**Julen Bohoyo Bengoetxea**

# SEGMENTACIÓN DE CANCER COLORECTAL MEDIANTE REDES NEURONALES CONVOLUCIONALES

**TRABAJO FINAL DE GRADO**

**dirigido por el Dr. Marc Sanchez Artigas**

**Doble Grado en Ingeniería Informática y en Biotecnología**

UNIVERSITAT ROVIRA I VIRGILI

**Tarragona**

**2022**

## Resum

Un mètode comunament utilitzat per diagnosticar el càncer de còlon és a través de l'anàlisi de mostres obtingudes mitjançant biòpsies; un procés que en l'actualitat es realitza de forma especialment lenta. Per a solucionar aquest inconvenient s'ha plantejat desenvolupar una eina de visió per computadora utilitzant xarxes neuronals convolucionals amb la finalitat d'agilitzar aquest procés. En concret, es proposa utilitzar un model per a segmentació semàntica basat en una U-Net combinada amb ResNet34 que segmenti els teixits i tumors presents en les mostres. U-Net és una arquitectura especialment dissenyada per a l'anàlisi d'imatges de l'àmbit mèdic i diversos estudis han demostrat el seu gran rendiment. Un obstacle comú en aquesta mena de projectes és la limitada o gairebé nul·la disponibilitat d'imatges de qualitat necessàries per a entrenar les xarxes neuronals. Per a superar aquesta limitació s'han aplicat diverses tècniques que permeten obtenir un sistema raonablement precís a partir d'un conjunt limitat d'imatges. En concret, s'ha recorregut a la utilització d'una xarxa preentrenada amb Imagenet, a l'aplicació de tècniques d'augmentat de dades i, especialment determinant, a la divisió de les imatges d'entrenament en petites rajoles amb la finalitat d'augmentar la disponibilitat de dades. S'ha aconseguit desenvolupar un sistema de segmentació de teixits i tumor funcional i amb potencial de ser utilitzat per professionals mèdics amb la finalitat d'agilitzar el procés de diagnòstic.

## Resumen

Un método comúnmente utilizado para diagnosticar el cáncer de colon es a través del análisis de muestras obtenidas mediante biopsias; un proceso que en la actualidad se realiza de forma especialmente lenta. Para solventar este inconveniente se ha planteado desarrollar una herramienta de visión por computadora utilizando redes neuronales convolucionales con el fin de agilizar este proceso. En concreto, se propone utilizar un modelo para segmentación semántica basado en una U-Net combinada con ResNet34 que segmente los tejidos y tumores presentes en las muestras. U-Net es una arquitectura especialmente diseñada para el análisis de imágenes del ámbito médico y diversos estudios han demostrado su gran rendimiento. Un obstáculo común en este tipo de proyectos es la limitada o casi nula disponibilidad de imágenes de calidad necesarias para entrenar las redes neuronales. Para superar esta limitación se han aplicado diversas técnicas que permiten obtener un sistema razonablemente preciso a partir de un conjunto limitado de imágenes. En concreto, se ha recurrido a la utilización de una red preentrenada con Imagenet, a la aplicación de técnicas de aumentado de datos y, especialmente determinante, a la división de las imágenes de entrenamiento en pequeñas baldosas con el fin de aumentar la disponibilidad de datos. Se ha logrado desarrollar un sistema de segmentación de tejidos y tumor funcional y con potencial de ser utilizado por profesionales médicos con el fin de agilizar el proceso de diagnóstico.

# Abstract

A commonly used method to diagnose colon cancer is through the analysis of samples obtained from biopsies; a process that currently is particularly slow. To overcome this drawback, it has been proposed to develop a computer vision tool using convolutional neural networks in order to speed up this process. Specifically, it is proposed to use a model for semantic segmentation based on a U-Net combined with ResNet34 that segments the tissues and tumours present in the samples. U-Net is an architecture specially designed for image analysis in the medical field and several studies have demonstrated its high performance. A common obstacle in this type of project is the limited or almost non-existent availability of quality images needed to train the neural networks. To overcome this limitation, several techniques have been applied to obtain a reasonably accurate system from a limited set of images. In particular, we have resorted to the use of a pre-trained network with Imagenet, to the application of data augmentation techniques and, especially decisive, to the division of the training images into small tiles in order to increase data availability. We have succeeded in developing a functional tumour and tissue segmentation system with the potential to be used by medical professionals in order to speed up the diagnostic process.

# Índice

## Índice de Tablas



## Índice de Ecuaciones



## Índice de Figuras

# 1 Introducción

## 1.1 Cáncer de Colon

El cáncer de colon fue la tercera clase de cáncer más diagnosticada, con 1,93 millones de nuevos casos, y la segunda que más muertes causó, 935 000 muertes, en el año 2020 según datos de la OMS[1] [1].

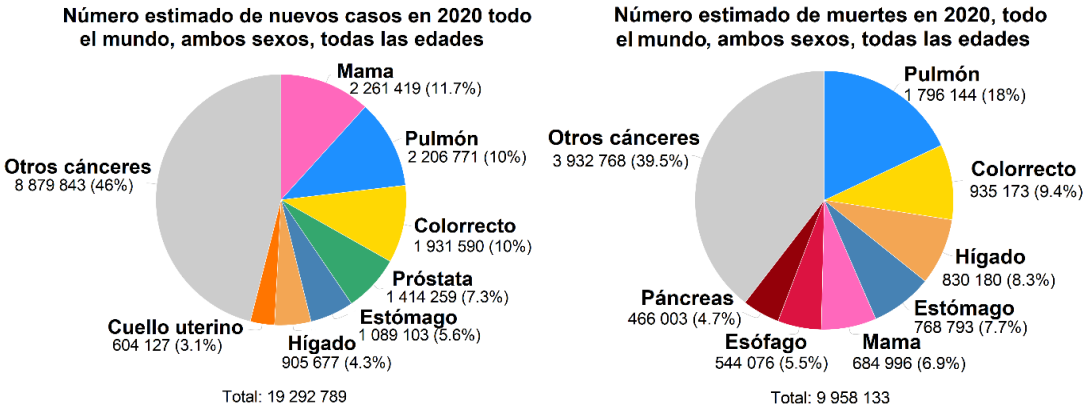



**Ilustración 1.** Cantidad de nuevos casos y muertes causadas por clases de cáncer en 2020, datos de GLOBOCAN, adaptado de [2]

Típicamente, el cáncer de colon comienza como un pólipo. Estos pólipos son crecimientos o agregaciones localizadas de células anormales dentro de la mucosa intestinal que sobresalen hacia la luz intestinal. Con el tiempo, las células que forman los pólipos pueden acumular los suficientes cambios genéticos como para adquirir la capacidad de invadir la pared intestinal. Eventualmente, pueden alterarse aún más y diseminarse por los ganglios linfáticos locales y finalmente a sitios metastásicos distantes. Afortunadamente, solo un pequeño porcentaje de los pólipos adquieren características malignas, e incluso para los que lo hacen, la progresión completa de pólipos a cáncer generalmente toma varios años o incluso una década [3].

[1] Organización mundial de la Salud

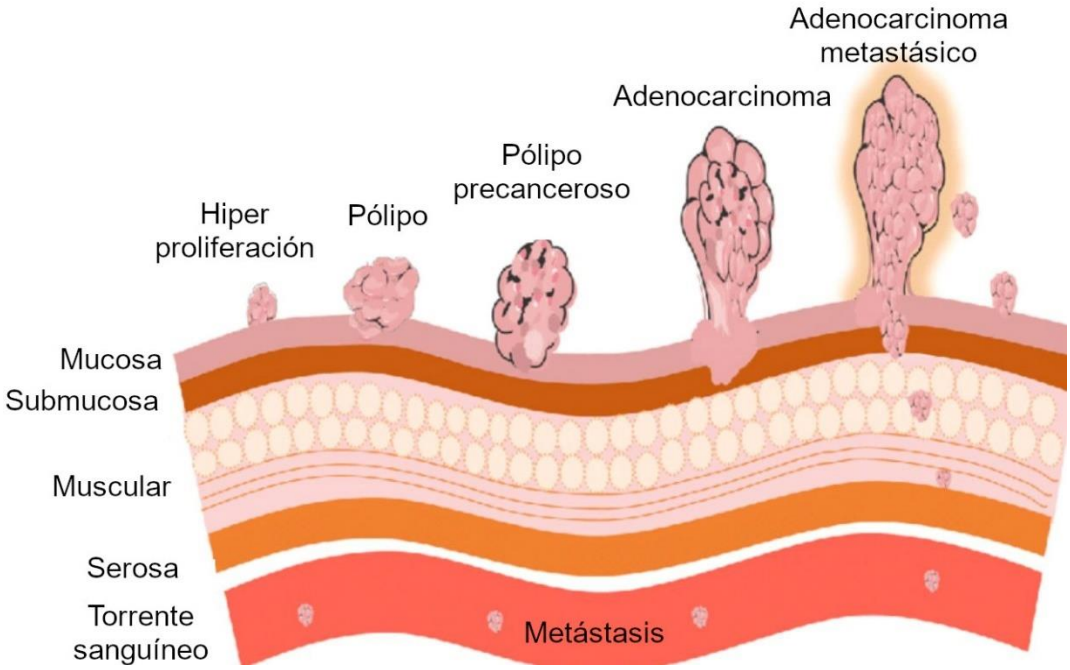



**Ilustración 2.** Evolución de un pólipo en un adenocarcinoma metastásico, adaptado de [4]

En cuanto a los factores de riesgo, aunque cualquier individuo puede desarrollar CCR[2], se han identificado varios factores que están asociados con un mayor riesgo de contraer la enfermedad. Algunos factores de riesgo son modificables, como la dieta, la obesidad, la falta de actividad física, el consumo de tabaco y el consumo de alcohol de moderado a intenso. Por el contrario, se ha informado que una mayor ingesta de fibra dietética, verduras de hoja verde, ácido fólico y calcio protegen contra el desarrollo de CCR.

A diferencia de los anteriores, existen otros factores de riesgo que no son modificables. Entre éstos se encuentran los antecedentes personales o familiares de pólipos colorrectales o CCR, afecciones hereditarias como el síndrome de Lynch, antecedentes personales de enfermedad inflamatoria intestinal, antecedentes raciales y étnicos, así como la presencia de diabetes tipo 2. Por último, es interesante señalar que la probabilidad de desarrollar CCR aumenta notablemente después de los 50 años. De hecho, el 90% de los casos nuevos y el 94% de las muertes asociadas con el CCR que ocurren a partir de dicha edad [3].

Si el CCR se detecta y trata en un estado temprano, cuando todavía se encuentra presente de forma localizada, la tasa de supervivencia estimada a 5 años es de un 91%. Por el contrario, si se comienza el tratamiento cuando ya se ha producido metástasis, la probabilidad de supervivencia es de únicamente un 14% [5]. Por lo tanto, es fundamental detectar y actuar cuanto antes con el fin de que el paciente tenga una mayor probabilidad de sobrevivir. Debido a que los síntomas provocados por el CCR son difícilmente identificables en los estados tempranos de la enfermedad, es necesario llevar a cabo programas de *screening*, especialmente en la población con mayor riesgo de sufrirlo [6].

[2] Cáncer colorrectal

| ESTADO | TASA DE SUPERVIVIENCIA A 5 AÑOS CÁNCER DE COLON | TASA DE SUPERVIVIENCIA A 5 AÑOS CÁNCER RECTAL |
|---|---|---|
| Localizado | 91% | 90% |
| Regional | 72% | 73% |
| Distante | 14% | 17% |
| Todos los estados combinados | 64% | 67% |

**Tabla 1.** Tasas de supervivencia de 5 años para pacientes de cáncer colorrectal, adaptado de [7]

Con este fin existen varias pruebas útiles para la detección de CCR, cada una de las cuales tiene distintas ventajas y limitaciones. La colonoscopia es el método de referencia actual para la detección del CCR y se recomienda emplearla cada 10 años en pacientes de riesgo promedio a partir de los 50 años o más. La capacidad de la colonoscopia para detectar lesiones cancerosas y precancerosas a través de la visualización directa ha sido ampliamente demostrada, de hecho, la sensibilidad de esta técnica para detectar el CCR es del 95%. Además, una de las mayores ventajas de la colonoscopia es la capacidad de poder eliminar lesiones cancerosas pequeñas y precancerosas en el mismo momento de la detección [3]. En algunos casos, se puede realizar una biopsia de tejido tumoral o pretumoral para analizar la muestra mediante microscopio y determinar qué tejidos han sido invadidos por el tumor. Esto es esencial para conocer cuánto ha penetrado el carcinoma en la pared del colon, y, por ende, poder estimar el riesgo de que llegue al riego sanguíneo y se pueda producir una metástasis.

## 1.2 Motivaciones y Objetivos

Los oncólogos del Hospital Universitario de Araba solicitaron al Departamento de Biología Computacional del ISS Biodonostia[3] el desarrollo de una herramienta que les facilitase la tarea de examinar las biopsias obtenidas de posibles pacientes con CCR. Actualmente, los médicos necesitan analizar manualmente las muestras, reconocer la región afectada por el tumor e identificar cuáles son los tejidos que están siendo invadidos por este. Para llevar a cabo dicha tarea utilizan el software de etiquetado médico QuPath [8]. Esta aplicación permite visualizar y examinar las imágenes a una alta resolución, al tiempo que ofrece herramientas para etiquetar las diferentes clases presentes, resaltándolas con diferentes colores. El problema reside en que dicho análisis es una tarea que requiere mucho tiempo a los oncólogos. Como se ha mencionado previamente, es esencial poder realizar *screenings* de la población para obtener un diagnóstico temprano y aumentar la tasa de supervivencia. Por lo tanto, resultaría muy conveniente disponer de una herramienta capaz de proveer un diagnóstico preliminar, de forma que los médicos puedan centrarse en examinar únicamente las muestras en las que se aprecie algún riesgo para la salud del paciente.

Por lo tanto, se debía desarrollar un sistema que, de forma autónoma, sea capaz de identificar las regiones afectadas por el tumor en imágenes de biopsias. Además, también debe ser capaz de clasificar los tejidos presentes en la muestra y reconocer cuáles de estos están siendo invadidos por el tumor. Finalmente, se debe presentar la información de forma similar a la representación que se obtiene un QuPath, resaltando cada tejido con un color diferente y la región afectada por el tumor destacada en color rojo. También se mostrará una lista con los tejidos que han sido invadidos por el tumor.

Se trata de una tarea de visión por computador de ámbito biomédico. En los últimos años, y gracias a los avances en *hardware*, capacidad computacional y disponibilidad de datos, el DL[4] ha experimentado grandes avances. En concreto, el DL ha demostrado tener un gran potencial en el ámbito biomédico. Actualmente, las principales áreas de aplicación del DL para análisis de imágenes médicas implican segmentación, clasificación y detección de anomalías utilizando imágenes generadas a partir de un amplio espectro de modalidades de imágenes clínicas [9]. Por lo tanto, se optó por usar una red neuronal con el fin de desarrollar el sistema solicitado por los oncólogos.

Respecto al punto de vista formativo, este proyecto resulta idóneo para combinar los conocimientos multidisciplinares cursados en el Doble Grado en Ingeniería Informática y en Biotecnología. Además, el aprendizaje profundo es una disciplina de la informática en auge y con gran proyección de futuro. No obstante, en el programa formativo no se profundiza especialmente en este aspecto, por lo tanto, considero que la experiencia adquirida durante este proyecto resulta de gran valor para mi próximo futuro laboral.

[3] Instituto de Investigación Sanitaria Biodonostia
[4] *Deep Learning*

### *1.3 Bulleted List*

A continuación se exponen las principales tareas que han se han llevado a cabo durante la ejecución del proyecto:

- Discutir diferentes opciones para desarrollar el proyecto y decidir la opción óptima.
- Exportar las imágenes desde QuPath y prepararlas para su utilización.
- Aplicar técnicas de preprocesado para subsanar la escasez de datos.
- Seleccionar un modelo de red neuronal adecuado para la tarea.
- Entrenar una U-Net + ResNet34 preentrenada con Imagenet utilizando la base de datos propia.
- Evaluar los resultados

# 2 Materiales

## 2.1 Entorno de Desarrollo

El ISS Biodonostia cuenta con un clúster computacional que está dividido en 6 nodos formados por CPUs y un nodo formado por GPUs. Cada uno de los nodos formados por procesadores convencionales, contiene 16 procesadores Intel Xeon Silver 4112. En cuanto al nodo formado por tarjetas gráficas, éste dispone de ocho tarjetas gráficas NVIDIA Tesla V100 de 32Gb. Cada uno de estos dispositivos dispone de 640 Tensor Cores, los cuales ofrecen hasta 112 teraFLOPS en tareas de aprendizaje profundo. Estas unidades son capaces de trabajar con redes neuronales mucho más rápido de lo que lo haría un procesador convencional. Esto es debido a la naturaleza de los cálculos que se llevan a cabo en el interior de las redes neuronales, consistentes en la realización de miles de multiplicaciones de pequeñas matrices con valores de tipo float32, y para los cuales están específicamente diseñadas las GPUs. El disponer de estas tarjetas gráficas ha ayudado a desarrollar el proyecto de forma ágil y rápida, además de permitir optimizar hiperparámetros a base de prueba y error sin tener que esperar largos periodos de tiempo para obtener resultados.

