UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Ayuda al diagnóstico de cáncer de colon mediante técnicas de aprendizaje automático

**TRABAJO FIN DE GRADO**

Grado en Ingeniería Informática

*Autor:* José Javier Calvo Moratilla

*Tutor:* Jon Ander Gómez Adrián

Curso 2020-2021

# Agradecimientos

A todas las personas que dedicaron parte de su tiempo para ayudarme a comprender que la sociedad siempre está en continuo cambio y que tenemos que ser partícipes de ello, a mis profesores/ras del IES Salvador Gadea de Aldaia, CE. Cheste por ayudarme a crecer, a mi tía Fanny Llorens por haber creído en mí desde Barcelona, a mi padre José Vicente Calvo por transmitirme su amor por la cultura y las nuevas tecnologías, a mi familia por estar ahí en los tiempos más difíciles, a mi equipo del Hospital Oftalmológico de Valencia, Pablo, Juanjo, Carlos, Gabi, Javi, Edu, Joel, Gustavo por ser y haber sido parte de mi segunda familia durante estos últimos cuatro años y en último lugar a mi tutor Jon Ander Gómez Adrián del equipo de investigación *Pattern Recognition and Human Language Technology (PRHLT)* por haberme facilitado el entorno de hardware y software necesario para el desarrollo del proyecto y por la oportunidad que me ha otorgado para poder adentrarme en el mundo del aprendizaje automático.

# Resum

Les llistes d'espera dels hospitals espanyols durant la pandèmia del virus *SARS-CoV-2* han augmentat considerablement i han dificultat el diagnòstic primerenc de malalties com el càncer. Per a incrementar el rendiment del personal sanitari reduint la càrrega de treball es proposa la utilització d'algorismes d'aprenentatge automàtic per a detectar cèl·lules canceroses dins de l'àrea colorectal. S'empra l'arquitectura *U-Net* amb diferents configuracions per a segmentar l'àrea de l'imatge mèdica on es detecta la patologia, ajudant al personal sanitari en la tasca del diagnòstic de malalties. El màxim rendiment obtingut ha estat amb la configuració *sigmoid* 1b amb un 0,802630 de *IoU*. Es demostra que l'arquitectura de xarxa proposada entrena correctament amb el conjunt de dades obtenint un bon rendiment.

**Paraules clau:** xarxes neuronals, U-Net, aprenentatge automàtic, deep learning, càncer, colon, patología, diagnostic, whole-slide images

# Resumen

Las listas de espera de los hospitales españoles durante la pandemia del virus *SARS-CoV-2* han aumentado considerablemente y han dificultado el diagnóstico temprano de enfermedades como el cáncer. Para incrementar el rendimiento del personal sanitario reduciendo la carga de trabajo se propone la utilización de algoritmos de aprendizaje automático para detectar células cancerígenas dentro del área colorectal. Se emplea la arquitectura *U-Net* con diferentes configuraciones para segmentar el área de la imagen médica donde se detecta la patología, ayudando al personal sanitario en la tarea del diagnóstico de enfermedades. El máximo rendimiento obtenido ha sido con la configuración *sigmoid* 1b con un 0,802630 de *IoU*. Se demuestra que la arquitectura de red propuesta entrena correctamente con el conjunto de datos obteniendo un buen rendimiento.

**Palabras clave:** redes neuronales, U-Net, aprendizaje automático, deep learning, cáncer, colon, patología, diagnóstico, whole-slide images

# Abstract

Waiting lists in Spanish hospitals during the SARS-CoV-2 pandemic have increased considerably and have hindered the early diagnosis of diseases such as cancer. To increase the performance of healthcare personnel by reducing the workload, the use of machine learning algorithms to detect cancer cells within the colorectal area is proposed. The *U-Net* architecture is used with different configurations to segment the area of the medical image where the pathology is detected, helping health personnel in the task of disease diagnosis. The maximum performance obtained has been with the configuration *sigmoid* 1b with a 0.802630 of *IoU*. It is demonstrated that the proposed network architecture trains correctly with the data set obtaining a good performance.

**Key words:** neural networks, U-Net, machine learning, deep learning, cancer, colorectal, pathology, diagnosis, whole-slide images

# Índice general

# Índice de figuras

# Índice de tablas

# CAPÍTULO 1
# Introducción

En el presente trabajo final de grado se describe la participación en el *Challenge* PAIP2020, organizado por *Pathology AI Platform (PAIP)* y *Seoul National University Hospital (SNUH)*.

Los organizadores proponen diferentes retos, la detección de la existencia de cáncer colorrectal, la clasificación del subtipo molecular y segmentación de las áreas afectadas.

Para participar en el reto se procede a utilizar las últimas técnicas basadas en aprendizaje automático e implementar un algoritmo capaz de detectar la existencia de células cancerígenas, realizando una segmentación semántica.

La segmentación semántica es una de las cuatro tareas más importantes dentro de la visión por computador junto a la clasificación, la detección de objetos y la segmentación de instancias, que actualmente no sólo resuelven problemas dentro del campo de la medicina, también se aplica en la conducción autónoma de vehículos a la hora de percibir objetos de su entorno y clasificarlos.

## 1.1 Motivación

Se estima que en el año 2020 9,6 millones de personas fallecieron por culpa del cáncer en el mundo [1], ocupando el cáncer de colon el tercer lugar con 1,93 millones de casos siguiendo el cáncer de pecho y el de pulmón.

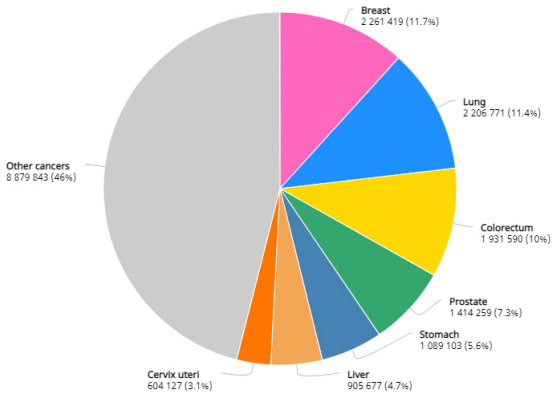


**Figura 1.1:** Número estimado de casos de cáncer en el año 2020, World Health Organization

La crisis económica ha incrementado seriamente las listas de espera de los hospitales españoles. El tiempo de espera medio de un paciente en la especialidad de cirugía general y digestiva dentro del sistema sanitario español en el año 2019 ha sido de 116 días [2].

El impacto del coronavirus en el año 2021 ha provocado una aceleración en la digitalización de la sociedad y la inteligencia artificial ha demostrado ser muy útil para resolver problemas dentro del ámbito sanitario para hacer frente al *SARS-CoV-2*.

Los equipos de investigación de empresas y universidades han visto la pandemia como la oportunidad de probar el estado de arte de la segmentación semántica para detectar los primeros síntomas del virus.

Las tareas de reconocimiento de patrones en imágenes médicas es un cometido que el cerebro humano puede resolver debidamente, pero la utilización de redes neuronales en la resolución del problema mejora enormemente la productividad de la unidad patológica de los hospitales en conjunción con la efectividad del factor humano.

La utilización de las nuevas tecnologías aplicadas al ámbito sanitario reducen la carga del personal médico dentro de los hospitales y agilizan la tarea de detección de células cancerígenas en los pacientes, incrementando la probabilidad de supervivencia, mejorando enormemente la calidad del servicio sanitario y reduciendo la lista de espera en los hospitales.

Se abarca el reto por la necesidad personal de querer ayudar al campo de la medicina a causa del fallecimiento de mi padre por un cáncer de colon terminal, por ello es homenaje a su memoria como a la de todas personas que padecieron la enfermedad y una primera aproximación en la tarea de detección en enfermedades en imágenes médicas.

Dentro del campo de la inteligencia artificial, la visión por computador es el área en la que más tiempo he dedicado y espero que el presente trabajo sea la lanzadera para poder realizar proyectos de mayor envergadura en el futuro.

## 1.2 Objetivos

El principal objetivo del trabajo es implementar un algoritmo capaz detectar células cancerígenas en imágenes médicas y demostrar la efectividad de las redes convolucionales con arquitectura U-Net para realizar dicho cometido de manera efectiva y fiable.

La red implementada recibe a la entrada una imagen tomada desde un equipo de electromedicina y genera una máscara dónde se indica la existencia o no de células tumorales, segmentando el área afectada.

Para lograr el cometido se tratan los siguientes seis puntos:

- Definición carpetas del proyecto y carga del dataset al servidor de *PRHLT*
- Preprocesamiento de las imágenes y máscaras de los conjuntos de entrenamiento, test y validación del dataset
- Implementación de la lanzadera de imágenes
- Implementación de la red
- Implementación del entrenamiento
- Implementación de la evaluación

En primera instancia se organiza la estructura de carpetas en los servidores del equipo de investigación *Pattern Recognition and Human Language Technology (PRHLT)*, almacenando el dataset para el desarrollo del proyecto y se instalan las librerías necesarias en el entorno, ya que implementan un gran número de funciones necesarias para el desarrollo del proyecto.

Seguidamente todo proceso de entrenamiento de redes neuronales precisa de un preproceso de datos, por ello se abarca una primera fase en la cual se preparan las imágenes para poder ser utilizadas dentro de las computadoras, superar las limitaciones de hardware existentes y asegurar un buen entrenamiento de los algoritmos de aprendizaje automático utilizados en el proyecto.

No tener disponible un conjunto de datos elevado e imágenes médicas de calidad también dificulta seriamente la tarea de segmentación semántica y puede provocar que los algoritmos no converjan a una solución exitosa, por ello a lo largo del proyecto se propone una solución para minimizar dichos problemas.

Una vez realizado el preproceso de datos se procede a la implementación del código necesario para la definición de la arquitectura de la red neuronal, la construcción del modelo, el entrenamiento y la evaluación de los diferentes modelos.

Para terminar el proyecto se procederá a realizar las pruebas de evaluación de los modelos de aprendizaje automático generados y así medir su rendimiento.

Dado que para realización del trabajo se realiza un desarrollo en cascada, los objetivos se distribuyen en un eje temporal indicando la fecha de inicio y final de cada uno de los puntos para el desarrollo del presente trabajo, obteniendo el siguiente diagrama de Gantt:

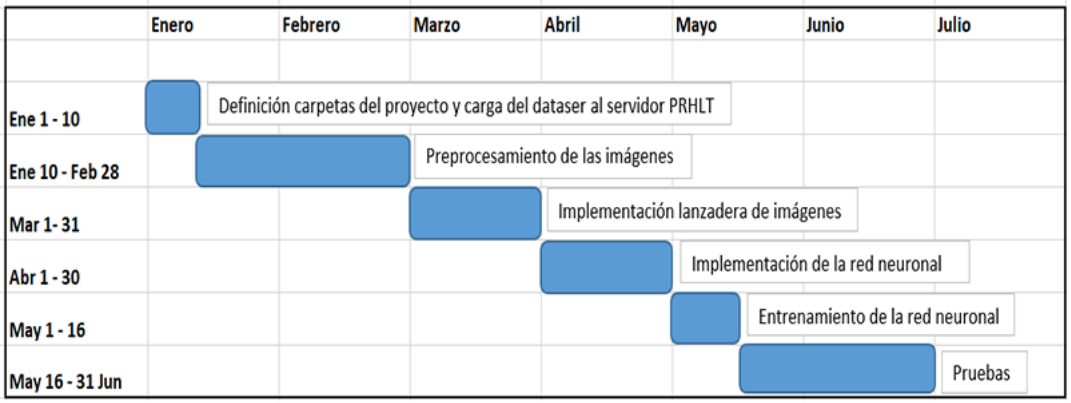


**Figura 1.2:** Diagrama de Gantt, Objetivos del proyecto

## 1.3 Estructura de la memoria

El trabajo se estructura en ocho capítulos.

En el primer capítulo se introduce el trabajo realizado, abordando las motivaciones más significativas para la elección del tema y los objetivos marcados para la consecución del proyecto.

En el segundo capítulo se aborda el marco teórico del aprendizaje automático, haciendo hincapié en los conceptos teóricos más relevantes que afectan al proyecto.

En el tercer capítulo se trata el estado de la cuestión, se investiga las herramientas actuales para la segmentación semántica, se describe la arquitectura modular y se analiza cada uno de sus puntos fuertes y débiles.

En el cuarto capítulo se marcan los requisitos que debe de tener la solución del presente proyecto para lograr los objetivos.

En el quinto capítulo se tratan todas las herramientas utilizadas y recursos necesarios para el desarrollo del trabajo.

En el sexto capítulo se muestra el trabajo desarrollado para el preproceso de los datos, la implementación de la red, el proceso de entrenamiento y evaluación de los modelos generados.

En el séptimo capítulo se realizan las pruebas necesarias para probar el rendimiento de la solución.

En el octavo y último capítulo se concluye el trabajo.

# CAPÍTULO 2
# Marco teórico

## 2.1 Aprendizaje automático

### 2.1.1. Fundamentos

El aprendizaje automático *machine learning* es una de las ramas de la inteligencia artificial que dota a la computadoras el poder del autoaprendizaje para la resolución de problemas. El término aglutina diferentes disciplinas que se desarrollan con anterioridad y que participan en la acuñación del término en los años 80, siendo las más importantes el reconocimiento de patrones *pattern recognition* y la inteligencia artificial.

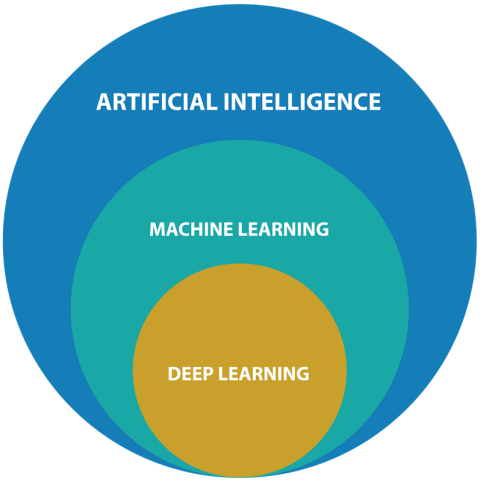


**Figura 2.1:** Artificial Intelligence. Medium

Se enumeran las tres técnicas de aprendizaje automático más importantes:

- Aprendizaje supervisado
- Aprendizaje no supervisado
- Aprendizaje semi-supervisado

En primer lugar se observan los sistemas con aprendizaje supervisado donde no se tiene conocimiento previo sobre el problema a resolver y se aprende mediante datos de entrenamiento de entrada y salida.

Se clasifican en cuatro tipos, inductivos, de memorización, deductivos y por analogía observados en problemas de clasificación o segmentación de imágenes.

Seguidamente se identifican los sistemas de aprendizaje no supervisado donde los datos de entrenamiento utilizados solo tienen datos de entrada, cuyo aprendizaje da como resultado un agrupamiento de muestras de datos en diferentes grupos o *clusters* que se clasifican en dos tipos, los jerárquicos y no jerárquicos.

Los clusters jerárquicos no precisan el número de *clusters* como punto de partida para ser encontrados, los algoritmos detectan todos los grupos posibles en los datos, en cambio en los no jerárquicos los algoritmos disponen del número de grupos a detectar como premisa.

En último lugar se contemplan los sistemas con aprendizaje semi-supervisado, un planteamiento que comparte características de los dos tipos enumerados en los dos párrafos anteriores, en los cuales los algoritmos disponen de datos etiquetados y no, buscando ser más efectivos que con un planteamiento supervisado o no supervisado.

La mejora de los algoritmos y el aumento del poder computacional a lo largo de los años ha permitido que el uso y aceptación del aprendizaje automático sea generalizado dentro del campo de la investigación y en el mundo empresarial, identificando los siguientes cuatro campos de de aplicación más importantes:

- Reconocimiento de imágenes y vídeo
- Reconocimiento de audio
- Reconocimiento de texto
- Análisis de datos

El reconocimiento de imágenes se aborda dentro del campo de la visión por computador y dota a cualquier sistema del poder de percepción de su entorno, siendo aplicado en tareas de reconocimiento facial en dispositivos móviles, en la conducción autónoma de vehículos o para la detección de enfermedades en imágenes médicas, siendo las arquitecturas convolucionales las más utilizadas para la resolución de problemas.

