



**UNIVERSIDAD
DE GRANADA**

Universidad de Granada

*Escuela Técnica Superior de Ingeniería Informática y
Telecomunicaciones*

Visión por Computador

Proyecto Final

*Adaptación de una red U-Net para la
segmentación de imágenes de bacterias y
células sanguíneas*

ALEJANDRO PINEL MARTÍNEZ

ÁNGEL DE LA VEGA JIMÉNEZ

Enero 2021

Índice general

1	Introducción	3
1.1	Descripción del problema	3
1.2	Conjunto de datos	4
1.2.1	Desbalanceo de clases	4
1.3	Motivación	5
1.4	Objetivos	6
2	Estado del Arte	7
2.1	Estado del arte en segmentación	7
2.2	Estado del arte en Bacteria detection with darkfield microscopy	7
3	Métodos	10
3.1	División en training, validación y test	10
3.2	Proceso de selección del mejor modelo	11
3.3	Procesamiento de datos	12
3.4	Métricas: Accuracy vs Dice	13
3.5	Pre-entrenamiento	15
3.6	Función de loss	16
3.6.1	Categorical crossentropy	16
3.6.2	Weighted categorical crossentropy	17
3.6.3	DICE	18
3.7	Arquitecturas Propuestas	18
3.7.1	U-Net	19
3.7.2	Nuestra propuesta para UNetv2	20
4	Experimentos	23
4.1	Data Augmentation	24
4.2	Pre-trained UNet	24
4.3	Funciones de pérdida	26
4.4	U-Net Clásica	27
4.5	U-Net v2	30
4.6	Comparación U-Net vs U-Netv2	32
4.7	Comparación Estado del arte vs U-Net v2	34

Índice general

5 Cota sobre el error	38
6 Conclusiones	39
Bibliografía	40
Anexo: Instrucciones de ejecución	43

SECCIÓN 1

Introducción

1.1 DESCRIPCIÓN DEL PROBLEMA

Este proyecto se centrará en la utilización de una red U-Net pre-entrenada para el problema de segmentación semántica de bacterias y células sanguíneas en imágenes tomadas por microscopio. La segmentación semántica consiste en, tomando una imagen como entrada, asignar a cada píxel una etiqueta, que representa cada una de las clases para las que el modelo es entrenado.

En este proyecto se analizarán los retos específicos de este problema y se utilizarán técnicas del estado del arte encontradas en la bibliografía para mejorar los resultados obtenidos.

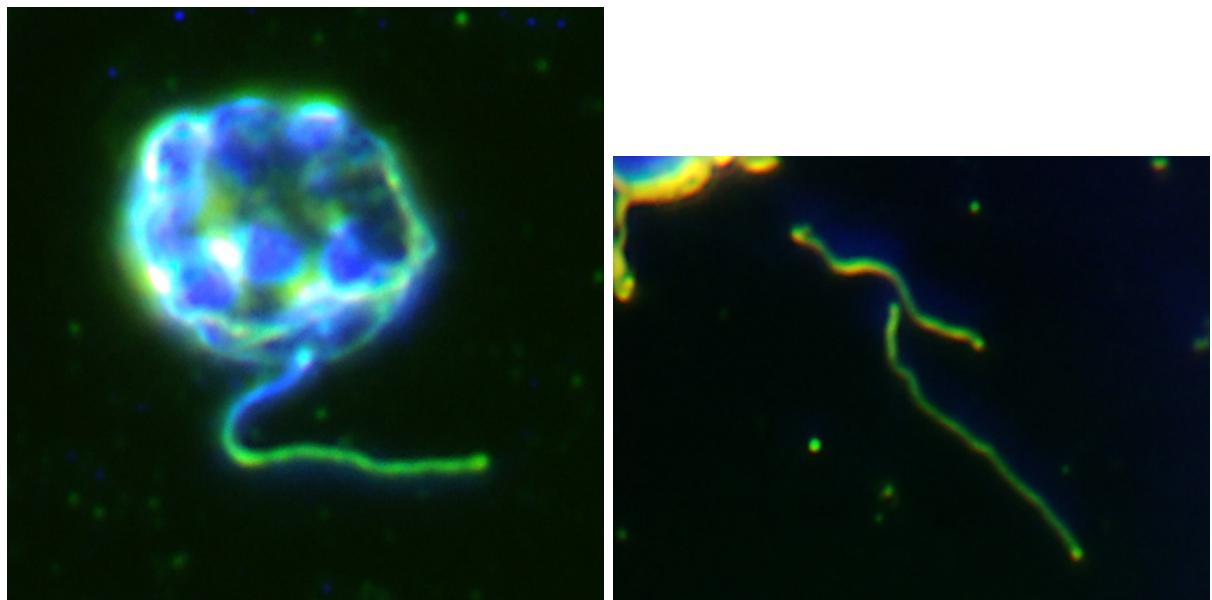


Figura 1.1: Muestras de Spirochaetes al microscopio

1.2 CONJUNTO DE DATOS

El conjunto de datos a tratar consiste en 366 imágenes de bacterias del filo *Spirochaetes*, junto con las máscaras anotadas a mano por estudiantes de la universidad de Heilbronn, Alemania.

Cada píxel pertenece a 3 clases posibles: fondo, que se refiere a la parte de la imagen en la que no hay ni bacterias ni células sanguíneas, célula sanguínea, y bacteria. Etiquetas 0-negro, 1-rojo y 2-verde respectivamente.