Para el desarrollo del proyecto se ha optado por utilizar la librería de TensorFlow [10] para Python frente a PyTorch debido a la gran cantidad de documentación disponible. Además, dada la nula experiencia en el campo del aprendizaje profundo, se ha hecho un amplio uso de las herramientas proporcionadas por Keras [11]. Keras es una librería capaz de ejecutarse sobre TensorFlow. Proporciona un conjunto de funciones más intuitivas y de alto nivel, haciendo más sencillo el desarrollo de modelos de aprendizaje profundo independientemente del *backend* computacional utilizado. Como editor de código se ha optado por utilizar Jupyter Notebook debido a la flexibilidad que aporta el poder ejecutar bloques independientes del código sin necesidad de volver a ejecutar el programa por completo. De esta forma, una vez cargado el *dataset* o entrenada la red neuronal, ha sido mucho más sencillo subsanar los errores que se hayan podido encontrar en las siguientes fases del código.

## 2.2 Datos

Todas las imágenes utilizadas en este proyecto provienen del equipo de oncólogos del Hospital Universitario Txagorritxu, en Araba. Se trata de imágenes histológicas de pacientes obtenidas como parte de su PCCR[5] [12]. Durante dicho proceso se realizan colonoscopias acompañadas de biopsias para el posterior análisis visual de los tejidos. Por lo tanto, se trata de datos proporcionados por especialistas en el sector de la salud que han dedicado parte de su tiempo a prepararnos estos datos de entrenamiento. Actualmente, uno de los mayores obstáculos en la aplicación de algoritmos de inteligencia artificial en el ámbito biomédico es la falta de disponibilidad de datos de calidad. Así pues, disponer de estas imágenes de entrenamiento preparadas por oncólogos experimentados es uno de los puntos que más valor añade a este proyecto.

Las muestras están teñidas con la tinción hematoxilina-eosina (H&E). Esta tinción ayuda a identificar diferentes tipos de células y tejidos, así como a obtener información importante sobre las características, la forma y la estructura celular de una muestra de tejido. Debido a estas propiedades, es ampliamente utilizada para el diagnóstico de enfermedades como el cáncer. Gracias a dicha tinción, en las imágenes obtenidas mediante esta técnica se pueden apreciar cinco tipos de tejidos claramente diferenciables por su color y textura: mucosa, linfocitos, submucosa, muscular y subserosa. Además, y debido a que son las capas que constituyen la estructura de la pared del colon, estos tejidos siempre se encuentran en el orden mencionado. Asimismo, en los pacientes que efectivamente sufran CCR, también se aprecia una región especialmente oscura, debido a la tinción, correspondiente al tumor.

La mayor limitación que se ha encontrado durante el desarrollo del proyecto ha sido la disponibilidad de tiempo por parte de los oncólogos para el etiquetado de las imágenes. En un principio se valoró clasificar cada uno de los tejidos presentes en la muestra como sano o enfermo. Es decir, que la región afectada por el tumor también tendría los diferentes tejidos etiquetados. Sin embargo, el equipo médico indicó que no era viable realizar este tipo de etiquetado tan preciso, ya que preparar cada imagen podría requerirles más de una tarde de trabajo. En consecuencia, con el único fin de facilitar la tarea de etiquetado, se optó por dividir el proyecto en dos redes neuronales que trabajaran de forma independiente. Más adelante, en el apartado de "Estructura del sistema" se profundizará en este aspecto.

Los datos de entrenamiento se han facilitado en forma de proyecto de QuPath. Este software almacena los datos de cada imagen en dos archivos diferenciados. En primer lugar, la propia imagen en formato .TIFF de muy alta resolución, en torno a 50.000 x 50.000 píxeles. En segundo lugar, mediante las herramientas proporcionadas por QuPath el usuario puede etiquetar los tejidos presentes en dicha muestra. Estos datos de etiquetado se almacenan en un archivo .qpdata. Este formato de datos no es válido para introducirlos en una red neuronal, por lo tanto, en primer lugar, es necesario exportarlos de forma apropiada para poder trabajar con ellos. Hay que señalar que Este formato de datos no es válido para introducirlos en una red neuronal, por lo que para trabajar con ellos será necesario exportarlos previamente a un formato compatible.

[5] Programa de Cribado de Cáncer Colorrectal

Aunque QuPath ofrece la posibilidad de exportar las imágenes con o sin etiquetado a formato .JPG, no dispone de forma nativa de ninguna herramienta para exportar los datos de etiquetado en forma de máscara de segmentación. Sin embargo, al trabajar con redes neuronales es necesario disponer del resultado esperado para las imágenes de entrenamiento codificado en forma de máscara de segmentación; aspecto en el que se profundizará más adelante. No obstante, QuPath sí ofrece la posibilidad de desarrollar *scripts* en Goovy para implementar herramientas propias. Por lo tanto, se optó por desarrollar un *script* encargado de automatizar la exportación de todas las muestras proporcionadas a los formatos necesarios, véase *Anexo 1*. Por un lado, la imagen original en formato .JPG y sin información de etiquetado, equivalente a las imágenes de nuevos pacientes que se introducirán en la red neuronal una vez esta sea funcional. Por otro lado, la máscara de segmentación en formato .PNG en escala de grises. Además, y debido a la gran resolución de las imágenes originales, también se decidió aplicar un reescalado de los datos, generando un *dataset* en el que se disminuye la resolución en un factor de 10 (x0.1) y otro en el que se disminuye en un factor de 40 (x0.025).

Para poder hacer uso de ciertas herramientas que proporcionan Keras y TensorFlow es necesario organizar el *dataset* siguiendo cierta disposición. La herramienta flow_from_directory [13], de la que dispone Keras, ofrece la posibilidad de utilizar un *dataset* para el entrenamiento sin necesidad de cargarlo en memoria y aplicando las transformaciones deseadas sobre los datos de entrenamiento en tiempo real. Sin embargo, esta herramienta está pensada para sistemas de clasificación, por lo que infiere la etiqueta a la que pertenece cada imagen a partir del subdirectorio en la que se encuentra. Más adelante, en el apartado de aumentado de datos, se profundizará en el código que ha sido necesario desarrollar para solventar este problema. En lo que corresponde a la disposición de los datos, ha sido necesario organizarlos de la siguiente forma:

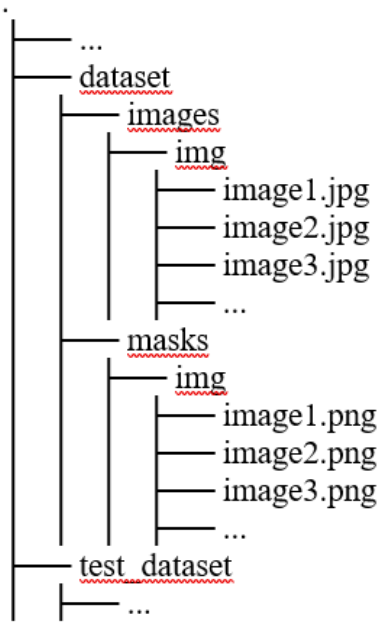



**Ilustración 3:** Estructura de directorios utilizada para almacenar el *dataset*.

# 3 Deep Learning

## 3.1 Estructura

Los componentes básicos de las redes neuronales artificiales son las neuronas artificiales. La neurona artificial recibe una o más entradas binarias ($x_m$), equivalentes a los potenciales postsinápticos excitatorios y potenciales postsinápticos inhibitorios de las neuronas biológicas. Estos valores de entrada se suman para producir una salida (y), que representará el potencial de acción de la neurona y se transmitirá a lo largo de su conexión de salida o axón. Por lo general, cada entrada se modula multiplicándola por un valor o peso ($w_{km}$). Finalmente, la suma ($v_k$) de todas las entradas moduladas se evalúa a través de una función no lineal conocida como función de activación ($\varphi$). Usualmente, el valor de entrada $x_0$ toma el valor de +1, el cual es modulado por $b_k$ para producir un valor *bias* [14].

Por lo tanto, la neurona artificial puede expresarse de forma matemática de la siguiente manera:

$$y_k = \varphi\left(\sum_{j=0}^{m} x_j \cdot w_{kj}\right) \quad (1)$$

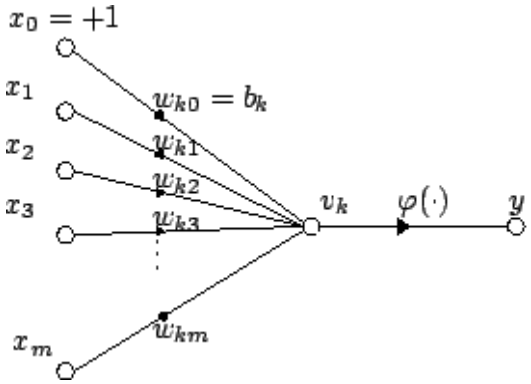



**Ilustración 4.** Representación de una neurona artificial [15].

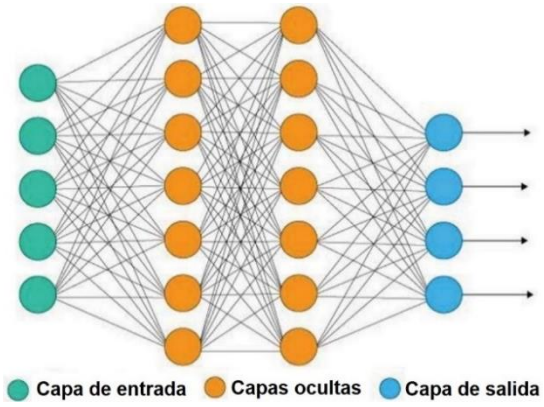



**Ilustración 5.** Representación de una red neuronal con dos capas ocultas, adaptado de [16].

Las neuronas se encuentran estrechamente interconectadas y organizadas en capas. La capa de entrada recibe los datos, mientras que la capa de salida genera el resultado final. Entre las dos, normalmente se intercalan una o más capas ocultas, como se puede observar en la *Ilustración 4* [16]. Existen diferentes tipos de capas que son apropiadas para diferentes tareas y tipos de datos. Se puede pensar en las capas como en piezas de un puzle, especialmente al trabajar con librerías de alto nivel como Keras. Para construir modelos de aprendizaje profundo en Keras se concatenan una serie de capas compatibles de forma que se logra un *pipeline* capaz de transformar y procesar los datos de entrada de la forma deseada.

Una vez que las neuronas han sido agrupadas en capas, las operaciones individuales llevadas a cabo por cada neurona pueden entenderse conjuntamente como operaciones de tensores. Cuando la red neuronal está destinada a procesar imágenes, se trabaja con tensores 3D, correspondiendo los dos primeros canales a la altura y la anchura. El tercer canal tendrá una profundidad de 3 si se trabaja con imágenes RGB, o de 1 si se trabaja con escala de grises. En este nuevo escenario x, w y b pasarán a ser tensores de valores (matrices bidimensionales en el caso de tratar con imágenes) que serán modulados por el tipo de capa ($f$):

$$y = f(x \cdot w) \tag{2}$$

Finalmente, estas capas se agrupan para formar un modelo de Deep Learning. [17].

## 3.2 Entrenamiento

Para que un determinado modelo realice la tarea deseada, debe contener el conocimiento necesario para llevarla a cabo. La capacidad de almacenar conocimiento y, por lo tanto, de llevar a cabo la tarea asignada a una determinada red neuronal, reside en los valores que toman los pesos y las funciones de activación de las neuronas que la conforman. Estos valores se inicializan de forma aleatoria por lo que, en consecuencia, es necesario ajustarlos gradualmente para lograr la funcionalidad requerida. En términos generales, el ajuste de pesos y umbrales en una red se suele llevar a cabo a través de un proceso iterativo de presentación repetida de ejemplos de la tarea requerida. En cada presentación, se efectúan pequeños cambios en los pesos y umbrales para alinearlos más con los valores idóneos. Este proceso se conoce como "entrenamiento de la red" y el conjunto de ejemplos se denomina conjunto de entrenamiento. A la red se le presenta un conjunto de patrones de entrada o vectores ($x_i$) y, para cada uno de estos, un vector de salida deseado u objetivo ($t_i$). Por tanto, la red ($f$) debería responder con $t_k$, dada la entrada $x_k$. Este proceso se conoce como entrenamiento supervisado, debido a que a la red se le indica o supervisa en cada paso lo que se espera que haga [14].

Esto se puede representar matemáticamente de la siguiente manera:

$$t_k = f(x_k) \tag{3}$$

### 3.3 Función de *Loss* o coste

La función de *loss* o función de coste representa una medida del error obtenido en una tarea en cuestión, por lo que se utiliza para estimar el grado de acierto obtenido por el resultado de la predicción en relación con el resultado esperado. Idealmente, se busca que el valor obtenido sea cero, es decir, sin divergencia entre el valor estimado y el esperado. Por tanto, a medida que se va entrenando el modelo, los pesos de las interconexiones de las neuronas se irán ajustando paulatinamente hasta obtener mejores predicciones [18].

En este proyecto se ha optado por utilizar el índice de Jaccard, también conocido como Intersection over Union como función de coste. Se profundizará mas adelante en este aspecto en el apartado de evaluación.

### *3.4 Backpropagation*

Una vez calculado el valor de *loss*, esta información se propaga hacia atrás. De ahí su nombre: *backpropagation*. Es decir, que a partir de la capa de salida, ese valor de *loss* se propaga a todas las neuronas de la capa oculta que contribuye directamente a la salida. Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la señal total de *loss*, en función de la contribución relativa que cada neurona haya aportado a la salida original. Este proceso se repite, propagándose hacia las capas precedentes, hasta que todas las neuronas de la red hayan recibido una señal de *loss* que describa su contribución relativa al valor de *loss* total. Una vez se ha transmitido esta información, se deben ajustar los pesos de las conexiones entre las neuronas. Mediante este proceso, lo que se busca es hacer que el valor de *loss* sea lo más cercano posible a cero la próxima vez que se utilice la red para una predicción. Para este fin, se emplea un optimizador. El optimizador indica como se han de modificar los pesos mediante pequeños ajustes con la ayuda del cálculo de la derivada (o gradiente) de la función de *loss*, lo que permite ver en qué dirección “descender” hacia el mínimo global [18].

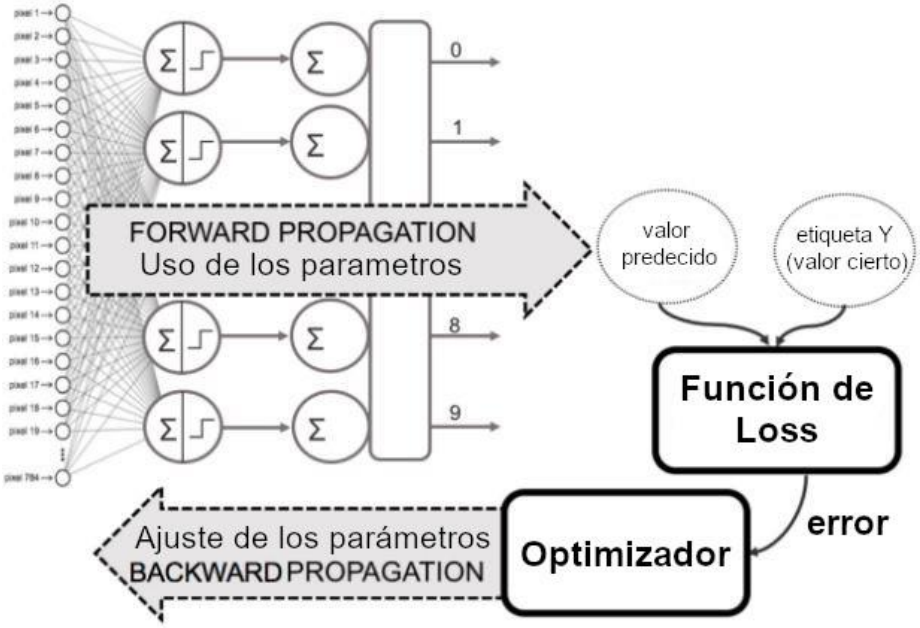



**Ilustración 6.** Optimización de los parámetros de una red neuronal [18].

## 3.5 Optimizador

En general, podemos ver el proceso de aprendizaje como un problema de optimización global, donde ciertos parámetros deben ajustarse de tal manera que se minimice el resultado de la función de *loss* presentado anteriormente. En la mayoría de los casos estos parámetros no se pueden resolver analíticamente. Generalmente, se pueden abordar mediante algoritmos de optimización. Por lo tanto, el optimizador es la herramienta que establece cómo se actualizarán los pesos la red neuronal en función del valor de *loss* transmitido por la capa consecuente. Existen diversos tipos de optimizadores, entre los más comunes se encuentran el GD[6] y el SGD[7].