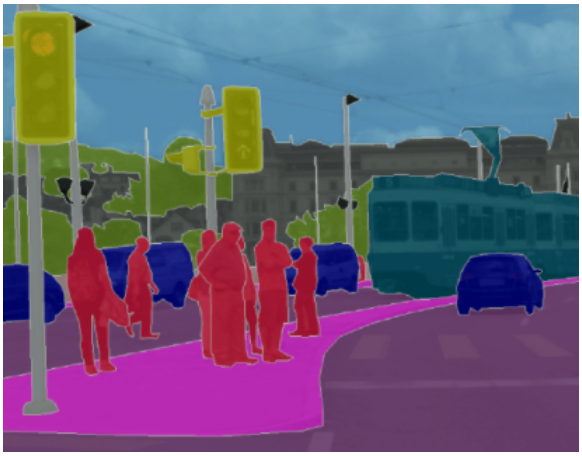

**Figura 2.2:** Ejemplo de segmentación semántica. Analyticsindiamag

El reconocimiento de texto también se engloba dentro del campo de la visión por ordenador, pero también en el procesamiento del texto natural *Natural Language Processing*

que se utiliza para digitalizar libros de formato físico a digital mediante el reconocimiento de texto manuscrito con técnicas de reconocimiento óptico de caracteres *OCR* con arquitectura convolucional o mediante el uso de redes recurrentes *RNN* o redes *Transformers* como estado de la cuestión.

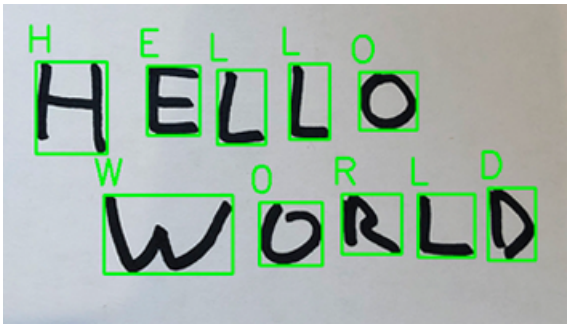


**Figura 2.3:** Ejemplo de reconocimiento de caracteres *(OCR)*. Pyimagesearch

El reconocimiento de audio realiza un proceso de transformación de audio a texto mediante arquitecturas convolucionales que detectan patrones con mucha efectividad, utilizado en dispositivos móviles para transcripción a texto o en la traducción instantánea, también en los sistemas de detección de voz para el control de dispositivos domóticos o para conocer el estado anímico de las personas en una conversación.

Una vez transformados los datos a texto se utilizan las arquitecturas de redes recurrentes utilizadas en el reconocimiento de texto descritas en el párrafo anterior.

El campo de aplicación descrito ha hecho posible la utilización de dispositivos de personas con diversidad funcional de diferente índole, pudiendo adaptar la tecnología a las necesidades del usuario.

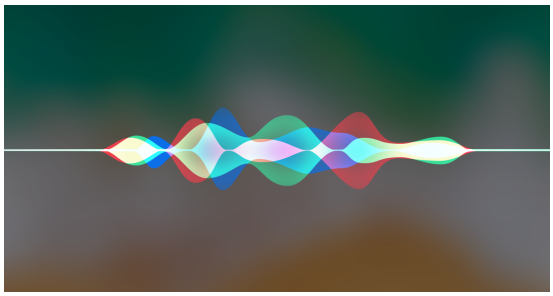

**Figura 2.4:** Ejemplo de reconocimiento de voz, Apple *(SIRI)*. Apple

Por último, el análisis de datos permite analizar gran cantidad de datos y detectar patrones en ello, utilizando algoritmos de clustering y ciencia de datos, siendo utilizado por grandes empresas para sus decisiones directivas o por equipos deportivos para fichar nuevos jugadores o estudiar a rivales por parte de los equipos técnicos.

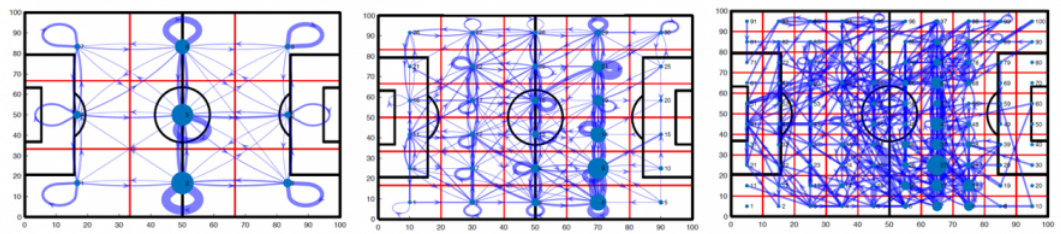

**Figura 2.5:** Ejemplo de reconocimiento de patrones de movimiento en un partido de fútbol. Laboratory of Biological Networks, Center for Biomedical Tecnhology (UPM)

El uso de datos está aumentando debido a que las grandes organizaciones están dedicando mayores esfuerzos en la recopilación de los mismos, cuidando la calidad y el etiquetado.

Todo ello es importante, pero sin olvidar la necesidad de poder computacional para el entrenamiento de arquitecturas más complejas que han ayudado al auge del *Cloud Computing* y la publicación de librerías como *Eddl* [3] o *TensorFlow* [4] reduciendo la complejidad matemática en la implementación de grandes soluciones.

### 2.1.2. Perceptrón

El perceptrón es un modelo probabilístico acuñado por Frank Rosenblatt a finales de los años 50 [6] basado en los trabajos previos desarrollados por McCulloch, Pitt y Hebb en los que se define el concepto de neurona artificial y su entrenamiento para desarrollar tareas de clasificación.

El modelo tiene tres principales objetivos: Entender cómo el mundo biológico detecta el mundo físico, la metodología seguida para almacenar la información y si los datos almacenados influyen en el reconocimiento del entorno físico.

Mediante el uso de un algoritmo el modelo realiza un proceso de aprendizaje en el que va actualizando su estado interno para converger en última instancia a una solución.

Una vez realizada la convergencia del algoritmo, el modelo puede ser utilizado para resolver problemas de clasificación.

El modelo intenta emular el funcionamiento biológico de las neuronas y la sinapsis, dando forma a la siguiente arquitectura:

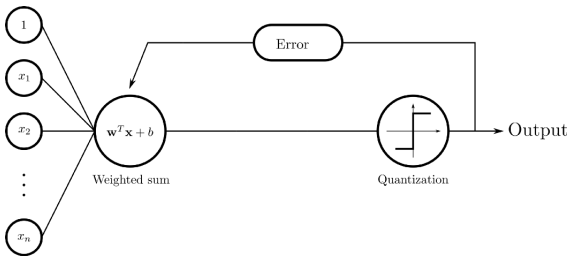


**Figura 2.6:** *Rosenblatt's Perceptron*, Towards Data Science

En primera instancia se observa la entrada del perceptrón, punto dónde el modelo recibe la información del mundo real cuyos datos de entrada se formatean en vectores y contienen en primer lugar por los datos de la muestra y en última instancia la etiqueta con la clase que identifica a la muestra.

Seguidamente se identifica la neurona, formada por una función discriminante lineal, una matriz de pesos y el parámetro sesgo *bias* para el entrenamiento.

En último lugar, la función discriminante lineal da como resultado un valor codificado que identifica la clase clasificada que en el caso que la clasificación sea errónea, la matriz de pesos de la neurona se actualiza.

El algoritmo se ejecuta en bucle hasta que se detiene cuando durante una etapa de entrenamiento no se produce ningún error de clasificación.

Es importante señalar que los datos deben de ser linealmente separables para la convergencia efectiva del algoritmo, por dicho problema se produce la época oscura de las redes neuronales, ya que no se contaba con los algoritmos necesarios para separar datos que no son linealmente separables.

### 2.1.3. Red neuronal

Las redes neuronales *NN* son un modelo conexionista basado en el funcionamiento del perceptrón que interconexionan cada una de estas pequeñas unidades, conformando una red de neuronas.

Después del planteamiento inicial del perceptrón surgieron problemas que se solucionaron con los años y que ayudaron a impulsar el planteamiento de red neuronal.

En un primer momento Marvin Minsky y Seymour Papert demostraron en el año 1969 el perceptrón no converge a una solución con los datos linealmente no separables [20].

En la década de los 70 se desarrollaron los algoritmos de cómputo matemático descenso por gradiente para poder conexionar perceptrones en cascada y poder ser entrenados, pero el costo computacional era bastante alto para poder llevarlo a cabo por los ordenadores de la época por ello se originó una época oscura dentro del campo de la inteligencia artificial.

En última instancia la invención del algoritmo de retropropagación *Backpropagation* permitió realizar el cómputo matemático de manera menos costosa promoviendo el avance en el desarrollo de las redes neuronales y realzando de nuevo el interés por dicho campo, después de la época oscura.

En la Figura 2.7 se observa la arquitectura de la red neuronal:

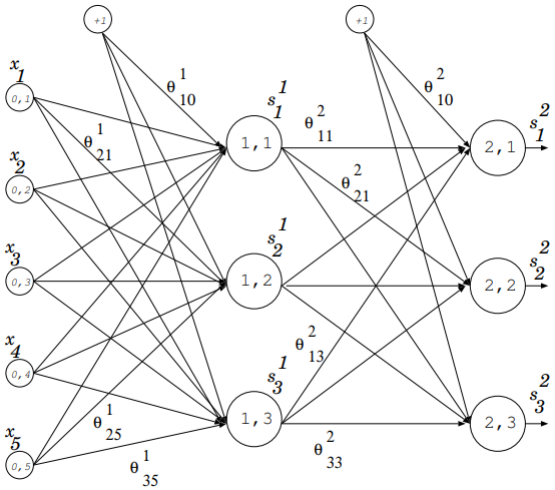


**Figura 2.7:** Red Neuronal, Asignatura de aprendizaje automático, *ETSINF*

La red neuronal se construye en forma matricial para facilitar el cómputo de los cálculos en el entrenamiento y minimizar el uso de memoria dentro del computador.

El proceso de entrenamiento se realiza en dos fases, la propagación hacia adelante y la retropropagación.

En la propagación hacia adelante o *(Forward Propagation)* cada neurona utiliza una función de activación y la matriz de pesos almacenada para calcular el valor de salida, todo antes de ser propagado a la capa siguiente.

Cuando las últimas neuronas propagan el resultado a la salida se calcula el gradiente de cada capa con el algoritmo de retropropagación *(Back Propagation)* y en dicho proceso se calcula el valor de la matriz de pesos de las neuronas que minimizan el error a la salida de la red, actualizando los valores de la matriz.

Actualmente las redes neuronales se utilizan en numerosas aplicaciones, especialmente en visión por computador, en el reconocimiento de lenguaje natural humano o en la detección de patrones en los datos.

### 2.1.4. Red convolucional

Una red convolucional *(CNN)* es un tipo de red neuronal utilizada para la detección de patrones en datos sin ser previamente identificados por el ser humano, por ello el uso de dicha arquitectura se ha popularizado en los últimos años gracias al aumento del poder computacional de los ordenadores, a la hora de realizar las operaciones matemáticas.

En la primera parte de la red las neuronas se especializan en la detección de patrones concretos existentes en la imagen, como localizar bordes, líneas verticales u horizontales. Si alguna de las neuronas detecta dichas características la función de activación de la neurona propaga el valor calculado a la capa posterior.

La detección previa de características realizadas por la red permite la automatización de la labor que con anterioridad era competencia absoluta del ser humano.

Una vez detectadas las características, las neuronas encargadas en la detección de patrones propagan los valores computados a una red neuronal posterior que ya se encarga exclusivamente de la tarea de clasificación o segmentación.

La aplicación de dicha arquitectura neuronal no sólo ayuda a detectar patrones en imágenes, también es efectiva en la detección de características en audio y vídeo.

En la Figura 2.8 se observa la arquitectura de una red convolucional:

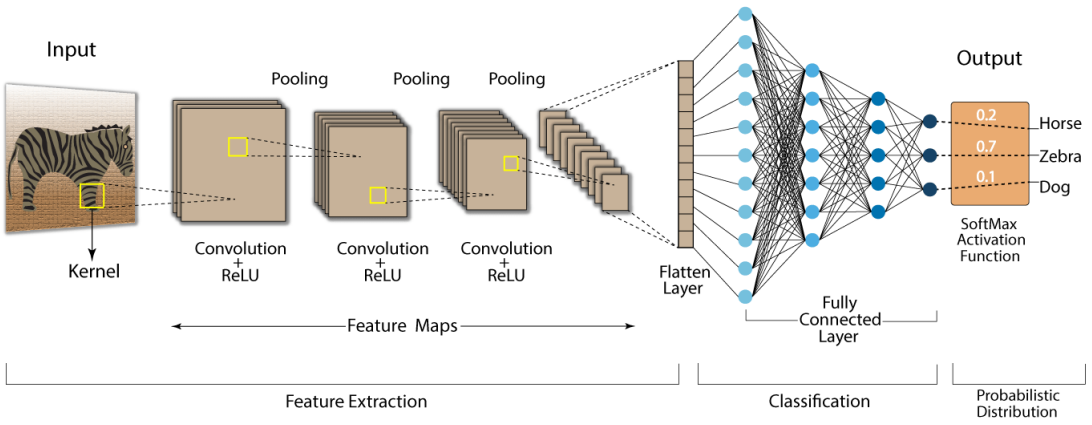


**Figura 2.8:** *Convolution Neural Network*, Developers Breach

Para la detección de características las neuronas utilizan las siguientes cinco operaciones:

- Convolución y deconvolución.
- Función activación *ReLU*
- *Pooling y UpSampling*

En primer lugar, la operación de convolución se encarga de aplicar un filtro o *kernel* para operar con una imagen de entrada y dar como resultado a la salida una imagen convolucionada, en la cual dependiendo del filtro utilizado se genera un mapa de características diferente.

Dado que una imagen tiene mucha información píxel a píxel para detectar características concretas hay que utilizar dichos filtros y así focalizar en unos rasgos seleccionados como la detección de bordes, importante para la tareas de visión por computador, discriminando el resto de información no relevante para la clasificación de dicha característica concreta.

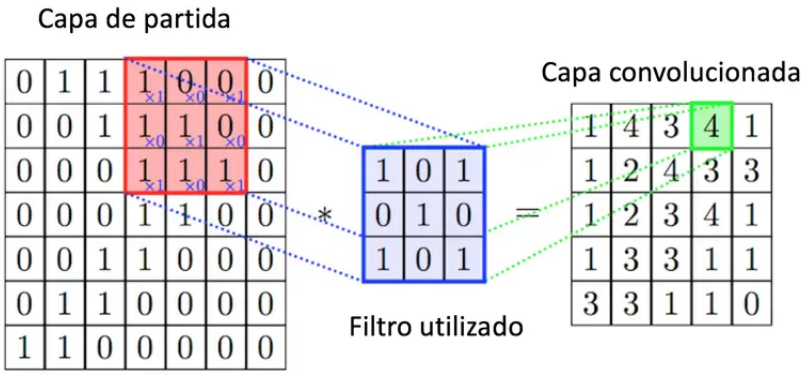


**Figura 2.9:** Convolución, Diego Calvo

En las arquitecturas de redes convolucionales, una vez se ha realizado el proceso de convolución en la fase de codificación, se debe de generar una imagen con las mismas dimensiones que la original, por ello una vez detectadas las características se utiliza la operación de convolución transpuesta o deconvolución.

Las operaciones convolucionales y deconvolucionales están acompañadas de funciones de activación *ReLU* se aseguran de que solo los filtros que detectan características propaguen datos a las capas siguientes, ya que dicha función iguala a cero todo valor negativo.

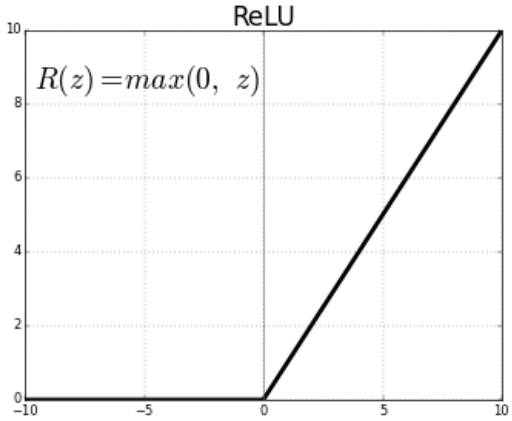


**Figura 2.10:** Función ReLU, ml4a

En último lugar la operación *Pooling* reduce la dimensionalidad de los mapas de características resultantes de las operaciones de convolución y de la función *ReLu*, facilitando así la detección de características más importantes detectadas, con un coste computacional reducido.

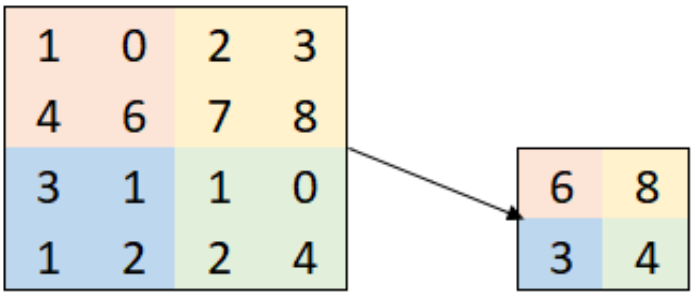


**Figura 2.11:** Operación de *Pooling*

En la Figura 2.11 se observa que en el resultado se ha obtenido el valor máximo de píxel observado en cada área de color.