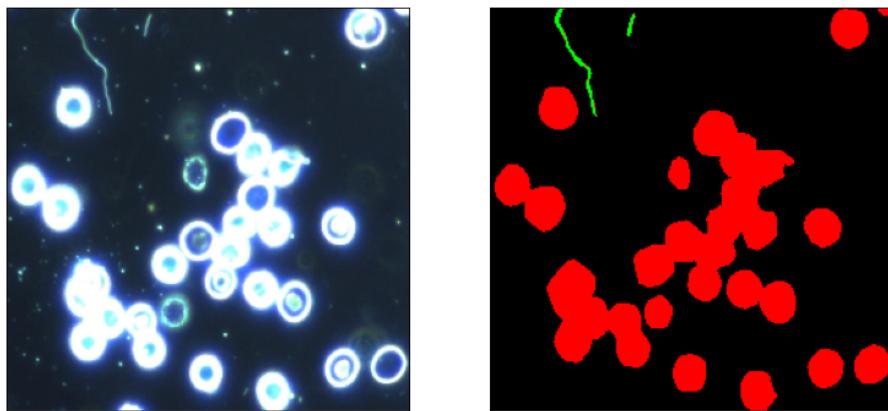


Figura 1.2: A la izquierda, una de las imágenes del conjunto de datos, a la derecha su máscara correspondiente

1.2.1 DESBALANCEO DE CLASES

El mayor reto de este conjunto de datos es que las clases están enormemente desbalanceadas, puesto que la mayor parte de los píxeles son del fondo. Las células sanguíneas tienen un tamaño relativamente grande, mientras que la proporción ocupada por las bacterias es mínima. En concreto, el porcentaje de fondo respecto del tamaño total es aproximadamente del 85 % mientras que las células sanguíneas ocupan entorno a un 14 % y las bacterias apenas un 1 %.

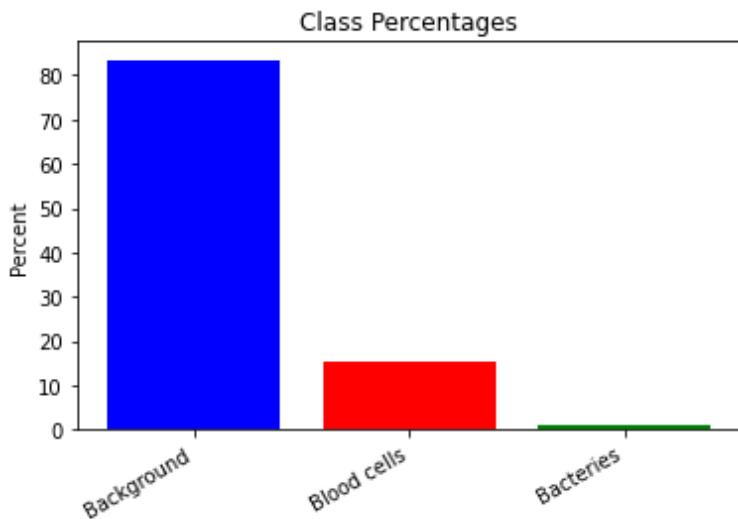


Figura 1.3: Porcentaje de aparición de cada clase

1.3 MOTIVACIÓN

Las bacterias del género Spirochaete son responsables de algunas patologías muy dañinas para los humanos, entre las que se encuentran [1]:

- **Leptospirosis**, causada por la especie Leptospira
- **Enfermedad de Lyme**, transmitida por garrapatas portadoras de la especie *Borrelia burgdorferi*
- **Fiebre reincidente**, por la especie *Borrelia recurrentis*
- **Sífilis y Pian**, causada por la especie *Treponema pallidum*
- **Spirochetosis Intestinal**, producidas por *Brachyspira pilosicoli* y *Brachyspira aalborgi*

Entrenar un modelo de segmentación para la detección de bacterias del género Spirochaete y de las células sanguíneas podría asistir al personal sanitario a la hora de diagnosticar estas patologías, ahorrando tiempo de personal especializado y, por tanto, ahorrando recursos muy valiosos para la sociedad.

Además, debido a que los problemas característicos de esta base de datos son comunes a otros problemas de segmentación (por ejemplo, el desbalanceo de las clases), cualquier progreso obtenido a este respecto, puede ser fácilmente aplicable a otros campos.

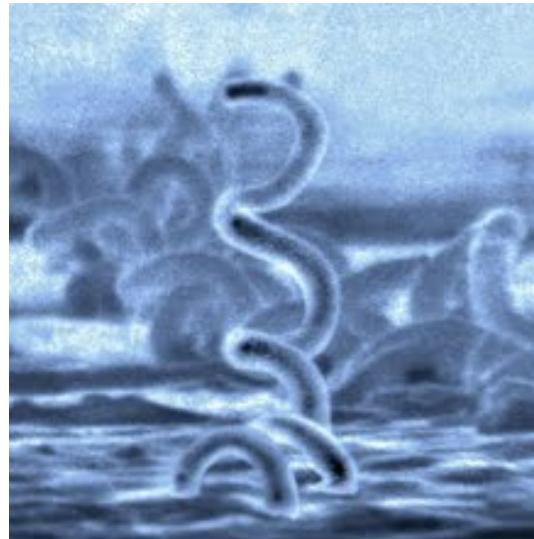


Figura 1.4: Bacteria causante de la sífilis, *Treponema pallidum*, del género Spirochaete

1.4 OBJETIVOS

El enunciado de este proyecto era:*Adaptar una FCN (Segnet, Unet, etc) ya entrenadas a la segmentación de imágenes de un nuevo problema de segmentación. (25 puntos)*

Partiendo de ahí, hemos concretado los siguientes objetivos:

- Adaptar una red U-Net pre-entrenada con otros datos al problema de segmentación de bacterias y células sanguíneas.
- Probar distintas mejoras del estado del arte en nuestro modelo para mejorar los resultados.
- Comparar nuestro modelo con el estado del arte utilizando una métrica adecuada.

SECCIÓN 2

Estado del Arte

2.1 ESTADO DEL ARTE EN SEGMENTACIÓN

Antes de trabajar en nuestro problema, hemos realizado una investigación en la literatura académica sobre el estado del arte, tanto en problemas de segmentación en general [2] [3] [4] como en nuestro problema en particular.

La segmentación semántica está dominada actualmente por las *fully convolutional networks*. Entre ellas, las más importantes son **SegNet**, **HRNet** y la que utilizaremos en este proyecto, **U-Net** [5], que a su vez dispone de algunas variantes como **V-Net** [6] y **U-Net++** [7].