El descenso de gradiente utiliza la primera derivada (gradiente) de la función de *loss* para actualizar los parámetros. El proceso consiste en encadenar las derivadas de del valor de *loss* de cada capa oculta a partir de las derivadas del valor de *loss* de su capa superior, incorporando en el cálculo su función de activación. En cada una de las iteraciones, una vez que todas las neuronas tienen el valor del gradiente de la función de *loss* que les corresponde, los valores de los parámetros se actualizan en sentido contrario al indicado por el gradiente. El gradiente, de hecho, siempre apunta en la dirección en que aumenta el valor de la función de *loss*. Por lo tanto, si se utiliza el negativo del gradiente, se obtiene la dirección en la que disminuye el valor de *loss*. En el ejemplo anterior se trabaja simultáneamente con el conjunto de datos completo.

[6] Descenso de gradiente
[7] Descenso estocástico de gradiente

Sin embargo, existe también la posibilidad de realizar estos cálculos individualmente para cada dato de entrenamiento e iterar sobre el conjunto completo uno por uno. Este procedimiento se conoce como SGD. Además, también es posible iterar sobre el conjunto de datos analizando pequeños subconjuntos o lotes (*batches*) de datos en cada iteración, en lugar de analizar cada dato individualmente. Este tipo de optimizador se denomina gradiente estocástico de *mini-batch* (MBSGD).

Estos dos métodos estocásticos presentan ciertos inconvenientes estadísticos frente al GD. Sin embargo, aportan la gran ventaja de simplificar considerablemente los cálculos efectuados y el espacio de memoria necesario para almacenar los datos usados en cada iteración del entrenamiento [19].

En este estudio se ha optado por el optimizador Adam con un *learning rate* de 0,01. Este hiperparametro determina el tamaño del paso en cada iteración mientras se desplaza hacia el mínimo de una función de coste. El algoritmo de optimización Adam es una extensión del descenso estocástico de gradiente que se basa en la estimación adaptativa de momentos de primer y segundo orden. Este optimizador añade un *learning rate* adaptativo en lugar de uno constante como SGD, lo que facilita el ajuste de este hiperparametro. Además, es uno de los optimizadores más eficientes y con menor requerimiento de memoria.

## *3.6 Overfitting*

El problema fundamental en el aprendizaje automático es la tensión entre la optimización y la generalización. La optimización se refiere al proceso de ajustar un modelo para obtener el mejor índice de acierto posible en los datos de entrenamiento (el aprendizaje en el aprendizaje automático), mientras que la generalización nos indica el grado de adecuación del modelo entrenado al trabajar con nuevos ejemplos nunca antes vistos [17].

Al comienzo del entrenamiento, la optimización y la generalización están correlacionadas, por lo que cuanto mayor sea la precisión en los datos de entrenamiento, mayor será la precisión en los datos de test. Mientras esto sucede, se dice que le modelo está subajustado o *underfitted*, todavía hay progreso por hacer; la red no ha tenido tiempo para modelar todos los patrones relevantes presentes en los datos de entrenamiento. Después de un cierto número de iteraciones en los datos de entrenamiento, la generalización y las métricas de validación dejan de mejorar para finalmente comenzar a degradarse. El modelo, en este punto, está comenzando a sobreajustarse (*overfit*). Es decir, está comenzando a aprender patrones que son específicos de los datos de entrenamiento, pero que son perjudiciales o irrelevantes cuando se tratan nuevos datos [17].

Para evitar que un modelo aprenda patrones engañosos o irrelevantes que se encuentran en los datos de entrenamiento, la mejor solución es ofrecerle un mayor conjunto de entrenamiento. Un modelo entrenado con mayor cantidad y variabilidad de datos generalizará mejor. Cuando eso no es posible, la siguiente mejor solución es modular la cantidad de información que el modelo puede almacenar o agregar restricciones sobre qué información se permite almacenar. Si una red solo puede permitirse memorizar una pequeña cantidad de patrones, el proceso de optimización le obligará a enfocarse en los patrones más destacados, que tienen más posibilidades de generalizarse bien [17].

## 3.7 Convolutional Neural Networks (CNN)

Cuando se trabaja con tareas de visión por computador se suele emplear un caso concreto de redes neuronales: las Redes Neuronales Convolucionales, también conocidas como *covnets*. Las CNN[8] son muy similares a los demás tipos de redes, excepto por el hecho de que asumen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades en la arquitectura para reconocer elementos específicos en las imágenes. Las *covnets* utilizan capas de convolución en lugar de las capas densas que suelen utilizarse en otras redes neuronales. Estas capas aprenden patrones locales, estos patrones se buscan mediante pequeñas ventanas o *kernels* 2D que recorren la entrada. Esto proporciona dos características esenciales a las CNN [17].

Por un lado, tras aprender un determinado patrón en cierta posición de la imagen, una *convnet* puede reconocerlo en cualquier otro lugar. En cambio, una red densamente conectada tendría que aprender de nuevo el patrón si apareciera en una nueva ubicación. Esto hace que los datos de las redes de convolución sean eficientes cuando se procesan imágenes (porque el mundo visual es fundamentalmente invariable a la traslación). Debido a esta invarianza de posición, necesitan menos muestras de entrenamiento para aprender representaciones que tienen poder de generalización [17].

Por otro lado, pueden aprender jerarquías espaciales de patrones. Una primera capa de convolución aprenderá pequeños patrones locales, como los bordes, mientras que una segunda capa de convolución aprenderá patrones más complejos compuestos por las características de las primeras capas, y así sucesivamente. Esto permite a las redes de convolución aprender de forma eficaz conceptos visuales cada vez más complejos y abstractos (véase *Ilustración 6*) [17].

La operación de convolución extrae parches del tensor de entrada y aplica la misma transformación a todos estos parches, produciendo un mapa de características de salida. Este mapa de características de salida sigue siendo un tensor 3D. Sin embargo, la profundidad de salida es un parámetro de la capa, y los diferentes canales de ese eje de profundidad ya no representan colores específicos, como en la entrada RGB, sino que representan filtros. Los filtros codifican aspectos específicos de los datos de entrada: en un nivel alto, un solo filtro podría codificar el concepto "presencia de una cara en la entrada", por ejemplo [17].

[8] *Convolutional neural network*

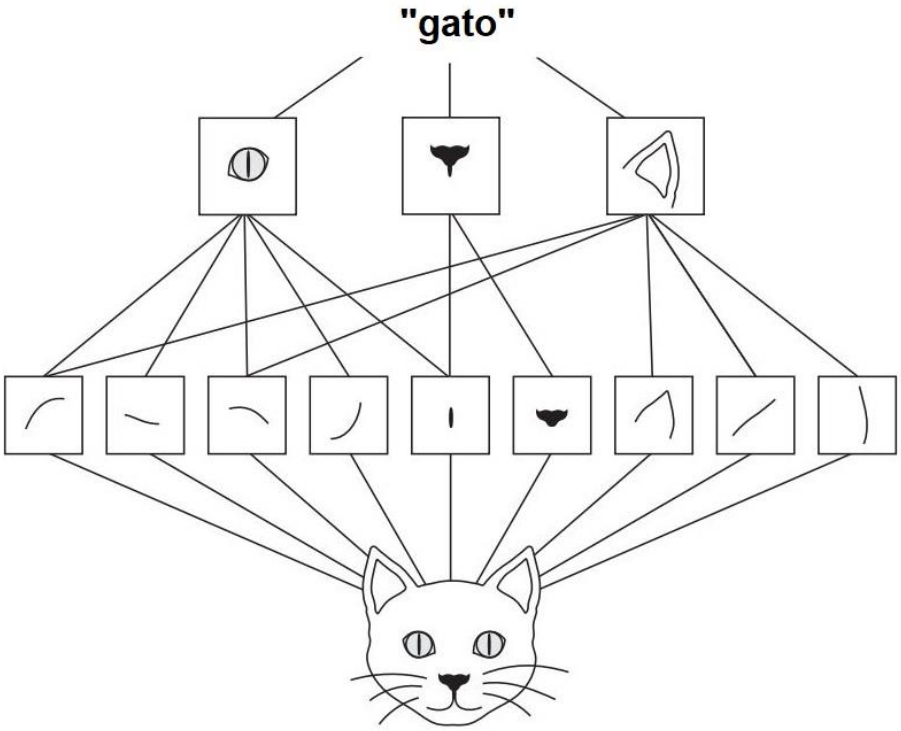



**Ilustración 7.** Jerarquía espacial de los módulos visuales, adaptado de [17].

### 3.8 U-net

En este caso, se ha optado por utilizar una *covnet* con una estructura de tipo U-Net [20]. Una U-Net es una arquitectura de CNN desarrollada específicamente para la segmentación de imágenes biomédicas. Las U-Nets son muy eficaces para tareas en las que la salida tiene un tamaño similar al de la entrada. Esta caracteristica las hace idóneas para crear máscaras de segmentación y para la generación de imágenes, como la superresolución o la coloración. Habitualmente, cuando las redes neuronales convolucionales se utilizan para la clasificación de imágenes, se toma la imagen y se reduce la muestra en una o más clasificaciones utilizando una serie de convoluciones que reducen cada vez el tamaño de la matriz o tensor que forma la imagen. Para poder generar una imagen del mismo tamaño que la de entrada, o más grande, es necesario que haya una ruta de expansión que aumente el tamaño de la matriz. Esto hace que el diseño de la red se asemeje a la forma de la letra U. Una U-Net está compuesta por una ruta de contracción o codificador, que forma la parte izquierda de la U, y es la responsable de extraer información a partir de la imagen de entrada. La U-Net también cuenta con una ruta de expansión o decodificador, que forma la parte derecha de la U y es la responsable de generar la máscara de segmentación a partir de las características extraídas por el codificador. Ambas rutas están formadas por cinco bloques de convolución o niveles. Como característica distintiva, para transmitir la información necesaria durante el proceso ascendente, la U-Net contiene conexiones de salto o *skip connections* que alimentan los bloques del decodificador con su bloque equivalente del codificador. En total, una U-Net convencional está formada por 23 capas convolucionales [20].

La ruta de contracción o codificación sigue la arquitectura típica de una red convolucional, con 5 bloques que reducen sucesivamente la resolución espacial para captar características de nivel cada vez superior. Cada bloque o nivel consiste en la aplicación repetida de dos convoluciones 3x3, cada una seguida de una ReLU[9] y una operación de 2x2 max pooling con desplazamiento 2. Así, en cada nivel de la codificación se disminuye la resolución espacial a la mitad, a la vez que se duplica el número de canales de características.

La ruta de expansión o decodificador trata de construir un mapa de segmentación a partir de las características codificadas. Cada bloque de la ruta expansiva duplica la resolución espacial y reduce a la mitad el número de canales de características mediante una convolución 2x2 ("convolución ascendente"). Seguidamente, concatena el resultado con la salida de su bloque simétrico en el codificador (la conexión de salto previamente mencionada). A continuación, aplica dos convoluciones 3x3, cada una de ellas seguida de una ReLU. En la capa final se utiliza una convolución 1x1 para asignar cada vector de características de 64 componentes al número de clases deseado.

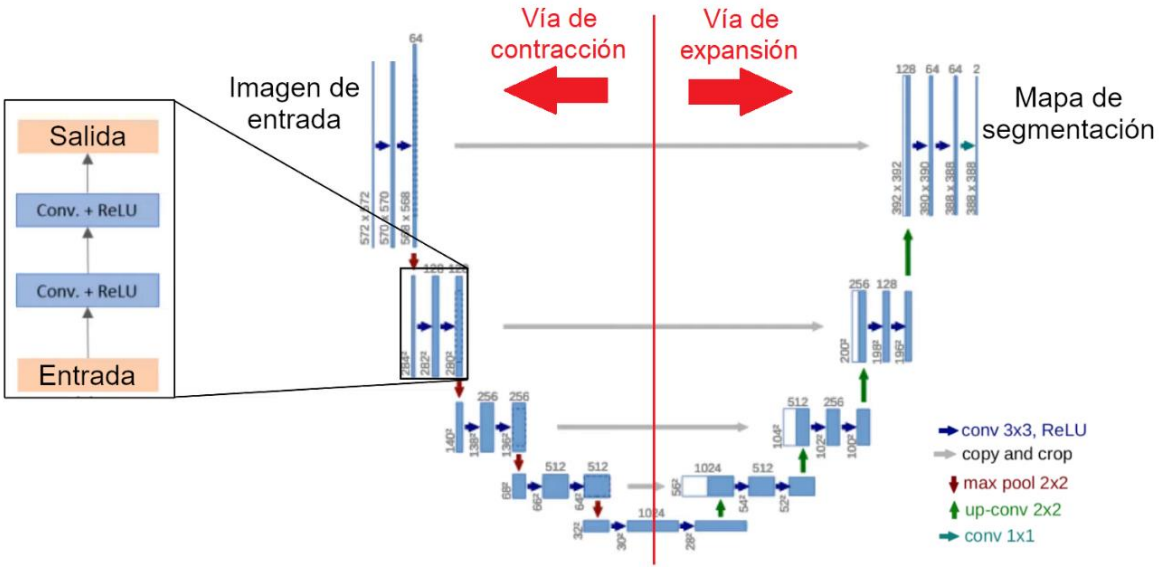



**Ilustración 8.** Estructura de una U-Net convencional [21]

Sin embargo, en el presente estudio se ha decidido sustituir el *backbone* original de U-Net, es decir, la ruta de contracción previamente expuesta, por un modelo diferente. Como consecuencia de esto, la ruta expansiva de U-net debe ser necesariamente modificada en función del *backbone* empleado. Con este propósito se ha utilizado la librería Segmentation Models [22]. Esta librería ofrece varios modelos, entre los que se encuentra U-Net. Además, proporciona múltiples *backbones* populares, entre los que se encuentran VGG, ResNet, Inception o MobileNet entre otros. En este caso se ha optado por utilizar ResNet34, una red residual de 34 capas de profundidad. [23].

[9] Unidad lineal rectificada

Como se ha descrito previamente, el valor de *loss* se transmite desde las capas finales hacia las capas iniciales mediante el *backprogapagation*. Cuando se multiplican las derivadas de muchas capas, el gradiente disminuye exponencialmente y se obtiene un valor muy pequeño que es inútil para el cálculo del gradiente. Este problema se conoce como el problema del gradiente desvaneciente (*vanishing gradient*). Las redes residuales o ResNets modifican los bloques convencionales que habitualmente contienen las redes convolucionales añadiéndoles nuevas conexiones internas que saltan capas intermedias. Por lo tanto, una capa puede estar conectada con la capa siguiente y, al mismo tiempo, con la capa 2 o 3 posiciones posterior. Gracias a estas conexiones, se pueden propagar gradientes mayores a las capas iniciales, y estas capas también podrán aprender tan rápido como las capas finales, lo que ayuda a obtener una curva de *loss* suave y proporciona la capacidad de entrenar redes más profundas. Estos bloques modificados se conocen como *ResBlocks*. Al haber sustituido el codificador original por ResNet34 es necesario modificar de igual manera el decodificador.

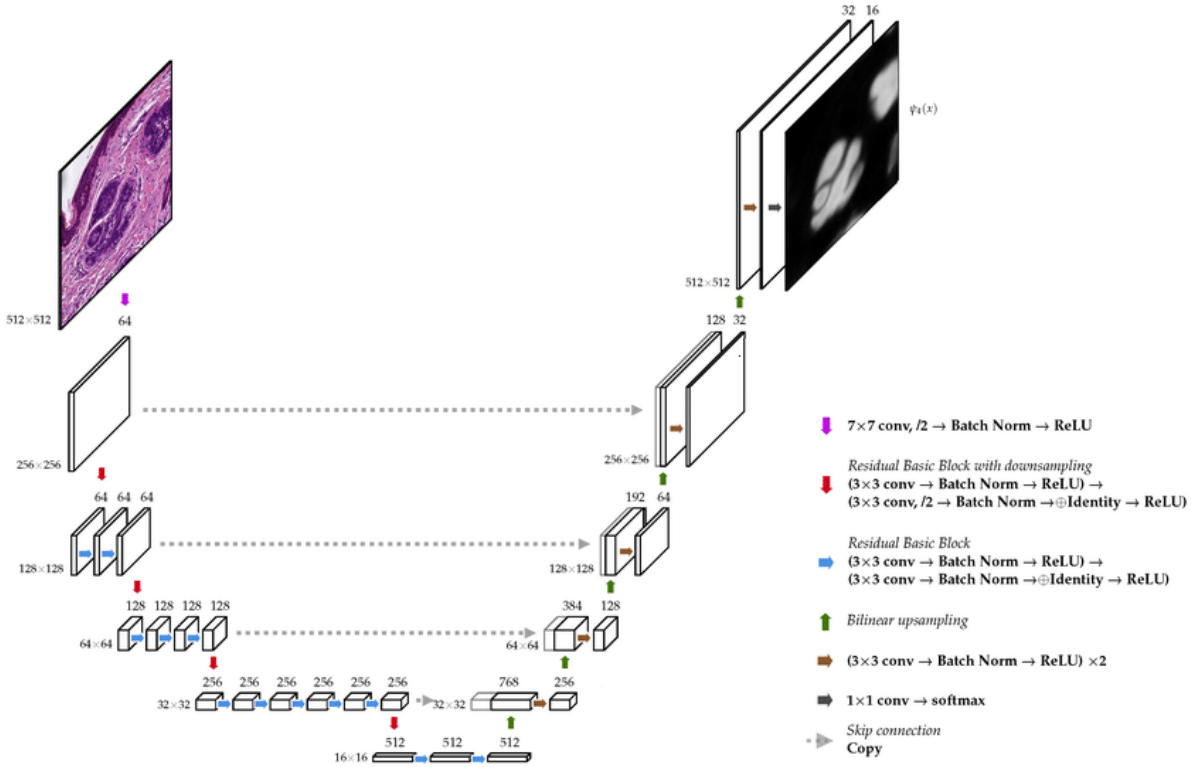



**Ilustración 9.** Estructura de una U-Net con ResNet34 como *backbone* [24].

Para una ilustración más detallada de la estructura de la red neuronal véase en enlace en el *Anexo* 5.