Como ha ocurrido con las operaciones de convolución, las imágenes necesitan recobrar las dimensiones de la imagen original, por ello una vez se recorre las capas profundas de la red que dan como resultado una máscara reducida, se realiza el proceso de decodificación donde se efectúan las operaciones de deconvolución para reducir el número de filtros y el *UpSampling* para aumentar la dimensión de las imágenes, así hasta lograr la imagen requerida a la salida de la red.

En el desarrollo del proyecto se utiliza una red convolucional de arquitectura *U-Net* con diferentes variantes para ver el rendimiento de cada una de ellas.

### 2.1.5. Red *U-Net*

La red *U-Net* es una arquitectura de red convolucional propuesta por Olaf Ronneberger, Philipp Fischer, y Thomas Brox en el año 2015 para la segmentación de imágenes dentro del campo de la biomedicina, en el artículo *U-Net: Convolutional Networks for Biomedical Image Segmentation* [9].

La arquitectura fue propuesta para la participación en el *EM segmentation challenge* en el año 2015 obteniendo el menor *Warping Error* de todos los participantes, demostrando el rendimiento a de la arquitectura para realizar tareas de segmentación en multitud de campos médicos.

La arquitectura utiliza las operaciones descritas en la red convolucional para detectar los patrones en la imagen original y generar a la salida de la red una máscara segmentada en un proceso de codificación y otro de codificación.

La arquitectura de la red se puede observar En la Figura 2.12:

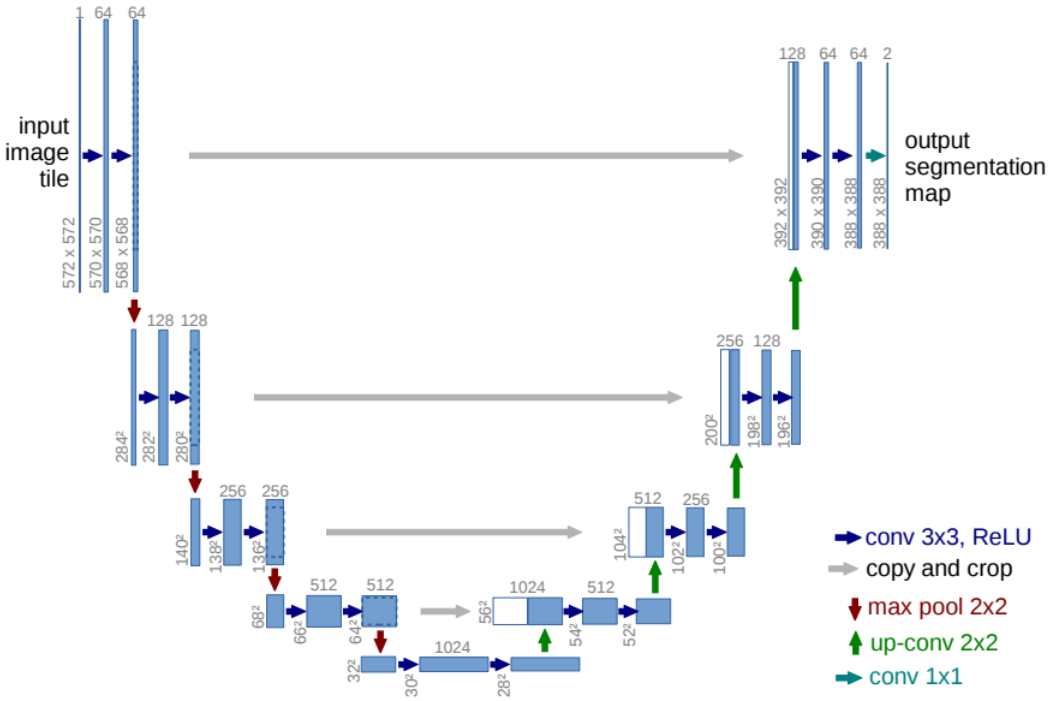


**Figura 2.12:** Arquitectura red convolucional, *U-Net*: Convolutional Networks for Biomedical Image Segmentation

En primer lugar en el área izquierda de la arquitectura se realizan operaciones convolucionales junto la función de activación *ReLU*, seguida de la operación *max pooling*, duplicando el número de canales o filtros de los mapas de características generados y reduciendo también las dimensiones.

Como se ha descrito dentro del punto de las redes convolucionales el uso de dichas operaciones permite detectar características importantes de la imagen y restablecer las dimensiones de la imagen original para generar una clasificación píxel a píxel a la salida.

Una vez realizadas las primeras operaciones finalizando la etapa de codificación se realiza la fase de decodificación en el área derecha la arquitectura, realizando las operaciones de deconvolución junto a la función de activación *ReLU* y las operaciones de *UpSampling* para regenerar las dimensiones de la imagen original.

Seguidamente se realizan las últimas operaciones convolucionales para la generación de la segmentación a la salida de la red, que dependiendo la tarea a realizar se configura de una manera u otra para obtener a la salida la información deseada.

En último lugar en cada una de las fases se contempla un camino *Copy and crop* que conecta las capas de codificación con las de decodificación, mejora propuesta en el artícu-

lo *The Importance of Skip Connections in Biomedical Image Segmentation* en el año 2016 [11] dónde se demuestra que mejora el rendimiento de la red.

La segmentación de la red da como salida una imagen con la mismas dimensiones que las imágenes de entrada pero en la que se clasifica cada píxel con la clase correspondiente, en el caso de detectar el objeto corresponde a un píxel blanco, en el caso opuesto de no detectar nada el píxel es de color negro.

La imagen de entrada tiene que ser de igual dimensión que las imágenes de salida para poder visualizar sobre la imagen original el área exacta donde se han detectado las características aprendidas.

## 2.2 Contenedores de imágenes

Para el desarrollo del proyecto se han utilizado diferentes contenedores de imágenes en los cuales se almacena la información utilizada para su preproceso, el entrenamiento y evaluación de los algoritmos de aprendizaje automático.

### 2.2.1. WSI *(Whole Slide Images)*

WSI es un contenedor de imágenes utilizado por los equipos de electromedicina desde los años 90, donde los escáneres obtienen la información directamente de los portaobjetos donde se introducen los componentes biológicos, adquiriendo la información en alta resolución.

El escáner lleva incorporado un robot que es el encargado de capturar las pequeñas áreas del portaobjetos, generando pequeños recortes llamados *slides* y viene dada por la naturaleza del escáner a la hora de obtener la información.

Para visualizar la imagen global escaneada se unen las *slides* mediante el uso de algoritmos para crear una imagen completa de alta calidad.

Las imágenes médicas del dataset aportadas por la organización y utilizadas en el presente proyecto tienen el formato WSI.

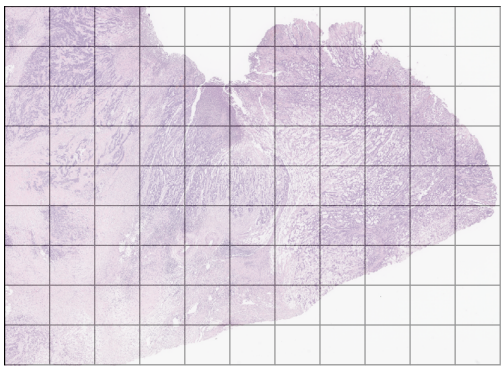

**Figura 2.13:** Formato de imagen *WSI*

Dado el peso de cada imagen para la manipulación de cada una de ellas y las limitaciones en el hardware se utilizan otros formatos de imágenes para poder manipularlas y utilizarlas en el proyecto como el formato *PNG* o *TIFF*.

### 2.2.2. *PNG (Portable Network Graphics)*

*PNG* es un formato de imagen estática que se presenta en el año 1996 cuyo algoritmo se basa en el trabajo de Jacob Ziv y Abraham Lempel relacionado con la comprensión secuencial de datos en el año 1977 [8]. Permite reducir el tamaño del archivo realizando una compresión sin pérdida.

Su nacimiento viene motivado por las limitaciones en el número de colores en la paleta del formato GIF y las limitaciones derivadas de la propiedad intelectual, que no permitían utilizar el formato con total libertad.


**Figura 2.14:** Formato de imagen *PNG*

Las imágenes en formato *PNG* se pueden configurar en tres canales RGB (Rojo, Verde, Azul) y permite añadir un cuarto canal con la funcionalidad de transparencia en las imágenes.

En el proyecto el formato *PNG* es utilizado por las imágenes médicas del dataset para el entrenamiento de la red neuronal y para las máscaras generadas.

### 2.2.3. *TIFF (Tagged Image File Format)*

*TIFF* es un formato creado en los 80 por Aldus y Microsoft para almacenar imágenes sin embargo actualmente pertenece a Adobe Systems.

Es un formato que se diferencia del resto por suprimir la cabecera de información y sustituirla por etiquetas, algo que permite añadir información de manera flexible

Permite utilizar diferentes algoritmos de compresión sin pérdida de información, cuando otros formatos solo pueden utilizar un esquema de compresión determinado.

Las unidades de información de la imagen se pueden formatear en diferentes tipos de datos y añadir los canales que se necesiten arbitrariamente, algo que ha propiciado que sea el formato elegido por el mundo de la ciencia para almacenar datos.

En el proyecto el formato *PNG* es el utilizado por las máscaras del dataset.

**Figura 2.15:** Muestra de una máscara del dataset

## 2.3 Representación de modelos

### 2.3.1. *ONNX (Open Neural Network Exchange)*

Es un formato de representación de modelos de aprendizaje automático de código abierto que permite almacenar en un archivo modelos generados, permitiendo importar con facilidad dichos modelos desde código y promover su distribución libre entre profesionales del sector de la inteligencia artificial.

El formato permite cargar un modelo con los pesos resultantes del entrenamiento incluidos desde el código implementado, para construir posteriormente de nuevo la red y utilizarla.

En el presente proyecto se ha utilizado el formato *ONNX* para almacenar los modelos entrenados etapa a etapa para su posterior evaluación, donde el tamaño en memoria ocupado por cada modelo *U-Net* entrenado junto a sus pesos es de aproximadamente unos 90MB.

## 2.4 Objetos

### 2.4.1. Tensor

Un tensor es un objeto matemático utilizado por las principales librerías de aprendizaje automático para el formateo de datos de entrada y salida en los modelos de aprendizaje automático utilizados, promovido por la librería *TensorFlow* de Google.

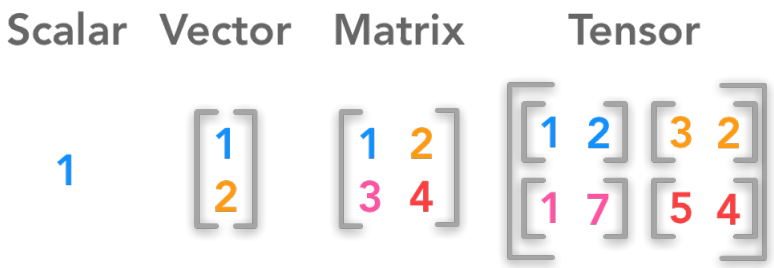


**Figura 2.16:** *Scalar, Vector, Matrix, Tensor*. Hadrienj.

Dicho objeto permite introducir en una misma estructura de datos un lote completo de imágenes con sus respectivos canales y dimensiones, facilitando significativamente la introducción de datos en los modelos y así realizar las operaciones matemáticas correspondientes con un coste computacional reducido.

En el presente proyecto los tensores se utilizan para la entrada de la red neuronal y en la segmentación semántica generada por el modelo, pudiendo realizar operaciones sencillas de conversión a *NumPy Array* para su posterior transformación en imágenes.

## 2.5 Funciones de pérdida

El proceso de entrenamiento de la red neuronal precisa internamente de estimadores para la convergencia de los algoritmos durante el proceso del descenso por gradiente, identificando las siguientes dos funciones, el error cuadrático y la entropía cruzada.

### 2.5.1. MSE

El error cuadrático medio es una función de pérdida que se utiliza para comprobar el poder de predicción del modelo entre el valor real y el estimado utilizado en el proceso de descenso por gradiente, en el cual se obtienen los pesos que minimizan el error a la salida del modelo.

Es una función simple para realizar operaciones computacionales.

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

**Figura 2.17:** Función de pérdida, error cuadrático medio (MSE)

La variable $Y_i$ es la muestra de referencia y la $\hat{Y}_i$ es la variable estimada, todas las diferencias se calculan al cuadrado y se suman, para luego obtener la media del error.

Dada la naturaleza de la ecuación el error cuadrático medio siempre es positivo, siendo el valor más próximo a cero el esperado para observar la convergencia del algoritmo.

Dicha función de pérdida se utiliza en el proyecto para los modelos que utilizan una función de activación tipo *sigmoid* a la salida de la red.

Los problemas de regresión lineal suelen utilizar mayoritariamente la función de pérdida del error cuadrático medio.

### 2.5.2. Entropía Cruzada

La entropía cruzada *(Cross Entropy)* es una función de pérdida que calcula el rendimiento de un modelo de clasificación probabilístico propuesta para la tarea de entrenamiento del gradiente en el artículo *Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels* de Zhilu Zhang Mert y R. Sabuncu en el año 2018. [16]

Surge como alternativa al *Mean Squared Error (MSE)* y *Mean Absolute Error (MAE)* para el cálculo del gradiente dentro del entrenamiento de modelos de aprendizaje automático, dado que los modelos profundos que utilizan dichas métricas necesitan grandes fases de entrenamiento para lograr un gran rendimiento, demostrando que la entropía cruzada es una alternativa válida para acortar la fase de convergencia de los algoritmos.

Poniendo de ejemplo un problema de clasificación el parámetro $p$ corresponde con la probabilidad que tiene una muestra de pertenecer a una clase específica y la variable $q$ la probabilidad estimada por el modelo de pertenencia a cada una de las clases.

$$H(p, q) = -\sum_{x} p(x)\, \log q(x).$$

**Figura 2.18:** Función de pérdida, entropía cruzada

La función recorre todas las clases y multiplica la probabilidad de que una muestra pertenezca a dicha clase por el logaritmo de la probabilidad estimada por el modelo, realiza un sumatorio de todo ello, donde el valor de entropía cruzada aumenta si la diferencia de probabilidad entre la etiqueta original y la estimada difieren.

El valor resultante está comprendido entre cero y uno, siendo el cero el valor de mayor verosimilitud.

Dicha función de pérdida se utiliza en el proyecto para los modelos que utilizan una función de activación tipo *softmax*, generalmente en los problemas de clasificación.

## 2.6 Métricas

Las métricas son utilizadas para comprobar la similitud entre una muestra de referencia y la imagen estimada, dentro del campo de la visión computador en las tareas de segmentación semántica, utilizando en el proyecto la puntuación *F1* y el índice de Jaccard *IoU*.

### 2.6.1. Coeficiente de Sorensen-Dice *(F1-Score)*

Métrica definida en los trabajos de Thorvald Sørensen [17] y Lee Raymond Dice [19] en la década de los 40, cuya métrica se utiliza para comprobar la similitud de dos muestras teniendo en cuenta dos variables, la precisión y el *recall*.

Las dos variables de la función se obtienen gracias a la casuística observada En la Figura 2.19:

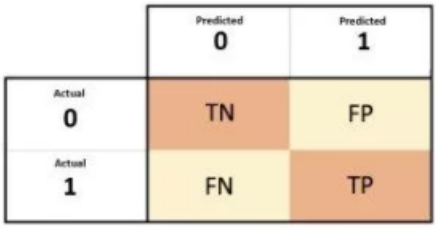


**Figura 2.19:** *F1-Score*, The Data Scientist

La precisión se utiliza para observar en un conjunto de predicciones realizadas por un modelo, cuantas son realmente positivas, ya que al observar la fórmula de la Figura 2.20 se comprueba que si se incrementan los casos de falsos positivos, el valor se decrementa, demostrando que el modelo no es preciso.

$$Precisión = \frac{TP}{TP + FP}$$

**Figura 2.20:** Cálculo de la precisión

Su uso es bastante relevante por ejemplo, para observar el buen funcionamiento de los filtros de *e-mails spam* de los clientes de correo, a la hora de desechar el correo que realmente útil para el usuario.

