Universidad Zaragoza
1542

Trabajo Fin de Máster

DEEP LEARNING PARA SEGMENTACIÓN CELULAR DE IMAGEN

DEEP LEARNING FOR CELLULAR IMAGE SEGMENTATION

Martin Teodoro Hörndler Gil

Director/es

Rubén Martínez-Cantín

Escuela de Ingeniería y Arquitectura de Zaragoza (EINA)

Año
2021

ÍNDICE

RESUMEN

Título: **Deep Learning para segmentación celular de imagen**

El estudio de la migración celular in vitro es fundamental para entender muchos procesos biológicos importantes, para ello, es necesario conocer e identificar la posición de cada célula en cientos de imágenes de microscopía. Por lo que la realización del trabajo se ha centrado en aplicar técnicas de *deep learning* a un conjunto de imágenes celulares con el objetivo de realizar la segmentación de forma automática. La utilización de las redes convolucionales mediante una arquitectura conocida como U-net, han permitido una notoria mejoría en el campo de la segmentación.

Mediante la utilización de esta arquitectura de red convolucional para un *dataset* cedido por el Grupo M2BE de la Universidad de Zaragoza, se ha entrenado la red para obtener un modelo que desempeñe de forma óptima la segmentación automática.

Una vez realizados los entrenamientos con las imágenes y máscaras se han podido obtener resultados satisfactorios, el modelo entrenado realiza la segmentación de imágenes que era el objetivo central del trabajo. El desempeño del modelo entrenado a través de la red ha sido evaluado a través de una métrica que permita conocer cómo se está realizando la segmentación. Aunque los resultados iniciales con los primeros entrenamientos son óptimos, se ha buscado la mejoría a través de entrenamientos con parámetros modificados. La tasa de aprendizaje de la red, así como las veces que la red procesa todas las imágenes y máscaras (*epochs*), han sido ajustadas, con el objetivo de obtener el mejor aprendizaje del modelo.

Un hándicap de trabajar con imágenes de la tipología utilizada es que no se dispone de gran cantidad de estas. Por lo que en el trabajo inicialmente se han alterado las características espaciales de las mismas para ampliar el *dataset* del entrenamiento, para posteriormente aplicar *data augmentation* totalmente implementado en el código de la red, lo que permite incorporar la variación de las características de las imágenes de forma aleatoria. Estos cambios han repercutido en obtener los mejores resultados de todo el trabajo.

Para verificar la generalización del modelo entrenado, se ha realizado la segmentación de imágenes desconocidas para el modelo entrenado, obteniéndose buenos resultados en la detección de bordes y contornos. Posteriormente se ha verificado que entrenar a la red con este nuevo conjunto de datos también permite obtener buenos resultados en la segmentación de ese tipo de imágenes.

Aunque el desempeño del modelo es bueno, no se consigue una segmentación perfecta, pero es cierto que mejora con creces la segmentación manual. El campo del *deep learning* se encuentra en continua innovación, y hay técnicas que se podrían incorporar a la metodología del trabajo de forma adicional. Los resultados obtenidos son los que a priori se habían marcado como objetivo, pero hay que tener en cuenta la capacidad de mejora en el desempeño del modelo con las técnicas actuales, y las que podrían llegar en un futuro cercano.

ABSTRACT

Title:**Deep learning for cell image segmentation**

The study of cell migration in vitro is essential to understand many important biological processes, for which it is necessary to know and identify the position of each cell in hundreds of microscopy images. Therefore, this work has focused on applying deep learning techniques to a set of cell images in order to perform segmentation automatically. The use of convolutional networks through an architecture known as U-net, has allowed a notorious improvement in the field of segmentation.

By using this convolutional network architecture for a dataset provided by the M2BE Group of the University of Zaragoza, the network has been trained to obtain a model that performs automatic segmentation in an optimal way.

Once the training with images and masks has been carried out, satisfactory results have been obtained, the trained model performs image segmentation, which was the main objective of the work.The performance of the trained model through the network has been evaluated by means of a metric that allows us to know how the segmentation is being carried out. Although the initial results with the first trainings are optimal, improvement has been sought through training with modified parameters. The learning rate of the network, as well as the number of times the network processes all the images and masks (epochs), have been adjusted, with the aim of obtaining the best learning of the model.

One handicap of working with images of the type used is that there are not a large number of images available. For this reason, the work has initially altered the spatial characteristics of the images to enlarge the training dataset, and then applied data augmentation fully implemented in the network code, which allows the variation of the characteristics of the images to be incorporated randomly. These changes have resulted in the best results of all the work.

To verify the generalisation of the trained model, the segmentation of unknown images for the trained model has been carried out, obtaining good results in edge and contour detection. Subsequently, it has been verified that training the network with this new dataset also allows obtaining good results in the segmentation of this type of images.

Although the performance of the model is good, perfect segmentation is not achieved, but it certainly improves on manual segmentation by far. The field of deep learning is in continuous innovation, and there are techniques that could be incorporated into the methodology of the work in an additional way. The results obtained are those that had been set as an objective a priori, but we must take into account the capacity for improvement in the performance of the model with the current techniques, and those that could come in the near future.

1.INTRODUCCIÓN

La migración celular es un evento fundamental en una amplia variedad de procesos fisiológicos, que abarca desde la embriogénesis, angiogénesis, osteogénesis, respuesta inflamatoria, respuesta inmune y cicatrización de heridas, para desarrollar enfermedades como el cáncer y la formación de metástasis.

Los experimentos in vitro se han vuelto cada vez más sofisticados con el fin de reproducir lo más preciso posible el entorno biológico natural de los organismos a partir de estudios in vitro. A medida que los estudios in vitro han aumentado su sofisticación, sus requisitos también han crecido en complejidad. Debido a la complejidad y el costoso trabajo de laboratorio de los experimentos in vitro, los estudios *in silico* tienen un papel complementario en la comprensión de la migración de células mesenquimales. Los modelos matemáticos basados en computadoras permiten realizar un gran número de experimentos controlados y reproducibles con costes asociados mucho más bajos [1].

El proceso de segmentación es de vital importancia para estudiar migraciones celulares, tiene el objetivo de dividir la imagen en diferentes segmentos otorgándoles una clasificación por pixeles. Con estos resultados se pueden realizar estudios de forma automática de las migraciones mencionadas. En el caso de las imágenes celulares lo interesante es discernir entre lo que es célula y lo que no lo es. De esta manera es posible determinar los contornos de las células, así como variaciones en las mismas.

La motivación de este trabajo es realizar la segmentación de imágenes celulares de forma automática como punto de inicio para poder estudiar los casos de migración celular y de esta manera estudiar los mecanismos que gobiernan este evento de gran importancia en el ámbito de la salud.

La segmentación de imágenes es uno de los problemas que con más frecuencia aborda el campo de la visión artificial englobado dentro del *deep learning* (aprendizaje profundo). Aunque las técnicas de *deep learning* son relativamente actuales, el problema de la segmentación de imágenes celulares se viene abordando desde hace tiempo.

El *deep learning* es un subconjunto del *machine learning*, que por otro lado es un subconjunto de la inteligencia artificial. Inteligencia Artificial es un término general que se refiere a las técnicas que permiten a las computadoras imitar el comportamiento humano. *Machine Learning* representa un conjunto de algoritmos entrenados con datos que hacen todo esto posible.

Los algoritmos de aprendizaje profundo intentan sacar conclusiones similares a las de los humanos analizando continuamente los datos con una estructura lógica dada. Para lograr esto, el aprendizaje profundo utiliza una estructura de múltiples capas de algoritmos llamados redes neuronales [2] (Figura 1).

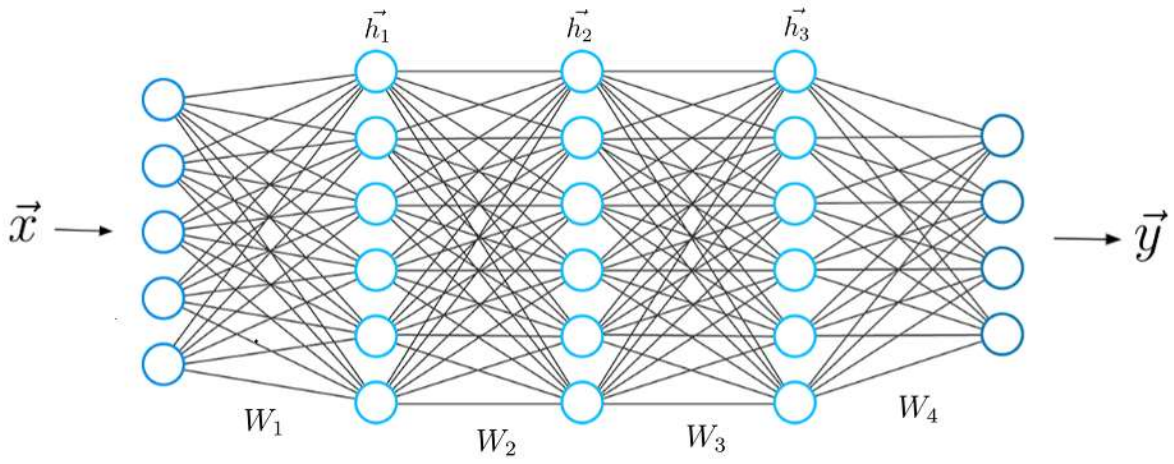


Figura 1. Red neuronal

Hasta la llegada de las redes convolucionales que son un tipo de red neuronal, la segmentación de imágenes se realizaba con otras técnicas, entre ellas manualmente. Por lo que la implementación de las redes convolucionales ha permitido una automatización de este proceso, pudiendo predecir la red entrenada la segmentación de la imagen de entrada (Figura 2 y 3).

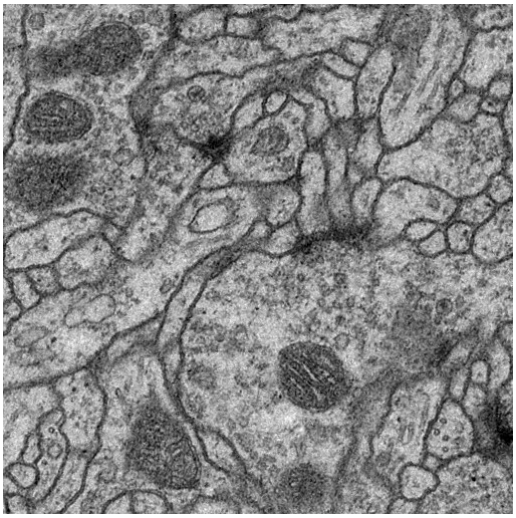


Figura 2. Imagen de entrada a la red

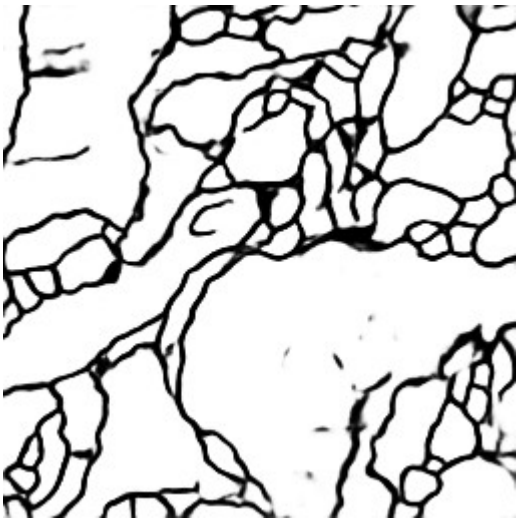


Figura 3. Imagen segmentada de forma automática por la red entrenada

2. OBJETIVOS Y ALCANCE

Este trabajo tiene como **objetivo principal** la segmentación de células en imágenes tomográficas usando técnicas de *deep learning,* utilizando como núcleo del trabajo la red convolucional U-net.

Dada la dificultad de obtener imágenes celulares se utilizarán técnicas de aumento de datos para generar mayor variabilidad en el entrenamiento del modelo.

Para verificar el grado de cumplimiento de los objetivos marcados se realizará una evaluación del desempeño del modelo respecto a la segmentación, comparando diferentes configuraciones del

modelo de aprendizaje y observando la evolución de la métrica a evaluar. El objetivo es alcanzar un modelo con la mejor métrica y que realice la segmentación lo mejor posible.
Una vez entrenado el modelo con las imágenes que se disponen, se debe verificar si este realiza una buena generalización con imágenes celulares con las que no ha sido entrenado.

3. CONCEPTOS TEÓRICOS DEL *DEEP LEARNING*

Como se ha comentado en la introducción, el *deep learning* es un subconjunto del *machine learning.* Las redes neuronales utilizadas en el *deep learning* (Figura 4) identifican patrones y clasifican diferentes tipos de información. Las diferentes capas de las redes neuronales sirven como filtro, yendo desde los elementos más generales a los más sutiles, aumentando la probabilidad de detectar y generar un resultado correcto [3].