# 4 Segmentación de Imágenes

Actualmente, se pueden encontrar gran variedad de técnicas de visión por computador, entre las que se encuentran, entre otras, la clasificación de imágenes, la detección de objetos o la segmentación. En el caso de este proyecto, se busca identificar visualmente tanto los distintos tejidos como el área que está siendo invadida por un posible tumor. Se trata, por lo tanto, de un problema de segmentación. La segmentación de imágenes se puede dividir en dos categorías principales: segmentación semántica y segmentación de instancias. La primera de estas se refiere a la definición más fundamental de segmentación de imágenes: la identificación, agrupación y etiquetado de píxeles en una imagen que forma un objeto completo. De manera similar, en la segmentación de instancias, se etiquetan las clases presentes en la imagen. Sin embargo, aquí cada nuevo objeto se etiqueta como una instancia diferente, incluso dentro de la misma clase. Además, existe una tercera variante: la segmentación panóptica, consistente en una tarea a medio camino entre las dos anteriores, puesto que se realiza una segmentación de instancias únicamente de los objetos determinados (como personas y coches) y no así de los "materiales" (como el cielo o la montaña) [25].

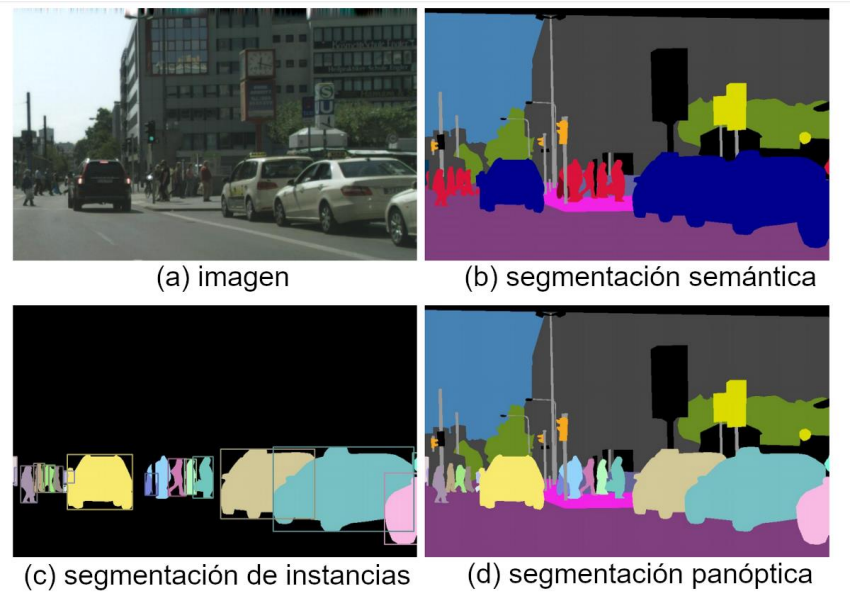



**Ilustración 10.** Tipos de segmentación de imágenes [25].

El presente proyecto pretende identificar el grado de penetración que ha alcanzado el tumor en base a los tejidos con los que se encuentra en contacto. Por lo tanto, en el caso de los tejidos es completamente irrelevante clasificar áreas independientes de un mismo tejido cómo instancias diferenciadas. En cuanto al tumor, si pudiera darse el caso de que en una misma imagen se encontrasen dos tumores independientes, sí podría ser útil una segmentación de instancias, o incluso una segmentación panóptica en la que los tejidos se segmenten semánticamente, pero identifiquen los diferentes tumores como instancias. Esto permitiría conocer el grado de penetración de cada tumor. Sin embargo, no se ha encontrado ninguna imagen en el conjunto de entrenamiento en la que se puedan apreciar dos tumores diferenciados. Además, el equipo médico confirmó que no sería necesario tener esta

posibilidad en consideración. En conclusión, se ha optado por un sistema de segmentación semántica, la cual permite diferenciar tanto los tejidos como el tumor presentes en la muestra como clases diferenciadas.

## 4.1 Codificación de las Imágenes

Como se ha mencionado previamente, al trabajar en un problema de segmentación semántica el input del sistema es la imagen que se desea segmentar, mientras que el output del sistema consiste en una máscara de segmentación. Una máscara de segmentación consiste en una imagen con idénticas dimensiones a las de la imagen a la que segmenta. Sin embargo, a diferencia de la imagen original, los píxeles no contienen información de color o nivel de gris. En su lugar, los píxeles contienen una etiqueta de clase representada por un número entero. Por lo tanto, ya sea para una imagen en escala de grises, de dimensiones (H x W x 1) o a color, de dimensiones (H x W x 3), la máscara de segmentación resultante será una imagen de (H x W x 1) en la que los píxeles tomarán valores enteros entre 0 y N-1, siendo N el número total de clases entre las que se deberá discernir.

Por último, es necesario destacar que, para que la máscara de segmentación pueda ser utilizada por la CNN, es necesario codificarla en formato *one-hot enconded*. Este formato de codificación consiste en transformar la máscara en una matriz de (H x W x N), donde cada etiqueta con un valor entero será sustituida por un 1 en su índice correspondiente en la tercera dimensión de la matriz.

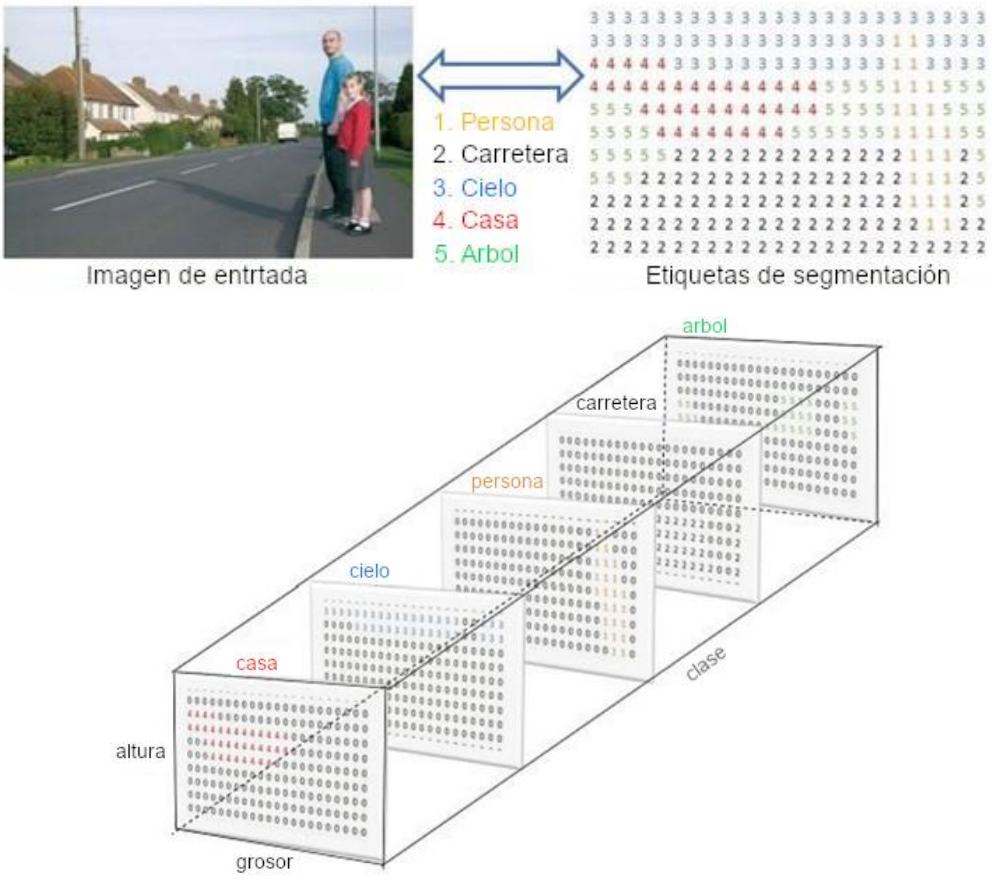



**Ilustración 11**. Representación de una imagen y su máscara de segmentación correspondiente [26].

# 5 Implementación

## 5.1 Estructura del Sistema

La forma más lógica de implementar el sistema requerido por el equipo médico es utilizar una única red neuronal que segmente la totalidad de las clases presentes en la imagen. Siguiendo esta premisa, se han planteado dos posibilidades. La primera opción consistiría en una CNN que segmente la imagen en once clases: el fondo y cinco tipos de tejidos posibles, los cuales a su vez pueden estar afectados o no por el tumor. Puesto que el objetivo final es identificar qué tejidos han sido invadidos por el tumor, esta opción es la que más información aporta y la que indica, sin lugar a duda, qué tejidos han sido afectados. Sin embargo, las modificaciones en la morfología celular producidas por el cáncer podrían suponer un impedimento para una red neuronal que aspire a segmentar los tejidos comprendidos dentro de la región del tumor. Se profundizará en este aspecto en el apartado de resultados. Una segunda opción es utilizar una CNN que segmente únicamente seis clases: el fondo, los cinco tipos de tejidos y el tumor, entendido éste como un tejido más.

A pesar de considerar que cualquiera de las dos opciones anteriores serían preferibles, ha sido necesario recurrir a una tercera opción debido a la limitación de tiempo por parte de los oncólogos, previamente mencionada en el apartado "Datos". Etiquetar imágenes en QuPath es un trabajo que ocupa grandes cantidades de tiempo. Por consiguiente, y por requerimiento del equipo médico, ha sido necesario reducir al mínimo posible el número de clases a etiquetar. Además, al comienzo del proyecto ya se disponía de cierta cantidad de imágenes segmentadas de pacientes sanos; es decir, con los 5 tejidos segmentados, pero sin tumor. Por estas dos razones, se ha optado por crear dos redes neuronales independientes. La primera de las redes neuronales, HTP[10], lleva a cabo la tarea de segmentar los tejidos, mientras que la segunda red, CTP[11], se encarga de segmentar el área que está siendo afectada por el tumor. Para mostrar el resultado final, se superpondrán las dos máscaras de segmentación obtenidas.

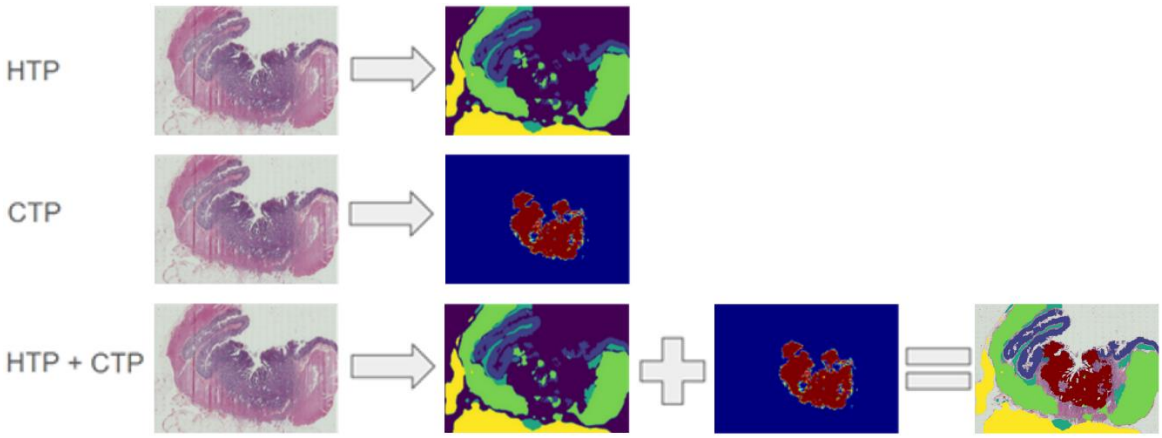



**Ilustración 12.** División del proyecto en dos redes neuronales independientes.

[10] *Healthy Tissue Predictor*
[11] *Cancer Tissue Preditor*

Habitualmente, al tratar con tareas de segmentación semántica suelen utilizarse miles de imágenes de entrenamiento [27]. Sin embargo, debido a los inconvenientes temporales anteriormente expuestos, la disponibilidad de datos de entrenamiento ha sido mucho menor. En un principio se contaba con únicamente 50 imágenes de pacientes sanos, es decir, para el entrenamiento de HTP. A pesar de que a lo largo del proyecto se han ido recopilando más imágenes, únicamente se ha alcanzado a recopilar apenas unas 180 imágenes para cada una de las CNN. En consecuencia, el principal desafío del proyecto ha sido el de lograr combinar las diferentes técnicas que han ayudado a mitigar el impacto negativo de dicho inconveniente. En concreto, se han aplicado tres técnicas: Aumentado de datos, Transferencia de aprendizaje y *Tiling*.

## 5.2 Aumentado de Datos

Como se ha mencionado previamente, la mejor solución para evitar el overfitting es proporcionar un mayor *dataset* a la red neuronal, lo que ofrece mayor diversidad durante el entrenamiento, logrando así que el sistema resultante generalice mejor. Cuando no es posible obtener más imágenes nativas, existe la posibilidad de aplicar transformaciones sintéticas a los datos de los que se dispone. Es decir, que en un escenario del mundo real podemos utilizar un *dataset* de imágenes tomadas en un conjunto limitado de condiciones. Sin embargo, se puede desear que la aplicación final continúe siendo funcional para una variedad de condiciones, como diferente orientación, ubicación, escala, brillo, etc. Esto se puede conseguir entrenando la red neuronal con datos adicionales modificados sintéticamente.

Las bibliotecas TensorFlow y Keras ofrecen image_data_generator()[13], una herramienta que permite realizar aumentado de datos de forma sencilla. Mediante dicha utilidad se pueden aplicar múltiples transformaciones de forma aleatoria sobre un conjunto de datos. Sin embargo, al tratarse de una tarea de segmentación semántica, es necesario aplicar exactamente las mismas transformaciones a la imagen de entrada y a su máscara de segmentación correspondiente. Debido a esto, hay algunas limitaciones al establecer los tipos de trasformaciones que se podrán aplicar. Por citar algún ejemplo, la rotación o el escalado requieren generar nuevos píxeles, cuyos valores son calculados por Keras mediante interpolación. Sin embargo, a pesar de que esta operación es válida en las imágenes de entrada, resulta problemática en las máscaras de segmentación. En concreto, y dado que en las máscaras de segmentación cada píxel tiene un valor entero correspondiente a la clase a la que pertenece, al calcular nuevos píxeles mediante interpolación se obtienen resultados con valores reales, lo cual no es válido. Para solventar este inconveniente, se valoró la opción de redondear los valores de cada píxel una vez realizado el aumentado de datos para obtener una máscara de segmentación válida. No obstante, de esta forma no se tendría la certeza de que estos píxeles estén siendo clasificados correctamente. Por consiguiente, se optó por descartar las transformaciones que puedan resultar problemáticas y recurrir únicamente a simetrías y modificaciones de brillo.

Por otra parte, image_data_generator() ofrece la posibilidad de utilizar, alternativamente, una lista, un *dataframe* de Pandas o de iterar sobre un directorio. En este caso se ha optado por emplear la tercera opción, a fin de evitar cargar en memoria la inmensa cantidad de imágenes que se han obtenido mediante la técnica de *tiling*, expuesta en el próximo apartado. Esto se lleva a cabo mediante la función flow_from_directory(), que está diseñada para ser empleada en tareas de clasificación de imágenes, por lo que genera lotes de imágenes y una lista con las etiquetas correspondientes. Estas etiquetas se infieren a partir de los directorios en los que están almacenadas las imágenes. Para utilizar esta herramienta en un problema de segmentación es necesario aplicar ciertas modificaciones al resultado retornado por la función.

En primer lugar, se crea un único generador de imágenes que aplique las modificaciones deseadas al *dataset*. En segundo lugar, es necesario llamar dos veces a la función flow_from_directory() del generador previamente declarado. La primera de las veces se especifica como fuente el directorio que contiene las imágenes de entrada, mientras que en la segunda llamada a la función se especifica el directorio que contiene las máscaras. Además, es imprescindible pasar una misma semilla en ambas llamadas a la función para que el recorrido siga el mismo orden en ambos directorios. Asimismo, se debe pasar el parámetro class_mode como None para que Keras no infiera automáticamente la clase a la que pertenece cada imagen a partir del directorio que la contiene. De esta forma se obtienen dos generadores de imágenes independientes. Uno será utilizado para modificar las imágenes de entrada y el segundo para modificar sus respectivas máscaras de segmentación, pero las modificaciones serán exactamente iguales en ambos casos. Para unir ambos generadores en un solo generador se utiliza la función zip(), la cual itera sobre varios iterables en paralelo, produciendo tuplas con un elemento de cada uno. Por último, se itera sobre el nuevo generador de tuplas aplicando el preprocesado deseado a las imágenes y se vuelven a unir con la función yield(). Véase *Anexo 2*.