En el caso del *recall* se utiliza para medir de todo el conjunto de positivos reales cuantos se han predicho como falsos positivos, observando el cálculo en la la Figura 2.21.

$$Recall = \frac{TP}{TP + FN}$$

**Figura 2.21:** Cálculo del *recall*

El *recall* es una buena métrica para ver si en un hospital donde se utilizan modelos predictivos se dan diagnósticos negativos de una enfermedad cuando el paciente la tiene en realidad.

El coeficiente de Sorensen-Dice utiliza los valores de precisión y recall como se indica en la Figura 2.22 para obtener un balance entre ellos:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

**Figura 2.22:** Métrica *F1-Score*

La métrica de puntuación *F1* se puede utilizar en el entrenamiento del modelo de aprendizaje automático implementado en el proyecto dentro del proceso de experimentación de los modelos, estando comprendido en el rango [0, 1] siendo el valor de uno el mejor resultado obtenido por la métrica.

### 2.6.2. *Intersection-Over-Union (Jaccard Index)*

El coeficiente de intersección sobre unión o índice de Jaccard fue propuesto por Paul Jaccard en el artículo *Distribution comparée de la flore alpine dans quelques régions des Alpes occidentales et orientales* [18] del año 1902.

La métrica es generalmente utilizada en problemas de segmentación semántica y es la elegida en el proyecto para efectuar las tareas evaluación de los modelos entrenados y así observar qué tan bueno es un modelo o no para la tarea a realizar, en el caso del proyecto actual en la detección de células cancerígenas.

Para la realización del cálculo las imágenes de entrada se interpretan como arrays, realizando la operación entre cada uno de los píxeles, obteniendo el resultado de la métrica a la salida.

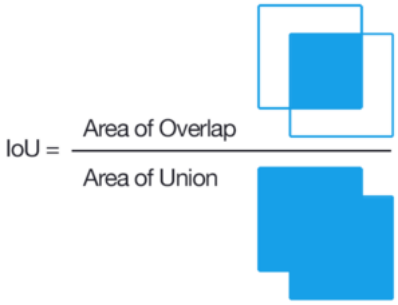


**Figura 2.23:** Métrica *IoU*, Towards Data Science

El numerador corresponde con el área de intersección entre las dos imágenes y el denominador el área de unión entre las dos imágenes, obteniendo la puntuación de la métrica, comprendida entre el valor cero y uno, siendo el valor uno la muestra de que la segmentación realizada es totalmente similar a la imagen original.

Para considerar que un modelo de aprendizaje automático ha realizado una buena predicción el valor resultante de la operación de *IoU* tiene que estar comprendido en el rango aproximado [0.5 - 1].

# CAPÍTULO 3
# Situación actual de la tecnología

## 3.1 *Fast Fully-connected network (FastFCN)*

La red *FastFCN* es una arquitectura propuesta por Huikai Wu, Junge Zhang, Kaiqi Huang en el año 2019 para la segmentación de imágenes [10], en la cual se mejora la red *FCN* original obteniendo un mapa de características de alta resolución con un mayor rendimiento.

Una de las operaciones utilizada por una de las actualizaciones posteriores de la red original FCN *(DilatedFCN)* es la convolución dilatada que precisa de una alta complejidad computacional y ocupa mucho espacio en memoria, por ello proponen una actualización de la red original para solucionar el problema existente, ya que ello reduce significativamente la posibilidad de ser utilizada en un gran número de aplicaciones a tiempo real.

A la arquitectura *DilatedFCN* se le añade un módulo nuevo llamado *Joint Pyramid Upsampling (JPU)* que reduce la complejidad computacional y la memoria necesaria, por consiguiente mejora en gran medida el rendimiento de la red tanto en tiempo de ejecución como en utilización de espacio en memoria.

En la Figura 3.1 se observa la arquitectura y sus diferentes módulos:

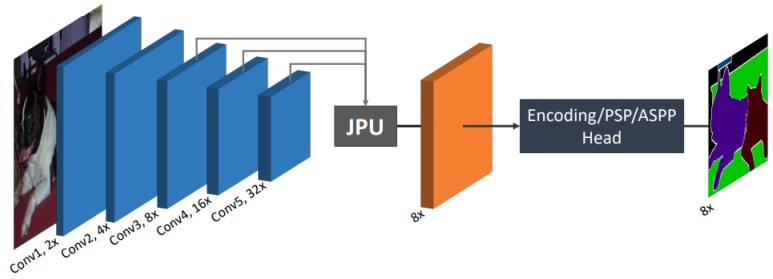


**Figura 3.1:** *Fast Fully-connected network, Institute of Automation, Chinese Academy of Sciences*

En una primera fase de la red se aplican operaciones de convolución reduciendo las dimensiones de la imagen original y duplicando el número de filtros y conectando las últimas tres convoluciones con el módulo *JPU*.

En la parte central se observa el módulo *Joint Pyramid Upsampling (JPU)* que recibe a la entrada una imagen en baja resolución y una imagen de guia en alta resolución que

transfiere a la imagen en baja resolución detalles y estructuras, obteniendo a la salida una imagen en alta resolución.

En última instancia se utiliza una cabecera codificadora convolucional, *Pyramid Scene Parsing Network* PSP [13] o *Atrous Spatial Pyramid Pooling* ASPP [12] para obtener la segmentación semántica en alta resolución a la salida de la red.

El funcionamiento de la arquitectura *FastFCN* se puede observar En la Figura 3.2:

**Figura 3.2:** Funcionamiento arquitectura *FastFCN* con el *benchmark PASCAL VOC 2012*

Para comprobar el rendimiento de la arquitectura propuesta utiliza el dataset de prueba *PASCAL VOC 2012*, obteniendo los siguientes resultados con la métrica *mIoU*.

| Dataset | Resultados |
|---|---|
| PASCAL VOC 2012 | 53.13 % |

**Tabla 3.1:** Resultados evaluación *FastFCN* con el *benchmark PASCAL VOC 2012*.

## 3.2 *Gated Shape CNNs*

*Gated-SCNN* es una arquitectura propuesta en el año 2019 para realizar tareas de segmentación semántica dentro del campo de la visión por computador [14], que funciona mejor que el resto de métodos del estado del arte como la arquitectura *PSP-Net* [13] y *DeepLabV3+* de Google [15].

En arquitecturas anteriores se presentan diversos problemas en tareas de segmentación, como la pérdida de detalle en distancias largas y el ruido existente en los límites de cada objeto visible, algo que limita su poder y su aplicación en la resolución de problemas reales, como por ejemplo la conducción autónoma de vehículos a gran velocidad, por ello la presente configuración minimiza los problemas significativamente.

La arquitectura de la red está formada por dos cadenas que trabajan en paralelo seguidas de un módulo de fusión.

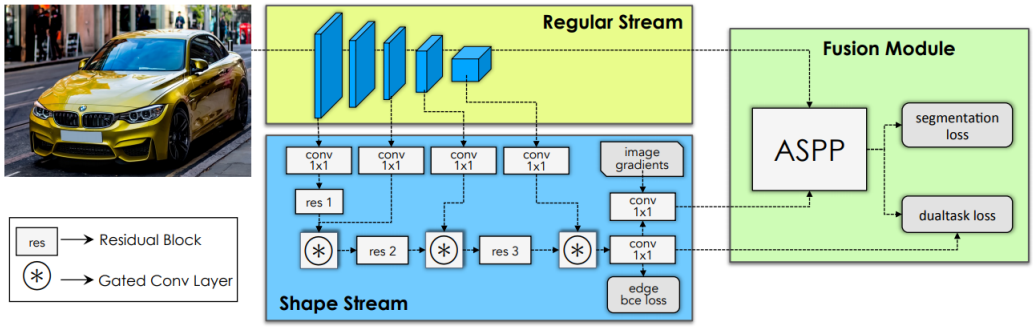


**Figura 3.3:** Arquitectura, *Gated-CNN*

La primera cadena es la regular y está formada por una segmentación clásica mediante arquitectura convolucional *ResNet*.

La segunda cadena de nombre *Shape* se encarga de descubrir los límites semánticos de cada imagen de los objetos que observa en las imágenes identificando los bordes de cada objeto, todo ello mediante bloques residuales intercalados junto a capas de convolución cerrada *GCL*, dotando al módulo de la habilidad de tratar sólo la información relevante para identificar cada una de las formas a clasificar.

En último lugar se identifica el módulo de fusión que recibe la información propagada de los módulos anteriores y los combina, obteniendo una segmentación mucho más exacta, utilizando una arquitectura *ASPP* [12].

Para comprobar el rendimiento de la arquitectura propuesta utiliza el dataset de prueba *Cityscapes*, obteniendo los siguientes resultados con la métrica *Mean Intersection Over Union (mIOU)*:

| Dataset | Resultados |
|---|---|
| Cityscapes | 82,8 % |

**Tabla 3.2:** Resultados evaluación *Gated Shape CNNs* con el *benchmark Cityscapes*.

El funcionamiento de la arquitectura con el dataset de pruebas *Cityscapes* se puede observar En la Figura 3.4:

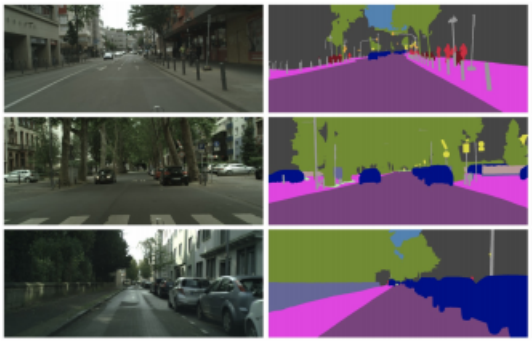

**Figura 3.4:** Funcionamiento arquitectura *Gated Shape CNNs* con el *benchmark Cityscapes*

## 3.3 *DeepLabV3+*

La versión avanzada de *DeepLab* tres es una arquitectura propuesta por Google el año 2018 [15] como actualización de la versión anterior llamada *DeepLabV3*, para realizar tareas de segmentación semántica dentro del campo de la visión por computador.

Se basa en una configuración *Encoder-Decoder* junto un módulo *Atrous Separable Convolution*, que combina una convolución simple en profundidad con una de punto a punto.

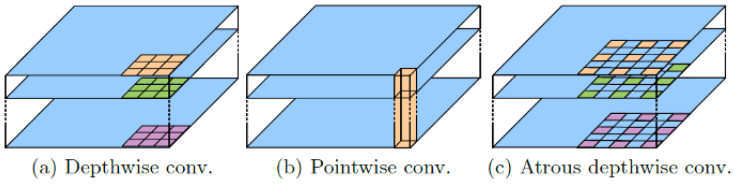


**Figura 3.5:** Ejemplo de *Atrous Separable Convolution*, Medium

La arquitectura tiene tres objetivos principales, el primero reducir la resolución de características, el segundo controlar la existencia de objetos de diferente escalas y el último resolver la falta de rendimiento de la segmentación en los bordes de los objetos.

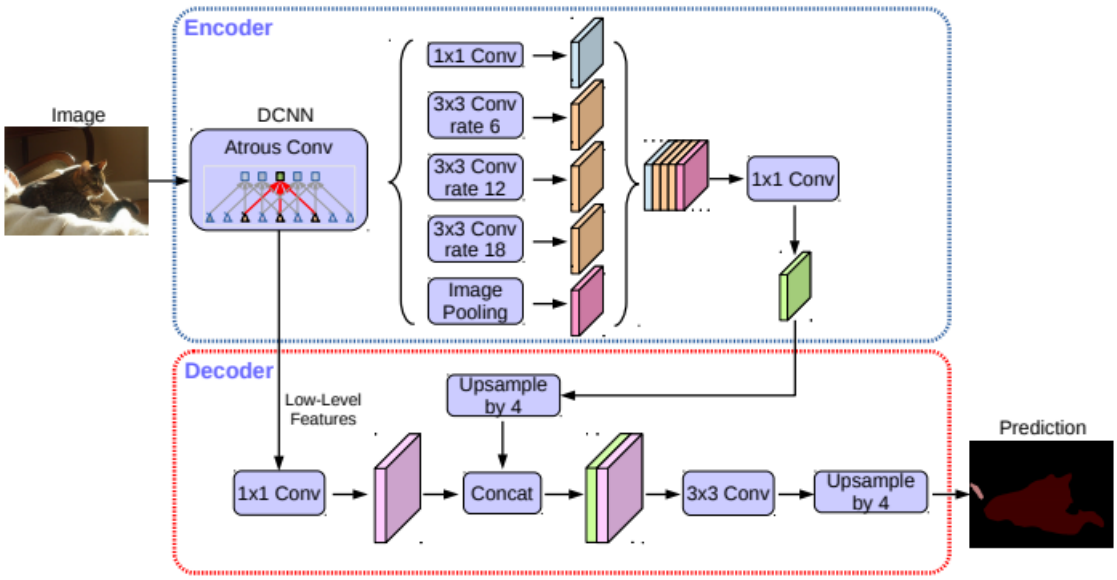


**Figura 3.6:** Arquitectura, DeepLab, Google

En el módulo *encoder* se codifica la información contextual aplicando una arquitectura *Atrous Spatial Pyramid Pooling (ASPP)* con un refinado del resultado de la arquitectura grácias al módulo *decoder*.

La arquitectura propone utilizar convoluciones *Depthwise Separable* para reducir el coste computacional y el número de parámetros a utilizar sin reducir el rendimiento.

Para comprobar el rendimiento de la arquitectura propuesta utilizan los datasets de prueba *PASCAL VOC 2012* y *Cityscapes*, obteniendo los siguientes resultados con la métrica *Mean Intersection Over Union (mIOU)*:

| Dataset | Resultados |
|---|---|
| PASCAL VOC 2012 | 89,0 % |
| Cityscapes | 82,1 % |

**Tabla 3.3:** Resultados evaluación *DeepLabV3+* con los *benchmarks PASCAL VOC 2012* y *Cityscapes*.

El funcionamiento de la arquitectura *DeepLabV3+* se puede observar En la Figura 3.7:

**Figura 3.7:** Funcionamiento arquitectura *DeepLabV3+* con el *benchmark PASCAL VOC 2012*

# CAPÍTULO 4
# Diseño

En el diseño de la solución se implementa una red neuronal convolucional de arquitectura *U-Net* para la segmentación del área afectada por las células cancerígenas.

La arquitectura elegida precisa de unas especificaciones que se deben de cumplir para el correcto funcionamiento de la red.

## 4.1 Especificación de requerimientos

El proyecto se organiza en carpetas para organizar la información de manera adecuada para el desarrollo del proyecto y su posterior evaluación

Las imágenes del dataset original se formatean en WSI y las máscaras en formato XML con unas dimensiones de (93000, 120000) píxeles aproximadamente, las cuales precisan de una resolución alta para que el algoritmo reconozca bien las características a detectar y clasificar.

Se transforman las imágenes WSI a *PNG* aplicando una reducción de escala del L3 logrando imágenes con unas dimensiones de (2800, 3800) píxeles aproximadamente, para poder proceder a realizar el recorte previo para lanzar las imágenes a la red neuronal.

Los ficheros XML se transforman al formato *PNG* con una reducción de escala L3 con unas dimensiones iguales que la imagen de referencia.

Las imágenes L3 se recortan a una dimensión de (128, 128) píxeles, con un desplazamiento horizontal y vertical de 64 píxeles, siendo almacenadas la imagen y la máscara correspondiente en formato *PNG* para poder lanzar las imágenes a la red.

Se configuran seis versiones de arquitectura, que corresponden con las dos versiones *softmax* y *sigmoid* con configuraciones *1a*, *1b* y *2*.

Las máscaras generadas por la red se reconstruyen con una máscara original con dimensiones de (2800, 3800) píxeles aproximadamente, para utilizar la métrica de evaluación *IoU* entre las máscaras originales y las predichas.

Las funciones de pérdida utilizadas para el entrenamiento del modelo serán la entropía cruzada para la versión *softmax*, el *MSE* para la versión *sigmoid* y la métrica *IoU* para el proceso de evaluación de las dos versiones.

## 4.2 Metodología

El objetivo del proyecto es la implementación de diferentes arquitecturas de red neuronal para su entrenamiento y evaluación, para ello se diferencian dos procesos, el primero relacionado con la escritura de la presente memoria y la obtención de bibliografía relevante y el segundo con la implementación del código.

En el primer proceso para la búsqueda de información se ha utilizado la herramienta *Google Académico* en la cual se pueden encontrar documentos de ámbito académico, científico relevantes para la escritura de los diferentes puntos de la memoria.

Para la búsqueda directa de artículos científicos relacionados con las nuevas tecnologías he utilizado el buscador de *ArXiv* de *Cornell University*.

Para la escritura de la memoria se ha utilizado *Overleaf online* sobre la plantilla facilitada por la escuela de informática de la UPV y para la corrección de errores, a parte de las herramientas aconsejadas por la escuela se ha utilizado la tecnología de *Google Docs* para la corrección ortográfica.