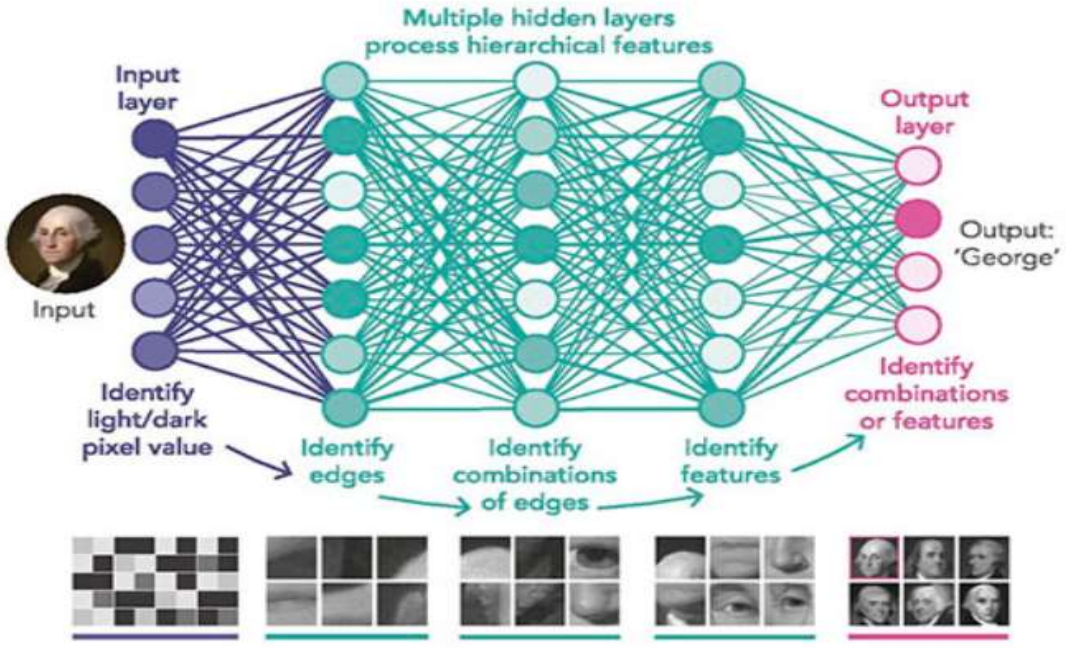


Figura 4. Aprendizaje de la red Neuronal, recuperado de: https://www.futurespace.es/

Entre los avances más reseñables dentro del campo del *deep learning* están: la traducción de imágenes con texto, reconocimiento de objetos y facial, la personalización enfocada a la venta y mejora de la eficacia sanitaria, sin embargo, el objetivo de este trabajo se centra en el campo de la visión artificial.
Las redes neuronales convolucionales son un algoritmo de *deep learning* que está diseñado para trabajar con imágenes, tomando estas como entrada, asignándole pesos a ciertos elementos en la imagen, para así poder diferenciar unos de otros. En los últimos años, las redes convolucionales profundas han superado el estado del arte en muchas tareas de reconocimiento visual. Estas redes

contienen varias capas ocultas, donde las primeras puedan detectar líneas, curvas y así se van especializando hasta poder reconocer formas complejas como un rostro, siluetas, etc. Las tareas más comunes de este tipo de redes son: detección o categorización de objetos, clasificación de escenas y clasificación de imágenes en general.

La red toma como entrada los pixeles de una imagen. El uso típico de redes convolucionales es en tareas de clasificación, donde la salida a una imagen es una etiqueta de clase única. El kernel en las redes convolucionales se considera como el filtro que se aplica a una imagen para extraer ciertas características importantes o patrones de esta. Entre las características importantes para lo que sirve el kernel son detectar bordes, enfoque, desenfoque, entre otros. Esto se logra al realizar la convolución entre la imagen y el kernel. Uno de los procesos más distintivos de estas redes son las convoluciones. El cual consiste en tomar un grupo de píxeles de la imagen de entrada e ir realizando un producto escalar con un kernel. El kernel recorrerá todas las neuronas de entrada y obtendremos una nueva matriz como se observa en la Figura 5 [4].

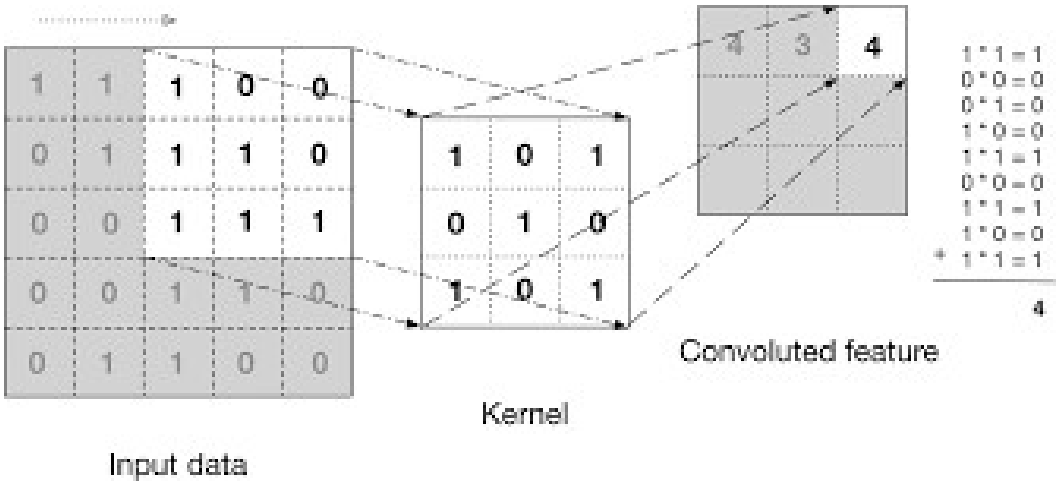


Figura 5.Aplicación del kernel a la imagen de entrada

La arquitectura de la red se observa en la Figura 6. Una modificación importante en la arquitectura de este trabajo es que en la parte de muestreo superior se tiene una gran cantidad de canales de características, que permiten que la red propague información de contexto a capas de mayor resolución. Como consecuencia, la trayectoria expansiva es más o menos simétrica a la trayectoria de contracción y produce una arquitectura en forma de U.

La U-net consta de una ruta de contracción (lado izquierdo Figura 6) y un camino expansivo (lado derecho Figura 6). La ruta de contracción sigue la arquitectura típica de una red convolucional. Consiste en la aplicación repetida de dos convoluciones de 3x3 (convoluciones sin relleno), cada una de ellas seguida de una unidad lineal rectificada (ReLU) y una operación de agrupación máxima de

2x2 con *stride* 2 para el muestreo descendente. En cada paso de muestreo descendente se duplica el número de canales.

Cada paso en la ruta expansiva consiste en un muestreo superior del mapa de características seguido de una convolución 2x2 (convolución ascendente) que reduce a la mitad el número de canales de características, una concatenación con el mapa de características recortado correspondientemente de la ruta de contratación y dos 3x3 convoluciones, cada una seguida de un ReLU que es la función de activación que devuelve el máximo entre 0 y el valor de entrada. El recorte es necesario debido a la pérdida de píxeles de borde en cada convolución. En la capa final, se usa una convolución 1x1 para mapear cada vector de características de 64 componentes al número deseado de clases.

En total, la red tiene 23 capas convolucionales que se muestran en la Figura 6, finalmente se obtiene la salida del mapa de segmentación [5].

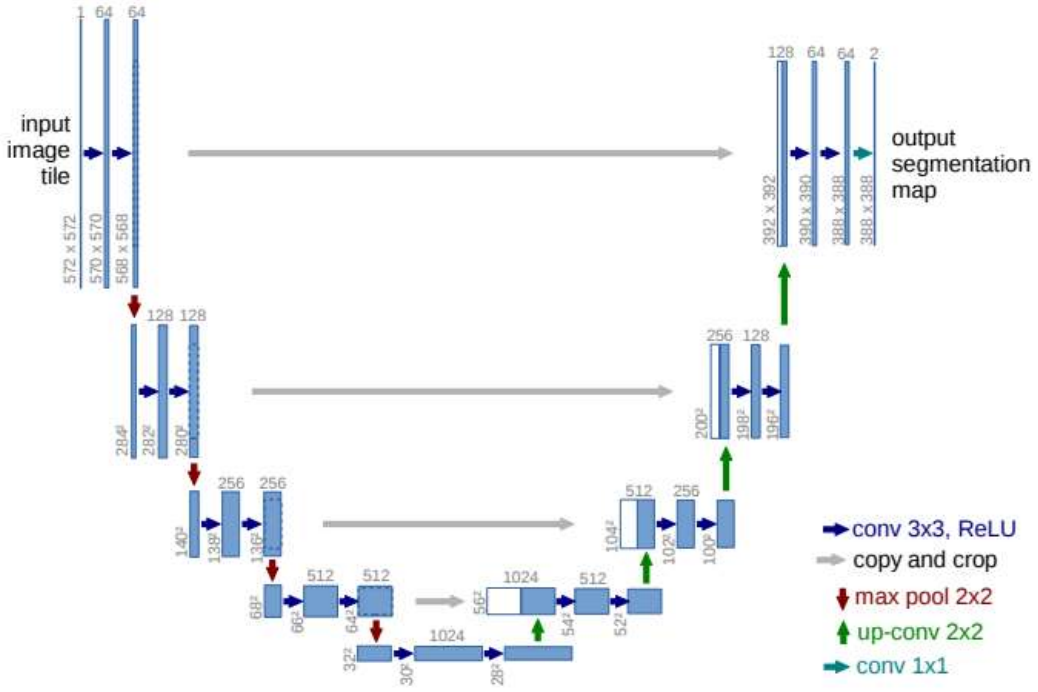


Figura 6. Red U-net, recuperado de https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

La red no tiene capas completamente conectadas y solo usa la parte válida de cada convolución, es decir, el mapa de segmentación solo contiene los píxeles, para los cuales el contexto completo está disponible en la imagen de entrada. Esta estrategia permite la segmentación perfecta de imágenes arbitrariamente grandes mediante una estrategia de mosaico superpuesto como la que se muestra en la Figura 7:

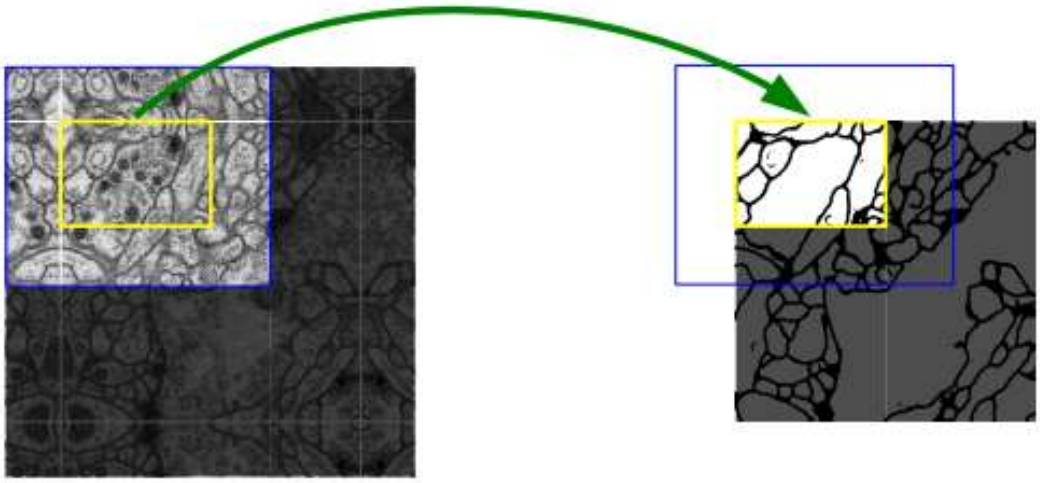


Figura 7. Superposición en mosaico

Para predecir los píxeles en la región del borde de la imagen, el contexto faltante se extrapola reflejando la imagen de entrada. Esta estrategia de mosaico es importante para aplicar la red a imágenes grandes, ya que de lo contrario la resolución estaría limitada por la memoria de la GPU (*Graphics Process Unit*).

En muchas de las técnicas de aprendizaje automático, este consiste en encontrar qué parámetros W (pesos para cada conexión nueronal) minimizan la función de coste. Esto es así para la regresión lineal y polinómica, la regresión logística, el deep learning, etc. El gradiente descendiente es un método de optimización numérica para estimar los mejores coeficientes y su objetivo es encontrar el conjunto de parámetros que minimizan la función de coste (error cuadrático medio). Como vemos en la Figura 8, el gradiente representa la pendiente en el punto que nos encontremos de la función de coste [6].