En cuanto a la función de preprocesado, en primer lugar, es necesario preprocesar la imagen de entrada exactamente de la misma forma en la que se preprocesaron las imágenes de preentrenamiento. Segmentation_models.get_preprocessing() ofrece las funciones de preprocesamiento necesarias para este paso. A continuación, se deberán preprocesar las máscaras de segmentación. debido a que el script de QuPath exporta cada etiqueta con un color aleatorio, por lo que, previamente, habrá que substituir el color respectivo de cada clase por el valor entero que se le ha decidido asignar. Esto se realiza de forma sencilla mediante la función numpy.where(). Finalmente, se debe codificar la matriz en formato *one-hot*, lo que se realiza de forma automática mediante keras.util.to_categorical().

## 5.3 Tiling

Otra de las técnicas que se decidió aplicar es el *tiling*. Esta técnica consiste en dividir las imágenes originales en ventanas más pequeñas. Se trata de un procedimiento habitualmente utilizado en imágenes de gran resolución (principalmente imágenes satelitales), las cuales, y debido a las limitaciones en el hardware actual, no pueden ser procesadas en su totalidad en su resolución original [28].

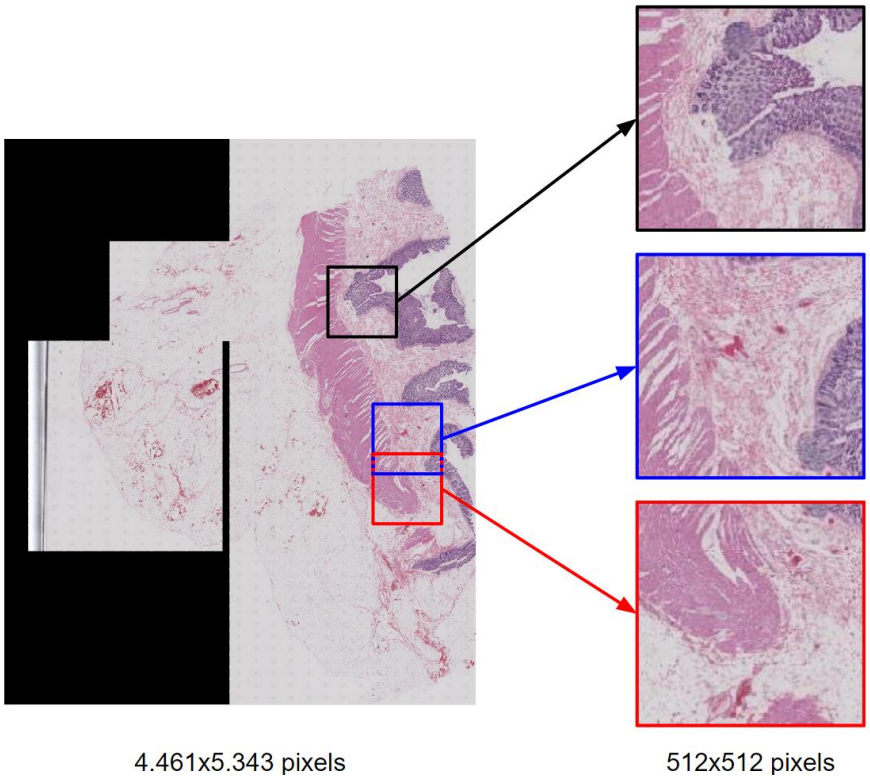



**Ilustración 13.** División de una imagen de gran resolución en tiles de 512 píxeles.

La forma más sencilla de aplicar esta técnica es el *random cropping*. Esto consiste en seleccionar pequeñas ventanas de la imagen original aleatoriamente, de esta forma, se logra generar múltiples imágenes que muestran diferentes partes de una misma muestra. Sin embargo, de esta forma no se asegura que se utilice la totalidad de cada imagen durante el entrenamiento, por lo que se estaría desaprovechando cierta parte de la información disponible para el entrenamiento. Esto es especialmente grave en casos como el presente trabajo, en el que la disponibilidad de datos es especialmente limitada. Para evitar este inconveniente se optó por utilizar una técnica diferente de muestreo. Se ha implementado un algoritmo que divide las imágenes originales en ventanas de forma ordenada. En primer lugar, es necesario definir el tamaño de las ventanas deseadas (N), el avance (A) de estas ventanas, y el porcentaje mínimo de contenido útil que deben tener estas ventanas para que no sean descartadas. Con estos parámetros, el script comienza a recorrer la imagen original desde la esquina superior izquierda y genera una nueva ventana a partir de los primeros NxN píxeles de la imagen. Después, la ventana se desplaza A píxeles a la derecha para generar una nueva ventana. Una vez se ha alcanzado el extremo derecho de la imagen, la ventana

vuelve al extremo izquierdo y baja A píxeles para continuar avanzando con el muestreo. Este proceso se realiza de igual manera para las imágenes como para sus máscaras de segmentación correspondientes. Una vez finalizado este proceso, se analizan las ventanas generadas a partir de las máscaras de segmentación en busca de ventanas en las que los píxeles clasificados como "no fondo" sean superior al porcentaje indicado inicialmente. Cuando se encuentra una máscara de segmentación que no cumple este criterio, se elimina tanto la máscara como la ventana de entrada correspondiente a esta. Véase *Anexo 3*.

Aplicar este procedimiento aporta múltiples ventajas. En primer lugar, las redes neuronales suelen tratar con imágenes cuadradas, sin embargo, el conjunto de datos aportado por los oncólogos está formada por imágenes que no respetan este formato. Es más, si las imágenes no fueran cuadradas, pero siguieran una relación de aspecto constante, cabría la posibilidad de deformarlas todas para hacerlas cuadradas, puesto que las transformaciones morfológicas serian equivalentes en todas las imágenes. Sin embargo, las imágenes no siguen un formato constante, lo que obliga a descartar dicha opción. Este problema se soluciona al dividir cada imagen en pequeñas muestras cuadradas, ya que de esta forma se pueden obtener muestras cuadradas a partir de imágenes de cualquier tamaño y relación de aspecto.

En segundo lugar, el mayor factor limitante durante el desarrollo de este proyecto ha sido la disponibilidad de datos. Al dividir las imágenes originales en pequeñas ventanas se logra generas una extensa base de dantos que resuelve el problema de la escasez de datos para el entrenamiento. Por ejemplo, el aplicar esta técnica en un *dataset* con 47 imágenes de una resolución del orden de 5000x5000 píxeles, se ha logrado generar 50.584 imágenes de 512x512 píxeles con una superposición del 25%. Sin embargo, muchas de estas ventanas únicamente contienen fondo, por lo que tras descartar estas nuevas imágenes que no aportan información relevante, se han obtenido 24.514 imágenes para el entrenamiento.

En tercer lugar, lograr un mejor rendimiento del modelo con resoluciones de imagen de entrada más bajas podría parecer inicialmente paradójico, pero, en varios paradigmas de aprendizaje automático, un número reducido de entradas o características es deseable como medio para reducir el número de parámetros que deben optimizarse, lo que a su vez disminuye el riesgo de sobreajuste del modelo. Sin embargo, la reducción de la resolución de la imagen elimina información útil para la clasificación. Además, la unidad de procesamiento gráfico puede tener limitaciones de memoria en las que el uso de una mayor resolución de imagen puede reducir el tamaño máximo de lote utilizable, y un mayor tamaño de lote puede permitir un mejor cálculo del gradiente con respecto a la función de pérdida [29]. Por consiguiente, gracias a la técnica aplicada, se obtienen los veneficios de trabajar con menores resoluciones sin necesidad de sacrificar información útil.

En cuarto lugar, en su mayoría, los conjuntos de datos utilizados habitualmente involucran imágenes de baja resolución (256x256) que incluyen objetos considerablemente grandes con una gran cobertura de píxeles. Por lo tanto, los modelos preentrenados brindan un rendimiento de detección muy exitoso para esos tipos de datos de entrada. En cambio, suelen tener una precisión significativamente menor en tareas de detección de objetos pequeños en imágenes de alta resolución. Al dividir las imágenes originales de muy alta resolución en las pequeñas ventanas de baja resolución, se logra producir un conjunto de datos en el que los objetos originalmente pequeños pasan a ocupar una gran parte de los píxeles [30].

Por lo tanto, esta técnica ha posibilitado generar un gran conjunto de datos a partir de las pocas imágenes disponibles inicialmente, a la vez que permite analizar imágenes de

alta resolución de forma óptima, y se mantiene una ejecución rápida con baja complejidad, mientras que se mantiene un pequeño requerimiento de memoria.

Se ha considerado trabajar con *tiles* de 256x256 píxeles, 512x512 píxeles y 1024x1024 píxeles. Al utilizar ventanas con tamaños menores se ha observado que se favorece la detección de cuerpos pequeños y claramente definidos como los linfocitos, mientras que se penaliza la identificación de cuerpos grandes y dispersos como la subserosa, en cuyas segmentaciones se presentaban huecos internos clasificados como fondo. Al trabajar con ventanas de mayor tamaño el efecto es el contrario, los cuerpos de mayor tamaño se clasifican de forma más eficiente pero el sistema presenta dificultad a la hora de identificar cuerpos pequeños y muy locales como los mencionados linfocitos. Por lo tanto, se ha optado por continuar con un tamaño de *tile* de 512x512 píxeles, el cual se ha considerado un tamaño "intermedio". Con este tamaño de ventana se ha encontrado un punto de equilibrio en la efectividad a la hora de clasificar los objetos de distintos tamaños.

Sin embargo, utilizar esta técnica agrega complejidad al sistema. Al haber entrenado la red para trabajar con estas pequeñas ventanas, es necesario realizar la inferencia con imágenes equivalente. Así pues, cuando se desea efectuar una predicción sobre una nueva imagen, también es necesario dividirla en ventanas, tratarlas por separado, y, finalmente, unir las máscaras de segmentación generadas para obtener el resultado final. Asimismo, el número de hiperparametros a tener en cuenta es mayor, ya que, tanto el tamaño de las baldosas utilizadas, como el nivel de superposición, afectan al funcionamiento del modelo.

## 5.4 Transfer Learning

La necesidad de recurrir a transfer learning se produce cuando hay un suministro limitado de datos de entrenamiento. Esto puede deberse a que los datos son escasos, a que son difíciles de recopilar y etiquetar o, simplemente, a que son inaccesibles. En cualquier caso, y dado que los repositorios de *big data* son cada vez más frecuentes, el uso de conjuntos de datos existentes que están relacionados con un dominio de interés, pero que no son exactamente iguales, hace que las soluciones de aprendizaje por transferencia sean un enfoque atractivo.

Dado un dominio de origen $D_O$ con una tarea de origen correspondiente $T_O$ y un dominio de destino $D_D$ con una tarea correspondiente $T_D$, la transferencia de aprendizaje consiste en el proceso de mejora de la función predictiva de origen $f_O(x)$ para obtener la función predictiva de destino $f_D(x)$ mediante el uso de la información útil de $D_O$ y $T_O$ [31]. Expresado de otra forma, en lugar de inicializar los pesos de la red neuronal de forma aleatoria, se toman como punto de partida los valores obtenidos al entrenar esta misma red neuronal con un conjunto de datos diferente al deseado. Con esta técnica la red neuronal aprende a identificar patrones generales que pueden resultar útiles para cualquier tarea del mismo ámbito, en este caso la segmentación de imágenes. Después, la red neuronal vuelve a entrenarse con el conjunto de datos propio acabando de ajustar los pesos para que lleven a cabo la tarea deseada.

Como se ha mencionado previamente, se ha utilizado la librería Segmentation Models para obtener un modelo de U-net combinado con ResNet34. Una de las ventajas que proporciona esta librería es que ofrece todos sus modelos disponibles preentrados con ImageNet. El proyecto ImageNet consiste en una gran base de datos de imágenes diseñada

para su uso en la investigación de software de reconocimiento de objetos visuales. El repositorio dispone de 14 millones de imágenes anotadas a mano y categorizadas en más de 20.000 clases. Por lo tanto, se ha utilizado la red neuronal preentrenada con esta gran base de datos para agilizar el proceso de aprendizaje y disminuir la cantidad de imágenes de entrenamiento necesarias.

# 6 Evaluación

Para evaluar el rendimiento del sistema implementado se han reservado 20 imágenes de pacientes con tumor. Aunque habitualmente suele reservarse una proporción considerablemente mayor, alrededor de un 20%, en este caso se ha optado por utilizar menos imágenes para el testeo debido a la escasez de imágenes de entrenamiento.

En muchos estudios se aborda el problema de la segmentación de imágenes empleando funciones de coste simples como Categorical Cross Entropy, que evalúa la precisión media del modelo píxel por píxel. Esto puede dar lugar a problemas. Por ejemplo, con un *dataset* en el que todas las muestras tengan gran cantidad de fondo se obtendrán buenos resultados clasificando todos los píxeles como fondo si únicamente se evalúa el porcentaje de acierto total. A tal efecto, se ha optado por utilizar el índice de Jaccard, o IoU[12], como medida del rendimiento obtenido. Dada una imagen, la medida IoU da la similitud entre la región predicha y la región etiquetada para un objeto presente en la imagen, y se define como la medida de la intersección dividido por la unión de las dos regiones [32]. IoU es más representativa que otras aproximaciones que evalúan la precisión por píxel. Sin embargo, el índice de Jaccard no es diferenciable y, por lo tanto, no se puede utilizar como función de coste. Para solucionar este problema se recurre a una aproximación de IoU. Cuando se utiliza esta aproximación la función se convierte en diferenciable y se puede utilizar como una función de pérdida [33].

$$IoU = \frac{Positivos\ ciertos}{Positivos\ ciertos + Positivos\ falsos + Negativos\ falsos} \qquad (1)$$

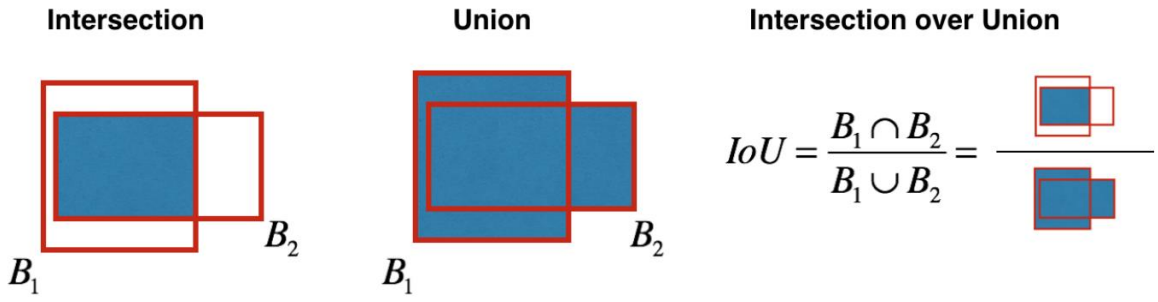



**Ilustración 14.** Representación gráfica de Intersection over Union [34].

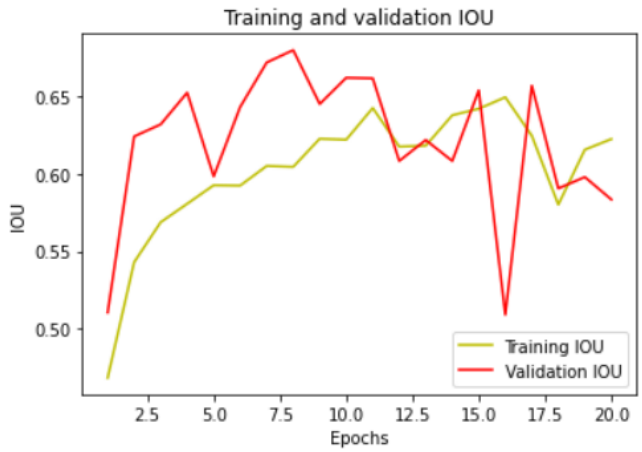
[12] *Intersection over Union*

Además, IoU no sólo puede ser calculado para evaluar el rendimiento medio de la red neuronal en todas las clases en su conjunto, sino que también es posible realizar esta operación para cada una de las clases por separado, de forma que se obtiene el índice de acierto para cada una de las clases individualmente. Esto aporta mucha más información sobre el funcionamiento del sistema y permite identificar qué clases, tejidos en este caso, resultan más problemáticas.

Por otra parte, durante el entrenamiento de la CNN es indispensable contar con una función de coste que proporcione un valor numérico para evaluar el modelo y resulta extremadamente útil contar con una métrica objetiva del rendimiento de éste. Sin embargo, debido a consideraciones tales como la complejidad del modelo desarrollado, su división en dos redes neuronales y a que los propios usuarios finales han sido parte activa del desarrollo, en última instancia, se ha tenido en gran consideración la evaluación manual de los resultados por parte del equipo médico.