En el diseño y modificación de las figuras utilizadas en la memoria se usa el programa *Adobe Photoshop*.

Para la consulta de documentación técnica del uso Overleaf Látex, se ha utilizado un buscador online básico.

Una vez explicado el proceso de escritura de la memoria y las herramientas utilizadas para dicho propósito se define la metodología utilizada para el desarrollo de la red neuronal, que corresponde a un modelo en cascada.

El trabajo que se realiza es secuencial, no se pueden realizar una fase posterior sin la consecución del hito de la fase anterior, por ello la elección del presente modelo.

En la Figura 4.1 se observa el diagrama de la metodología:

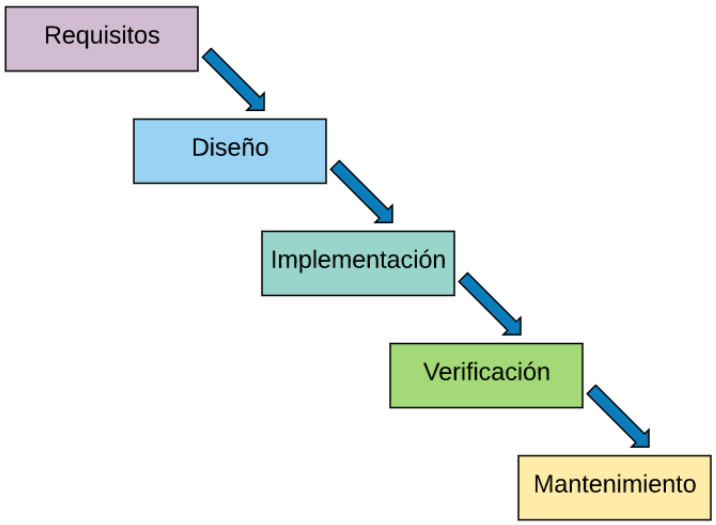


**Figura 4.1:** Metodología cascada, Openclassrooms

En un primer momento se planifica el desarrollo del proyecto especificando los requerimientos y las fases que van a realizarse para su consecución.

Los requerimientos son importantes dada la complejidad de la tecnología utilizada, ya que los datos de entrada del código implementado precisa de unas condiciones de formato, dimensionalidad para que cada una de las piezas del proyecto puedan conectarse y poder evitar problemas de sobreaprendizaje en el entrenamiento de los modelos.

También se definen las herramientas a utilizar en cada una de las fases de diseño para facilitar la búsqueda de las herramientas y la investigación de todas ellas desconocidas al inicio del proyecto.

Una vez realizada la especificación de requisitos se aborda la fase de diseño donde se define lo que se tiene que desarrollar en la fase implementación.

Para el diseño se ha utilizado como herramienta lápiz y papel para hacer esbozos de baja fidelidad, para después ser traducido todo a pseudocódigo, para facilitar la implementación de código.

Al pasar a la fase de implementación se ha utilizado el programa *Visual Studio Code* y *Google Collab* para la escritura de código y la realización de pruebas.

Para la escritura correcta de etiquetas en el código se ha utilizado el traductor *DeepL Traductor* y el corrector *Grammarly*.

La conexión con el entorno de trabajo facilitado por el *PRHLT* se ha utilizado la consola de comandos de Windows y el programa *WinSPC* para la conexión con el protocolo *ssh*.

Cuando se finaliza la implementación de código de los archivos relacionados, se realiza una prueba de funcionamiento para ver si la ejecución del programa es correcta, observando si se cumplen requisitos especificados inicialmente para proseguir con la implementación del código.

Una vez implementado todo el código se ejecuta el proceso de entrenamiento y evaluación de la red neuronal para verificar el correcto funcionamiento de la aplicación.

Se ejecuta el código *Python* en la consola de windows, conectando por *ssh* a los servidores.

En última instancia se contempla la entrega de la solución y la resolución de problemas que se pueden observar en la integración de la solución en diferentes entornos.

# CAPÍTULO 5
# Herramientas utilizadas

## 5.1 Software

### 5.1.1. Látex, *overleaf online*

Para la escritura de la memoria se ha utilizado Latex, un programa muy potente que te permite escribir documentos científicos de manera sencilla, cumpliendo el estándar utilizado para la escritura de trabajos de nuestro campo.

La aplicación Overleaf te permite escribir el trabajo Latex desde cualquier dispositivo móvil, almacenando la última versión en la nube.

Cuenta con una pantalla dónde se guardan los archivos importantes para la memoria, incluyendo las imágenes, en la parte central otra ventana donde se puede escribir el texto, código y a la derecha una tercera pantalla donde se visualiza a tiempo real la vista previa de la memoria en formato pdf.

### 5.1.2. *WinSCP*

Programa para la transmisión de datos entre el servidor y el ordenador. El software dispone de una interfaz con la que puedes manipular y editar archivos directamente, sin necesidad de escribir código por consola. También se dispone de una ventana que te permite observar cuando se realizan las subidas o descargas al servidor, útil cuando quieres tener la seguridad de que la versión que se ejecuta en el entorno está actualizada.

Dado que el proyecto se desarrolla en un entorno cedido por el equipo de investigación *PRHLR* del *DSIC*, he utilizado el programa para transmitir el dataset a dichos servidores.

### 5.1.3. *Google Collab*

Herramienta en línea de google para la creación de cuadernos de código colaborativos en los que se puede escribir código y ejecutar código en python en la nube, sin la necesidad de instalar las librerías más conocidas y utilizadas.

Es una herramienta potente para realizar pruebas de código y comprobaciones de funcionamiento previas de manera rápida y sencilla, para posteriormente pasar el código al entorno natural del proyecto.

La herramienta ha sido muy útil para diferentes asignaturas de la carrera y para la realización del proyecto.

### 5.1.4. *SSH Windows Shell*

Para la ejecución de código del proyecto se ha utilizado la Windows Shell y el protocolo de comunicación SSH. Después de realizar la conexión se utiliza el entorno de anaconda para la descarga de librerías y la ejecución de los programas implementados en el proyecto.

### 5.1.5. *Visual Studio Code*

Programa de edición de código de Microsoft utilizado para la implementación de las soluciones para el proyecto. El programa es muy potente, permite descargarse complementos para la indexación correcta del código y para la depuración de errores.

La utilización de WinSCP te da la opción de seleccionar el editor de código predeterminado, por ello al clicar sobre el programa en la ventana de exploración del programa se abre automáticamente Visual Studio Code.

### 5.1.6. **Asana**

Herramienta online para el desarrollo de diagramas de Gantt que te permite diseñar un eje temporal de objetivos para la organización y dirección de proyectos.

La herramienta ha sido utilizada para crear el diagrama con el eje temporal de objetivos llevados a cabo en el proyecto para conseguir una visualización clara del tiempo que se ha necesitado para la realización de cada uno de los puntos.

La aplicación web permite añadir tareas a realizar con duración, personas implicadas, urgencia y presupuesto necesario, pudiendo visualizar diferentes tipos de diagramas dependiendo de las necesidades del usuario.

### 5.1.7. *Google Scholar*

Buscador de información de ámbito académico y científico para la búsqueda de los artículos relevantes para la escritura del presente proyecto.

Tiene la apariencia del buscador normal pero prioriza la búsqueda de documentos relevantes para el desarrollo del trabajo.

### 5.1.8. *Adobe Photoshop*

Es el programa más utilizado dentro del diseño gráfico y dispone de un gran número de herramientas con las que poder editar imágenes.

Con el programa se han podido confeccionar y redimensionar las figuras utilizadas en la escritura del proyecto.

### 5.1.9. *Grammarly*

Programa de corrección de lengua inglesa que te permite escribir en inglés sin faltas de ortografía, siendo muy efectivas sus correcciones.

El programa permite integrarse dentro del explorador web donde detecta el texto de los cuadros de texto visualizados en cualquier página web, como los de *Google Translator* extendiendo el uso del traductor.

El corrector permite configurar el registro utilizado y te sugiere pequeños cambios para que el texto escrito se ajuste a dicho parámetro.

Dispone de una versión profesional que añade más funcionalidades y recomendaciones de corrección más completas.

Se ha utilizado para la escritura de etiquetas en lengua inglesa dentro del código implementado.

### 5.1.10. *DeepL Translator*

Traductor basado en *Deep Learning* que traduce texto al inglés con con un margen de error muy reducido.

Por defecto se utiliza la versión de prueba que te permite traducir un número exacto de palabras, que al ser utilizado en la red universitaria se sobrepasa, ya que identifica la red a la que te conectas, pero se puede solucionar entrando en modo incógnito con el explorador web o adquiriendo el modo profesional, que no tiene limitaciones de palabras en la traducción.

Se ha utilizado para traducir al inglés los comentarios del código que inicialmente fueron escritos en lengua castellana.

### 5.1.11. Anaconda

Plataforma utilizada en la realización de proyectos de ciencia de datos extendida en todo el mundo de la inteligencia artificial.

Permite crear entornos de ejecución donde descargar versiones concretas de librerías almacenadas en sus respectivos repositorios.

Nace dada la necesidad en la dependencia entre librerías y versiones concretas a la hora de utilizar las funciones que se utilizan en los proyectos de ciencia de datos, pudiendo emular el entorno en otras máquinas, garantizando el funcionamiento correcto de las aplicaciones.

Uno de los problemas más comunes dentro del mundo de la ciencia de datos son que los cambios que se realizan en las librerías provocan la no ejecución de estas por los cambios que se efectúan en las funciones.

La plataforma se instala en los servidores del equipo de investigación *PRHLR* y se utiliza para la ejecución del código en los servidores y la utilización de tarjetas gráficas en el entrenamiento de los modelos.

## 5.2 Lenguajes de programación

### 5.2.1. *Python*

Lenguaje de programación de alto nivel creado por Guido Van Rossum en el año 1991. El nombre hace referencia al gusto del creador por los *Monty Python*.

Es un lenguaje bastante extendido en el mundo del aprendizaje automático por su corta curva de aprendizaje, por la facilidad en la descarga y uso de librerías.

Al ser un lenguaje interpretado la depuración de código es sencilla ya que en caso de error te indica la línea inequívoca de código dónde ha ocurrido dicho problema.

Es un lenguaje con el que me siento familiarizado después de la realización de la rama de computación y tras la realización de diferentes cursos en línea utilizando Python.

## 5.3 Librerias

### 5.3.1. *Pyeddl*

Implementación de la librería *European Distributed Deep Learning Eddl* para Python.

Forma parte del proyecto *Deep Health* desarrollado por diferentes universidades y equipos de investigación europeos para la utilización de la inteligencia artificial en la resolución de problemas dentro del ámbito de la medicina.

Cuenta con las últimas herramientas y modelos del campo del aprendizaje automático para la detección de patrones en imagen médica, por ello se utiliza para la implementación de la red convolucional completa *fully convolutional network U-Net*.

### 5.3.2. *NumPy*

*NumPy* es una librería de python que permite crear vectores o matrices para el almacenamiento y procesamiento de datos dentro del campo científico.

El uso de arrays permite estructurar el valor de los píxeles de las imágenes y sus respectivos canales para el tratamiento de imagen mediante el uso de filtros, para la conversión en imágenes *PNG* o en objetos de tipo Tensor.

En el proyecto se utiliza para la manipulación de imágenes píxel a píxel y para la generación de tensores de entrada a la red neuronal.

### 5.3.3. *OpenCV*

*OpenCV* es una librería que aporta muchas herramientas para el preprocesamiento y tratamiento de imágenes.

No solo se utiliza para leer y crear imágenes a partir de *Numpy Arrays*, también se puede utilizar para detectar formas en imágenes, encontrar formas como rectángulos, cuadrados con unas dimensiones concretas y recortar dichas áreas específicas.

Para el cometido del proyecto se ha utilizado para la carga y descarga de imágenes, para el recorte de imágenes y para el estudio de la importancia del canal *alpha* en el funcionamiento de la red.

### 5.3.4. *Matplotlib*

Librería de python para la creación de gráficas utilizada en el proyecto para la visualización de los resultados de evaluación del proyecto y para visualizar la máscara segmentada superpuesta en la imagen médica.

Mediante el uso de *Matplotlib* se obtiene la serie de datos de la gráfica desde un fichero de texto y formateado en una lista y se diseña la gráfica, pudiendo modificar los colores de las series, la leyenda utilizada, añadir líneas *grid* a la gráfica, demostrando la versatilidad de la librería para el diseño del recuadro donde se muestra la información

También dispone de documentación en línea muy extensa y una gran comunidad detrás para resolver dudas.

# CAPÍTULO 6
# Implementación

En el siguiente capítulo se lleva a cabo toda la implementación de la solución previamente diseñada, explicando detalladamente cada una de las fases marcadas, los problemas que se han producido en el proceso y su posterior solución para seguir con el desarrollo del proyecto.

La implementación engloba en primera instancia la organización de la estructura del entorno de servidor cedido por el grupo de investigación *PRHLT*, en el cual se almacena el dataset aportado por los organizadores del *PAIP2020 challenge*, el código implementado en el proyecto, los modelos entrenados, el código *script* para la ejecución de la fase de entrenamiento, evaluación y las gráficas generadas para las pruebas.

Es importante tener una estructura clara de carpetas para organizar debidamente la información para que sea visible por el código en ejecución y tener instaladas las diferentes librerías necesarias para el desarrollo del proceso.

Una vez terminada la fase organizativa se implementa el código necesario para el proyecto almacenado en la carpeta *python*.

Cuando se acaba con la implementación se escribe un *script* en el que se ejecuta de manera secuencial desde línea de comandos el código con los parámetros de ejecución esperados.

Una vez realizada la ejecución del código se obtienen los resultados de la métrica *IoU* para las versiones, configuraciones propuestas y se generan las gráficas necesarias para la evaluación de los modelos y la demostración del funcionamiento del código.

## 6.1 Creación de la estructura de carpetas del proyecto

Para la realización del proyecto se define una estructura determinada de carpetas organizando debidamente todos los archivos utilizados.

Tener una buena organización de los archivos permite encontrar los archivos desde el código implementado para la carga y creación de nuevos archivos como imágenes, ficheros de texto y modelos.

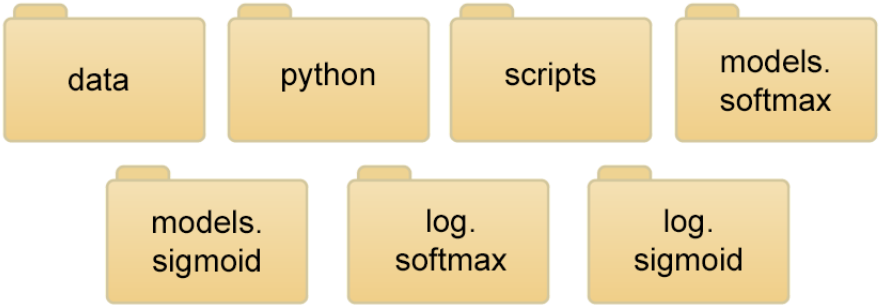


**Figura 6.1:** Estructura de carpetas en el proyecto

En la carpeta *data* se almacena el conjunto de datos de entrenamiento, validación y test de los modelos utilizados por la red *U-Net*.

En la carpeta *python* se almacena todo el código implementado para su ejecución desde la consola de comandos.

En la carpeta *scripts* se almacena el código generado para realizar la tarea de entrenamiento y evaluación de la red en los servidores del *PRHLT*.

En la carpeta *models* se almacenan los modelos generados en el entrenamiento de la red *U-Net* para las versiones *softmax* y *sigmoid*.

En la carpeta *log* se almacenan la salida de la consola generada por la ejecución del código.

## 6.2 Preprocesamiento de las imágenes y máscaras

Las imágenes aportadas por la organización del *Challenge* tienen un formato WSI ya que se han escaneado desde un equipo de electromedicina, están debidamente etiquetadas por profesionales del campo de la medicina, generando un fichero XML donde se especifican el área dónde aparecen células tumorales.

Las dimensiones iniciales de las imágenes de entrenamiento y validación WSI son de (90000, 120000) píxeles aproximadamente con un tamaño de 2,9GB que dificulta su posterior tratamiento, por ello la organización del *challenge* ofrece el fichero *xml2mask.py* que permite reducir la dimensión de las imágenes y el formato.

Por el tamaño de la imagen se debe de realizar una reducción de tamaño para poder utilizarla, por ello el programa permite realizar diferentes reducciones de escala, siendo la reducción L3 la elegida.