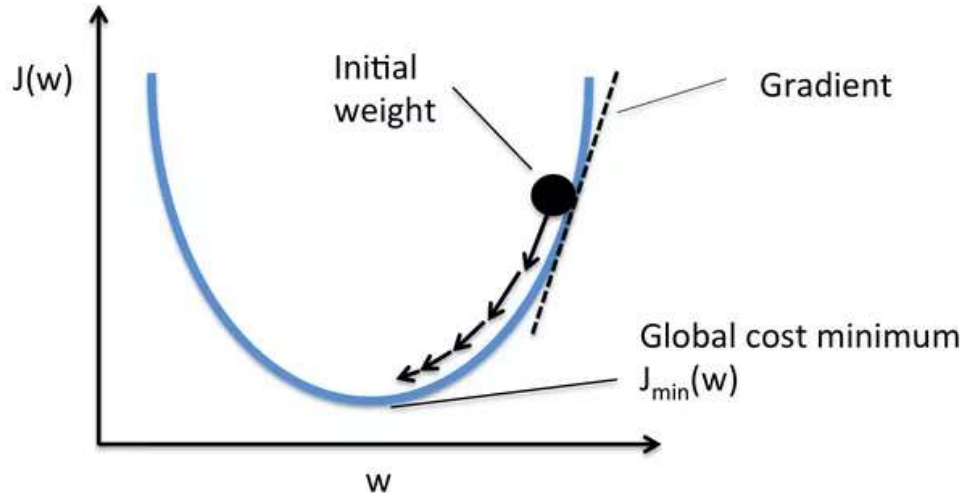


Figura 8. Gradiente de función de perdida, recuperado de: https://medium.com/data-science-group-iitr/

El optimizador utilizado para entrenar es RMSProp [7], es un optimizador de velocidad de aprendizaje adaptativo propuesto por Tieleman & Hinton (2012). La motivación es que la magnitud de los gradientes puede diferir para diferentes pesos, y puede cambiar durante el aprendizaje. RMSProp aborda esto manteniendo una media móvil del gradiente cuadrado y ajustando las actualizaciones de peso en esta magnitud. Como función para la optimiazción se utiliza BCEWithLogitsLoss que forma parte de Pytorch, combina una capa sigmoidea y el BCELoss *(Binary Cross Entropy)* en una sola clase [8]. La *cross entropy* (entropía cruzada) se puede utilizar para definir una función de pérdida en el aprendizaje automático y la optimización. La verdadera probabilidad p_i es la etiqueta verdadera, y la distribución dada q_i es el valor predicho del modelo actual.

La regresión logística,que (entre otras cosas) se puede utilizar para clasificar las observaciones en dos clases posibles (a menudo simplemente etiquetadas como 0 y 1). La salida del modelo para una observación dada, dado un vector de entidades de entrada x, puede interpretarse como una probabilidad, que sirve de base para clasificar la observación. La probabilidad se modela utilizando la función logística:

$$g(z) = \frac{1}{(1 + e^{-z})}$$

donde z es alguna función del vector de entrada x, comúnmente una función lineal. La probabilidad de la salida y=1 está dado por:

$$q(y = 1) = \frac{1}{(1 + e^{-w*x})}$$

donde el vector de pesos W se optimiza a través de algún algoritmo apropiado, como el descenso de gradiente [9].

4. METODOLOGÍA

Las arquitecturas de redes han sido probadas con imágenes celulares y el objetivo es aplicarla a un *dataset* (conjunto de datos) cedido por Unizar para alcanzar resultados similares. La forma de evaluar si el trabajo cumple con los objetivos, se verifica a través de métricas que miden el desempeño de la red, así como de la capacidad de segmentar imágenes celulares por medio de los modelos entrenados. Una vez realizados los entrenamientos, modificando parámetros de la red o del entrenamiento, se podrá ver cómo realiza la segmentación de las imágenes respecto a las máscaras con las que se ha entrenado. Una de las finalidades de realizar este tipo de técnicas para segmentar es poder controlar la migración celular.

Para poder trabajar la segmentación de imágenes celulares con *deep learning* es necesario un entorno de trabajo basado principalmente en la programación. Como lenguaje se utiliza Pyhon (*Python Software Foundation*), es un lenguaje de programación de alto nivel interpretado, orientado a objetos y con semántica dinámica [10]. A través de Python se utilizará la biblioteca Pytorch que permite trabajar las técnicas de *deep learning* con sus modelos de forma eficiente. Entre sus funciones más recalcables, tiene la funcionalidad de integrar *data augmentation,* dotando de entrenamientos con imágenes creadas artificialmente, que permiten aumentar *datasets* que son de un tamaño reducido.

Entre las ventajas que se tienen por utilizar Pytorch para técnicas de *deep learning* se encuentran las siguientes: está basado en Python y dispone de una gran compatibilidad, permite la utilización de gráficos dinámicos para ver el desarrollo del entrenamiento, gran velocidad de computo, aumento de la productividad gracias a la depuración de código integrada y aceleramiento con la GPU del ordenador.

En el caso del trabajo se ha accedido a los códigos almacenados en un repositorio para poder entrenar con la red convolucional U-net.,este repositorio se encuentra disponible de forma pública en la pagina web de GitHub [11].

Para poder gestionar todas las librerías de código necesarias y entornos de programación, se ha utilizado Anaconda, que permite crear entornos de trabajo individualizados sin crear dependencias globales con la instalación de otras versiones de librerías en el ordenador.

En el caso del trabajo realizado y debido a que es necesario compartir el mismo ordenador con otros usuarios, era crucial crear entornos propios de trabajo, teniendo la posibilidad según el tipo de código utilizado y su compatibilidad con ciertas librerías, cambiar facilmente de un entorno a otro. Además esta herramienta permite instalar practicamente todas las librerías necesarias para ejecutar las técnicas utilizadas dentro del campo del *deep learning.*

La metodología seguida durante el trabajo fue:

1.-En primer lugar, familiarizarse con los entornos de programación necesarios.

2.-Obtener un *dataset* (Unizar Dataset) al que aplicar segmentación de imagen con *deep learning.*

3.-Una vez conseguido entrenar el modelo con imágenes, se pasó a la fase de evaluación de resultados con una métrica específica.

4.- A través de la modificación de parámetros del entrenamiento y utilizando técnicas de optimización se procedió a mejorar la métrica de evaluación, así como observar el desempeño del modelo entrenado segmentando las imágenes.

Como se disponía de muy pocos datos de entrenamiento, se precisaba utilizar técnicas de aumento de datos aplicando deformaciones elásticas a las imágenes disponibles. Esto permite que la red

aprenda la invariancia a tales deformaciones, sin la necesidad de ver estas transformaciones en las propias imágenes originales. Esto es particularmente importante en la segmentación biomédica, ya que la deformación suele ser la variación más común en el tejido y las deformaciones realistas se pueden simular de manera eficiente.

Para la realización de los entrenamientos, se precisa de capacidad de cálculo por lo que se ha utilizado de forma remota un ordenador a través de SSH (*Secure Shell*) para los entrenamientos. El ordenador utilizado en remoto forma parte de la universidad, está dotado de una tarjeta gráfica de última generación que permite realizar entrenamientos largos y pesados de una forma ágil.

Puesto que algunos de los entrenamientos que se realizan pueden durar más de 12 horas es inviable mantener encendido el ordenador que se ejecuta en remoto. Por ello se utiliza una funcionalidad nativa de Linux, *Screen*, que permite administrar diferentes sesiones y ejecuciones aun cerrándose el terminal.

Para poder realizar pruebas de los cambios en el código obtenido de GitHub, se ha utilizado una plataforma de google llamada Colab (Google Colab), mediante la cual es posible ejecutar modelos de deep learning en la nube. Esto permite depurar el código de forma previa, de esta manera la gráfica de elevado poder de procesamiento se reserva para entrenamientos definidos y optimizados. Es de gran utilidad esta herramienta, ya que permite tener los archivos almacenados en la nube y no consume recursos del ordenador local.

5. SEGMENTACIÓN CELULAR MEDIANTE DEEP LEARNING: ANTECEDENTES

La red que se ha utilizado en el trabajo está basada en un trabajo de investigación de la universidad de Friburgo (2015). La arquitectura consta de una ruta de contracción para capturar el contexto y una ruta de expansión simétrica que permite una localización precisa [5].

El uso típico de redes convolucionales es en tareas de clasificación, donde la salida a una imagen es una etiqueta de clase singular. En muchas tareas visuales, especialmente en el procesamiento de imágenes biomédicas, el resultado deseado debe incluir la localización, es decir, se supone que se asigna una etiqueta de clase a cada píxel.

La arquitectura de la red ha sido modificada para que sea capaz de segmentar de manera correcta cuando se dispone de una cantidad de imágenes reducidas, como es el caso del campo de aplicación de este trabajo Fin de Máster.

En un principio el código que se utilizó para este trabajo de investigación fue realizado con la librería Caffe la cual fue descartada porque no tiene actualmente el mismo desarrollo que otras librerías. Actualmente son más utilizadas librerías como Tensorflow y Pytorch. Al comienzo del trabajo se utilizó la adaptación de la U-net en Tensorflow [12] como librería de referencia, llegando a obtener

resultados óptimos en su funcionamiento, pero las incompatibilidades de versiones de librerías que eran necesarias también condujeron a la utilización de Pytorch, que era compatible con otras funcionalidades que se requerían en el trabajo [11].

Dentro del trabajo de investigación, los investigadores adaptaron sus trabajos para conseguir obtener con un *dataset* reducido, el mejor resultado con la métrica escogida. En su caso se utilizó la métrica IoU (Figura 9) y en el caso del trabajo Dice (Figura 10), que es como se indica en la ecuación:

$$IoU = \frac{Area\ de\ coincidencia}{Area\ de\ la\ unión} \qquad Dice = \frac{2*(Area\ de\ coincidencia)}{Area\ total\ de\ ambas\ imágenes}$$

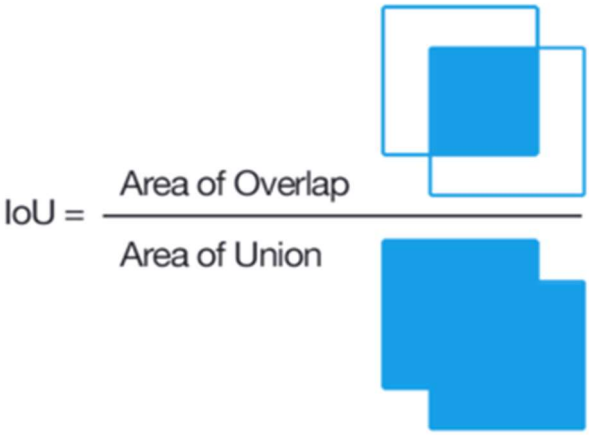


Figura 9. IoU, recuperado de: https://es.wikipedia.org/

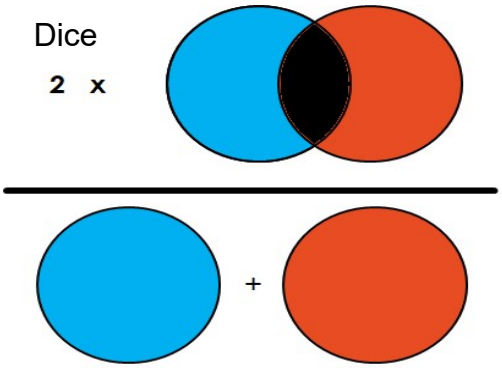


Figura 10. Dice, recuperado de: https://towardsdatascience.com

El éxito del trabajo fue tal que superó con gran distancia a otras adaptaciones para resolver este mismo problema, como se puede observar en la Tabla 1.