Aunque el objetivo final se segmentar las imágenes completas, por las razones anteriormente expuestas, se ha optado por segmentar pequeñas baldosas de forma individual. En consecuencia, durante el proceso de entrenamiento únicamente cabe la posibilidad de calcular el índice de Jaccard para las baldosas. Sin embargo, en la fase de testeo se puede calcular este índice tanto para las baldosas de forma independiente como para una imagen completa después de haber sido reconstruida. La máscara final se obtiene a partir de calcular medias de baldosas con cierto grado de superposición, por lo que puede haber cierta discrepancia en los valores de IoU obtenidos en baldosas e imágenes completas. Por esta razón a continuación se exponen tanto los valores calculados con baldosas individuales como con imágenes reconstruidas.

En cuanto a los resultados de la segmentación de tejidos realizada con HTP, en la mayoría de los casos las predicciones han resultado correctas. Todos los tejidos han quedado claramente delimitados con diferentes colores, a excepción de las regiones afectadas por el tumor (analizado más adelante). Al realizar un análisis visual, en la mayoría de las situaciones el rendimiento ha sido el esperado, e incluso superior. Prácticamente la totalidad de las etiquetas coinciden a la perfección con los tejidos que se debían identificar. Además, las etiquetas resultantes forman cuerpos continuos, uniformes y sin huecos en su interior, lo cual representaba una de las principales preocupaciones al dividir la imagen original en pequeñas baldosas.




```
Maximum accuracy reached:  0.6794504523277283
Maximum accuracy reached at epoch:  8
```

**Ilustración 15.** Evaluación de la precisión (IoU) obtenida durante el entrenamiento de Healthy Tissue Predictor.

| Tejido | IoU baldosas individuales | IoU máscara reconstruida |
|---|---|---|
| Media | 0.77 | 0.85 |
| Fondo | 0.76 | 0.86 |
| Mucosa | 0.84 | 0.90 |
| Linfocitos | 0.67 | 0.82 |
| Submucosa | 0.72 | 0.80 |
| Muscular | 0.91 | 0.94 |
| Subserosa | 0.72 | 0.77 |

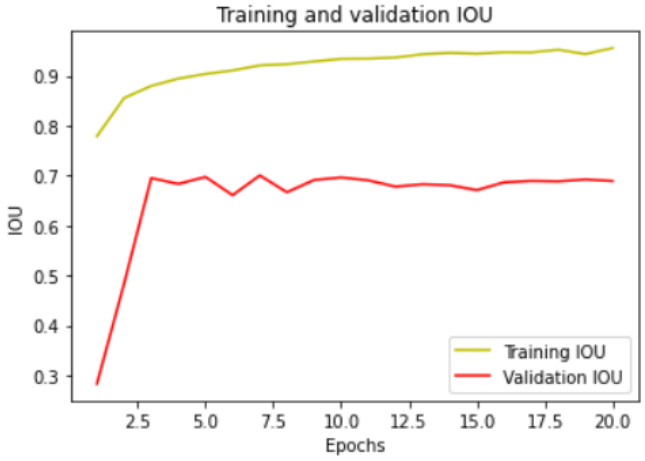
**Tabla 2.** Valores de IoU obtenidos en Healthy Tissue Predictor para el conjunto de test.

Los valores de IoU obtenidos muestran una tasa de acierto media del 85%. Los resultados son especialmente notables al tratar el tejido muscular y la mucosa. Presumiblemente, esto se debe a que se trata de tejidos muy diferenciados y que toman colores que contrastan especialmente con el resto de la imagen. Por el contrario, los tejidos más difusos como la submucosa o la subserosa han obtenido valores de IoU menores. Esto se debe a dos motivos: por un lado, al tratarse de tejidos menos diferenciados, son más difíciles de segmentar y, consecuentemente, el rendimiento del sistema es algo menor. Por otro lado, y por la misma razón, los oncólogos tampoco han tenido totalmente claro por dónde delimitar los tejidos, y, por lo tanto, en algunos casos el sistema ha clasificado correctamente partes de estos tejidos que no habían sido etiquetadas por el equipo médico. Este último punto ha sido confirmado por los médicos que colaboran en el proyecto.

También se han encontrado casos excepcionales en los que la segmentación ha resultado completamente errónea. No obstante, esto ha sucedido al procesar imágenes discordantes que no se asemejan en absoluto al conjunto de entrenamiento. Uno de los casos en los que se ha dado este problema es el expuesto en la imagen 14-5731 del *Anexo* 4. En dicha imagen no se reconocen claramente los tejidos presentes, lo que contrasta con el resto de la casuística, en la que se aprecia una clara diferencia tanto en el color, la textura y la estructura de cada tejido.

Se ha observado que la predicción de tejidos no arroja ningún resultado en las áreas afectadas por tumor. Aunque fuera previsible que en estas regiones la predicción no fuera correcta debido a las alteraciones sufridas por los tejidos, se esperaba que los tejidos fueran clasificados como tejido, aunque ello resultara erróneo (véase *Anexo* 4 imagen 14-5804). En cambio, en la mayoría de los casos se ha encontrado que la región afectada por el tumor es clasificada como fondo. En un principio esto ha resultado desconcertante y no se ha encontrado una explicación clara. Una hipótesis podría ser que los cambios son de tal relevancia que el sistema no encuentra más similitud con otros tejidos que con el fondo y, debido al desequilibrio de clases, la clase "fondo" resulta privilegiada durante el entrenamiento.

En primera instancia resulta extraño que los valores de IoU obtenidos en el conjunto de test sean notablemente superiores a los valores obtenidos en el conjunto de validación durante el entrenamiento. Esto se debe a que durante el entrenamiento se realiza la predicción de cada baldosa y, posteriormente, se evalúa la predicción obtenida individualmente. Por el contrario, al evaluar el sistema con el conjunto de test, se realiza la predicción de cada baldosa, se aplica el algoritmo que unifica las baldosas reconstruyendo la máscara de segmentación completa y, por último, se calcula el valor de IoU de esta. Se ha observado que el realizar las predicciones de las baldosas el sistema tiende a clasificar los márgenes de estas como fondo. Esto explica porque al unir varias baldosas con superposición, el resultado global es considerablemente superior al resultado medio de cada una de las partes. Este mismo fenómeno se ha dado en la red CTP.




```
Maximum accuracy reached:  0.7005333304405212
Maximum accuracy reached at epoch:  7
```

**Ilustración 16.** Evaluación de la precisión (IoU) obtenida durante el entrenamiento de Cancer Tissue Predictor.

| Tejido | IoU baldosas individuales | IoU máscara reconstruida |
|---|---|---|
| Media | 0.67 | 0.97 |
| No tumor | 0.72 | 0.99 |
| Tumor | 0.62 | 0.94 |

**Tabla 3.** Valores de IoU obtenidos en Cancer Tissue Predictor para el conjunto de test.

En cuanto a la segmentación de tumores por parte de CTP, el resultado ha sido algo confuso. En algunos de los casos nos hemos encontrado con que el tumor estaba completamente segmentado. En otros casos, hemos podido observar que únicamente algunas partes del tumor habían sido segmentadas y permanecían zonas interiores del tumor sin clasificarse como tumor. Además, en la práctica totalidad de los casos las zonas más limítrofes del tumor tampoco han sido catalogadas como cáncer, lo que resulta en una mayor dificultad para identificar automáticamente los tejidos invadidos por éste. Sin embargo, los resultados de IoU obtenidos para CTP han resultado muy satisfactorios, lo que no concuerda con el análisis visual.

Este problema se consultó con el equipo de oncólogos, los cuales se manifestaron satisfechos con el resultado. En las imágenes de entrenamiento únicamente se etiquetó el estroma del tumor, parte muy importante en el crecimiento, invasión, vascularización, metástasis y resistencia a la penetración del tratamiento farmacéutico. Por lo tanto, CTP ha aprendido a identificar exclusivamente el estroma del tumor, excluyendo las partes más periféricas de este. Esto explica la incoherencia entre los resultados observados a simple vista y los valores numéricos calculados.

Al superponer las dos máscaras resultantes de sus respectivas redes neuronales se presenta una imagen clara, en la que se distingue de forma completamente evidente cada tejido y cuáles de éstos están siendo afectados por el tumor. Dejando a un lado los casos aberrantes en como la imagen 14-5731 del *Anexo* 4, la principal carencia encontrada es la distancia entre los limites del tumor y el tejido adyacente. Esto es problemático, ya que en un principio se ha propuesto inferir el grado de penetración a partir de la superposición entre ambas máscaras. A pesar de esto, se ha encontrado que en la mayoría de los casos sí se da cierta superposición en una pequeña cantidad de los píxeles más limítrofes, dando lugar a una deducción del estado del tumor correcta. Se espera que al volver a entrenar la red neuronal con el *dataset* corregido de tumor este problema pueda ser corregido, obteniendo una superposición de tumor y tejidos más pronunciada y, en consecuencia, proporcionando una estadificación mejor justificada.

# 7 Diagrama de Gantt

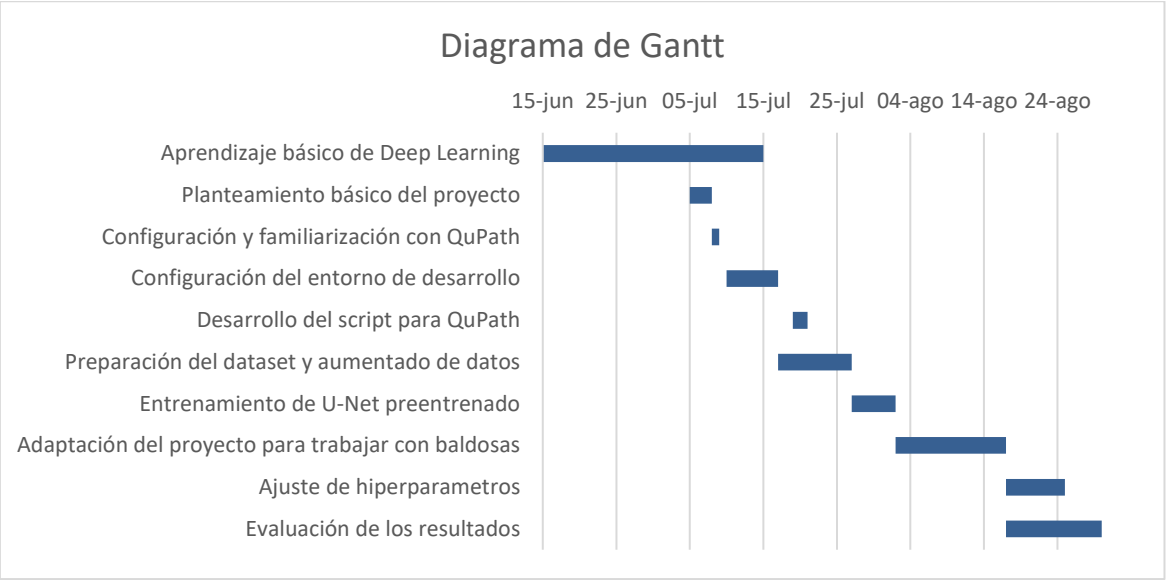



**Ilustración 17.** Diagrama de Gantt del desarrollo del proyecto.

# 8 Conclusiones y Perspectivas de Futuro

Como conclusión, podemos afirmar que se ha desarrollado un sistema de segmentación de tejidos y cáncer colorrectal funcional y con potencial para ser utilizado en el día a día por los profesionales sanitarios para agilizar el trabajo de estos. El principal escollo que se ha encontrado durante el desarrollo ha sido la limitada disponibilidad de datos de entrenamiento. Para solventar este problema, se ha recurrido a múltiples técnicas. De todas ellas, cabe destacar que la división de las imágenes originales en pequeñas baldosas de baja resolución ha sido el punto determinante que ha permitido sortear la escasez de datos.

Durante el proyecto se ha experimentado con diversas técnicas y posibilidades, además de desarrollar numerosas funciones orientadas a una cómoda visualización y análisis de los resultados obtenidos. Además, y en previsión de que gran parte del código desarrollado podría ser útil en futuros proyectos, se han reunido todo este código en un documento llamado segmentation_utils, de forma que sea fácilmente accesible a modo de librería. De hecho, ya ha resultado útil en otro proyecto desarrollado en ISS Biodonostia.

Respecto a las perspectivas de futuro cabe destacar, en primer lugar, que los oncólogos están inmersos en el proceso de volver a etiquetar las imágenes tumorales; esta vez tratando de abarcar la totalidad de las regiones afectadas por el cáncer en lugar de etiquetar exclusivamente el estroma del tumor. Se espera que esta modificación solucione el problema descrito. En vista del prometedor rendimiento del sistema, se procurará no solo corregir el etiquetado de las imágenes ya utilizadas, sino también recopilar nuevas imágenes para poder entrenar la ANN con una mayor cantidad de datos.

En cuento al problema de la clasificación de tejidos en las regiones afectadas, se ha considerado que las variaciones visuales ocasionadas por el cáncer son tales que confunden completamente a la red neuronal. Debido a la división en dos redes neuronales, este problema parece difícilmente subsanable. No obstante, los oncólogos han expresado que este punto no representa un problema especialmente relevante, dado que, el resultado obtenido les sirve de igual manera.

Otro aspecto para tener es cuenta es el backbone empleado. Aunque en un principio se ha optado por emplear ResNet34, existen otras opciones a tener en consideración. VGG, por ejemplo, está orientado a reducir el número de parámetros que contiene la red neuronal, reduciendo el coste computacional del entrenamiento. Otra opción interesante podría ser Inception, que emplea *kernels* de diferentes tamaños, que contribuyen a la identificación correcta tanto de características locales como generales. Esto podría ser conveniente debido al contraste entre las pequeñas regiones de linfocitos frente a las grandes y dispersas regiones de subserosa.

Cuando el proyecto alcance un nivel de desempeño que se considere suficiente para ser empleado por los médicos, será necesario desarrollar una interfaz gráfica de usuario que haga sencilla la utilización del sistema para éstos. Además, es necesario recordar que las redes neuronales requieren de gran capacidad de cómputo, más aún al trabajar con imágenes o videos de gran resolución. Por lo tanto, se considera necesario desplegar la aplicación en el clúster computacional de ISS Biodonostia para que los usuarios finales puedan acceder a ella remotamente desde sus ordenadores personales.

# Referencias


[1] "OMS Cancer," 2022. https://www.who.int/news-room/fact-sheets/detail/cancer (accessed Nov. 30, 2021).

[2] "Cancer Today," 2022. https://gco.iarc.fr/today/home (accessed Nov. 30, 2021).

[3] K. Simon, "Colorectal cancer development and advances in screening," *Clin Interv Aging*, vol. 11, pp. 967–976, Jul. 2016, doi: 10.2147/CIA.S109285.

[4] N. Dalal *et al.*, "Omics technologies for improved diagnosis and treatment of colorectal cancer: Technical advancement and major perspectives," *Biomedicine and Pharmacotherapy*, vol. 131, Nov. 2020, doi: 10.1016/J.BIOPHA.2020.110648.

[5] F. Petrelli *et al.*, "Prognostic survival associated with left-sided vs right-sided colon cancer a systematic review and meta-analysis," *JAMA Oncol*, vol. 3, no. 2, pp. 211–219, Feb. 2017, doi: 10.1001/JAMAONCOL.2016.4227.

[6] R. Labianca *et al.*, "Colon cancer," *Crit Rev Oncol Hematol*, vol. 74, no. 2, pp. 106–133, May 2010, doi: 10.1016/J.CRITREVONC.2010.01.010.

[7] "American Cancer Society," 2022. https://www.cancer.org/cancer/colon-rectal-cancer/detection-diagnosis-staging/survival-rates.html (accessed Jul. 24, 2022).

[8] P. Bankhead *et al.*, "QuPath: Open source software for digital pathology image analysis," *Sci Rep*, vol. 7, no. 1, Dec. 2017, doi: 10.1038/S41598-017-17204-5.

[9] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, "Medical Image Analysis using Convolutional Neural Networks: A Review," *J Med Syst*, vol. 42, no. 11, Nov. 2018, doi: 10.1007/S10916-018-1088-1.

[10] "TensorFlow." https://www.tensorflow.org/?hl=es-419 (accessed Dec. 03, 2022).

[11] "Keras: the Python deep learning API." https://keras.io/ (accessed Dec. 03, 2022).

[12] Osakidetza, "Programa de cribado de cáncer colorrectal," 2022. https://www.osakidetza.euskadi.eus/enfermedad-cancer/-/programa-cribado-cancer-colorrectal/ (accessed Aug. 04, 2022).

[13] TensorFlow, "tf.keras.preprocessing.image.ImageDataGenerator," 2022. https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator (accessed Aug. 07, 2022).

[14] K. Gurney and N. York, "An introduction to neural networks," 1997.

[15] "Artificial neuron," 2022. https://en.wikipedia.org/wiki/Artificial_neuron#External_links (accessed Jul. 28, 2022).