Para entender la reducción de dimensionalidad se utiliza la siguiente fórmula:

$$DimL_x = (Dim_{orig})/(4^{Nivel}/2)$$

Tomando como referencia una dimensión cercana a las imágenes en formato *WSI*, concretamente la correspondiente a (100.000, 100.000) píxeles, se obtiene la siguiente correlación de dimensiones para cada uno de los niveles:

| Nivel | Dimensiones aproximadas |
|---|---|
| Referencia | (100.000, 100.000) |
| 1 | (50.000, 50.000) |
| 2 | (12.000, 12.000) |
| 3 | (3.125, 3.125) |

**Tabla 6.1:** Tabla de conversión de dimensiones código *xml2mask.py*

El programa lee en primer lugar las imágenes WSI desde su carpeta y su respectiva máscara dentro de un fichero XML, generando una imagen *PNG* y una máscara en formato TIFF, almacenando las imágenes generadas en las carpetas *png_img_l3* y *mask_img_l3*, todo ello dentro de la carpeta *data*.

La dimensión de las imágenes después de realizar la transformación L3 es de (2800, 3800) píxeles aproximadamente, ya que depende de las dimensiones de la imagen original.

Se ha tenido que modificar el código ya que al ejecutar el programa las máscaras en formato *TIFF* se transponen y no se equiparan a las dimensiones de las imágenes *PNG*, por lo tanto una vez realizadas las modificaciones tanto las imágenes médicas como sus anotaciones tumorales las dimensiones se equiparan en su totalidad.

Posteriormente se comprueba si el canal *alpha* de las imágenes contiene información relevante para el entrenamiento de la red neuronal, por ello se hace un experimento con el programa *remove_alpha.py*.

En el programa se importa una imagen *PNG* transformada en el paso anterior y se elimina el canal *alpha*, seguidamente se generan las imágenes resultantes para ser visualizadas en una ventana y así analizar las diferencias, obteniendo el siguiente resultado:

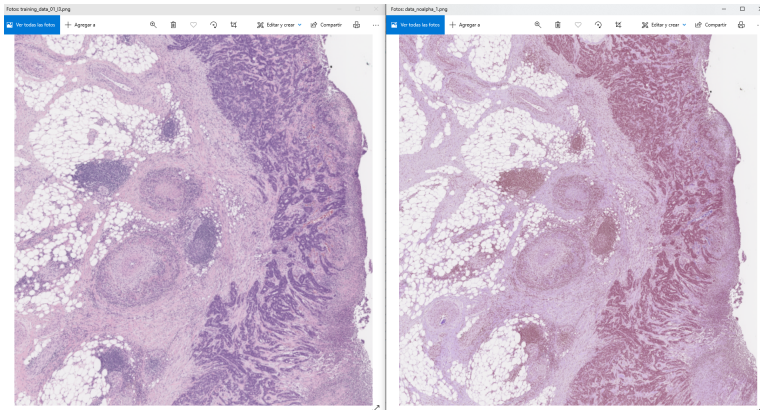

**Figura 6.2:** Comprobación significatividad canal alpha. A la izquierda *PNG* de cuatro canales, a la derecha *PNG* de tres canales

Al retirar el canal alpha se observa que afecta a toda la imagen por ello se concluye que el canal alfa es significativo ya que contiene información del escaneo de las imágenes WSI.

Para terminar con el preproceso, tanto las imágenes médicas como las máscaras necesitan ser troceadas a (128, 128) píxeles para poder ser introducidas en la red *U-Net*, por ello se implementa el programa llamado *crop_l3_images.py*.

En el programa en primera instancia se importan las imágenes y las máscaras, posteriormente se realiza un recorte con las dimensiones estipuladas, con un desplazamiento vertical y horizontal de 64 píxeles.

Una vez recortadas las imágenes se almacenan en las carpetas *png_img_l3.128x128* y *mask_img_l3.128x128* anotando en el nombre de la imagen información sobre la imagen origen y el desplazamiento vertical y horizontal relativo, ya que es de vital importancia para la reconstrucción posterior de la máscara predicha por la red para realizar la evaluación correspondiente.

Durante el proceso de imágenes se prueba la librería *Pillow* y *OpenCV* para abrir imágenes, eligiendo finalmente la segunda librería por las funciones disponibles para manipular las imágenes.

En el cambio de librerías se observa un problema con las imágenes recortadas, para investigar el origen del problema se implementa el código *inspect_data.py*.

El código permite comparar entre una carpeta de imágenes y otra de máscaras si las dimensiones de cada una de las imágenes corresponden con las de la máscara respectiva.

El código se ejecuta con los siguientes parámetros:

| Parámetro | Valor |
|---|---|
| '--folder-png' | Carpeta de imágenes |
| '--folder-mask' | Carpeta de máscaras |

**Tabla 6.2:** Descripción parámetros *inspect_data.py*

El tamaño de las imágenes reducidas L3 tienen unas dimensiones de (2800, 3800) píxeles aproximadamente, por ello cuando se realiza el recorte de tamaño (128, 128) píxeles con una ventana de desplazamiento de 64 píxeles y se llega a los límites verticales y horizontales las imágenes recortadas no son del tamaño correspondiente, ya que se quiere recortar un área donde no hay imagen.

Para solucionar el problema se genera un *NumPy* array lleno de ceros del tamaño de (128, 128) píxeles para luego copiar el área correspondiente recortada de la imagen original, teniendo al final todas las imágenes la dimensión requerida para el entrenamiento de la red.

En el recorte de las imágenes del subconjunto de entrenamiento, de un número de 47 imágenes WSI se obtienen 46.000 imágenes *PNG* aproximadamente, con unas dimensiones (128, 128, 4).

Dado que la organización no ha facilitado las anotaciones de los lotes de datos de validación y test se han utilizado las imágenes de entrenamiento para generar los lotes de validación y test, obteniendo la siguiente correlación de tamaños:

| Subconjunto | Porcentaje |
|---|---|
| Entrenamiento | 88 |
| Validación | 6 |
| Test | 6 |

**Tabla 6.3:** Tamaño de lotes de entrenamiento, validación y test

Una vez preparados los lotes se recortan las imágenes y máscaras tanto para las imágenes de entrenamiento, validación y test para su posterior utilización.

A la hora de realizar el recorte de las máscaras en formato *TIFF* se recortan en formato *PNG* para facilitar la ejecución de la métrica *IoU* al comparar las máscaras con las predichas por la red *U-Net*.

## 6.3 Implementación de la lanzadera de imágenes

Una vez realizado el preproceso de las imágenes se implementa una lanzadera encargada de enviar pequeños lotes *batches* de imágenes junto a las máscaras correspondientes a la red neuronal.

Se decide utilizar una lanzadera ya que a la hora de almacenar en un *NumPy array* grandes cantidades de imágenes se presentan problemas de memoria al ejecutar el código, ya que el espacio en memoria necesario para almacenar imágenes con las dimensiones previas al troceo es bastante grande y no permite trabajar con las imágenes adecuadamente.

El programa llamado *dataGenerate.py* crea un hilo de ejecución que va preparando pequeños lotes de imágenes y máscaras en formato *NumPy*, un tipo de objeto utilizado en las principales librerías de aprendizaje automático existentes para introducir datos de entrada a las redes neuronales.

Se identifican las siguientes variables para la llamada del código por parte de la red:

| Parámetro | Descripción |
|---|---|
| verbose | Visualización del proceso de ejecución en consola |
| folder_png | Carpeta de donde obtiene las imágenes en *PNG* |
| folder_mask | Carpeta de donde obtiene las máscaras en *PNG* |
| batch_size | Tamaño de lote entregado por la lanzadera |
| suffle | Indica si los datos se deben de mezclar al final de cada iteración |
| for_softmax | Formatea datos de entrega de lanzadera para red softmax |
| return_filenames | Indica el retorno del nombre de las imágenes |

**Tabla 6.4:** Descripción parámetros *dataGenerate.py*

En cada iteración de entrenamiento la red llama al programa y se van enviando los lotes, realizando posteriormente una mezcla de las imágenes para que los *batches* estén formados por imágenes de diferentes imágenes originales y regiones, así se evitan posibles problemas de sobreaprendizaje conocido por el nombre de *overfitting*.

Dependiendo de la tarea a realizar se configura la llamada de la clase con unos parámetros determinados, ya que si se realiza una tarea de evaluación es importante no mezclar los datos y retornar el nombre de las imágenes correspondientes, algo muy relevante si se quiere identificar a qué área de la imagen original corresponden dichas imágenes.

Por la naturaleza de la arquitectura *U-Net* se realiza un recorte de las imágenes (128, 128) píxeles, dimensiones adecuadas para la entrada de la red neuronal.

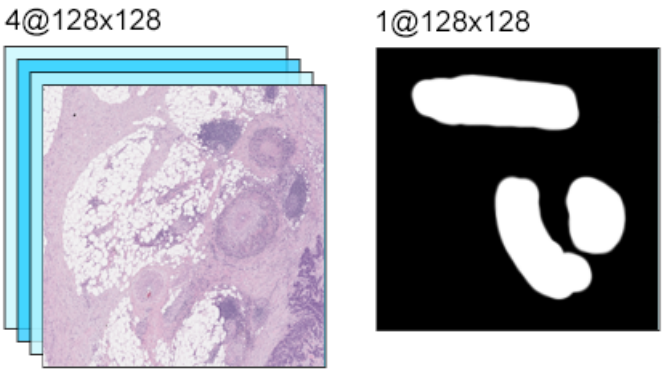


**Figura 6.3:** Imagen y máscara de entrada en la red

El recorte se contempla en el preproceso de las imágenes, por ello se realiza un desplazamiento *stride* de 64 píxeles verticales y horizontales, para poder cubrir toda la imagen y contar con más imágenes disponibles para el entrenamiento, validación y de test.

## 6.4 Implementación de la red

Para la implementación de la red se ha utilizado la librería *pyeddl*, *wrapper* de la librería en c++ de *European Distributed Deep Learning* de *Deep health project*, ya que está diseñada expresamente para la solución de problemas con el uso de algoritmos de aprendizaje automático dentro del campo de la medicina.

Otro de los motivos clave para la utilización de la librería es potenciar el uso de herramientas europeas dentro del campo de la inteligencia artificial, por encima de herramientas estadounidenses o chinas que cuentan con una legislación menos ética con la privacidad de las personas.

En la librería *pyeddl* se disponen de todas las operaciones necesarias para construir cualquier arquitectura de red y sus respectivas herramientas para la creación de tensores, funciones pérdida, métricas.

Para empezar con la implementación dentro del fichero *unet.py* se configura la red *U-Net* con pequeñas variaciones para experimentar posteriormente con el modelo y poder elegir la configuración que obtiene la mejor puntuación con la métrica *IoU*, por ello a la salida del modelo se utiliza la función de activación *softmax* y *sigmoid*.

Para cada función de activación de salida se implementan tres configuraciones diferentes:

- Configuración 1a
- Configuración 1b
- Configuración 2

Cada configuración activa una función diferente *UNetWithPadding_X* dentro del código *unet.py* para definir la arquitectura como paso previo a construcción del modelo.

Las configuraciones 1a, 1b activan la función *UNetWithPadding_1* y la configuración 2 *UNetWithPadding_2* respectivamente.

Como primera observación generalista de la arquitectura de red, primero se identifica la fase de contracción o *encoder*, en la que se van aplicando las diferentes operaciones convolucionales, *maxpooling* y la función de activación *ReLu* después de cada convolución.

Con cada operación de convolución se dobla el nivel de filtros de la imagen facilitando así la detección de características junto a la función de activación *ReLu* y con la operación de *maxpooling* se reducen las dimensiones de la imagen para quedarse con las características más relevantes detectadas.

Una vez realizada la fase de codificación donde se han obtenido las características se introducen en las capas densas para el entrenamiento de la red y luego propaga la información a la fase *decoder* donde se realizan las operaciones de deconvolución y de *upsampling* para que los datos propagados por la red densa tengan en cada capa las dimensiones que la capa correspondiente en la fase de codificación.

Cobra especial importancia que las capas de la fase de codificación y decodificación tengan las mismas dimensiones para así poder realizar una operación de concatenación entre dichas capas donde se mejora la tarea de detección de características en la red, pudiendo también añadir dos convoluciones seguidas de la función de activación *ReLu* antes de cada concatenación para mejorar aún más el proceso.

Los filtros utilizados para las operaciones de *maxpooling* y *upsampling* son de (2x2) mientras en las operaciones convolucionales y deconvolucionales el filtro utilizado es de (3x3).

Para implementar las diferentes configuraciones un mismo código puede ser utilizado tanto para la función de salida de la red *softmax* y *sigmoid* donde solo cambia la última operación de deconvolución y la función de activación saliente, como también para las tres configuraciones enumeradas con anterioridad, en la que se pueden discriminar diferentes áreas de código relevantes a la hora de implementar cada una de las soluciones.

En primera instancia se implementa la red con configuración *1a* en la cual las concatenaciones entre capas y fases de codificación, decodificación son directas sin operaciones de convolución y funciones de activación *ReLu* de paso intermedio.

Para la configuración *1b* se utiliza la misma arquitectura pero a diferencia de la *1a*, antes de cada concatenación se realizan dos convoluciones seguidas de la función de activación *ReLu*.

En la siguiente imagen se observa la configuración *1a* y *1b* con la función de activación *softmax* en la cual en los dos casos la máscara de salida tiene unas dimensiones de (1, 128, 128) píxeles:

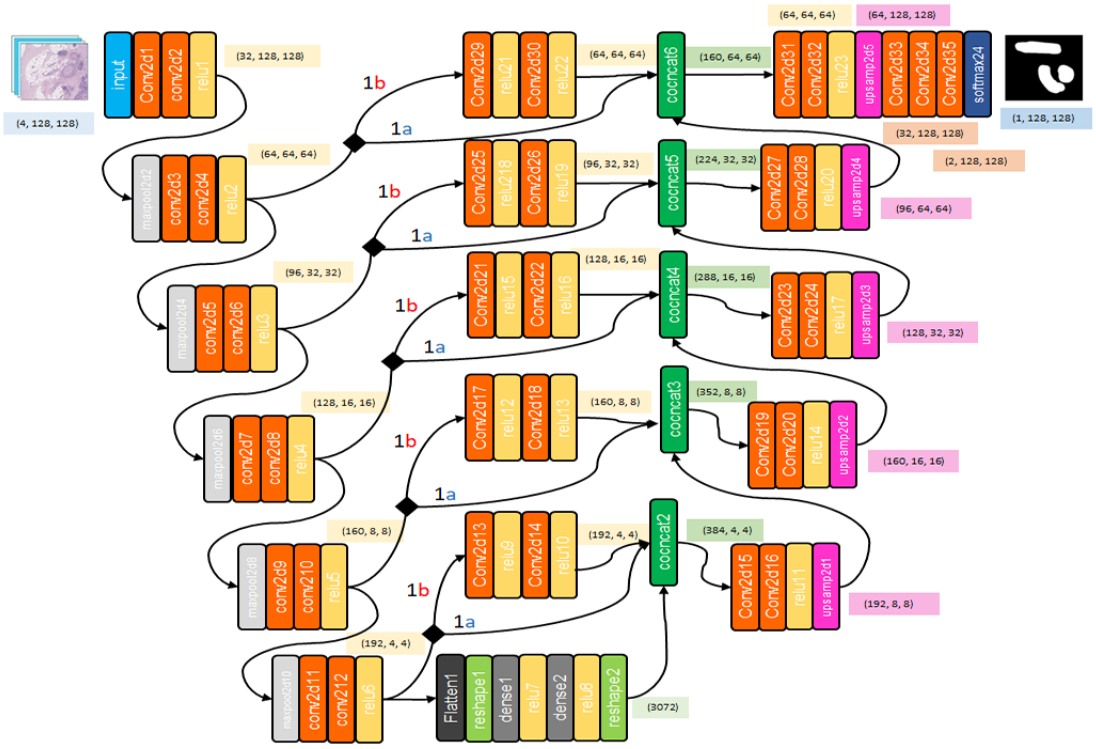


**Figura 6.4:** Arquitectura *U-Net*, configuración 1a, 1b, función de activación Softmax

Una vez implementadas las configuraciones *1a* y *1b* se configura la versión *2* en la cual se eliminan las concatenaciones, obteniendo la siguiente arquitectura:

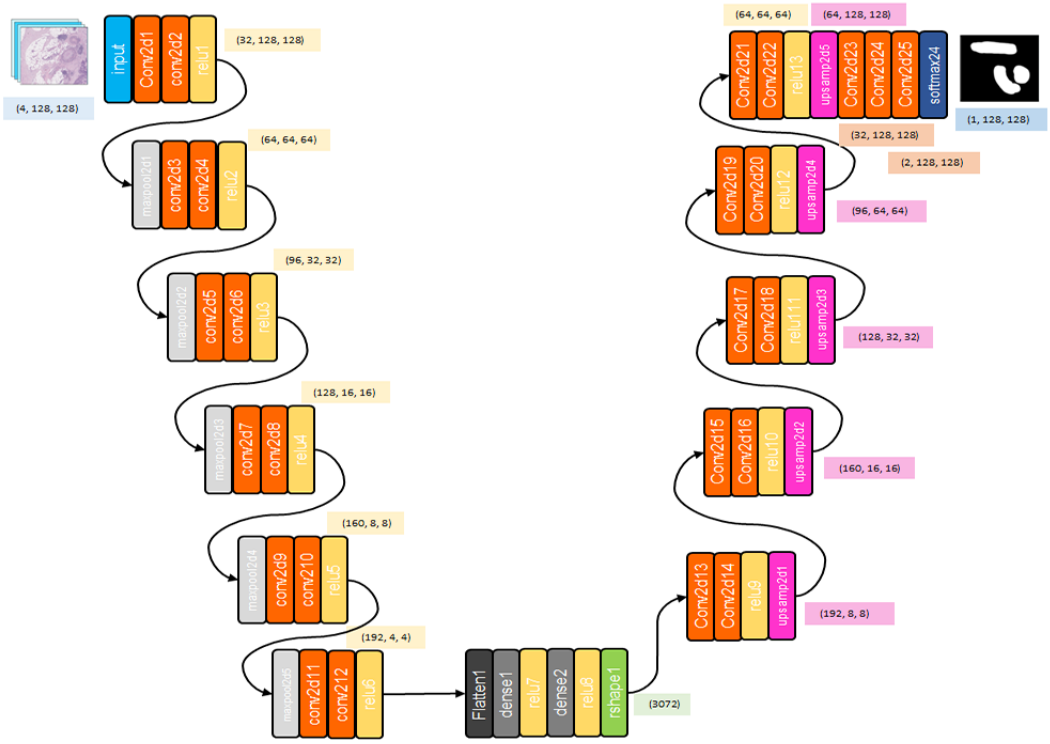


**Figura 6.5:** Arquitectura *U-Net*, configuración 2, función de activación Softmax

Para la configuración *2* se han eliminado las concatenaciones, para así comprobar la efectividad de la variante en la tarea de segmentación en el proyecto.