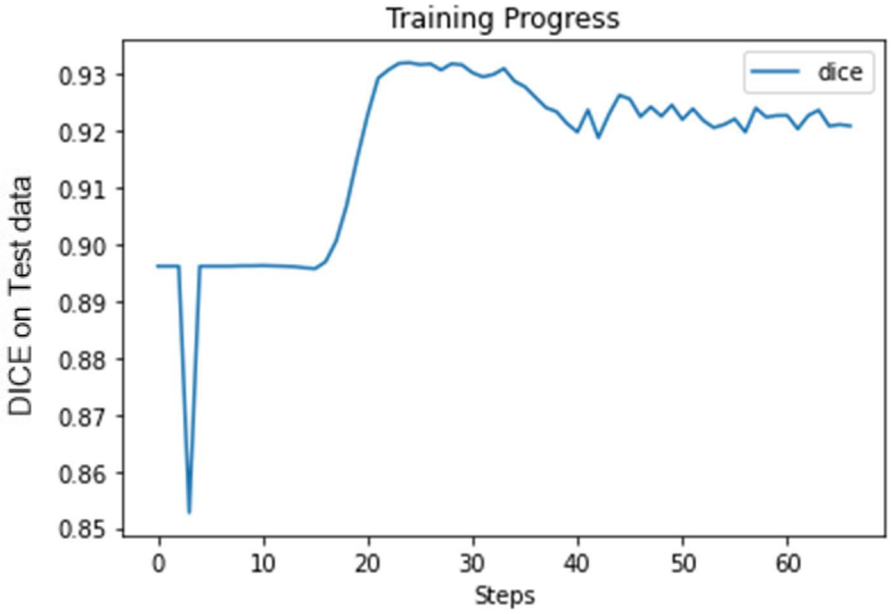
Tabla 1. Resultados de segmentación (IOU) en el desafío de seguimiento de células de ISBI 2015

Name	Phc-U373	DIC-hela
IMCB-SG(2014)	0.2669	0.2935
KTH-SE(2014)	0.7953	0.4607
HOUS-US(2014)	0.5323	-
Second-best 2015	0.83	0.46
U-net (2015)	**0.9203**	**0.7756**

Se obtuvo para el *dataset* PhC-U373 una métrica del 0,9203 superando con diferencia al segundo mejor, en cambio para DIC-HeLa se alcanzó un 0,7756.

Por lo tanto, el objetivo del trabajo queda claro con esta introducción a este desarrollo de investigación. Es por ello por lo que se realizarán entrenamientos basados en esta metodología con un *dataset* cedido por Unizar con la intención de emular los resultados y alcanzar métricas similares.

Para verificar que con la programación que se estaba trabajando en GitHub se obtendrían resultados similares a los conseguidos en el trabajo, se entrenó la red neuronal convolucional con las mismas imágenes que para Phc-U373, obteniéndose un Dice coeficiente representado en la Gráfica 1 obtenida mediante TensorBoard:



Gráfica 1. Dice coeficiente para *dataset* Universidad de Friburgo (2015)

Se puede observar que se alcanza un resultado elevado, similar al representado en el estudio de la universidad de Friburgo, con lo que respalda el inicio del trabajo con las imágenes propias. Aunque se obtiene un *IoU y Dice* similar, se ha realizado el entrenamiento con la métrica Dice para poder comparar los resultados de la manera más igualitaria, puesto que va a ser la métrica utilizada durante todo el trabajo.

El coeficiente Sørensen-Dice, también conocido por otros nombres tales como el índice de Sørensen, coeficiente de Dice, es un estadístico utilizado para comparar la similitud de dos muestras. La fórmula original de Sørensen estaba destinada a ser aplicada a datos de presencia/ausencia, puede ser visto como una medida de similitud sobre conjuntos. Donde A y B son el número de especies en las muestras A y B, respectivamente, y la intersección es el número de especies compartidas por las dos muestras; Dice es el cociente de similitud y varía de 0 a 1. Esta versión cuantitativa del índice de Sørensen también se conoce como Czekanowski índice. El índice de Sørensen es idéntico al coeficiente de Dice que siempre está en [0, 1] como rango. [13]

6. METODOLOGÍA APLICADA A DATASET UNIZAR

6. ENTRENAMIENTO DEL MODELO SIN AUMENTAR EL NÚMERO DE IMÁGENES

Mediante el *dataset* de Unizar cedido por el Grupo M2BE de la Universidad de Zaragoza, se pretende adaptar y aplicar las mismas técnicas que han sido explicadas con anterioridad. Se han obtenido 175 imágenes de distintas células, en distintos geles. Cada imagen tiene asignada una máscara con la segmentación correspondiente hecha a mano.

El primer paso es organizar dentro del código obtenido mediante GitHub las 175 imágenes y máscaras. Se introducen en un directorio diferenciado por *images* (imágenes*)* y *masks* (máscaras).

Dentro del código del entrenamiento se ha estipulado un tamaño del conjunto de datos de evaluación del 25%, el 75% restante es utilizado para el entrenamiento del modelo. En las Figuras 11 y 12 se muestra una imagen del conjunto de datos junto a su máscara segmentada a mano.

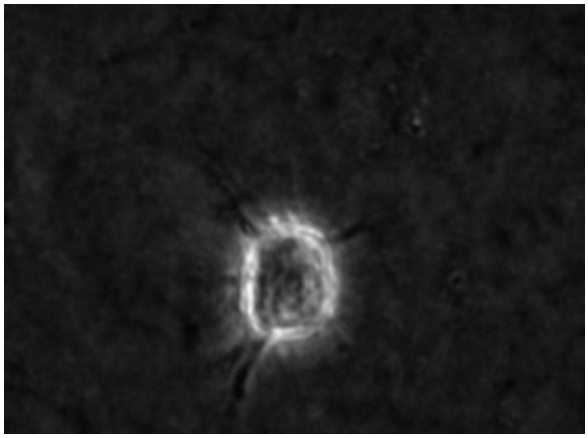


Figura 11. Imagen original

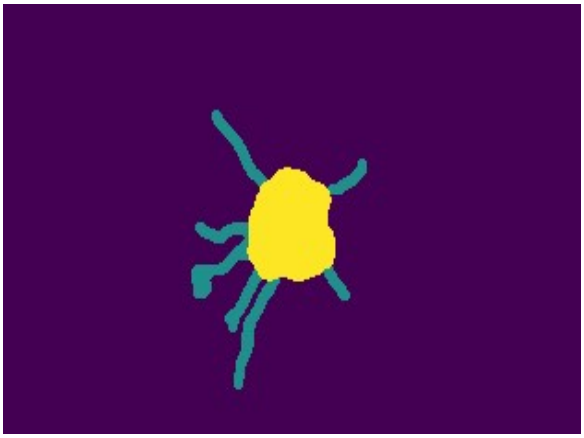


Figura 12. Máscara segmentada a mano

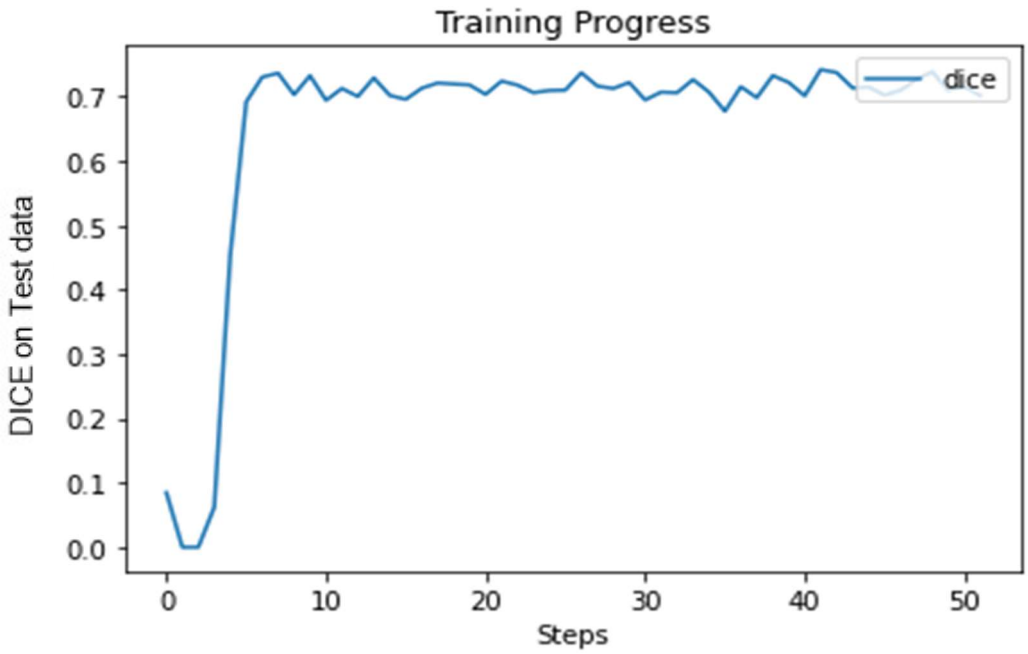
Una vez ubicado el *dataset,* se procede a adaptar el código de carga de datos, que como suele ser habitual lo que busca es el nombre de los ficheros, los carga y finalmente los introduce a la red para entrenar el modelo.

Dado que en este caso particular se tenían las imágenes en formato .png y las máscaras eran tensores (matrices con píxeles como elementos), se ha tenido que transformar las imágenes a formato tensor para poder cargarlas al entrenamiento. Cada tensor, tanto de imagen como de máscara contiene 3 posibles valores por píxel que representan cuerpo celular [0], protrusión [1] y todo lo demás [2]. Como se quiere realizar de forma binaria, se ha transformado todos los pixeles que eran mayores que 0 como 1. Por lo que ya se está en la situación de segmentación de imágenes binarias, es por ello por lo que diferenciará lo que es célula (cuerpo celular + protusión) y lo que es fondo, pasando de una máscara con 3 clases como la de la Figura 12 a máscaras binarias.

En una red neuronal, el paso hacia delante es un conjunto de operaciones que transforman la entrada de red en el espacio de salida. Durante la etapa de inferencia, la red neuronal se basa únicamente en el paso hacia delante. Para el paso hacia atrás, para poder empezar a calcular gradientes de error, primero, se tiene que calcular el error (es decir, la pérdida general). Se puede ver toda la red neuronal como una función compuesta. Usando la Regla de Cadena, se puede encontrar la derivada de una función compuesta. Esto permite obtener los gradientes individuales. En otras palabras, se puede usar la regla cadena para repartir el error total a las distintas capas de la red neuronal. Esto representa el gradiente que se minimizará utilizando *Gradient Descent* [14].

Tras adaptar el entorno de trabajo con la herramienta Anaconda, se instalaron todas las librerías que son necesarias para ejecutar el código y comenzar a entrenar. Una vez realizado 5 *epochs*, que hace referencia a cuando un conjunto de datos se pasa hacia delante y hacia atrás a través de toda la red neuronal, se procede a realizar el entrenamiento. Dado que una época es demasiado grande para alimentar a la computadora a la vez, se divide en varios lotes más pequeños, conocido como tamaño de *batch*. Pasar todo el conjunto de datos a través de una red neuronal no es suficiente. Es necesario pasar el conjunto de datos completo varias veces a la misma red neuronal. Por lo tanto, actualizar los pesos con una sola pasada o una época no es suficiente.

Se han obtenido los siguientes resultados (Gráfica 2):



Gráfica 2. Dice coeficiente 175 imágenes con 5 *epochs*

Tras realizar todo el entrenamiento se observa que se alcanza entorno a un 0.7 en la Gráfica 2, que comparado con los resultados obtenidos en el trabajo de investigación en el que nos basamos, está próximo del rango de los resultados que obtuvieron.

La arquitectura que se ha utilizado de la red es la que se muestra en código, en la que se pueden observar las capas de contracción donde se duplican los canales en cada etapa, y la de expansión donde se reducen a la mitad, así como las operaciones de convolución.

```
self.inc = DoubleConv(n_channels, 64)
self.down1 = Down(64, 128)
self.down2 = Down(128, 256)
self.down3 = Down(256, 512)
factor = 2 if bilinear else 1
self.down4 = Down(512, 1024 // factor)
self.up1 = Up(1024, 512 // factor, bilinear)
self.up2 = Up(512, 256 // factor, bilinear)
self.up3 = Up(256, 128 // factor, bilinear)
self.up4 = Up(128, 64, bilinear)
self.outc = OutConv(64, n_classes)
```

A continuación, se muestran unas imágenes originales (Figuras 14 y 16) y la segmentación obtenida con el modelo entrenado de forma correcta (Figuras 13 y 15):

Figura 13. Imagen segmentada con el modelo entrenado

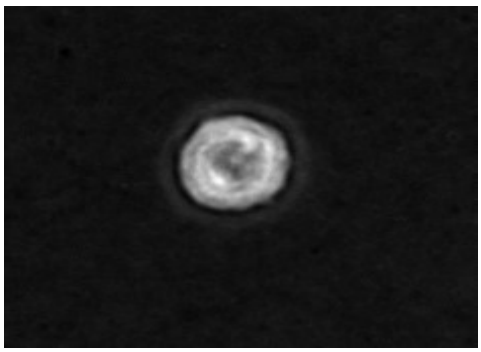


Figura 14. Imagen original

Figura 15. Imagen segmentada con el modelo entrenado

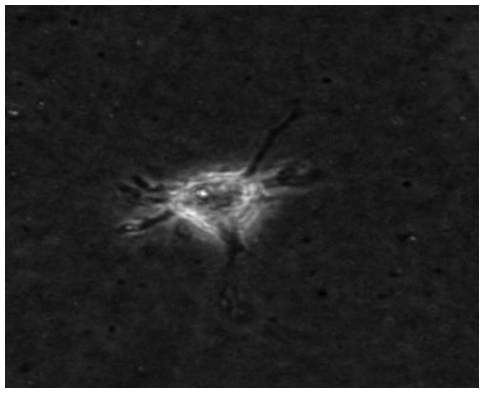


Figura 16. Imagen original

En otros casos se puede observar como el modelo tiene sus limitaciones, con ciertas protusiones celulares puede llegar a no tener una segmentación perfecta (Figura 18):