[16] R. Dastres and M. Soori, "Artificial Neural Network Systems," 2021. [Online]. Available: https://www.researchgate.net/publication/350486076

[17] F. Chollet, *DEEP LEARNING with Python*. 2017.

[18] J. Torres, "Learning Process of a Deep Neural Network," *Towards Data Science*, 2022. https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651 (accessed Aug. 01, 2022).

[19] E. Irby, "Gradient Descent vs Stochastic GD vs Mini-Batch SGD," *Analytics Vidhya*, 2021. https://medium.com/analytics-vidhya/gradient-descent-vs-stochastic-gd-vs-mini-batch-sgd-fbd3a2cb4ba4 (accessed Oct. 12, 2022).

[20] W. Weng and X. Zhu, "U-Net: Convolutional Networks for Biomedical Image Segmentation," *IEEE Access*, vol. 9, pp. 16591–16603, May 2015, doi: 10.48550/arxiv.1505.04597.

[21] Christopher Thomas, "U-Nets with ResNet Encoders and cross connections. ," *Towards Data Science*, 2019. https://towardsdatascience.com/u-nets-with-resnet-encoders-and-cross-connections-d8ba94125a2c (accessed Nov. 04, 2022).

[22] P. Iakubovskii, "Segmentation Models," *GitHub repository*, 2019. https://github.com/qubvel/segmentation_models#models-and-backbones (accessed Dec. 12, 2022).

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", Accessed: Jan. 02, 2023. [Online]. Available: http://image-net.org/challenges/LSVRC/2015/

[24] J. L. Arrastia *et al.*, "Deeply supervised unet for semantic segmentation to assist dermatopathological assessment of basal cell carcinoma," *J Imaging*, vol. 7, no. 4, Apr. 2021, doi: 10.3390/JIMAGING7040071.

[25] V. V.Kumar, "Panoptic Segmentation with UPSNet," *Towards Data Science*, 2019. https://towardsdatascience.com/panoptic-segmentation-with-upsnet-12ecd871b2a3 (accessed Oct. 14, 2022).

[26] S. Shekhar, "Pytorch implementation of Semantic Segmentation for Single class from scratch.," *Analytics Vidhya*, 2019. https://medium.com/analytics-vidhya/pytorch-implementation-of-semantic-segmentation-for-single-class-from-scratch-81f96643c98c (accessed Oct. 15, 2022).

[27] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks," *Int J Multimed Inf Retr*, vol. 7, no. 2, pp. 87–93, Jun. 2018, doi: 10.1007/S13735-017-0141-Z.

[28] G. A. Reina, R. Panchumarthy, S. P. Thakur, A. Bastidas, and S. Bakas, "Systematic Evaluation of Image Tiling Adverse Effects on Deep Learning Semantic Segmentation," *Front Neurosci*, vol. 14, p. 65, Feb. 2020, doi: 10.3389/FNINS.2020.00065/BIBTEX.

[29] C. F. Sabottke and B. M. Spieler, "The Effect of Image Resolution on Deep Learning in Radiography," *https://doi.org/10.1148/ryai.2019190015*, vol. 2, no. 1, p. e190015, Jan. 2020, doi: 10.1148/RYAI.2019190015.

[30] F. O. Unel, B. O. Ozkalayci, and C. Cigla, "The power of tiling for small object detection," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2019-June, pp. 582–591, Jun. 2019, doi: 10.1109/CVPRW.2019.00084.

[31] K. Weiss, T. M. Khoshgoftaar, and D. Wang Background, "A survey of transfer learning," *J Big Data*, 2016, doi: 10.1186/s40537-016-0043-6.

[32] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10072 LNCS, pp. 234–244, 2016, doi: 10.1007/978-3-319-50835-1_22/COVER.

[33] F. van Beers, "Using Intersection over Union loss to improve Binary Image Segmentation," Jul. 2018.

[34] Tzanidou Giounona, *Evaluating detection (Intersection Over Union)* , vol. Hands-On Convolutio…. Accessed: Dec. 14, 2022. [Online]. Available: https://www.oreilly.com/library/view/hands-on-convolutional-neural/9781789130331/a0267a8a-bd4a-452a-9e5a-8b276d7787a0.xhtml

# 9 Anexo 1: Exportado de Imágenes de QuPath

```
import qupath.lib.gui.images.servers.RenderedImageServer
import qupath.lib.gui.viewer.overlays.HierarchyOverlay

// Select a downsample factor (final img size = original size / downsample)
double downsample = 20

// Define export paths
def path_images = buildFilePath(PROJECT_BASE_DIR, './exports/images')
def path_annotations = buildFilePath(PROJECT_BASE_DIR,
                                     './exports/individual_annotations')
mkdirs(path_images)
mkdirs(path_annotations)
// Get filename
filename = getProjectEntry().getImageName()
filename = filename[0..<filename.lastIndexOf('.')] // Remove the extension

// Get the viewer & current image
def viewer = getCurrentViewer()
def imageData = getCurrentImageData()

// Original pixels server
def server1 = new RenderedImageServer.Builder(imageData)
    .downsamples(downsample)
    .build()

// Rendered rgb server
def server2 = new LabeledImageServer.Builder(imageData)
    .backgroundLabel(0, ColorTools.BLACK) // Specify background label (usually 0
                                     or 255)
    .downsample(downsample)    // Choose server resolution; this should match
                                     the resolution at which tiles are
                                     exported
    .addLabel('Mucosa', 1)      // Choose output labels (the order matters!)
    .addLabel('mucosa normal', 1)
    .addLabel('Linfocitos', 2)
    .addLabel('Immune cells', 2)
    .addLabel('Submucosa', 3)
    .addLabel('submucosa', 3)
    .addLabel('Muscular', 4)
    .addLabel('muscular propia', 4)
    .addLabel('Subserosa', 5)
    .addLabel('subserosa', 5)
    .lineThickness(0)          // Optionally export annotation boundaries with
                                     another label
    .setBoundaryLabel('Boundary*', 0) // Define annotation boundary label
    .multichannelOutput(false) // If true, each label refers to the channel of a
                                     multichannel binary image (required
                                     for multiclass probability)
    .build()

// Write the images
writeImage(server1, path_images+'/'+filename+'.jpg')
writeImage(server2, path_annotations+'/'+filename+'.png')
```

# 10 Anexo 2: Aumentado de Datos

```
def get_generator_from_directory(img_path, mask_path, size, mode,
                                 preprocess_function, augmentation=True,
                                 val_split=0.2, batch_size=32, seed=123):

    if(augmentation): data_gen_args = dict(validation_split=val_split,
                                           horizontal_flip=True, vertical_flip=True,
                                           fill_mode='reflect')

    else: data_gen_args = dict(validation_split=val_split)

    #same arguments in order to transform images and masks equally
    image_datagen = ImageDataGenerator(**data_gen_args)

    image_generator = image_datagen.flow_from_directory(img_path,
                                           target_size=(size, size),
                                           subset=mode,  #train or validation
                                           batch_size=batch_size, shuffle=True,
                                           class_mode=None, seed=seed)

   mask_generator = image_datagen.flow_from_directory(mask_path,
                                           target_size=(size, size),
                                           subset=mode,  #train or validation
                                           batch_size=batch_size,
                                           color_mode='grayscale', shuffle=True,
                                           class_mode=None, seed=seed)

    generator = zip(image_generator, mask_generator)
    for (img, mask) in generator:
        img, mask = preprocess_function(img, mask)
        yield (img, mask)


def preprocess_data(img, mask):
        scaler = MinMaxScaler()
        img = scaler.fit_transform(img.reshape(-1, img.shape[-1])).reshape(img.shape)
        # same as img = (img.astype('float32')) / 255.
        preprocess_input = sm.get_preprocessing(BACKBONE)
        img = preprocess_input(img)  #Preprocess based on the pretrained backbone...

        #change mask colors by class labels
        #notice that labels must be consistente with the CLASS dictionary
        mask=np.where(mask==225, 1, mask) #mucosa=1      #backgroud=0
        mask=np.where(mask==178, 2, mask) #linfocitos=2
        mask=np.where(mask==96,  3, mask) #submucosa=3
        mask=np.where(mask==131, 4, mask) #muscular=4
        mask=np.where(mask==105, 5, mask) #subserosa=5
        #Transform 1 channel labels to hot encoded (NUM_CLASSES channels)
        mask = tf.keras.utils.to_categorical(mask, num_classes=NUM_CLASSES)
        return (img, mask)
```

```
if __name__ == "__main__":
        # Get the generators
        train_generator = get_generator_from_directory(IMG_PATH, MASK_PATH,
                                                       size=TILE_SIZE, mode='training',
                                                       preprocess_function=preprocess_data,
                                                       augmentation=True,
                                                       val_split=VAL_SPLIT,
                                                       batch_size=BATCH_SIZE)

        # Validation data should never be augmented → augmentation = False
        val_generator = get_generator_from_directory(IMG_PATH, MASK_PATH,
                                                       size=TILE_SIZE, mode='validation',
                                                       preprocess_function=preprocess_data,
                                                       augmentation=False,
                                                       val_split=VAL_SPLIT,
                                                       batch_size=BATCH_SIZE)
```

# 11 Anexo 3: Tiling

```
def get_image_tiles(path, tile_size, step=None, print_resize=False,
                    dest_path=None):

    print('Reading images:')
    if(not step): step=tile_size

    image_list = []
    for directory_path in glob.glob(path):
        paths = sorted(glob.glob(os.path.join(directory_path, "*.jpg")))
        for img_path in paths:
            #update progress var
            percentage = 1/(len(paths)/(paths.index(img_path)+1))
            drawProgressBar(percentage, barLen = 50)

            img = cv2.imread(img_path, 1) #1 for reading image as BGR

            # Cut each image to a size divisible by tile_size
            original_width=img.shape[1] # useful for crop locations
            original_height=img.shape[0] # useful for crop locations
            width = (img.shape[1]//tile_size)*tile_size # get nearest width
                                                        divisible by tile_size
            height = (img.shape[0]//tile_size)*tile_size # get nearest height
                                                         divisible by tile_size
            img = Image.fromarray(img)
            img = img.crop((original_width-width ,0, original_width, height))
                    #Crop from top right corner ((left, top, right, bottom))
            img = np.array(img)
            if (print_resize): print('Cropped image size:', img.shape)

            # Extract patches from each image
            patches_img = patchify(img, (tile_size, tile_size, 3), step=step)
            for i in range(patches_img.shape[0]):
                for j in range(patches_img.shape[1]):
                    single_patch_img = patches_img[i,j,:,:]
                    single_patch_img = single_patch_img[0] #Drop the extra
                                                           unecessary dimension
                                                           that patchify adds.
                    image_list.append(single_patch_img)

                    # Saving the image
                    if dest_path is not None:
                        filename = img_path.rsplit( ".", 1 )[ 0 ]
                    #remove extension
                        filename = filename.rsplit( "/")[ -1 ]
                    #remove original path
                        filename = filename+' '+str(i)+'-'+str(j)+'.jpg'    # add
                    tile indexes
                        cv2.imwrite(dest_path+filename, single_patch_img)

    image_array = np.array(image_list)
    print('\nGot an image array of shape', image_array.shape, image_array.dtype)
    return(image_array)
```

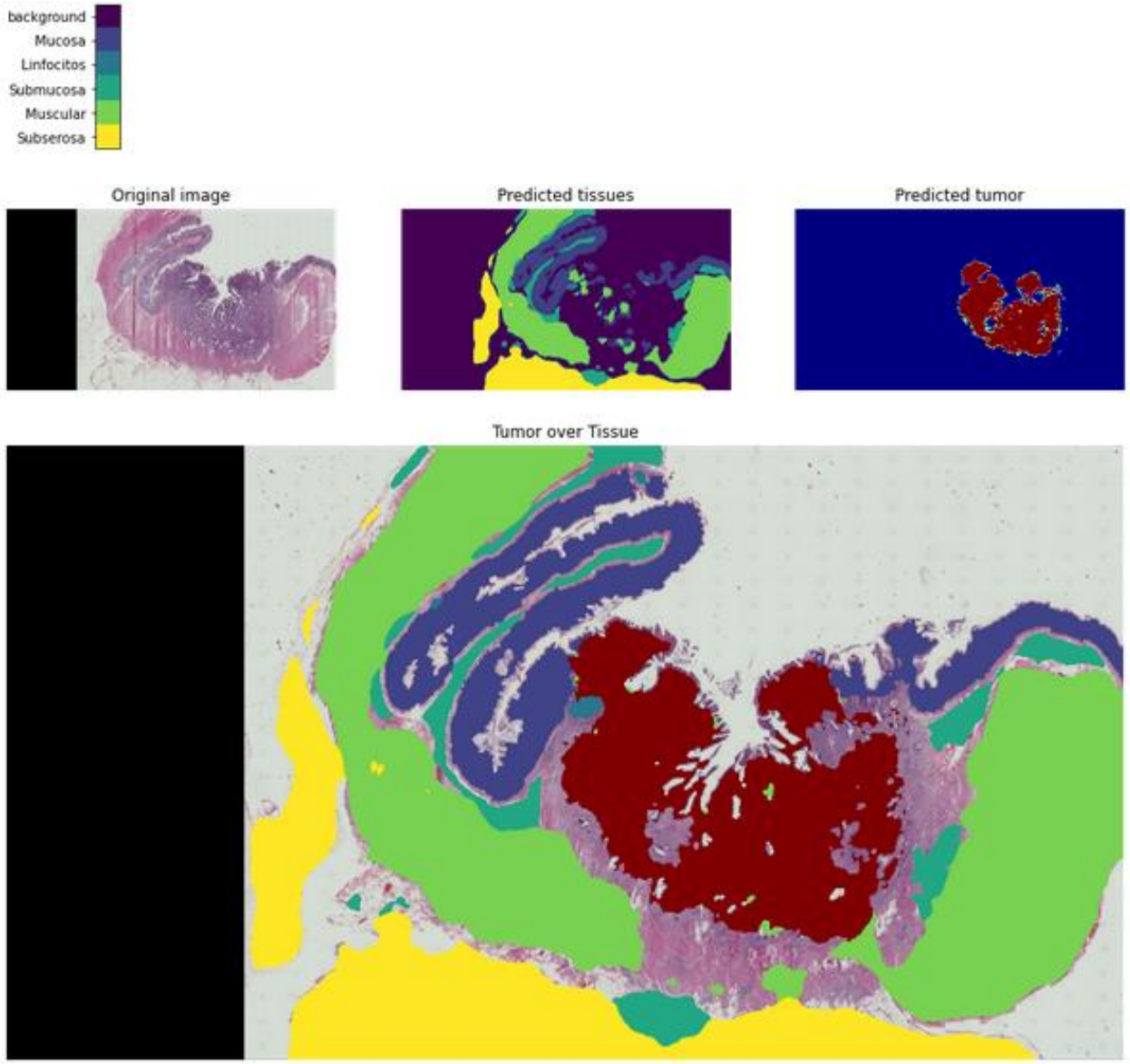
```
def get_mask_tiles(path, tile_size, step=None, print_resize=False,
                   dest_path=None):

    print('Reading images:')
    if(not step): step=tile_size

    mask_list = []
    for directory_path in glob.glob(path):
        paths = sorted(glob.glob(os.path.join(directory_path, "*.png")))
        for mask_path in paths:
            #update progress var
            percentage = 1/(len(paths)/(paths.index(mask_path)+1))
            drawProgressBar(percentage, barLen = 50)

            mask = cv2.imread(mask_path, 0) #0 for reading image as greyscale

            # Cut each image to a size divisible by tile_size
            original_width=mask.shape[1] # useful for crop locations
            original_height=mask.shape[0] # useful for crop locations
            width = (mask.shape[1]//tile_size)*tile_size # get nearest width
                                                         divisible by tile_size
            height = (mask.shape[0]//tile_size)*tile_size # get nearest height
                                                          divisible by tile_size
            mask = Image.fromarray(mask)
            mask = mask.crop((original_width-width ,0, original_width, height))
            #Crop from top right corner ((left, top, right, bottom))
            mask = np.array(mask)
            if (print_resize): print('Cropped mask size:', mask.shape)

            # Extract patches from each mask
            patches_mask = patchify(mask, (tile_size, tile_size), step=step)
            for i in range(patches_mask.shape[0]):
                for j in range(patches_mask.shape[1]):
                    single_patch_mask = patches_mask[i,j,:,:]
                    mask_list.append(single_patch_mask)

                    # Saving the mask
                    if dest_path is not None:
                        filename = mask_path.rsplit( ".", 1 )[ 0 ]
                    #remove extension
                        filename = filename.rsplit( "/")[ -1 ]
                    #remove original path
                        filename = filename+' '+str(i)+'-'+str(j)+'.png'
                    # add tile indexes
                        cv2.imwrite(dest_path+filename, single_patch_mask)

    mask_array = np.array(mask_list)
    print('\nGot a mask array of shape', mask_array.shape, mask_array.dtype,
          'with values', np.unique(mask_array))

    return(mask_array)
```

```
def get_useful_images(IMG_DIR, MASK_DIR, USEFUL_IMG_DIR, USEFUL_MASK_DIR,
                      PERCENTAGE=0.05):

    # needs to be sorted as linux doesn't list sorted
    img_list = sorted(os.listdir(IMG_DIR))
    msk_list = sorted(os.listdir(MASK_DIR))
    useless=0  #Useless image counter
    for img in range(len(img_list)):

        percentage = 1/(len(img_list)/(img+1))
        drawProgressBar(percentage, barLen = 50)

        img_name = img_list[img]
        mask_name = msk_list[img]

        temp_image = cv2.imread(IMG_DIR+img_list[img], 1)
        temp_mask = cv2.imread(MASK_DIR+msk_list[img], 0)

        val, counts = np.unique(temp_mask, return_counts=True)
        if (1 - (counts[0]/counts.sum())) > PERCENTAGE:
         #At least 5% useful area with labels that are not 0
            cv2.imwrite(USEFUL_IMG_DIR+img_name, temp_image)
            cv2.imwrite(USEFUL_MASK_DIR+mask_name, temp_mask)
        else: useless +=1;

    print("\nTotal useful images are: ", len(img_list)-useless)
```

```
import os
import sys
import shutil
import time
from time import gmtime, strftime
import segmentation_utils as su

if __name__ == "__main__":

    start_clock, start_time = time.clock(), time.time()

    ORIGINAL_PATH = sys.argv[1]+'/'      #path to the original images
    TILE_PATH = sys.argv[2]+'/'          #destination path for tiles
    TILE_SIZES = [1024, 512, 256, 128]   #list of tile sizes to generate

    for TILE_SIZE in TILE_SIZES:

        IMG_DIR = TILE_PATH+'/'+str(TILE_SIZE)+'_images/img/'
        MASK_DIR = TILE_PATH+'/'+str(TILE_SIZE)+'_masks/img/'
        USEFUL_IMG_DIR = TILE_PATH+'/'+str(TILE_SIZE)+'_useful_images/img/'
        USEFUL_MASK_DIR = TILE_PATH+'/'+str(TILE_SIZE)+'_useful_masks/img/'

        #Remove tile directories if exists and create
        if os.path.exists(IMG_DIR): shutil.rmtree(IMG_DIR)
        os.makedirs(IMG_DIR)
        if os.path.exists(MASK_DIR): shutil.rmtree(MASK_DIR)
        os.makedirs(MASK_DIR)
        if os.path.exists(USEFUL_IMG_DIR): shutil.rmtree(USEFUL_IMG_DIR)
        os.makedirs(USEFUL_IMG_DIR)
        if os.path.exists(USEFUL_MASK_DIR): shutil.rmtree(USEFUL_MASK_DIR)
        os.makedirs(USEFUL_MASK_DIR)

        #generate tiles
        print("_____________________________________________")
        print('\nCREATING', TILE_SIZE,'x',TILE_SIZE,'TILES\n')
        STEP = int(TILE_SIZE/4)
        su.get_image_tiles(ORIGINAL_PATH+'/images/img/', TILE_SIZE, step=STEP,
                           dest_path=IMG_DIR)
        su.get_mask_tiles(ORIGINAL_PATH+'/masks/img/', TILE_SIZE, step=STEP,
                           dest_path=MASK_DIR)

        #now, let us copy images and masks with real information to a new folder.
        print('\nChoosing the useful', TILE_SIZE,'x',TILE_SIZE,'tiles')
        su.get_useful_images(IMG_DIR, MASK_DIR, USEFUL_IMG_DIR, USEFUL_MASK_DIR)

    print('\nRun time:', strftime("%H:%M:%S", gmtime(time.time() - start_time)))
    print('CPU time:', strftime("%H:%M:%S", gmtime(time.clock() - start_clock)))
```