Una vez generadas las tres configuraciones se procede al entrenamiento de la red *U-Net*.

## 6.5 Implementación del entrenamiento

El entrenamiento de la red *U-Net* se implementa dentro del archivo *unet.py* donde previamente se ha construido la arquitectura.

Para la posible ejecución de diferentes tareas en el mismo código, dependiendo de la fase a realizar se ha añadido una lectura de parámetros de entrada que acompañan a la ejecución del código python desde consola.

La ejecución del código está pensada para hacerse secuencialmente desde un *script* que ejecuta el código necesario para el entrenamiento antes de realizar la fase de evaluación.

| Comando | Descripción |
|---|---|
| --gpu | Máscara de gpu |
| --softmax | Versión softmax |
| --sigmoid | Versión sigmoid |
| --batch-size | Tamaño de lotes |
| --threshold | Umbral de clasificación |
| --model-id | Id del modelo utilizado |
| --task | Tarea a realizar |
| --model-filename | Nombre del modelo a utilizar |
| --starting-epoch | Etapa de inicio |
| --data-dir | Carpeta imágenes de entrada |
| --output-dir | Carpeta imágenes de salida |

**Tabla 6.5:** Descripción parámetros *unet.py*

Para realizar el entrenamiento si el parámetro de entrada correspondiente al comando *--task* es *'training'*, se ejecuta dicha área de código.

Una vez dentro del área primero se prepara el tensor de entrada en la red *U-Net* indicando las dimensiones (4, 128, 128).

En el paso siguiente, dependiendo del parámetro *--model-id* se ejecuta la función *UNet-WithPadding_X* correspondiente para seleccionar la arquitectura elegida, que en el caso de las configuraciones *1a* y *1b* hay que añadir el parámetro del tipo de variante en la ejecución, ya que son variantes de la misma arquitectura.

La variable de entrada *_in* queda definida y la variable de salida *out* se almacena la arquitectura elegida con anterioridad.

Una vez seleccionada la entrada y salida se genera el modelo llamado *segnet* para proceder a definir posteriormente la función de pérdida, el optimizador y la métrica.

Para la versión *sigmoid* se utiliza como función de pérdida el *Mean squared error* y en la versión *softmax* se utiliza como métrica la entropía cruzada.

En el caso de los optimizadores en los dos casos se utiliza *adam*, pero con diferentes ratios de entrenamiento, en la versión *sigmoid* de (1.0e-5) y en *softmax* de (1.0e-6).

La métrica utilizada en los dos casos es el coeficiente de *Sorensen-Dice (F1 Score)*.

Una vez definidas las diferentes piezas se construye la red *U-Net* con la función *eddl.build* con los siguientes parámetros de entrada:

| Parámetro | Valor |
|---|---|
| Optimizador Adam | Ratio de aprendizaje |
| Función de pérdida | Funciones de pérdida para el entrenamiento |
| Métrica | Métrica para evaluar el entrenamiento de la red |
| GPU | Define el grado de utilización de la GPU |

**Tabla 6.6:** Descripción parámetros de entrada construcción *U-Net*

Una vez realizado el *build* del modelo, se escribe un resumen de los diferentes módulos de la red a la salida de la ejecución y un archivo en formato *PDF* con la visión gráfica de la arquitectura elegida.

Una vez almacenada la información descriptiva de la red se procede a describir el código que realiza el entrenamiento.

En cada época se llama a la lanzadera de imágenes encargada de enviar lotes de un tamaño determinado a la red para su entrenamiento donde las imágenes están formateadas en un array de *NumPy* para su posterior transformación en tensores.

Los parámetros de la lanzadera son los siguientes:

| Parámetro | Valor |
|---|---|
| Carpeta *PNG* | Carpeta imágenes de entrada |
| Carpeta máscara | Carpeta máscaras de entrada |
| Tamaño lote | Tamaño del lote a obtener |
| Mezcla lotes | Mezcla de lotes en la tarea de entrenamiento |
| Traza ejecución | Visualizar texto en consola |
| Softmax | Función de activación utilizada |
| Caché | Uso de la caché para obtener imágenes |

**Tabla 6.7:** Descripción parámetros lanzadera de imágenes

Una vez obtenidas las imágenes se generan los tensores de entrada a la red con la función de eddl *Tensor.fromarray* almacenando los tensores en las variables *x_da*, *y_da*, que posteriormente se utilizan para entrenar la red con la función *eddl.train_batch* pasando también el modelo construido con *eddl.build* llamado *segnet*.

En cada iteración la lanzadera de imágenes tiene un método donde realiza una mezcla de las imágenes que lanza a la *U-Net*.

Una vez realizado el entrenamiento de la etapa se imprime por pantalla el error de predicción cometido por la red y se almacena el modelo en formato *ONNX* para su futura experimentación y evaluación.

Para el entrenamiento de la red se ha utilizado un entorno creado en los servidores del equipo de investigación *PRHLT* en el cual se disponen de tarjetas gráficas GeForce GTX 1080, donde se realiza el entrenamiento con las diferentes configuraciones.

El entrenamiento provoca la utilización de la GPU en su totalidad, por ello hasta que no se acabe el entrenamiento no se puede proceder a las pruebas de experimentación y evaluación de la red. *U-Net*.

## 6.6 Implementación de la evaluación

La evaluación de la red se realiza en el código de *unet.py* seleccionando el parámetro *--task* "evaluate" que permite ejecutar el área de código correspondiente en la que se realizan predicciones.

Para realizar el proceso se utiliza la función *eddl.predict* junto a la red *segnet* entrenada con anterioridad para realizar las predicciones del lote de imágenes médicas recibidas por la lanzadera, que dada la naturaleza de la arquitectura, la máscara generada por la red tiene unas dimensiones de (1, 128, 128) píxeles, por lo tanto hay que realizar una predicción de todas las imágenes recortadas y posteriormente utilizar un programa que reunifica las máscaras relativas.

Las imágenes segmentadas se almacenan en la carpeta *mask_img_l3_predicted.128x128* y se implementa el archivo *join_mask.py* para realizar la tarea de unificación, almacenando las imágenes en la carpeta *mask_img_l3_predicted* y el posterior cómputo de la puntuación *IoU* con la función *compute_iou* implementada dentro del código.

El archivo *join_mask.py* recibe los siguientes parámetros de entrada:

| Parámetro | Valor |
|---|---|
| '--base-dir' | Carpeta máscaras generadas por la *U-Net* |
| '--output-file' | Carpeta máscaras reconstruidas por el código |

**Tabla 6.8:** Descripción parámetros *join_mask.py*

Una vez implementado el código del fichero *join_mask.py* se implementa código *script* llamado *compute-evolution.sh* en el que se automatiza el proceso de evaluación de la red *U-Net* a la hora de ejecutar comandos en el *bash* del servidor.

La tarea de evaluación se realiza para los datos de entrenamiento y validación por ello se ejecuta el mismo código con las imágenes de los dos *batches*.

Una vez realizada la ejecución del fichero *unet.py* seleccionando la tarea de "Evaluación" se generan las máscaras segmentadas partir de las imágenes originales, se almacenan en la carpeta saliente, posteriormente se computa la puntuación *IoU* y los resultados etapa a etapa se guardan en un fichero de texto para ser utilizados en la fase de pruebas.

Para realizar la evaluación y generar las gráficas para el apartado de pruebas se implementa el código *plot.py* que permite abrir los ficheros de texto donde se ha guardado la puntuación *IoU* con anterioridad y generar su respectiva gráfica.

El código se ejecuta con los siguientes parámetros de entrada:

| Parámetro | Valor |
|---|---|
| '--model-id' | Modelo de red |
| '--fun' | Función de activación utilizada en la arquitectura |
| '--output' | Carpeta salida de la gráfica generada |
| '--best' | Mejor valor obtenido por la arquitectura en test |

**Tabla 6.9:** Descripción parámetros *plot.py*

Una vez generada las gráficas se implementa un último código llamado *view_png_mask.py* que permite generar una gráfica en la que se visualiza a la izquierda la imagen médica

introducida en la red *U-Net* y a la derecha la misma imagen con la máscara predicha superpuesta.

El código se ejecuta con los siguientes parámetros de entrada:

| Parámetro | Valor |
|---|---|
| '--folder-png' | Carpeta imagen médica L3 |
| '--folder-mask' | Carpeta máscara predicha por la red |
| '--number-img' | Seleccionar el número de imagen a visualizar |
| '--folder-out' | Carpeta donde se almacena la gráfica |

**Tabla 6.10:** Descripción parámetros *view_png_mask.py*

Una vez implementado todo el código se procede a la fase de pruebas.

# CAPÍTULO 7
# Pruebas

Una vez realizada la fase de implementación se ejecutan las pruebas pertinentes para analizar la efectividad de cada una de las versiones y configuraciones *U-Net* implementadas en el proyecto, utilizando la métrica *IoU* para medir el funcionamiento de la red *U-Net* entrenada, pasando como parámetro de entrada los *batches* de entrenamiento y validación.

Las pruebas se realizan en primera instancia con el subconjunto de datos de entrenamiento, que durante un número especificado de etapas se calcula la puntuación *IoU* obtenida con los mismos datos de entrenamiento.

Seguidamente se utilizan los datos del subconjunto de validación con los modelos generados en la fase de entrenamiento, obteniendo la puntuación *IoU*, valor de referencia para la elección del modelo que se va a utilizar para la prueba final, que corresponde con el modelo con la puntuación máxima.

Una vez seleccionado el modelo que da la mejor puntuación con el subconjunto de validación se utiliza con los datos del subconjunto de prueba obteniendo así la mejor puntuación de *IoU* que puede dar dicha versión de red *U-Net*.

Para visualizar los resultados se ha utilizado una gráfica para cada una de las seis arquitecturas definidas previamente en la fase de diseño, donde se recoge la evolución de la puntuación de *IoU* etapa a etapa, para los subconjuntos de datos de entrenamiento y validación, anotando con una constante el mejor valor obtenido por el subconjunto de prueba.

## 7.1 Ejecución código

Para realizar las pruebas se ejecuta en el *bash* del servidor, el fichero *compute-evolution.sh*, que pone en marcha el código *unet.py* con los siguientes parámetros de entrada:

| Comando | Descripción |
|---|---|
| --batch-size | 24 |
| --gpu | [0, 1] |
| --model-id | {1, 2} |
| --task | ”evaluate” |
| --model-filename | {models.sigmoid, models.softmax} |
| --data-dir | /paip2020/{training, validation} |
| --output-dir | mask_img_l3_predicted.128x128 |

**Tabla 7.1:** Parámetros ejecución código *unet.py*

Tras la ejecución del modo de evaluación de *U-Net*, para cada imagen médica de entrada se genera su respectiva máscara, siendo almacenadas en la carpeta *mask_img_l3_predicted.128x128* y seguidamente se ejecuta el código de *join_mask.py* con los siguientes parámetros de entrada:

| Comando | Descripción |
|---|---|
| --base-dir | /paip2020/{training, validation} |
| --output-file | mask_img_l3_predicted |

**Tabla 7.2:** Parámetros ejecución código *join_mask.py*

Una vez ejecutado *compute-evolution.sh* se genera un archivo en la carpeta raíz con los resultados de la evaluación utilizados para la evaluación de las redes entrenadas, visualizando la puntuación *IoU* en la etapa correspondiente del entrenamiento de la *U-Net*.

Para la generación de las gráficas de resultados se implementa el código en el fichero *plot.py*, encargado de obtener la información de los ficheros de texto generados en la evaluación, los transfiere en una lista de valores enteros para las *epochs* y de tipo *Float* para el *IoU*, dibuja las diferentes gráficas y las almacena en formato *PNG*.

Con los los resultados obtenidos se utiliza la librería *matplotlib* para la generación de las diferentes gráficas, dónde se obtiene la puntuación *ÍoU* utilizando los subconjuntos de entrenamiento y validación.

La información se extrae del fichero de texto desde el código de *plot.py* para la generación de las diferentes gráficas, obteniendo los siguientes resultados para las diferentes versiones y configuraciones de la red *U-Net*.

Para la generación de las gráficas se ejecuta el fichero *plot.py* con los siguientes parámetros de entrada:

| Comando | Descripción |
|---|---|
| --model-id | {1a, 1b, 2} |
| --fun | {softmax, sigmoid} |
| --output | Carpeta de salida para la gráfica |
| --best | Mejor valor de *IoU* en test |

**Tabla 7.3:** Parámetros ejecución código *plot.py*

En último lugar para la visualización un ejemplo de segmentación semántica generada por una de las redes entrenadas en el proyecto también se utiliza la librería *matplotlib*.

Se crea una gráfica donde se visualiza a la izquierda la imagen médica introducida por el personal sanitario y a la derecha se visualiza la imagen con la máscara generada por la red *U-Net* superpuesta con la original, para ello se ejecuta el fichero *view_png_mask.py* con los siguientes parámetros de entrada:

| Parámetro | Valor |
|---|---|
| '--folder-png' | ./data/paip2020/test/png_img_l3/ |
| '--folder-mask' | ./data/paip2020/test/mask_img_l3_predicted/ |
| '--number-img' | 2 |
| '--folder-out' | ./plots |

**Tabla 7.4:** Parámetros ejecución código *view_png_mask.py*

Una vez ejecutado todo el código necesario para las pruebas de procede a explicar los resultados obtenidos por las diferentes variantes de red *U-Net* entrenadas y se muestra un ejemplo de detección de células cancerígenas, cumpliendo el principal objetivo del proyecto.

## 7.2 Resultados

Después de la ejecución del código necesario se han generado las gráficas donde se visualiza el rendimiento de cada uno de los modelos propuestos en la fase de diseño, en la tarea de detección de células cancerígenas con el conjunto de entrenamiento, validación y test.

Con la evaluación realizada con los diferentes conjuntos de datos se puede observar si los modelos han entrenado bien y si no se producen los fenómenos comunes de *overfitting* o *underfiting* y si la puntuación de *IoU* es aceptable para la detección de células cancerígenas.

### 7.2.1. Softmax 1a

Se observan los resultados obtenidos por la arquitectura Softmax 1a.