2.2 ESTADO DEL ARTE EN BACTERIA DETECTION WITH DARKFIELD MICROSCOPY

Bacteria detection with darkfield microscopy, nuestra base de datos, ha sido utilizada en varios *notebooks* publicados en la propia página de Kaggle donde está la base de datos. U-Net es la arquitectura más popular para este problema, por lo que exponemos los siguientes notebooks como “*competidores*”:

Nota: Para comparar los resultados utilizaremos el coeficiente **dice** como métrica principal. Las razones de esta elección se explican en detalle en la sección 3.4.

- S. Seal [8] [9] propone dos modelos de UNet: UNet clásica y una versión utilizando una técnica llamada “*Squeeze and Exitation*”. Sin embargo, su modelo solo realiza clasificación binaria (fondo o no fondo), luego no está resolviendo el problema completo. Es posible observar en los ejemplos que proporciona que sus redes no son capaces de detectar ninguna bacteria, sólo células sanguíneas, por lo que no

hemos optado a considerarlo parte de los modelos a batir. Como muestra, se incluye uno de sus resultados.

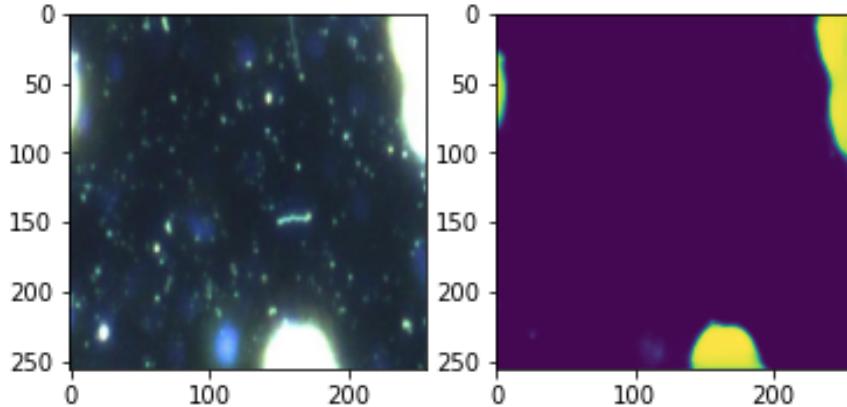


Figura 2.1: La bacteria del centro es ignorada

- L. Nguyen [11], que es el investigador original que subió la base de datos, propone la arquitectura *UNet Plus Plus* [7]. Utiliza una función de loss que combina *categorical cross entropy*, *intersection over union* y *dice*. Proclama un coeficiente dice final del 94 %, pero este porcentaje de dice no es comparable con el nuestro, pues nosotros lo calculamos de una forma más exigente, como más tarde se explicará. Hemos revisado y re-ejecutado su código con nuestra medida dice para poder comparar su modelo. Sus resultados finales son:

Dice: 60.26 % y Accuracy: 91.14 %

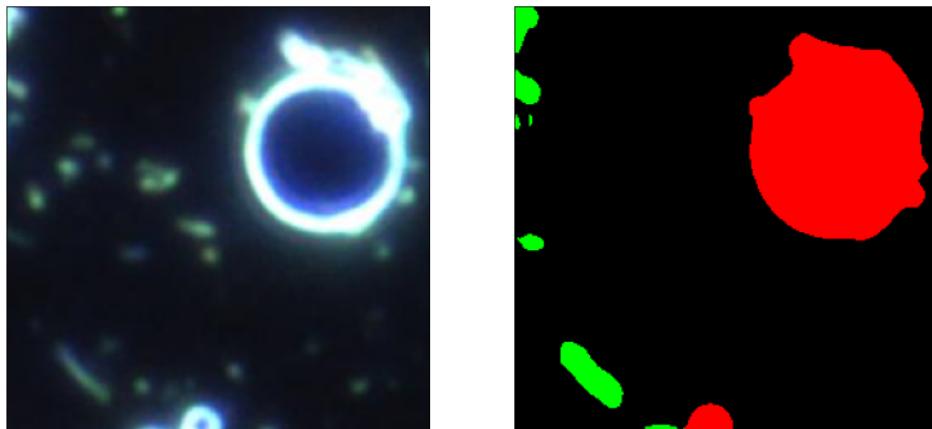


Figura 2.2: En general, sus resultados no son excesivamente buenos. Por ejemplo, esta imagen vemos la predicción de un ejemplo aleatorio en el que ha clasificado demasiados píxeles como bacteria.

Sección 2 Estado del Arte

- A. Le [10] presenta una aproximación más sofisticada. Su red UNet es una versión simplificada de UNet clásica con convoluciones transpuestas en el decoder. Utiliza una función de loss más elaborada para lidiar con el desbalanceo (weighted cross entropy). A pesar de que, por algunos errores en la implementación, su código no llega a utilizar todo su potencial, es el estado del arte actual. Sólo utiliza *accuracy* como métrica (alcanzando el 96 %) y ha sido necesario reproducir sus experimentos para obtener también su valor de dice. Al final, los valores obtenidos por Le son:

Dice: 70.42 % y Accuracy: 97.52 %

Además de su alto coeficiente dice, sus resultados son visualmente agradables y, por ello, él será nuestro principal rival a mejorar.