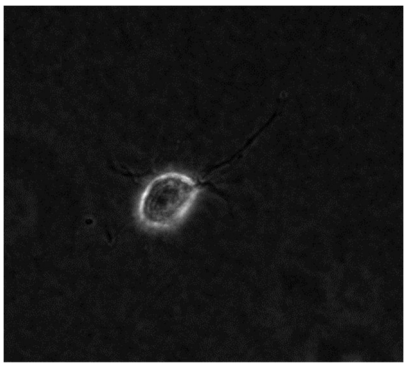


Figura 17. Imagen original

Figura 18. Imagen segmentada por el modelo

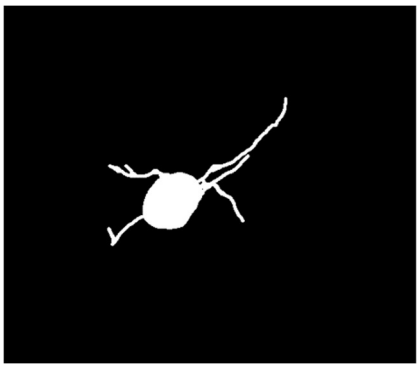


Figura 19. Mascara segmentada a mano

No obstante, el trabajo no termina con la obtención de este hito, dentro de las técnicas de mejora del rendimiento de los entrenamientos, hay parámetros que se pueden modificar como son las veces que pasamos por todo el *dataset (epochs)* o modificar parámetros como la tasa de aprendizaje (*learning rate).*

Dada una tasa de aprendizaje perfectamente configurada, el modelo aprenderá a aproximarse mejor a la función dados los recursos disponibles (el número de capas y el número de nodos por capa) en un número determinado de épocas de entrenamiento (pasa a través de los datos de entrenamiento). En general, una mayor tasa de aprendizaje permite que el modelo aprenda más rápido, a costa de llegar a un conjunto final de pesos subóptimo. Una tasa de aprendizaje más pequeña puede permitir que el modelo aprenda un conjunto de pesos óptimo o incluso globalmente óptimo, pero puede tardar mucho más en entrenarse.

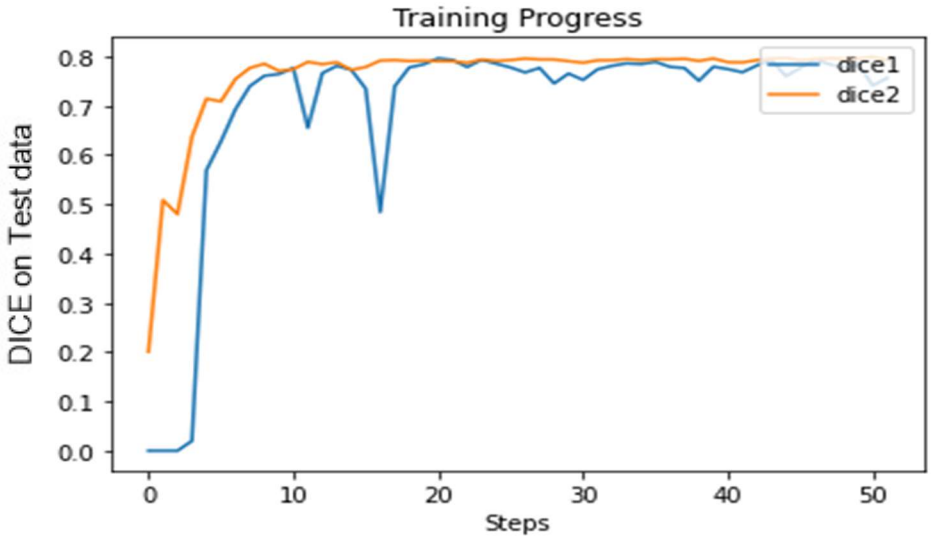
En los extremos, una velocidad de aprendizaje demasiado grande dará como resultado actualizaciones de peso que serán demasiado grandes y el rendimiento del modelo (como su pérdida en el conjunto de datos de entrenamiento) oscilará a lo largo de las épocas de entrenamiento. Una tasa de aprendizaje que es demasiado pequeña puede que nunca converja o puede quedarse atascada en una solución subóptima [15].

El *learning rate* puede ser modificado en nuestro código y entrenar en referencia al mismo, por convenio el entrenamiento se comienza con un “lr” de 1e-4. Si se establece este parámetro de manera elevada, se puede producir que se llegue a una optimización demasiado temprana y que no sea la correcta. En principio una bajada de este para el comienzo del entrenamiento hasta cierto punto puede suponer una mejora en los resultados.

Se ha realizado de nuevo un entrenamiento del modelo con un “lr” de 1e-5 (dice1), se puede confirmar que disminuir este parámetro para comenzar el entrenamiento ha conseguido que la red

aprenda de una manera óptima. Anteriormente se había alcanzado un Dice de 0.7 y ahora se ha terminado con un 0.76.

Continuando de la misma manera se ha reducido el parámetro de la tasa de aprendizaje a 1e-6 (dice2), se muestran los resultados comparativos en la Gráfica 3:



Gráfica 3. Dice comparativo con lr 1e-5 (dice1) y lr 1e-6 (dice2)

Se puede observar que al disminuir este parámetro ha conducido una mejoría en la métrica, llegando a 0.78, por lo que se puede determinar que sigue siendo favorable reducirlo. No obstante, la siguiente reducción no resultó favorable por lo que se determina que para este modelo el mejor *learning rate* es de 1e-6.

7. INCORPORACIÓN DE TÉCNICAS DE DATA AUGMENTATION

Como se ha comentado en el anterior capítulo, la incorporación de nuevas imágenes y máscaras transformándolas de forma exterior con cambios espaciales no es la forma más correcta de realizarlo. Se debe garantizar una incorporación de transformaciones que se implemente de forma aleatoria en cada *epoch.* Es importante tener en cuenta también que la misma transformación es aplicada tanto en la imagen como en la máscara, puesto que sino el aprendizaje de la red se vería mermado.

Los avances recientes en los modelos de aprendizaje profundo se han atribuido en gran medida a la cantidad y diversidad de datos recopilados en los últimos años. El aumento de datos es una estrategia que permite a los profesionales aumentar significativamente la diversidad de datos disponibles para los modelos de entrenamiento, sin necesidad de recopilar realmente nuevos datos [16].

Las técnicas de aumento de datos como el recorte, rotación etc. se utilizan comúnmente para entrenar grandes redes neuronales. En la Figura 20 se puede observar el proceso, en el que en cada iteración las imágenes cambian su aspecto espacial y permiten dar variabilidad al entrenamiento que sirve como ejemplo.

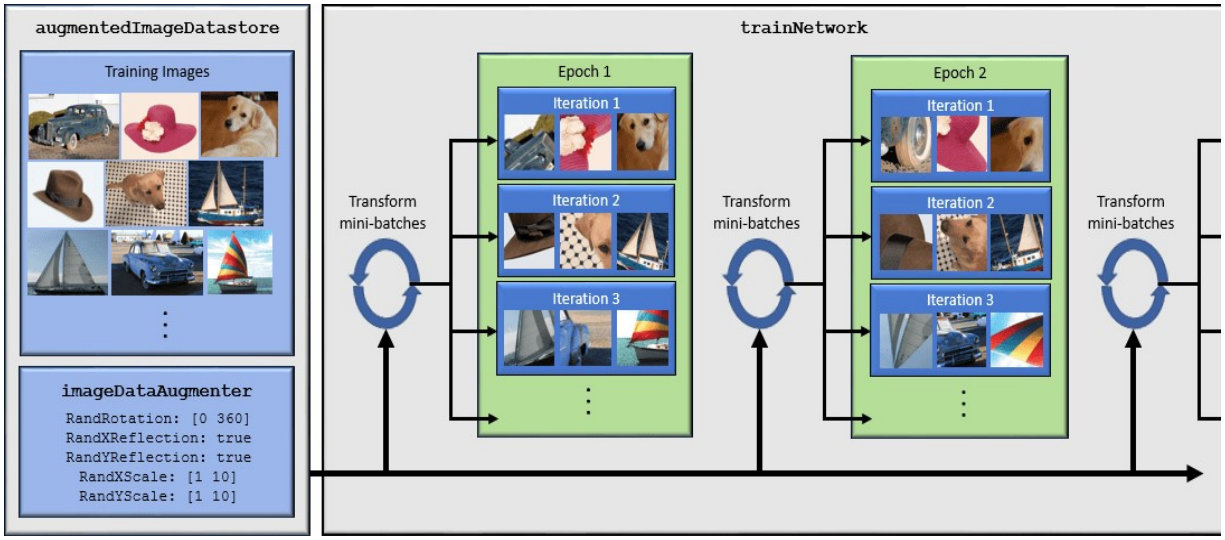


Figura 20. Transformaciones data augmentation Matlab

7.1 ENTRENAMIENTO DEL MODELO AUMENTANDO EL NÚMERO DE IMÁGENES A MANO

Uno de los problemas más frecuentes en este tipo de imágenes celulares es que no se disponen de una gran cantidad de imágenes con su segmentación realizada a mano. Normalmente este tipo de modelos aprenden de una forma más eficaz mediante la incorporación de nuevas imágenes de la tipología. Inicialmente se dispone de 175 imágenes y máscaras. Estas han sido tratadas para realizarles transformaciones de forma externa y así poder dar variabilidad al entrenamiento.

Pytorch es la librería que se ha utilizado para la realización de trabajo y dispone de diferentes transformaciones para incorporar en nuestro *dataset*, especialmente en su extensión de TorchVision que tiene infinidad de variaciones a realizar.

Las transformaciones que más sentido tienen son las que cambian características espaciales como rotaciones y giros de las imágenes, que en referencia al modelo funcionan como imágenes y máscaras completamente nuevas.