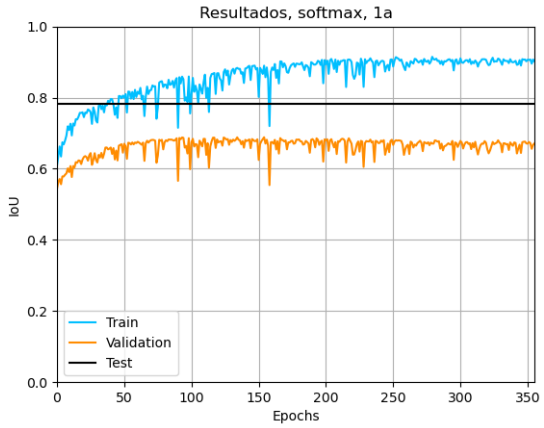


**Figura 7.1:** Gráfica pruebas, versión Softmax, configuración 1a

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 154, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.782867.

| Parámetro | Valor |
|---|---|
| Etapa | 154 |
| *IoU* | 0.782867 |

**Tabla 7.5:** Resultado del conjunto de pruebas para la versión Softmax 1a

### 7.2.2. Softmax 1b

Se observan los resultados obtenidos por la arquitectura Softmax 1b.

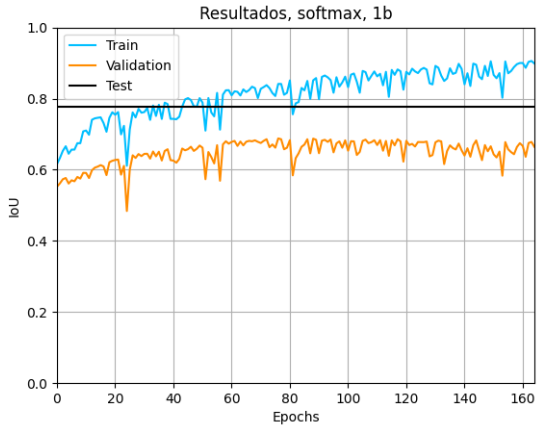


**Figura 7.2:** Gráfica pruebas, versión Softmax 1b

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 74, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.777880.

| Parámetro | Valor |
|---|---|
| Etapa | 76 |
| *IoU* | 0.777880 |

**Tabla 7.6:** Resultado del conjunto de pruebas para la versión Softmax 1b

### 7.2.3. Softmax 2

Se observan los resultados obtenidos por la arquitectura Softmax 2.

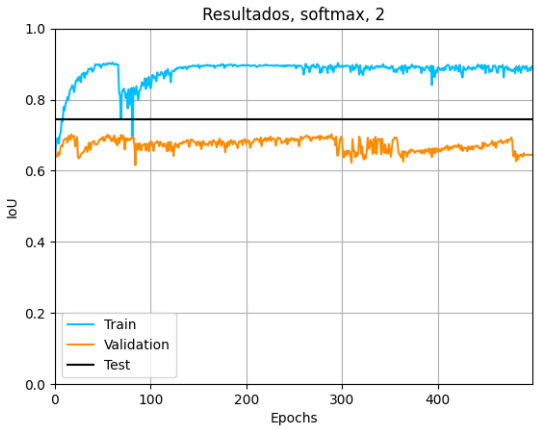


**Figura 7.3:** Gráfica pruebas, versión Softmax 2

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 289, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.745014.

| Parámetro | Valor |
|---|---|
| Etapa | 289 |
| *IoU* | 0.745014 |

**Tabla 7.7:** Resultado del conjunto de pruebas para la versión Softmax 2

### 7.2.4. Sigmoid 1a

Se observan los resultados obtenidos por la arquitectura Sigmoid 1a.

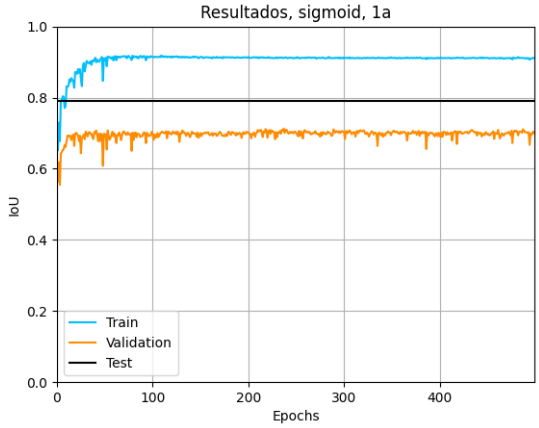


**Figura 7.4:** Gráfica pruebas, versión Sigmoid 1a

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 237, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.791934.

| Parámetro | Valor |
|---|---|
| Etapa | 237 |
| *IoU* | 0.791934 |

**Tabla 7.8:** Resultado del conjunto de pruebas para la versión Sigmoid 1a

### 7.2.5. Sigmoid 1b

Se observan los resultados obtenidos por la arquitectura Sigmoid 1b.

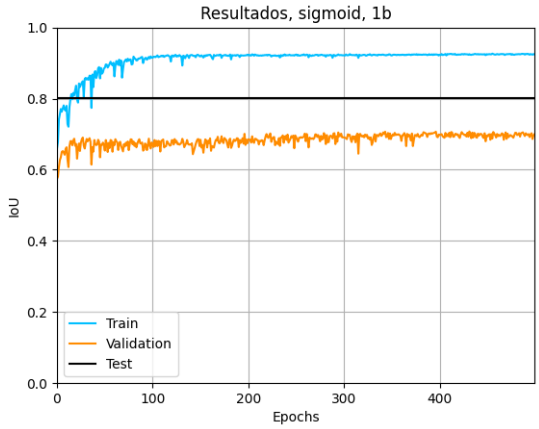


**Figura 7.5:** Gráfica pruebas, versión Sigmoid 1b

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 431, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.802630.

| Parámetro | Valor |
|---|---|
| Etapa | 431 |
| *IoU* | 0.802630 |

**Tabla 7.9:** Resultado del conjunto de pruebas para la versión Sigmoid 1b

### 7.2.6. Sigmoid 2

Se observan los resultados obtenidos por la arquitectura Sigmoid 2.

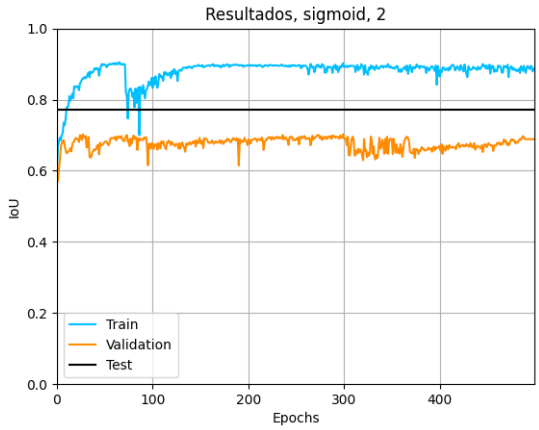


**Figura 7.6:** Gráfica pruebas, versión Sigmoid 2

Con la utilización del subconjunto de validación la mayor puntuación de *IoU* se obtiene en la etapa 299, cuyo modelo se utiliza para realizar las pruebas con el subconjunto de *test* obteniendo una puntuación de 0.771651.

| Parámetro | Valor |
|---|---|
| Etapa | 299 |
| *IoU* | 0.771651 |

**Tabla 7.10:** Resultado del conjunto de pruebas para la versión Sigmoid 1b

### 7.2.7. Demostración detección células cancerígenas

La imagen resultante de la ejecución del fichero *view_png_mask.py* se observa 7.7 donde las áreas afectadas con células cancerígenas se visualizan en la imagen de color verde para facilitar su interpretación.

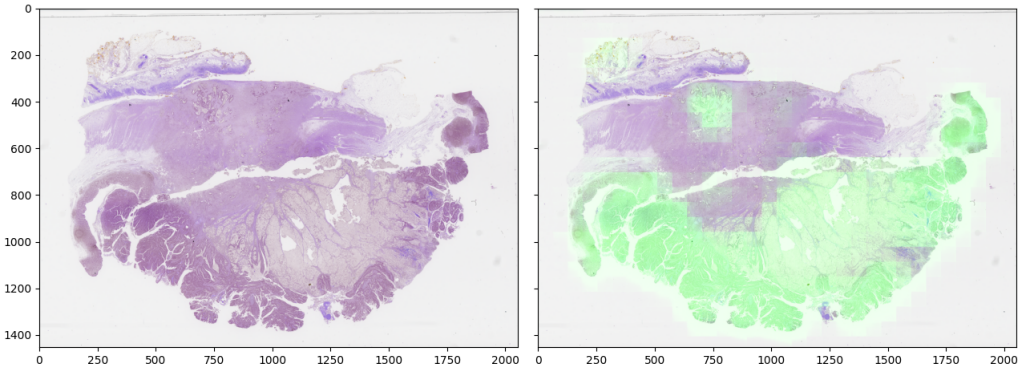


**Figura 7.7:** Gráfica de imagen con máscara superpuesta tras ejecución de *view_png_mask.py*

Se ha elegido una de las imágenes del conjunto de datos de prueba y una de las máscaras generadas por uno de los seis modelos ejecutados, para ver el resultado obtenido.

### 7.2.8. Crítica y discusión de los resultados

Después de la generación de las gráficas donde se recoge la evolución de la puntuación *IoU* para los diferentes subconjuntos de entrenamiento, validación y pruebas, se discuten los aspectos más relevantes.

El modelo que ha obtenido la mejor puntuación *IoU* ha sido *sigmoid 1b* con un valor de 0.802630, ya que en general los modelos *sigmoid* han obtenido una mejor puntuación que los modelos *softmax*.

Los modelos de arquitectura *1a*, *1b* muestran un mayor rendimiento que la arquitectura *2*.

En el caso de los modelos *sigmoid* la mejor puntuación se obtiene en la arquitectura *1b*.

Las versiones *sigmoid* tienen menos fluctuaciones que las versiones *softmax* a lo largo del entrenamiento.

Las arquitecturas *2* muestran grandes fluctuaciones de convergencia en la fase inicial, demostrando ser inestable.

La puntuación obtenida con el subconjunto de datos de entrenamiento generalmente es la más alta, ya que se han utilizado para la evaluación las mismas imágenes que en el entrenamiento, por ello se observa sobreaprendizaje *overfitting* pero importante para tener un límite superior definido.

Generalmente los resultados obtenidos con el subconjunto de datos de validación es más bajo que el subconjunto de test, demostrando que no se ha producido ningún efecto de sobreestimación y que los modelos trabajan adecuadamente.

Después de tratar los hechos más significativos se demuestra que las arquitecturas de red *U-Net* que utilizan concatenaciones entre la fase de codificación y decodificación *1a* y *1b* entrenan mejor que las que no tienen la concatenación *2*.

En las arquitecturas con concatenación la versión *sigmoid* funciona ligeramente mejor con las capas convolucionales y la función de activación intermedia, que con una concatenación directa, sin embargo en la versión *softmax* la concatenación directa tiene un rendimiento ligeramente más alto.

### 7.2.9. Trabajos futuros

Después de la realización del proyecto es interesante intentar aplicar el conocimiento adquirido para utilizar las arquitecturas de red *U-Net* propuestas, para realizar tareas de segmentación semántica para otros campos de la medicina, como por ejemplo el campo de la dermatología, dado que el cáncer de piel es una de las enfermedades que más afectan a los países donde hay una gran incidencia de rayos ultravioleta o para la detección de otros cánceres más comunes que el cáncer de colon como el de pecho o de pulmón.

Otro de los puntos interesantes sería aprender el funcionamiento de otras arquitecturas estado que se han tratado en la sección del estado de la tecnología actual en el proyecto para aplicarlas a realización de la tarea de segmentación semántica, en la detección células cancerígenas y observar el rendimiento de dichos modelos para la detección de tumores.

# CAPÍTULO 8
# Conclusiones

El estado de la tecnología dentro del campo del aprendizaje automático está en continuo cambio, cada vez los algoritmos tienen un rendimiento mayor para realizar tareas de segmentación semántica, dicha evolución se ve claramente observando la arquitectura *U-Net* en comparativa con las tres arquitecturas en el capítulo de la situación actual de la tecnología.

Cada vez las arquitecturas son más complejas y precisan mayor poder de cómputo para ser ejecutadas, por tanto utilizar modelos no tan potentes dependiendo de cual sea la tecnología disponible para el entrenamiento no es algo del todo negativo, ya que el gran poder de cálculo depende de los supercomputadores de grandes empresas y universidades, de momento no está disponible para todo el mundo.

Realizar el trabajo ha sido una oportunidad para tener una primera toma de contacto más profundo con la ciencia, ya que he aprendido a buscar artículos de investigación, he mejorado mi nivel en comprensión lectora inglesa a la hora de interpretar dichos artículos, también he profundizado dentro del campo de la visión por computador, concretamente en la segmentación semántica, he conocido el estado de la tecnología actual y he podido profundizar mi conocimiento sobre la arquitectura *U-Net*.

Personalmente el trabajo ha sido un reto personal, con mucha carga emocional detrás, ya que mi padre murió de cáncer de colon terminal hace ocho años, la etapa de aprendizaje desde entonces no ha sido sencilla, ha habido mucho esfuerzo detrás tanto dentro el camino de la formación profesional como en la universidad, por ello acabar una etapa con el presente trabajo ha sido bastante gratificante, así que espero tener la oportunidad de seguir creciendo dentro del mundo de la ciencia.

# Bibliografía


[1] El cáncer es una de las causas principales de muerte en todo el mundo: casi 10 millones de fallecimientos en 2020. Consultado en https://www.who.int/es/news-room/fact-sheets/detail/cancer.

[2] Tiempo medio de espera de pacientes por especialidad de cirugía general y digestiva durante el año 2019 en el sistema sanitario español. Pag 3. Consultado en https://www.mscbs.gob.es/estadEstudios/estadisticas/inforRecopilaciones/docs/LISTAS_PUBLICACION_dic19.pdf.

[3] Estimated number of new cases in 2020, worldwide, both sexes, all ages Consultado en https://gco.iarc.fr/today/online-analysis-pie?v=2020&mode=cancer&mode_population=continents&population=900&populations=900&key=total&sex=0&cancer=39&type=0&statistic=5&prevalence=0&population_group=0&ages_group%5B%5D=0&ages_group%5B%5D=17&nb_items=7&group_cancer=1&include_nmsc=1&include_nmsc_other=1&half_pie=0&donut=0.

[4] Librería Eddl Consultado en https://deephealthproject.github.io/eddl/.

[5] Librería TensorFlow Consultado en https://www.tensorflow.org/.

[6] Frank Rosenblatt. The Perceptron: A probabilistic model for information storage and organitzation in the brain. *Psychological Review*, Vol. 65, No. 6, 1958.

[7] Whole slide imaging in pathology: advantages, limitations, and emerging perspectives Farahani N, Parwani A, Pantanowitz L *Pathology and Laboratory Medicine International*, 11 June 2015 Volume 2015:7 Pages 23—33.

[8] A Universal Algorithm for Sequential Data Compression Jacob Ziv, Fellow, IEEE, Abraham LempelL, *IEEE Transactions on Information Theory*, vol. It-23, No. 3, Yay 1977

[9] U-Net: Convolutional Networks for Biomedical Image Segmentation Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany* , 18 May 2015.

[10] FastFCN: Rethinking Dilated Convolution in the Backbone for Semantic Segmentation Huikai Wu, Junge Zhang, Kaiqi Huang *Institute of Automation, Chinese Academy of Sciences*, 28 Mar 2019.

[11] The Importance of Skip Connections in Biomedical Image Segmentation Michal Drozdzal, Eugene Vorontsov, Gabriel Chartrand, Samuel Kadoury and Chris Pal. *Imagia Inc., Ecole Polytechnique de Montreal, Universite de Montreal, CHUM Research Center, Montreal Institute for Learning Algorithms, Montreal, Canada*, 22 Sep 2016.

[12] DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille, Fellow *Institute of Electrical and Electronics Engineers (IEEE)*, 12 May 2017.

[13] Pyramid Scene Parsing Network Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia *The Chinese University of Hong Kong, SenseTime Group Limited)*, 12 May 2017.

[14] Gated-SCNN: Gated Shape CNNs for Semantic Segmentation Towaki Takikawa, David Acuna, Varun Jampani, Sanja Fidler *NVIDIA, University of Waterloo, University of Toronto, Vector Institute*, 12 Jul 2019.

[15] Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam *Google Inc*, 22 Aug 2018.

[16] Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels Zhilu Zhang, Mert R. Sabuncu *Electrical and Computer Engineering, Meinig School of Biomedical Engineering, Cornell University*, 29 Nov 2018.

[17] 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 Thorvald Sørensen *The Royal Danish Sciences Society Biological Writings*, Volume V, No. 4, 1948.

[18] Distribution comparée de la flore alpine dans quelques régions des Alpes occidentales et orientales. Paul Jaccard *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 1902.

[19] Lee Raymond Dice. *The biotic provinces of North America*. University of Michigan Press, viii + 78. 1943.

[20] Marvin Minsky, Seymour A. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT, Massachusetts Institute of Technology. 1969.