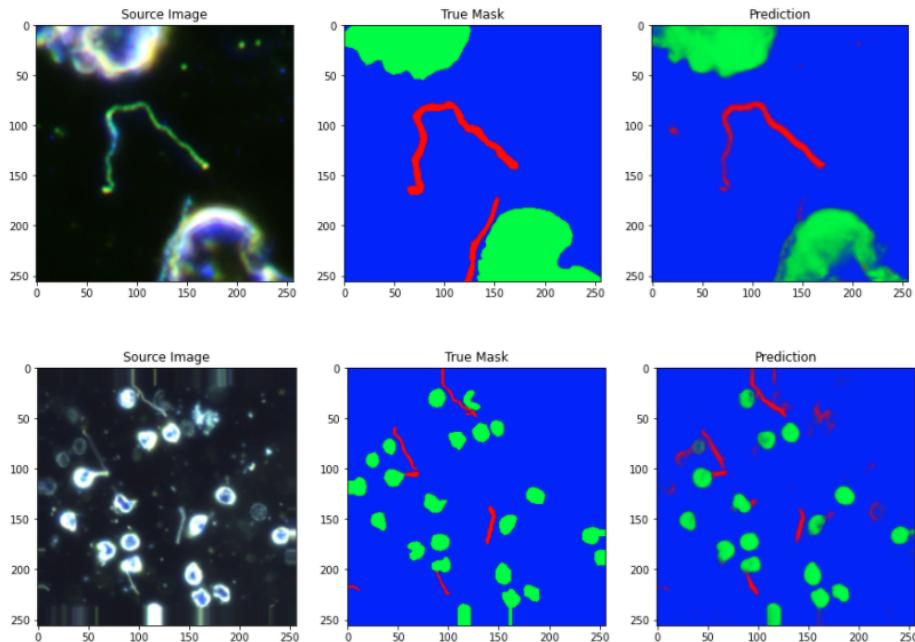


Figura 2.3: Resultados del estado del arte

SECCIÓN 3

Métodos

Nuestro acercamiento al problema será adaptar una UNet clásica y una versión mejorada, ambas pre-entrenadas, a nuestros datos. Utilizaremos distintas técnicas para mejorar los resultados que se explicarán a continuación.

3.1 DIVISIÓN EN TRAINING, VALIDACIÓN Y TEST

Como siempre en este tipo de problemas, el objetivo es dar un modelo final que consiga realizar la segmentación semántica con la máxima calidad posible.

Algo que siempre es importante es que además de obtener este modelo final, debemos dar una estimación del error final que cometerá nuestro modelo ante la llegada de nuevos datos. Con este objetivo sacaremos una pequeña parte de nuestro conjunto de datos para test, y no lo utilizaremos en ningún momento hasta que el mejor modelo haya sido seleccionado, ya que estaríamos sesgando el error de forma optimista.

Concretamente hemos decidido separar un **20 %** de los datos para test, haciendo esta división tras realizar una permutación aleatoria de los datos, con el objetivo de evitar los problemas que podría dar que los datos siguieran un cierto orden (el conjunto de test podría no ser representativo de los datos en ese caso). El motivo de esta proporción es que es un número que se sabe que en general funciona bien, ya que nos deja suficientes datos para entrenamiento, y por otra parte es suficientemente grande para que la estimación del error realmente sea representativa.

De los datos de training restantes, se tomará un **10 %** de ellos como validación, que nos servirá para poder seleccionar el mejor modelo.

3.2 PROCESO DE SELECCIÓN DEL MEJOR MODELO

A continuación vamos a hacer una breve descripción del proceso general que hemos seguido hasta llegar a la selección del mejor modelo final, aunque posteriormente se explicarán los pasos en más detalle.

En primer lugar, partiremos de una implementación propia de la red Unet descrita en el paper original, sin modificaciones.

A partir de esta arquitectura de base, realizaremos una serie de experimentos iniciales para tomar decisiones importantes como la del uso o no uso de aumento de datos, la selección de la función de pérdida que se adecúe más a nuestro problema, o comprobar hasta qué punto obtenemos mejora por el hecho de usar una red pre-entrenada o entrenarla desde cero.

En estos primeros experimentos, en los que no buscamos aún la obtención del modelo final completamente entrenado, utilizaremos un número de épocas relativamente bajo que simplemente nos permita poder saber qué decisiones podrían ser buenas o malas. El modo de trabajo para tomar estas primeras decisiones será el uso de 3-fold cross validation¹, que como sabemos consiste en dividir nuestro conjunto de training en 3 partes disjuntas, y entrenar 3 veces seleccionando 2 partes para entrenamiento, y utilizando la parte restante para estimar el valor de la métrica que se obtiene. Una vez realizados estos 3 entrenamientos, se calcula la media de los resultados. Calculado este error de cross-validation, iremos tomando las decisiones basándonos en él.

Después haber tomado estas decisiones básicas iniciales, entrenaremos la red U-Net durante un número grande de épocas y estudiaremos el proceso de aprendizaje (esencialmente la existencia de overfitting o underfitting) así como la bondad obtenida por este modelo. En este caso hemos preferido no utilizar cross-validation, sino que simplemente sacaremos un conjunto de validación a partir de train, ya que consideramos que el tiempo necesario para entrenar este modelo durante un número grande de épocas es considerablemente largo, por lo que no nos lo podemos permitir.

A partir del estudio de los resultados obtenidos por este modelo inicial de Unet, hemos estudiado una serie de posibles mejoras para adaptar Unet a nuestro problema. Como hemos dicho, la forma de decidir si una mejora se debe aplicar o no, será principalmente mediante el estudio de la curva de aprendizaje y de la bondad obtenida en el conjunto de validación (a pesar del sesgo que se introducirá inevitablemente al tomar decisiones usando sólo validación y no validación cruzada).

¹La ventaja de utilizar X-fold en lugar de utilizar simplemente una separación en training y validación es el hecho de que con X-fold obtenemos una estimación más robusta del error cometido por cada modelo que con hold-out. Cuanto mayor sea el X, mejor será la estimación, pero está claro que hay que buscar un compromiso, ya que el tiempo de cómputo utilizando un mayor número de divisiones sería mayor

Una vez seleccionado el mejor modelo y entrenado con el conjunto de training, calcularemos el error cometido sobre el conjunto de test, que nos valdrá para poder dar una estimación del error que cometerá el modelo ante nuevos datos.

