfister Narayanan Kasthuri, Kenneth Jeffrey Hayworth and Jeff William Lichtman.

Saturated reconstruction of a volume of neocortex. *Cell*, 162(3):648–661, Jul 2015.

[11] Winfried Denk Kevin L. Briggman, Moritz Helmstaedter. Wiring specificity in the direction-selectivity circuit of the retina. *Nature*, 471:183 EP –, Mar 2011. Article.

[12] Narayanan Kasthuri Kenneth J Hayworth Richard Schalek Daniel R Berger-Cristina Guatimosim H. Sebastian Seung-Jeff W Lichtman Juan C Tapia, John D Wylie. Pervasive synaptic branch removal in the mammalian neuromuscular system at birth. *Neuron*, 74(5):816–829, Jun 2012.

[13] Srinivas C. Turaga Viren Jain H. Sebastian Seung Winfried Denk Moritz Helmstaedter, Kevin L. Briggman. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500:168 EP –, Aug 2013. Article.

[14] Aleksandar Zlateski Kisuk Lee Mark Richardson Srinivas C. Turaga Michael Purcaro Matthew Balkam Amy Robinson-Bardia F. Behabadi Michael Campos Winfried Denk H. Sebastian Seung the EyeWirers Jinseop S. Kim, Matthew J. Greene. Space-time wiring specificity supports direction selectivity in the retina. *Nature*, 509:331 EP –, May 2014. Article.

[15] Viren Jain, H Sebastian Seung, and Srinivas C Turaga. Machines that learn to segment images: a crucial technology for connectomics. *Current Opinion in Neurobiology*, 20(5):653 – 666, 2010. Neuronal and glial cell biology – New technologies.

[16] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, Apr 2015.

[17] Ignacio Arganda-Carreras, Srinivas C. Turaga, Daniel R. Berger, Dan Cireşan, Alessandro Giusti, Luca M. Gambardella, Jürgen Schmidhuber, Dmitry Laptev, Sarvesh Dwivedi, Joachim M. Buhmann, Ting Liu, Mojtaba Seyedhosseini, Tolga Tasdizen, Lee Kamentsky, Radim Burget, Vaclav Uher, Xiao Tan, Changming Sun, Tuan D. Pham, Erhan Bas, Mustafa G. Uzunbas, Albert Cardona, Johannes Schindelin, and H. Sebastian Seung. Crowdsourcing the creation of image segmentation algorithms for connectomics. *Frontiers in Neuroanatomy*, 9:142, 2015.

[18] Dmitry Laptev, Alexander Vezhnevets, Sarvesh Dwivedi, and Joachim Buhmann. Anisotropic sstem image segmentation using dense correspondence across sections. volume 15, pages 323–30, 10 2012.

[19] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33:978–994, 01 2011.

[20] Mojtaba Seyedhosseini, Ritwik Kumar, Elizabeth Jurrus, Rick Giuly, Mark Ellisman, Hanspeter Pfister, and Tolga Tasdizen. Detection of neuron membranes in electron microscopy images using multi-scale context and radon-like features. volume 14, pages 670–7, 09 2011.

[21] Ting Liu, Elizabeth Jurrus, Mojtaba Seyedhosseini, Mark Ellisman, and Tolga Tasdizen. Watershed merge tree classification for electron microscopy image segmentation. volume 2013, pages 133–137, 11 2012.

[22] Lee Kamentsky, Thouis Jones, Adam Fraser, Mark-Anthony Bray, David Logan, Katherine Madden, Vebjorn Ljosa, Curtis Rueden, Kevin Eliceiri, and Anne Carpenter. Improved structure, function and compatibility for cellprofiler: Modular high-throughput image analysis software. *Bioinformatics (Oxford, England)*, 27:1179–80, 02 2011.

[23] Vaclav Uher and Radim Burget. Automatic 3d segmentation of human brain images using data-mining techniques. pages 578–580, 07 2012.

[24] Sun C Tan X and Pham TD. Membrane extraction using two-step classifier and post-processing. 2012.