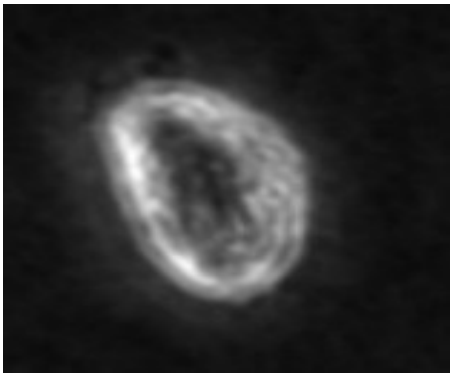


Figura 21. Imagen rotada 45º

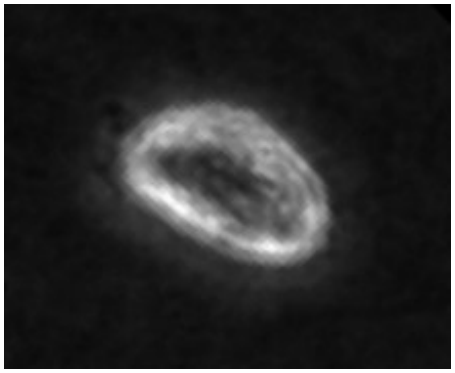


Figura 22. Imagen original

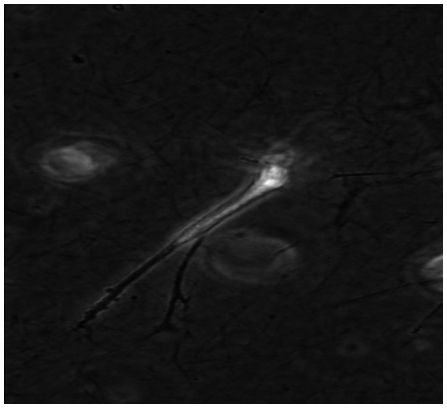


Figura 23. Imagen invertida

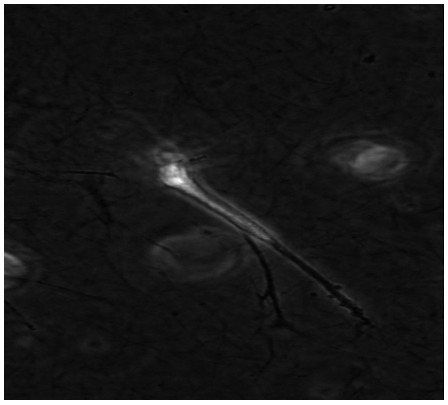
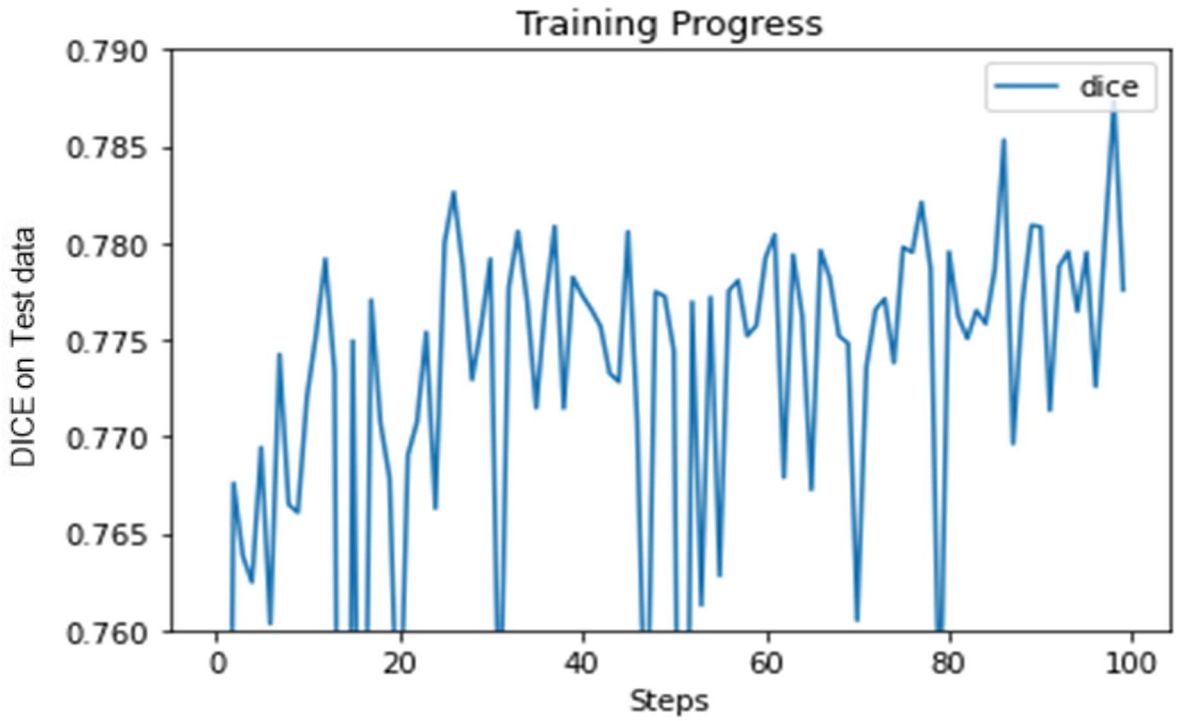


Figura 24. Imagen original

Mediante las transformaciones que permite Pytorch, se han transformado tanto las imágenes como las máscaras para garantizar la integridad del entrenamiento con las dos transformaciones mostradas en las Figuras 21 y 23.

Tras realizar ambas transformaciones, el *dataset* original pasa a triplicar su tamaño. Por lo que ya se está en disposición de iniciar un nuevo entrenamiento con esta mayor variabilidad de imágenes celulares.

En este caso se aumentó el número de *epochs* a 40, con el objetivo de obtener un aprendizaje mejor de la red neuronal convolucional y manteniendo la tasa de aprendizaje estándar para poder comparar las métricas. Los resultados se muestran en la Gráfica 4 del siguiente gráfico de TensorBoard.



Gráfica 4. Dice coeficiente con el *dataset* aumentado

Se puede observar en la gráfica 2 que al realizar el entrenamiento con las 175 imágenes se alcanzaba un coeficiente de 0.7, sin embargo, con la introducción de un mayor tamaño de imágenes y máscaras se ha incrementado hasta un 0.79 (Gráfica 4).

Para realizar *data augmentation* de una forma óptima se debe garantizar que las transformaciones se configuran de forma aleatoria a lo largo de cada *epoch*. Esto permite que el aprendizaje se realice de una forma óptima y es un paso previo al nuevo apartado, en el que se va a incorporar estas técnicas de aumento en el *dataset* de forma nativa en el código que carga las imágenes y máscaras.

7.2 ENTRENAMIENTO DEL MODELO CON DATA AUGMENTATION INCORPORADO EN EL CÓDIGO

A continuación, se muestran las modificaciones realizadas en el código donde se carga el *dataset* para posteriormente entrenar y validar la red.

```
def __getitem__(self, i):
    idx = self.ids[i]
    mask_file1 = glob(self.masks_dir + idx + '.*')
    mask_file = str(mask_file1)[2:-2]
    img_file = glob(self.imgs_dir + idx + '.*')
    img1 = Image.open(img_file[0])
    mask1= torch.load(mask_file)
    seed = np.random.randint(2147483647) # make a seed with numpy generator
    random.seed(seed) # apply this seed to img tranfsorms
    torch.manual_seed(seed) # needed for torchvision 0.7
    if self.transform is not None:
        img = self.transform(img1)
    random.seed(seed) # apply this seed to target tranfsorms
    torch.manual_seed(seed) # needed for torchvision 0.7
    if self.transform is not None:
        mask = self.transform(mask1)
    img2=transforms.Grayscale(num_output_channels=1)(img)

    return {
        'image': transforms.ToTensor()(img2),
        'mask': mask
    }
```

Para poder realizar la misma transformación, lo que se ha modificado es la incorporación de una *random.seed* (semilla aleatoria) que garantiza que la misma transformación se ejecuta tanto en imagen como en su máscara. Posteriormente se introduce un agregado de transformaciones que están incluidas en *self.transform*.

Se muestra también las distintas modificaciones que se realizan a través de la extensión de Pytorch conocida como TorchVision. Todas ellas son aplicadas de forma secuencial y aleatoria, siendo incluidas en la función de las transformaciones llamada “compose”, que las agrupa en el código siguiente.

```
def __init__(self, imgs_dir, masks_dir, scale=1, mask_suffix=''):
    self.imgs_dir = imgs_dir
    self.masks_dir = masks_dir
    self.scale = scale
    self.mask_suffix = mask_suffix
    self.transform = transforms.Compose([
            transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.9, 1.1), shear=0),
            transforms.RandomHorizontalFlip(p=0.3),
            transforms.RandomVerticalFlip(p=0.3),
            transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 2.0))
            ])
```

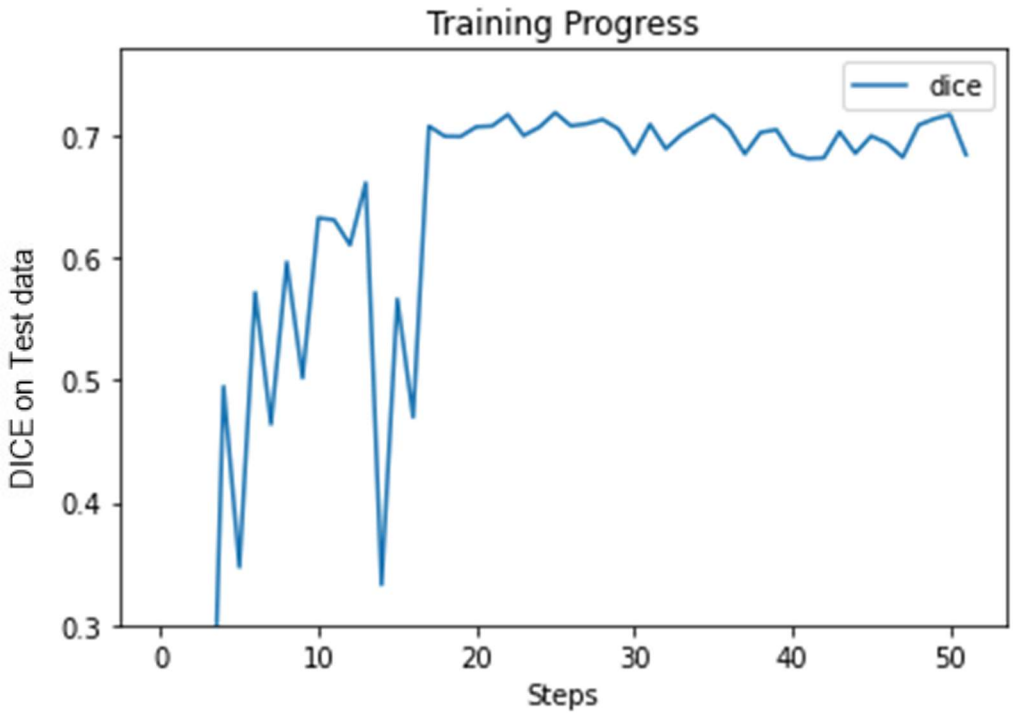
Las diferentes transformaciones que se han aplicado corresponden a las siguientes acciones sobre nuestro *dataset* [17]:

- **RandomAffine**: Transformación aleatoria de la imagen manteniendo el centro invariante. Incluye las siguientes transformaciones: Rango de grados de rotación, traslaciones horizontales y verticales e intervalo del factor de escala.
- **RandomHorizontalFlip**: Voltear horizontalmente la imagen dada aleatoriamente con una probabilidad dada.
- **RandomVerticalFlip**: Voltear verticalmente la imagen dada aleatoriamente con una probabilidad dada.
- **GaussianBlur**: Desenfoque de la imagen con modificación gaussiana elegida al azar. Precisa de tamaño de Kernel y de la desviación estándar.

Todas ellas son aplicadas de forma aleatoria, de manera que en cada *epoch* el aprendizaje de la red se ve variado, lo que garantiza que aprenda a diferenciar aspectos más concretos de cada imagen celular. Una vez se tienen las transformaciones definidas, se puede realizar entrenamientos con estas técnicas de *data augmentation*.

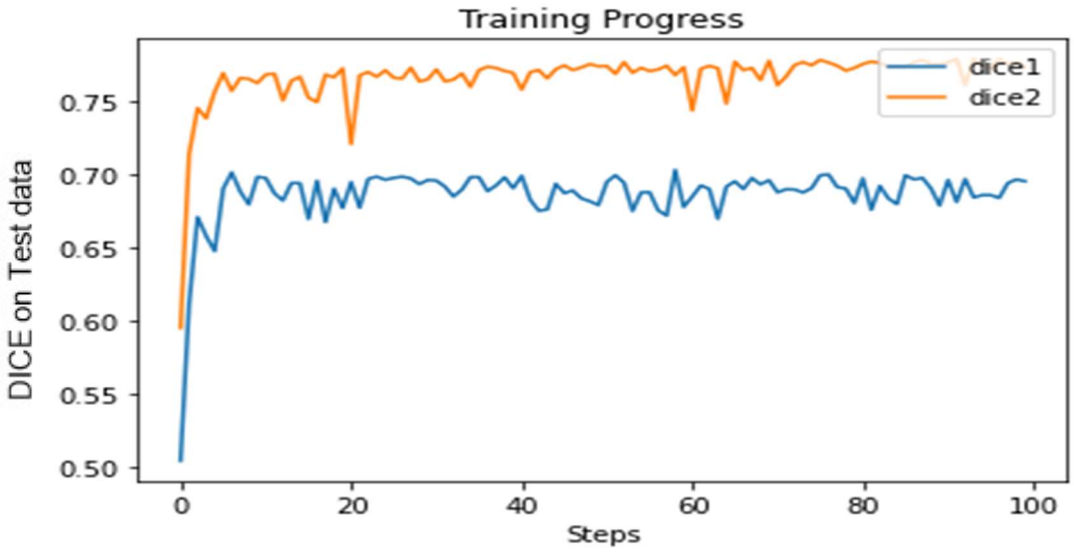
Primero de todo se ha realizado un entrenamiento con la tasa de aprendizaje estándar de 1e-4 y 5 *epochs.* Para poder comparar con los resultados sin técnicas de *data augmentation.*

Los resultados se muestran en la Gráfica 5 y reflejan una mejoría, puesto que al final del entrenamiento se alcanza una métrica de 0.72 mayor que 0.7, que se alcanzaba sin incorporar estas técnicas.



Gráfica 5. Dice coeficiente con data augmentation y 5 epochs

El siguiente paso ha sido incrementar la magnitud del entrenamiento con 40 *epochs* y manteniendo la tasa de aprendizaje inicial constante con (1e-4) y posteriormente modificarla (1-e-5.) Se puede observar en el gráfico comparativo del Gráfica 6, que en este caso el gasto computacional no ha sido efectivo y la convergencia del entrenamiento no necesita de tantas *epochs* (dice1) puesto que se obtiene un resultado similar al anterior. No obstante, se ha realizado de nuevo el entrenamiento con una tasa de aprendizaje de 1e-5, obteniéndose los siguientes resultados de la gráfica 6 (dice2).