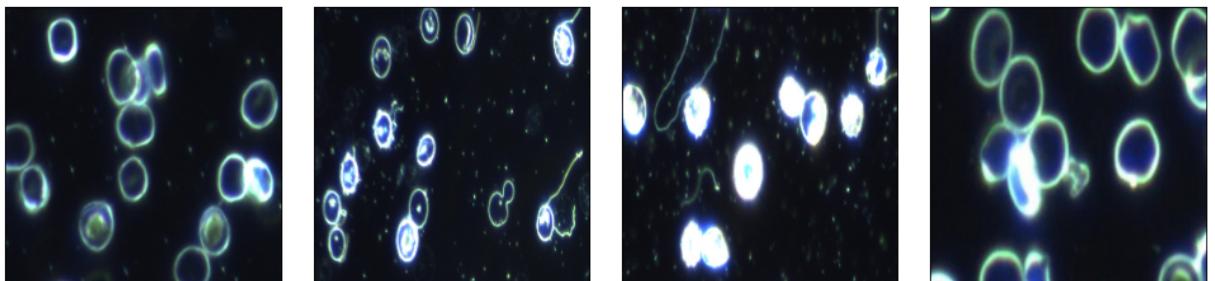
Por último, en el caso de utilizar esta red en producción, el modelo que devolveríamos sería el modelo final elegido, pero entrenado con training + test, ya que cuantos más datos utilicemos para el entrenamiento, mayor capacidad de generalización obtendremos.

3.3 PROCESAMIENTO DE DATOS

Bacteria detection with darkfield microscopy está formada por 366 imágenes con sus respectivas máscaras de segmentación. Las máscaras están codificadas con enteros que representan la clase de cada píxel (0: fondo, 1: células, 2:bacterias). Por tanto, es necesario transformar estos valores enteros en tensores de 3 valores reales utilizando la conocida función de keras, *to_categorical()*.

Por otra parte, la diferencia de tamaños y factores de forma entre las imágenes del conjunto de datos es muy grande, lo que hará que tengamos que utilizar algún método para unificar la forma de todas estas imágenes, y que puede tener cierta importancia debido a la posible introducción de deformaciones en los datos. En concreto, a todas las imágenes se les ha realizado un *resize* a (256, 256, 3). Este tamaño único no mantiene la relación de aspecto de las imágenes, pero las deformaciones resultantes no aparentan ser graves a simple vista.

Figura 3.1: Deformaciones en las imágenes



Como se puede observar, las células y bacterias siguen siendo reconocibles como tales. Además, dado que las células son estructuras deformables, este tipo de deformaciones son comunes en su estado natural.

Para mejorar los resultados de nuestro modelo, una mejora que hemos barajado es el **data augmentation**. Esta técnica consiste en conseguir un conjunto de datos mayor a partir de los datos de los que disponemos. Aumentar el número de ejemplos de entrenamiento mejora la generalización de la red y además ayuda a reducir el posible overfitting.

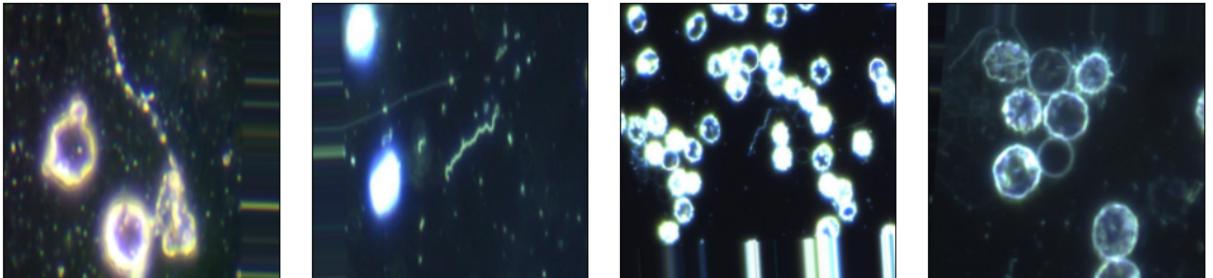
Sección 3 Métodos

Para utilizarla, en cada época de entrenamiento, las imágenes serán alteradas de distintas formas para que los valores de los píxeles sean diferentes. La aumentación de datos es especialmente importante en problemas donde se tienen pocos datos, como en este caso. Los parámetros escogidos han sido los siguientes:

- `width_shift_range=0.1, height_shift_range=0.1` Controlan el desplazamiento de las imágenes horizontal y verticalmente. Las zonas nuevas se llenan con el último píxel del borde por defecto. El valor escogido ha sido 0.1 que significa hasta un 10 % de desplazamiento.
- `horizontal_flip=True, vertical_flip=True` Este parámetro permite que las imágenes se inviertan horizontal y verticalmente. Tiene sentido activarlo especialmente en imágenes al microscopio.
- `rotation_range=5` Indica el número de grados que una imagen puede ser rotada.
- `zoom_range=0.2` Indica la cantidad de zoom que se aplicará como máximo, es decir, como mucho se aplicará un zoom de 1.2x

Para escogerlos, se ha optado por tomar los parámetros más utilizados en los modelos previos que se han utilizado con este problema [8] [10].

Figura 3.3: Ejemplos de aumentación de datos



3.4 MÉTRICAS: ACCURACY VS DICE

El desbalanceo en las clases de nuestro problema presenta un enorme reto. La dificultad a la que nos enfrentamos es encontrar una métrica que podamos utilizar para comparar el rendimiento de varios modelos reflejando fielmente la bondad de cada modelo.

La métrica más utilizada en los problemas de clasificación (clasificación píxel a píxel en este caso) es la **accuracy**, el porcentaje de píxeles clasificados correctamente. Sin embargo, no es una buena elección para nuestro problema debido a los problemas de desbalanceo. Como se ha podido ver en el gráfico 1.3 de la introducción , hasta un 85 % de los píxeles son de la clase fondo. Un modelo que sólo devolviese la clase “background”

Sección 3 Métodos

para todos los píxeles tendría una accuracy muy buena que contrastaría con lo absurdo de su funcionamiento [12]. También, un modelo que aprendiese a segmentar únicamente células sanguíneas, podría alcanzar una accuracy de hasta el 99 %, sin detectar ninguna bacteria, que es al fin y al cabo, la parte más importante del problema.