[25] Bas E and Uzunbas MG. Contextual grouping in a concept: a multistage decision strategy for em segmentation. 2012.

[26] Dan Ciresan, Alessandro Giusti, Luca M. Gambardella, and Jürgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2843–2851. Curran Associates, Inc., 2012.

[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[28] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[29] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2015.

[30] Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury, and Chris Pal. The importance of skip connections in biomedical image segmentation. *Lecture Notes in Computer Science*, page 179–187, 2016.

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[32] Michal Drozdzal, Gabriel Chartrand, Eugene Vorontsov, Mahsa Shakeri, Lisa Di Jorio, An Tang, Adriana Romero, Yoshua Bengio, Chris Pal, and Samuel Kadoury. Learning normalized inputs for iterative estimation in medical image segmentation. *Medical Image Analysis*, 44:1–13, Feb 2018.

[33] Thorsten Beier, Constantin Pape, Nasim Rahaman, Timo Prange, Stuart Berg, Davi D. Bock, Albert Cardona, Graham W. Knott, Stephen M. Plaza, Louis K. Scheffer, Ullrich Koethe, Anna Kreshuk, and Fred A. Hamprecht. Multicut brings automated neurite segmentation closer to human performance. *Nature Methods*, 14(2):101–102, 2017.

[34] Beier, t. et al. ieee conf. comp. vision patt. recog., 3507-3516 (2015).

[35] Wei Shen, Bin Wang, Yuan Jiang, Yan Wang, and Alan Yuille. Multi-stage multi-recursive-input fully convolutional networks for neuronal boundary detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[36] Aprendizaje de filtros orientables para cnn equivalentes de rotación. In *La Conferencia IEEE sobre visión por computadora y reconocimiento de patrones (CVPR)*.

[37] C. Xiao, J. Liu, X. Chen, H. Han, C. Shu, and Q. Xie. Deep contextual residual network for electron microscopy image segmentation in connectomics. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 378–381, April 2018.

[38] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[39] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957.

[40] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.

[41] Rmsprop. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

[42] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[43] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[45] Dropout: una forma simple de evitar que las redes neuronales se sobreajusten. *Journal of Machine Learning Research*.

[46] Keras. `https://keras.io`.

[47] Google colaboratory. `https://colab.research.google.com/notebooks/welcome.ipynb`.

[48] The mnist database of handwritten digits. `http://yann.lecun.com/exdb/mnist`.

[49] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[50] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[51] T.J. Sørensen. *A Method of Establishing Groups of Equal Amplitude in Plant Sociology Based on Similarity of Species Content and Its Application to Analyses of the Vegetation on Danish Commons*. Biologiske skrifter. I kommission hos E. Munksgaard, 1948.

[52] R. Unnikrishnan, C. Pantofaru, and M. Hebert. Toward objective evaluation of image segmentation algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):929–944, June 2007.

[53] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(5):898–916, May 2011.

[54] Maximin affinity learning of image segmentation. In *Avances en los sistemas de procesamiento de información neuronal*.

[55] William M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850, 1971.

[56] Juan Nunez-Iglesias, Ryan Kennedy, Toufiq Parag, Jianbo Shi, and Dmitri B Chklovskii. Machine learning of hierarchical clustering to segment 2d and 3d images. *PloS one*, 8(8):e71715, 2013.

[57] Marina Meilă. Comparing clusterings: an axiomatic view. In *In ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 577–584. ACM Press, 2005.

[58] Thorben Kroeger, Shawn Mikula, Winfried Denk, Ullrich Koethe, and Fred A. Hamprecht. Learning to segment neurons with non-local quality measures. In Kensaku Mori, Ichiro Sakuma, Yoshinobu Sato, Christian Barillot, and Nassir Navab, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2013*, pages 419–427, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[59] Özgün Çiçek, Ahmed Abdulkadir, Soeren S. Lienkamp, Thomas Brox, and Olaf Ronneberger. 3d u-net: Learning dense volumetric segmentation from sparse annotation. *Lecture Notes in Computer Science*, page 424–432, 2016.