Gráfica 6. Dice coeficiente con data augmentation,40 epochs y learning rate 1e-5

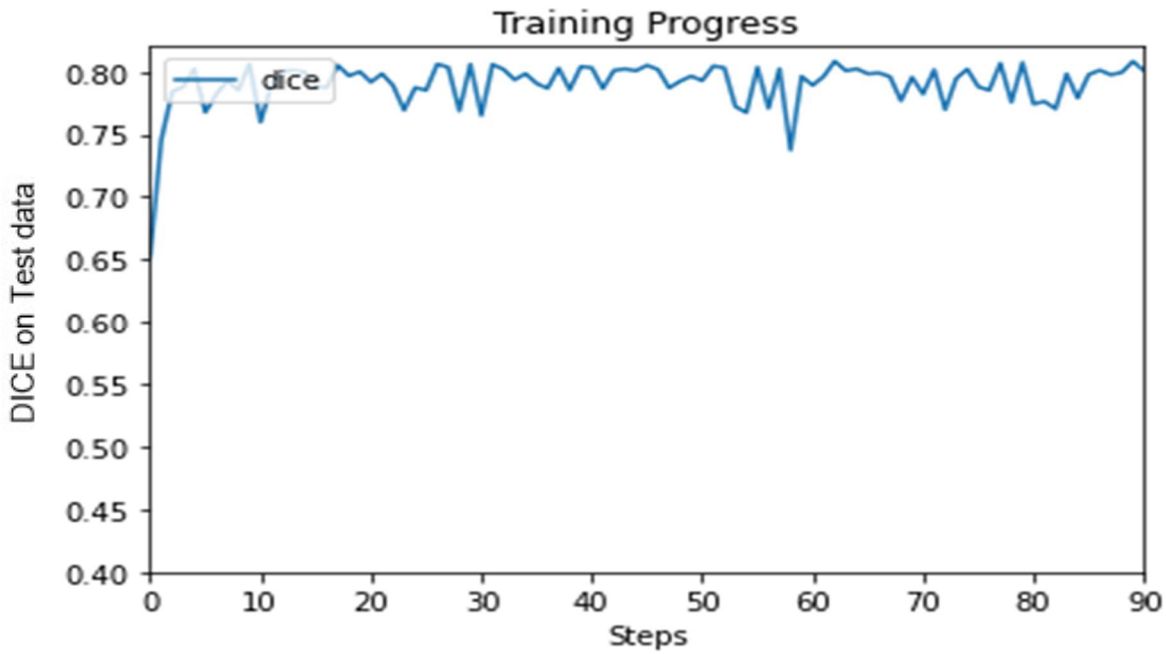
Se puede observar como el parámetro más importante que marca la diferencia es la tasa de aprendizaje, consiguiéndose una métrica de alrededor de 0.775, lo que supone una gran diferencia respecto a la anterior métrica de 0.7.

Posteriormente se prosiguió aumentando la duración del entrenamiento con tasas de aprendizaje disminuidas, pero todo ello no repercutió en mejoras en la métrica del entrenamiento.

Como se comenta en el trabajo de investigación [5], se realizaron transformaciones como la aplicada en el código propio de transformaciones, llamada GaussianBlur. Esta transformación realiza el desenfoque gaussiano en la imagen por un kernel determinado. Precisa de entrada una imagen o tensor y una desviación estándar.

La característica de desenfoque gaussiano se obtiene suavizando una imagen utilizando una función gaussiana para reducir el nivel de ruido. Se puede considerar como un filtro de paso bajo no uniforme que conserva la baja frecuencia espacial y reduce el ruido de la imagen y los detalles insignificantes en una imagen. Por lo general, se logra mediante la convolución de una imagen con un núcleo gaussiano. El valor de σ controla la varianza alrededor de un valor medio de la distribución gaussiana, que determina la extensión del efecto de desenfoque alrededor de un píxel [18].

Los resultados al aplicar esta transformación son los más satisfactorios, obteniéndose la siguiente gráfica de TensorBoard tras el entrenamiento que se muestra en la Gráfica 7



Gráfica 7. Dice coeficiente Data augmentation con GaussianBlur

Se puede observar que al final del entrenamiento se consigue una métrica superior a 0.8, aproximadamente 0.82, siendo esta la más alta conseguida en todo el trabajo.

A continuación, se muestra un ejemplo de cómo realiza el modelo la segmentación (Figura 26), comparándose con la máscara inicial (Figura 27).

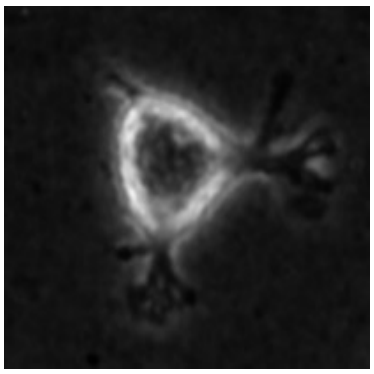


Figura 25. Imagen Original

Figura 26. Predicción de segmentación

Figura 27. Máscara segmentada a mano

8. RESUMEN COMPARATIVO DE LOS DISTINTOS RESULTADOS

A continuación, se muestra una tabla comparativa (Tabla 2) con los resultados obtenidos variando distintos parámetros del entrenamiento y utilizando o no *data augmentation*. La métrica utilizada para valorar el desempeño de la segmentación del modelo entrenado es Dice coeficiente.

Tabla 2. Resultados de los modelos entrenados

U-net modelo	Número de epochs	Learning rate	Data Augmentation	Dice
Dataset [5]	5	0.0001	No	0.92
Dataset Unizar	5	0.0001	No	0.7
Dataset Unizar	5	0.00001	No	0.76
Dataset Unizar	5	0.000001	No	0.78
Dataset Unizar	40	0.0001	No, pero alteración de imágenes para aumentar el *dataset*	0.79
Dataset Unizar	5	0.0001	Sí	0.72
Dataset Unizar	40	0.0001	Sí	0.69
Dataset Unizar	40	0.00001	Sí	0.775
Dataset Unizar	40	0.00001	Sí, Gaussian Blur añadido	0.82

Se puede observar en la tabla 2 como a lo largo de los entrenamientos realizados para entrenar el modelo se ha ido variando la métrica del trabajo. Para reducir el ruido en los resultados obtenidos en los entrenamientos se debería haber varios para cada caso, pero esto es inviable debido al recurso computacional limitado. Después de realizar los distintos entrenamientos se está en situación de decir que:

1.- Que la mejor tasa de aprendizaje para comenzar el entrenamiento es 1e-5.
2.- La implementación de técnicas de *data augmentation* ha permitido que se mejore el aprendizaje, dotando al entrenamiento de una mayor variabilidad de imágenes celulares.
3.- Dentro de todas las transformaciones implementadas dentro de Pytorch, sin duda la que mejor resultados ha producido es Gaussian Blur.
4.- El objetivo principal del trabajo que era conseguir un modelo que realizará correctamente la segmentación de las imágenes obtenidas ha sido alcanzado. Esto se puede confirmar puesto que la metodología ha sido aplicar distintas transformaciones espaciales, para observar que impacto tenían en el resultado final.

Gracias a que se ha tenido acceso a un equipo en remoto propio de la universidad de Zaragoza, se ha podido realizar entrenamientos más largos, lo que ha repercutido también en mejores resultados. Sin la disponibilidad de una gráfica potente hubiera sido difícil realizar este tipo de entrenamientos, debido a que la capacidad de cálculo necesaria para este tipo de entrenamientos es elevada.

9. EVALUACIÓN EN UN DATASET DISTINTO

Una señal de que el modelo entrenado es óptimo y capaz de generalizar es aplicándolo a un *dataset* de imágenes celulares diferente con el que se ha entrenado la red.
En el repositorio de GitHub utilizado, se encuentra un *dataset* con sus máscaras segmentadas. El objetivo de este repositorio era mejorar la segmentación automática de los tumores multiformes de glioblastoma (GBM), que son un tipo de cáncer que se da en el cerebro o en la médula espinal, mediante el uso de *deep Learning* [19].
Antes de entrenar con estas nuevas imágenes se ha procedido a segmentar con el modelo anteriormente entrenado las imágenes del *dataset* Unizar (fibroblastos), para asegurar que no ha aprendido de las imágenes que procede a segmentar. Como se puede observar en las Figuras 28 y 29 el modelo anterior, sin haber sido entrenado con este tipo de células ha sido capaz de generalizar, detectando contornos y formas de las imágenes que tenía que predecir.

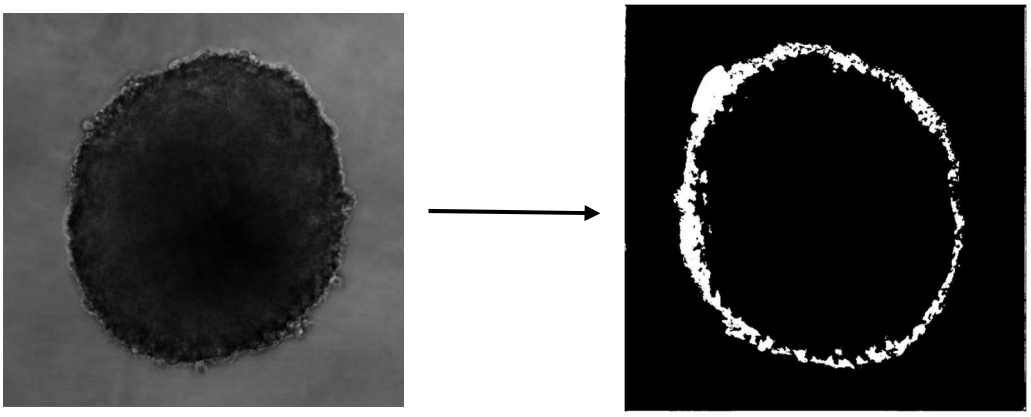


Figura 28. Imagen original y su segmentación

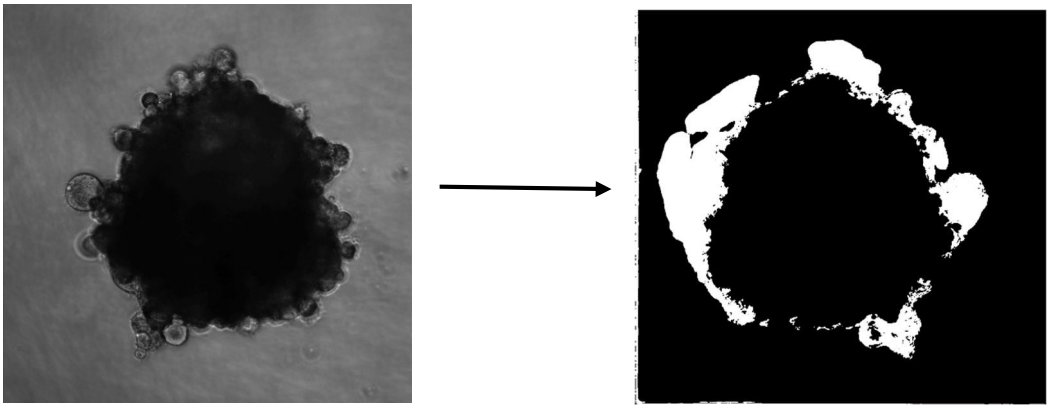
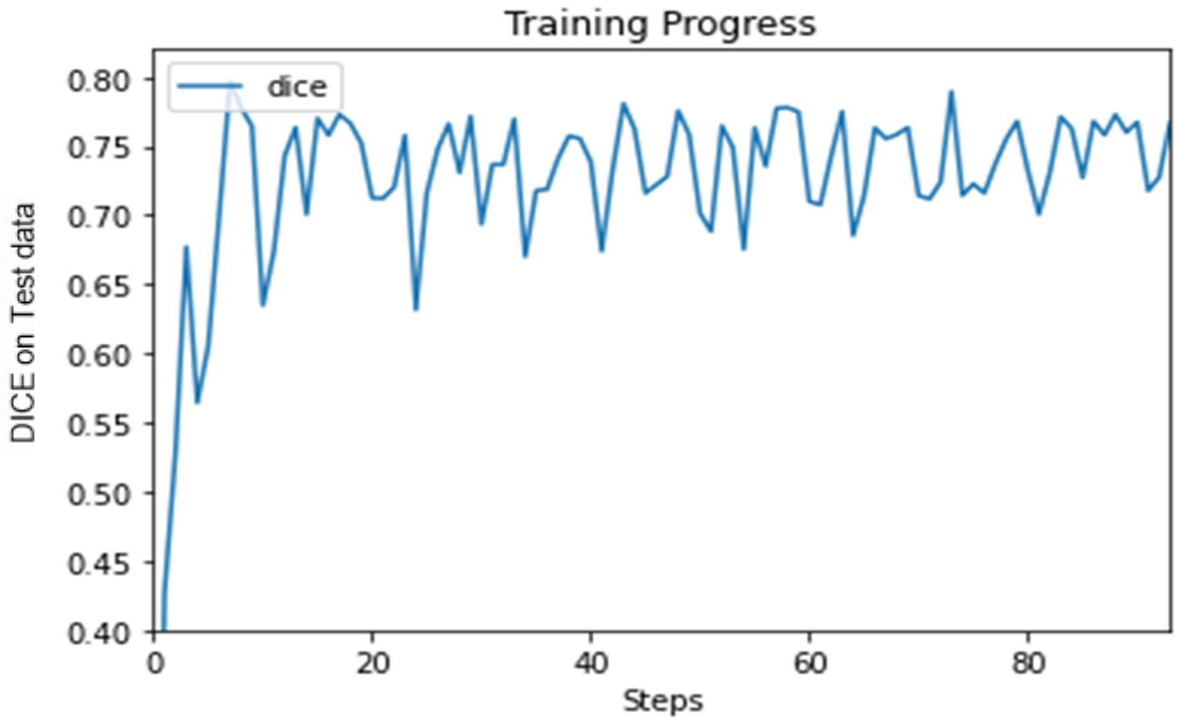