Como alternativa a la accuracy, utilizaremos **Mean Dice** [13]. **Dice** se trata de una métrica acotada entre 0 y 1 que habla de la cantidad de solapamiento entre el ground truth y la máscara predicha. Dadas A la máscara predicha y B el ground truth para una determinada clase, el coeficiente de Dice se calcula como:

$$Dice(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

Es decir, el coeficiente Dice para una determinada clase se calcularía como: dos por el número de píxeles bien predichos (intersección) dividido por la suma del número píxeles de cada máscara. Esta expresión también se puede escribir en términos de los elementos de la matriz de confusión de la forma

$$Dice = \frac{2TP}{2TP + FP + FN}$$

Está claro que esta métrica sólo dará valor 1 en el caso de que el solapamiento entre la máscara predicha y el ground truth sea perfecto, es decir, cuando todos los píxeles de la clase en cuestión se hayan etiquetado correctamente, y tendrá un valor próximo a 0 en caso de que el solapamiento sea bajo.

Hasta ahora hemos definido esta métrica para una clase, pero tenemos que recordar que en nuestro problema tenemos **3 clases**, por lo que tenemos que adaptarla. Una primera opción (curiosamente bastante usada [11]) sería la de aplicar Dice sin distinguir entre clases, pero está claro que si hacemos esto estaríamos ante el mismo problema que con el accuracy, ya que un buen solapamiento de la clase del fondo con el ground truth ya sería suficiente para obtener un buen valor.

Dicho esto, el enfoque que nosotros hemos tomado es el de definir “mean dice” como la media de calcular el valor de Dice para cada una de las clases de forma independiente, y hacer la media del resultado. De esta forma estamos diciendo que el valor de nuestra métrica únicamente será próximo a 1 cuando las máscaras predichas para tres clases tengan un buen solapamiento con el ground truth.

Entonces, nuestra métrica final para C clases utilizada para la evaluación de los modelos se calculará como:

$$MeanDice = \frac{1}{C} \sum_{i \in C} \frac{2TP_i}{2TP_i + FP_i + FN_i}$$

3.5 PRE-ENTRENAMIENTO

El pre-entrenamiento de una red es una práctica común en el campo del deep learning. Consiste en entrenar un modelo de red con un problema conocido y utilizar los pesos adquiridos como inicialización para, a continuación entrenar con los datos del problema principal que se pretende resolver (*fine-tuning*).

En nuestro caso, no hemos encontrado ninguna red U-Net ya preentrenada, por lo que nos hemos visto obligados a pre-entrenarla nosotros. Para ello, hemos utilizado los datos de **DRIVE** [14]. Esta conocida base de datos contiene imágenes del fondo de la retina y su objetivo es clasificar a nivel de píxel los vasos sanguíneos. Es una base de datos ampliamente estudiada y utilizada generalmente como benchmark en problemas parecidos.

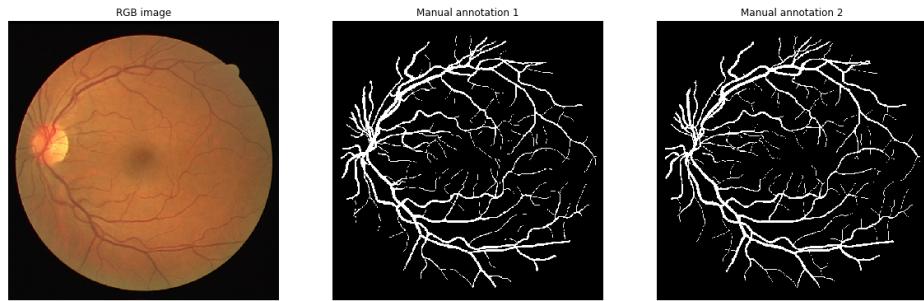


Figura 3.5: Ejemplo de imagen de DRIVE junto a máscaras anotadas (imagen de [14])

El único punto que debemos considerar a la hora de pre-entrenar nuestra red es tener en cuenta que los datos de DRIVE sólo tienen un canal de salida (cada píxel tiene una etiqueta entre $[0.0, 1.0]$) mientras que nuestro problema es segmentar en tres categorías discretas. Esto implica que la última capa de la red debe ser sustituida antes y después del pre-entrenamiento.

- Entrenando con DRIVE, el output de la última capa debe ser un único mapa de características, aplicándose una función de activación *sigmoide*.
- En nuestro problema, la última capa debe proporcionar tres mapas de características que indique los porcentajes de certeza de pertenencia a cada clase. Para ello, la activación será *softmax*.

Los resultados con los datos de DRIVE son excelentes, todos los modelos pre-entrenados allí han dado como resultado más del 99 % de accuracy. Se puede observar en las gráficas de loss y accuracy de nuestros modelos durante el pre-entrenamiento:

Figura 3.6: Pre-entrenamiento U-Net

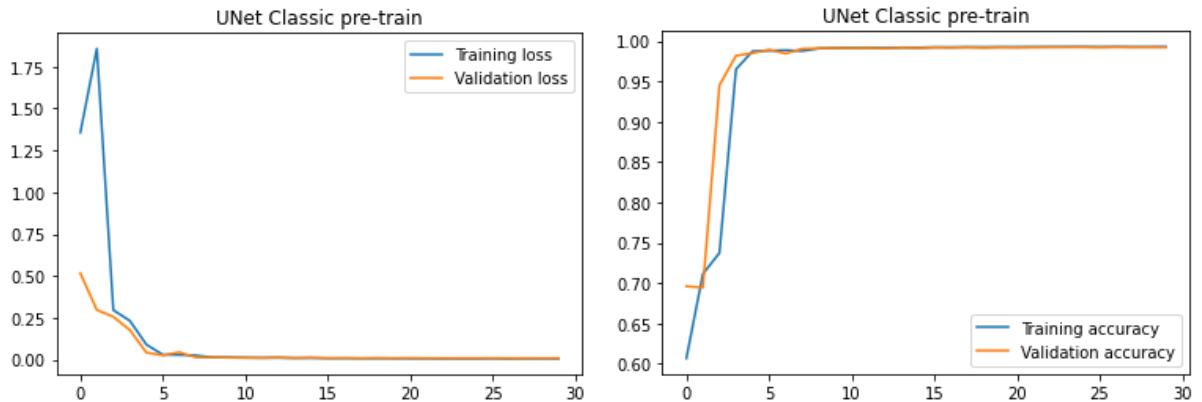
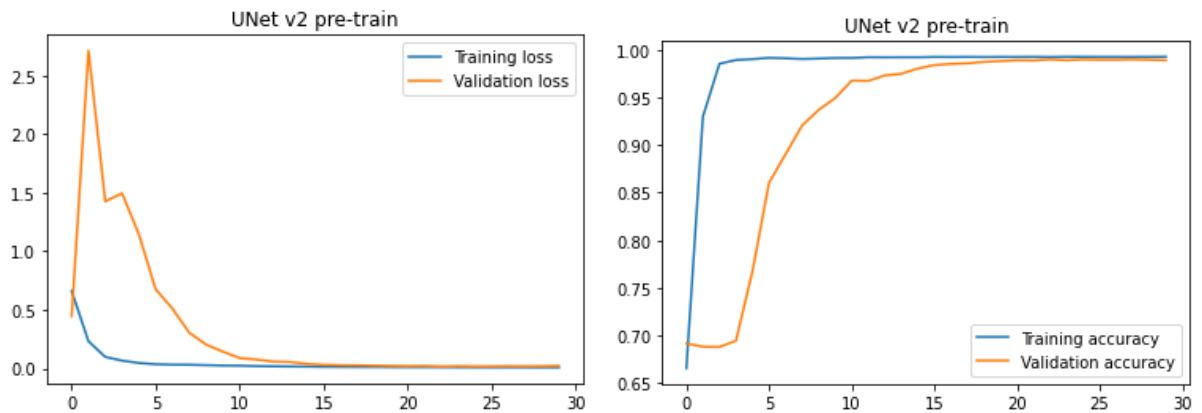


Figura 3.8: Pre-entrenamiento U-Net v2



3.6 FUNCIÓN DE LOSS

La función de pérdida es un elemento clave en el entrenamiento de cualquier red, ya que será la minimización o maximización de esta función la que guiará el proceso de aprendizaje. Dada su gran importancia, vamos a ver las posibles funciones de pérdida que hemos contemplado para nuestro problema [15].

3.6.1 CATEGORICAL CROSSENTROPY

Dado que la salida de nuestra última capa de nuestra red utiliza como función de activación softmax (pudiéndose interpretar la salida como la probabilidad de pertenencia de cada píxel a cada clase), una primera posible función de pérdida que podía ser adecuada para nuestro problema es la **categorical cross entropy**, que se calcularía para cada

Sección 3 Métodos

píxel de la siguiente forma, suponiendo codificación categórica (la clase de cada píxel se codifica con un vector de ceros que tiene un 1 en la posición correspondiente a la clase), siendo t la etiqueta real del píxel siguiendo esta codificación, y out la salida de softmax:

$$\text{categorical_crossentropy} = - \sum_i^c t_i \log(out_i)$$

Dado que estamos utilizando la codificación categórica, el vector t será 0 en todos sus componentes a excepción de la que corresponda a la clase verdadera. Se puede expresar esta función de pérdida como [19]:

$$\text{categorical_crossentropy} = -\log(out_j) \text{ con } j \text{ la etiqueta real del píxel}$$

Luego, la pérdida total cometida en cada imagen se calculará como la media de los errores cometidos en cada píxel, aunque este promedio es algo que Keras hace internamente en la función fit, por lo que nosotros sólo tendremos que calcular el error para cada píxel individual [20].

3.6.2 WEIGHTED CATEGORICAL CROSSENTROPY

A pesar de que la anterior es una de las funciones de pérdida más comúnmente utilizadas para clasificación multiclase, hemos pensado que existe la posibilidad de que aplicar esta función de forma directa no se adecúe bien a nuestro problema, debido al alto desbalanceo de clases.

Si nos fijamos, la función anterior penalizará por igual el hecho de clasificar mal un píxel de etiqueta bacteria, célula o fondo, es decir, su valor será 0 en el caso de que la clase real se prediga con probabilidad total, e “infinito” si se predice con cero probabilidad. Pero parece lógico pensar que equivocarse en una bacteria (de las que hay muy pocas) debería “costar” más caro que clasificar mal un píxel del fondo.

Por tanto, vamos a tratar de dar una función de pérdida que obligue a la red a aprender a clasificar correctamente la clase minoritaria. Para conseguir este comportamiento hemos creado una nueva función **weighted categorical crossentropy**, que tiene en cuenta los pesos de cada clase, y que se calcularán de forma inversamente proporcional al número de píxeles pertenecientes a cada clase en el conjunto de entrenamiento. Concretamente, la forma de calcular los pesos de cada clase es

$$w_i = \frac{\text{número de píxeles de la clase i}}{\text{número de píxeles de la clase mayoritaria}}$$

Utilizando esta fórmula, los pesos calculados para nuestro problema son [**Fondo: 1.00**
Células: 5.47, Bacterias: 82.69]

Sección 3 Métodos

Y la función de esta entropía cruzada con pesos, se calculará de la forma para cada píxel:

$$\text{categorical_crossentropy} = -\log(\text{out}_j)w_j \text{ con } j \text{ la etiqueta real del píxel}$$

Por supuesto, el método propuesto de calcular los pesos y la función de pérdida puede ser alterado y refinado. Formas más sofisticadas pueden ser mejores que las propuestas pero investigar esto recae fuera del alcance de este trabajo.

3.6.3 DICE

Como hemos dicho, nuestra intención es que la función de pérdida obligue a dar igual importancia a cada una de las clases, y esto nos ha llevado a estudiar la posibilidad de utilizar como función de pérdida “Mean soft Dice Loss”.

La idea intuitiva de esta función de pérdida es muy similar a la de la función Mean Dice que explicamos anteriormente, con una diferencia sutil.

Cuando utilizábamos Mean Dice como métrica, los valores de las etiquetas predichas eran discretos, es decir, o se acierta la clasificación, o se falla, pero no existen valores intermedios.