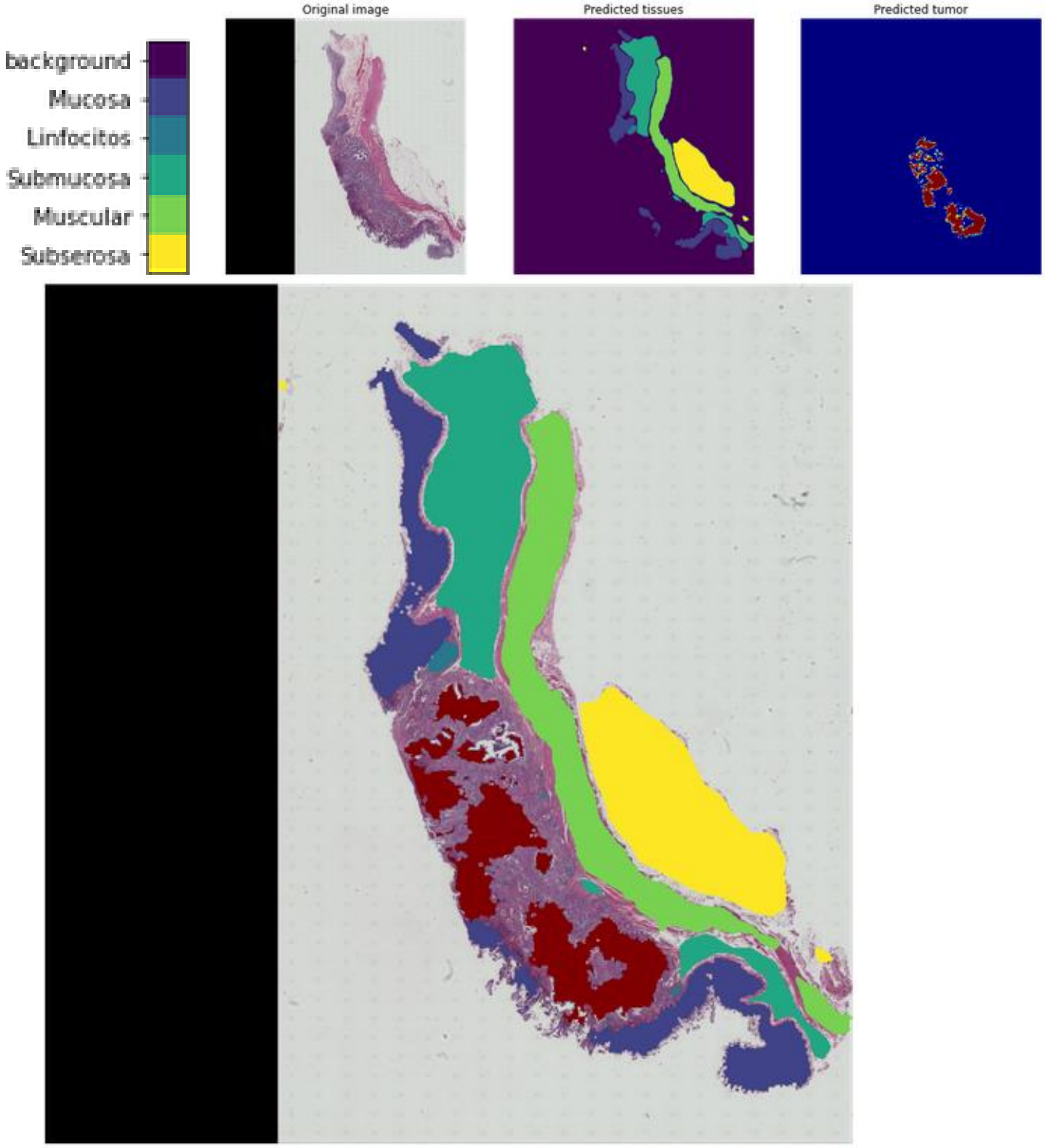
# 12 Anexo 4: Predicciones

## 12.1 Imagen: 14-833

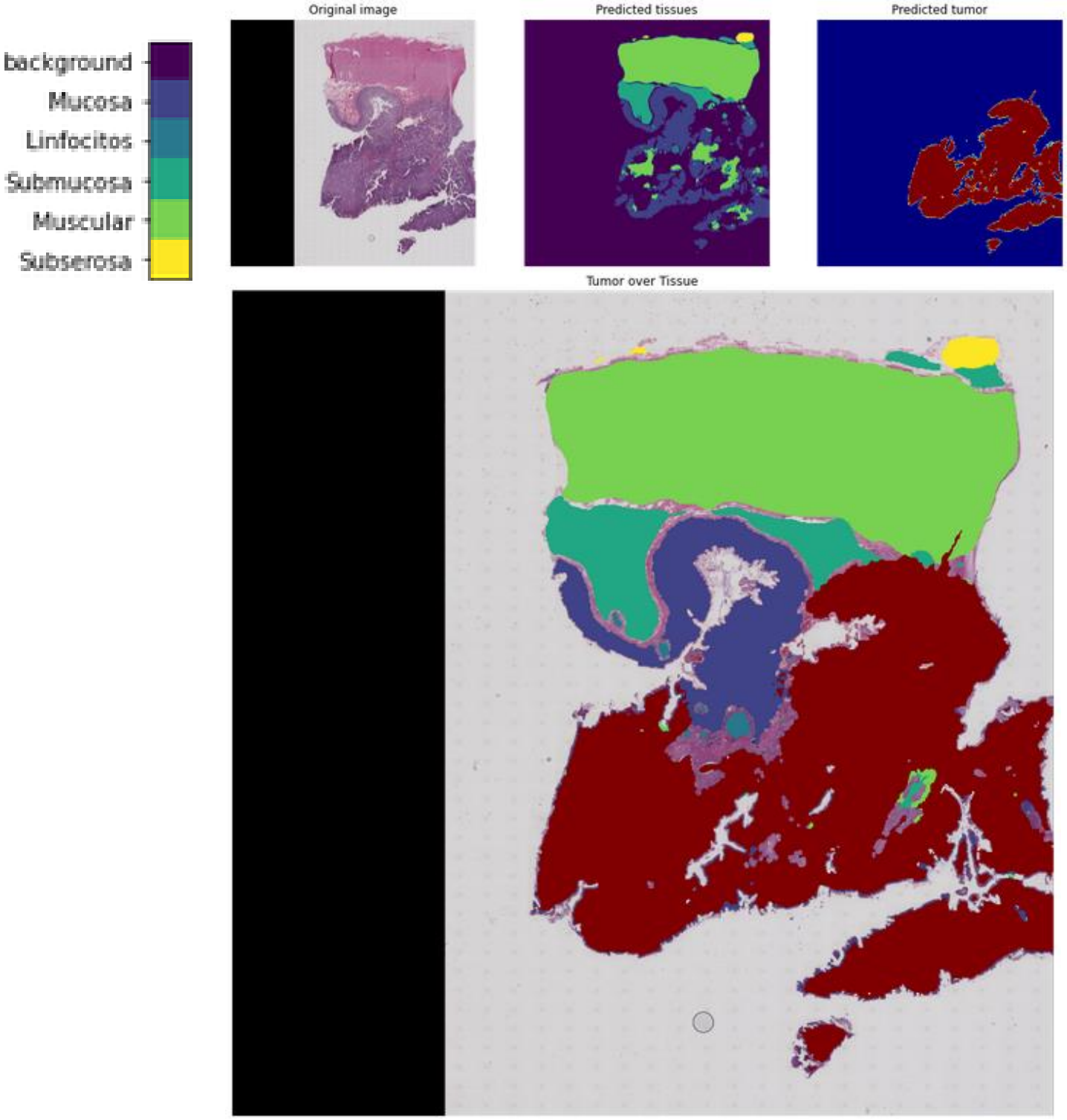



```
Tumor is invading tissues:
   Mucosa
   Linfocitos
   Submucosa
   Muscular
Tumor is stage T2
```

### 12.2 Imagen: 14-4707

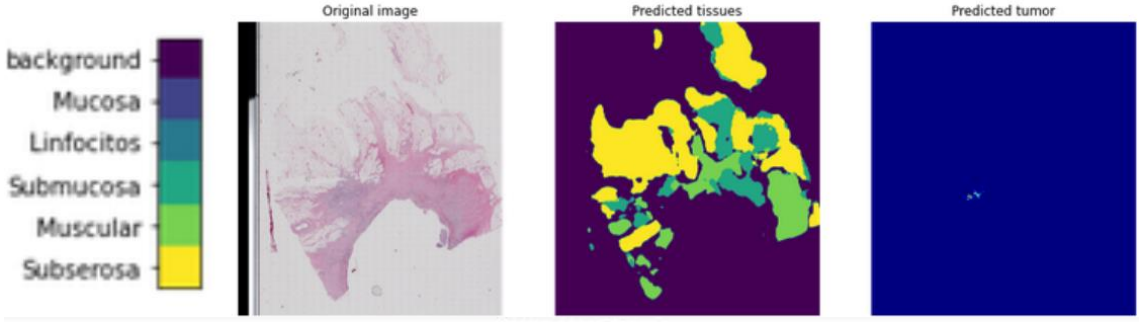


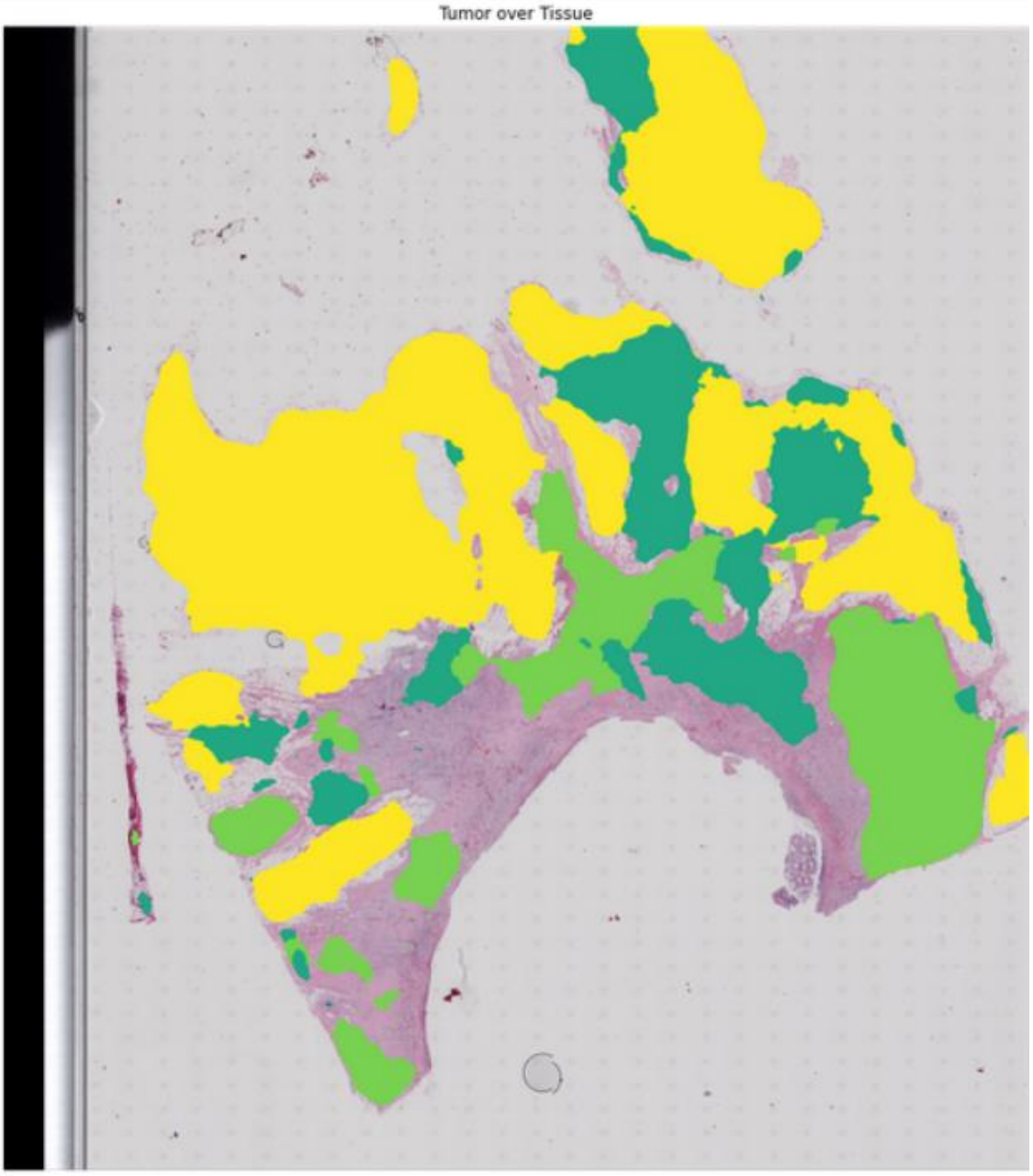
### 12.3 Imagen: 14-5804




```
Tumor is invading tissues:
   Mucosa
   Linfocitos
   Submucosa
   Muscular
Tumor is stage T2
```

### 12.4 Imagen: 14-5731




```
Tumor is invading tissues:
   Submucosa
Tumor is stage T1
```

# 13 Anexo 5: Ilustracion de la Red Neuronal

https://github.com/julenbhy/biomedical_segmentation/blob/master/U-Net%20%2B%20ResNet34%20plot.png


Escola Tècnica Superior
d'Enginyeria
UNIVERSITAT ROVIRA I VIRGILI