Figura 29. Imagen original y su segmentación

Por otro lado, al disponer de este nuevo *dataset,* era interesante ver los resultados de la red U-net entrenada con estas nuevas imágenes. Tras haber realizado un entrenamiento con las 366 imágenes nuevas y sus máscaras correspondientes (75% tamaño de entrenamiento y 25% tamaño de validación) y aplicándoles las técnicas anteriormente utilizadas de *data augmentation* se han obtenido los siguientes resultados (Gráfica 8)



Gráfica 8. Dice Coeficiente para Glioblastoma

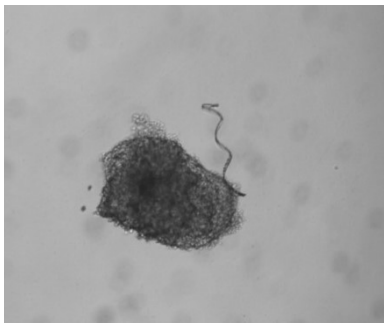


Figura 30. Imagen original

Figura 31. Imagen segmentada

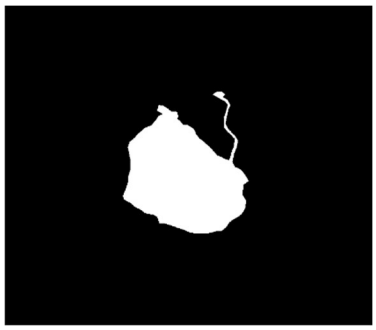


Figura 32. Mascara segmentada a mano

Se observa como la métrica alcanzada para 20 *epochs* y tasa de aprendizaje de 1e-5 es de entorno al 0.76. Los resultados son óptimos y similares a los que se tenían con el anterior *dataset*. Se muestra también el desempeño del modelo realizando la predicción de la segmentación de la imagen original (Figura 30), se observa como el resultado es muy positivo (figura 31) y prácticamente igual que la máscara segmentada a la perfección (figura 32).

10. CONCLUSIONES Y LÍNEAS FUTURAS

Por un lado, el objetivo principal del trabajo era conseguir la segmentación automática mediante técnicas de *deep learning* del tipo de imágenes que se disponían. Tras realizar varios entrenamientos con el *dataset* Unizar con distintos parámetros y optimizaciones se ha obtenido un modelo que realiza esta segmentación con una precisión bastante alta.

También se quería evaluar el desempeño de la red entrenada con una métrica, que finalmente ha sido el coeficiente dice. Un objetivo que se fijó fue conseguir a través de optimizaciones en los entrenamientos mejoría de esta métrica, lo cual se ha conseguido mediante la modificación de parámetros como la tasa de aprendizaje y la duración del entrenamiento, así como con la implementación de *data augmentation*.

Aunque los resultados del trabajo indican que los objetivos han sido cumplidos, se debe mencionar que hay ciertas imágenes en las que el desempeño del modelo tiene sus limitaciones, como en las protusiones de las células. Sin embargo, dados los continuos avances en *deep learning*, con trabajo adicional se podrían llegar a segmentaciones casi perfectas. Se pretende dejar marcadas las líneas de actuación futuras, de manera que se pueda seguir mejorando el trabajo realizado.

Una vez se tienen los resultados finales, se podrían intentar aplicar distintos enfoques con la intención de obtener una mejor actuación del modelo entrenado.

Sin duda, tener mayores *datasets* con imágenes celulares, permitiría que el modelo pudiera entrenar de forma más profunda, obteniendo aprendizaje con un mayor grado de detalle y mejorándose las métricas de segmentación.

Por otro lado, existen distintas técnicas basadas en los principios del *deep learning*, como serían las de *transfer learning y fine tuning.*

Cuando hablamos de *transfer learning* (Figura 33), nos referimos a transferir las características aprendidas en un problema y aprovecharlas en un problema nuevo y similar. El aprendizaje de transferencia se realiza normalmente para tareas en las que el conjunto de datos tiene muy pocos datos para entrenar un modelo desde cero. Constaría de las siguientes etapas este proceso:

1. Tomar capas de un modelo entrenado previamente.
2. Congelarlas, a fin de evitar la destrucción de cualquier información que contienen durante futuras rondas de entrenamiento.
3. Agregar algunas capas nuevas y entrenarlas sobre las capas congeladas. Aprenderán a convertir las características antiguas en predicciones en un nuevo conjunto de datos.
4. Entrenar las nuevas capas en el *dataset*.

Finalmente se puede realizar un ajuste fino (*Fine tuning*), que consiste en descongelar todo el modelo que se obtuvo anteriormente (o parte de él) y volver a entrenarlo con los nuevos datos con una tasa de aprendizaje muy baja. Esto puede potencialmente lograr mejoras significativas, mediante la adaptación incremental de las características previamente entrenadas a los nuevos datos [20].

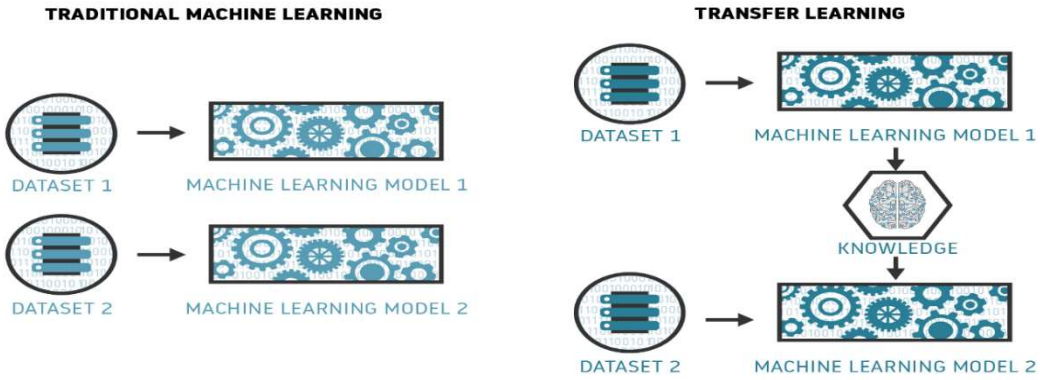


Figura 33. Metodología tradicional vs Transfer Learning

AGRADECIMIENTOS

Agradecer la labor del director de este trabajo, Rubén Martínez Cantín que me ha acompañado durante estos meses en la realización de este y me ha enseñado conceptos con los que he podido conseguir los resultados esperados y mejorar mi formación en el campo del *deep learning.*

Por otro lado, también agradecer que, una parte del trabajo se basa en los desarrollos con las imágenes celulares y sus etiquetados realizados por Francisco Merino y María José Gómez del Departamento de Ingeniería Mecánica.

Agradezco a toda mi familia y amigos el apoyo que me han brindado durante todo mi desarrollo académico en la universidad, que sin duda en los momentos más duros han sido capaces de sacar lo mejor de mí

BIBLIOGRAFÍA

1. Merino-Casallo, F., Gomez-Benito, M. J., Juste-Lanas, Y., Martinez-Cantin, R., & Garcia-Aznar, J. M. (2018). Integration of in vitro and in silico models using Bayesian optimization with an application to stochastic modeling of mesenchymal 3D cell migration. Frontiers in physiology, 9, 1246.
2. What is Deep Learning and How does it work? | Towards Data Science. (2021, 28 febrero). Medium.https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work-2ce44bb692ac
3. Carrera, L. (2021, 26 abril). Deep learning: ¿qué es, qué tipos hay y para qué sirve? TIC Portal. https://www.ticportal.es/glosario-tic/deep-learning-dl
4. Ai, B. (2019, 27 noviembre). Intro a las redes neuronales convolucionales Medium.https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8
5. Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.Heras, J. M. (2020, 20 septiembre). Gradiente Descendiente para aprendizaje automático. IArtificial.net. https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/
6. 6. Heras, J. M. (2020, 20 septiembre). Gradiente Descendiente para aprendizaje automático. IArtificial.net. https://www.iartificial.net/gradiente-descendiente-para-aprendizaje-automatico/
7. RMSProp Explained. (s. f.). RMSProp. Recuperado 5 de junio de 2021, de https://paperswithcode.com/method/rmsprop
8. BCEWithLogitsLoss — PyTorch 1.9.0 documentation. (s. f.). Pytorch. Recuperado 21 de junio de 2021, de https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html
9. Wikipedia (s. f.). Cross entropy. Wikipedia. Recuperado 21 de junio de 2021, de https://en.wikipedia.org/wiki/Cross_entropy#Relation_to_log-likelihood
10. What is Python? Executive Summary. (s. f.). Python.Org. Recuperado 26 de mayo de 2021, de https://www.python.org/doc/essays/blurb/
11. M. (s. f.). milesial/Pytorch-UNet. GitHub. Recuperado 26 de mayo de 2021, de https://github.com/milesial/Pytorch-UNet
12. zhixuhao/unet. (s. f.). GitHub. Recuperado 21 de junio de 2021, de https://github.com/zhixuhao/unet
13. Dice, Lee R. (1945). «Measures of the Amount of Ecologic Association Between Species». Ecology 26 (3): 297–302. (s. f.). Ecology 26 (3): 297–302.

14. An elegant way to represent forward propagation and back propagation in a neural network. (s. f.). Data Science Central. Recuperado 21 de junio de 2021, de https://www.datasciencecentral.com/profiles/blogs/an-elegant-way-to-represent-forward-propagation-and-back
15. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series) (English Edition). The MIT Press.
16. Seita, D. (2019, 7 junio). 1000x Faster Data Augmentation. The Berkeley Artificial Intelligence Research Blog. https://bair.berkeley.edu/blog/2019/06/07/data_aug/
17. Torchvision.transforms — Torchvision master documentation. (s. f.). Pytorch. Recuperado 1 de junio de 2021, de https://pytorch.org/vision/stable/transforms.html
18. Machine Learning for Subsurface Characterization, 2020. (2020). Machine Learning for Subsurface Characterization, 2020. Published.
19. Lacalle Castillo, D. (s. f.). WaterKnight1998/Deep-Tumour-Spheroid. GitHub. Recuperado 14 de junio de 2021, de https://github.com/WaterKnight1998/Deep-Tumour-Spheroid
20. Team, K. (s. f.). Keras documentation: Transfer learning & fine-tuning. Keras. Recuperado 5 de junio de 2021, de https://keras.io/guides/transfer_learning/

ANEXO I REPOSITORIO EN GITHUB

A continuación, se adjunta un enlace mediante GitHub, en el que se pueden encontrar en el repositorio los códigos utilizados y modificados para el desarrollo del trabajo:

https://github.com/martinhorndler/MartinHorndlerTFM.git

Dentro del repositorio creado, en el archivo README.md se ha resumido de forma general los cambios realizados sobre los códigos originales.

La utilización de la plataforma GitHub ha sido un punto clave en el desarrollo del trabajo, puesto que muchos usuarios reportaban información para adaptar los códigos de este tipo de redes a los casos concretos de estudio. Además, se ha utilizado la herramienta de Google *Collab*, que es un *notebook* en el que se pueden clonar los repositorios de GitHub si se desea y poder realizar pruebas, antes de realizar los entrenamientos de larga duración en la máquina virtual.

ANEXO II EJEMPLOS DE IMÁGENES SEGMENTADAS DEL DATASET UNIZAR

Se muestran las imágenes a la izquierda y su correspondiente segmentación a la derecha.