La diferencia esencial cuando usemos mean dice como función de loss, es que esta se aplicará sobre la salida de softmax en lugar de aplicarse sobre la predicción final. Y como ahora la salida no es discreta, es posible que la clasificación de un píxel sea “casi correcta” (de aquí el nombre soft), por lo que se redefine la intersección como la multiplicación del valor de la etiqueta real por el valor de softmax en esa etiqueta.

La segunda diferencia para que esta métrica pueda funcionar como función de pérdida, es que tendremos que devolver el valor en negativo, ya que por defecto Keras tratará de minimizar la función de pérdida.

Como podemos ver en sitios como [18], es posible que esta función de pérdida no vaya a darnos buenos resultados a pesar de que la intuición parezca adecuada. De forma resumida, este hecho se debe a que el gradiente de esta función no tiene una forma fácil de minimizar mediante backpropagation.

3.7 ARQUITECTURAS PROPUESTAS

Para este problema utilizaremos dos modelos de redes: la U-Net clásica [5] y una versión mejorada de esta, diseñada por nosotros mismos.

3.7.1 U-NET

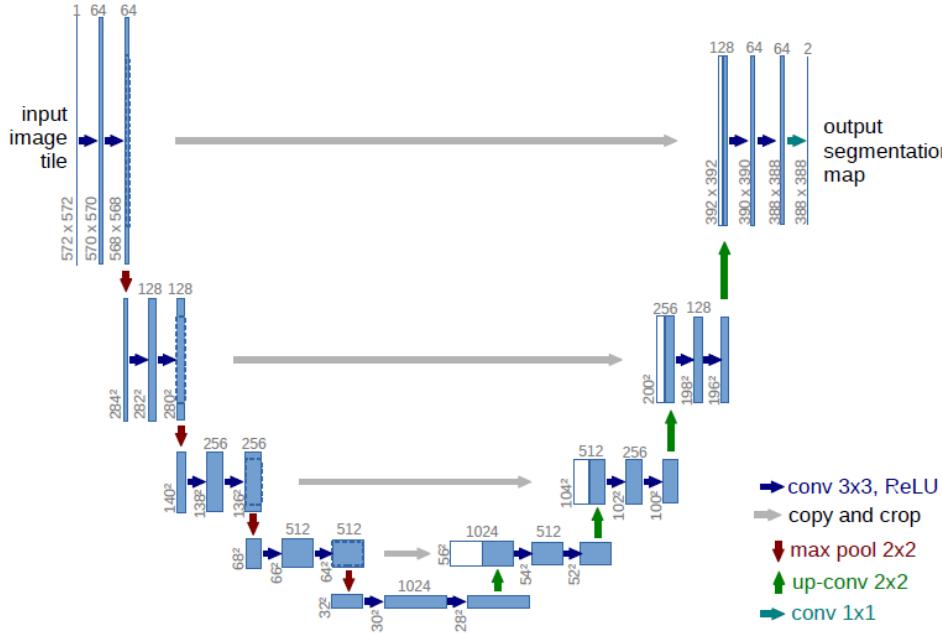


Figura 3.10: Esquema del funcionamiento de una U-Net, imagen extraída de [5]

U-Net es el modelo base con el que vamos a trabajar. Es un modelo fully convolutional formado por dos partes: el **encoder** y el **decoder**.

El encoder está formado por capas de convoluciones 3x3 y capas de max pooling, su objetivo es procesar la imagen y extraer (resumir) su información en mapas de características.

El decoder se conecta a la salida del encoder y utiliza capas de upsampling y convolution 3x3. En la última capa, realiza una convolución 1x1 devolviendo 3 mapas (el número de clases) utilizando la activación *softmax*. Su objetivo es devolver una imagen del **tamaño original** en que cada mapa de la salida representa la probabilidad de que cada píxel pertenezca a cada clase.

La principal innovación de U-Net, y la razón de su nombre, es que su arquitectura es prácticamente simétrica y esto permite concatenar los mapas de características intermedios extraídos del encoder con los mapas devueltos por las capas de upsampling del decoder.

Es una red muy utilizada y estudiada para problemas de segmentación, especialmente problemas médicos.

La implementación se ha llevado a cabo con la librería Keras de python utilizando el modelo funcional para las skip connections.

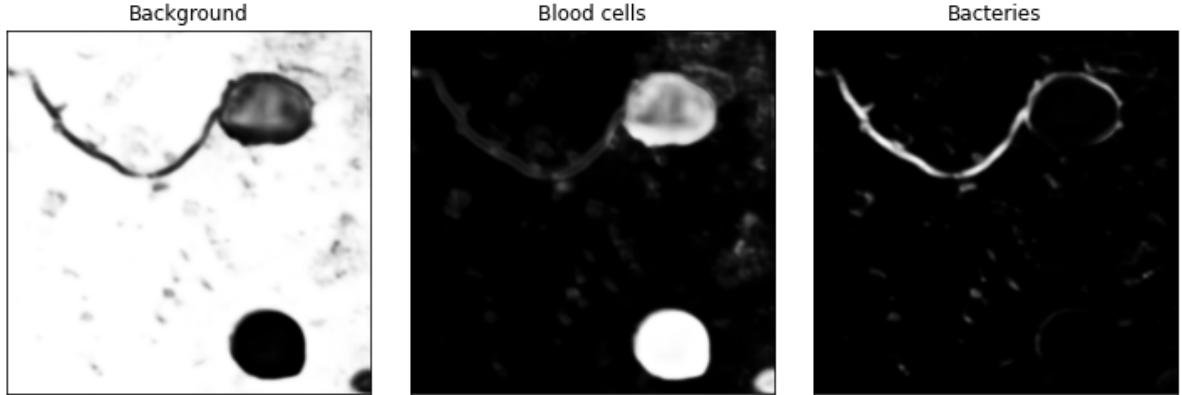


Figura 3.11: Cada uno de los tres mapas del output de una U-Net. Representan las probabilidades de pertenencia a cada clase.

3.7.2 NUESTRA PROPUESTA PARA UNETV2

Para mejorar la red, nos hemos inspirado en los modelos del estado del arte de nuestro problema [10] [11], pero manteniendo las dimensiones originales de UNet.

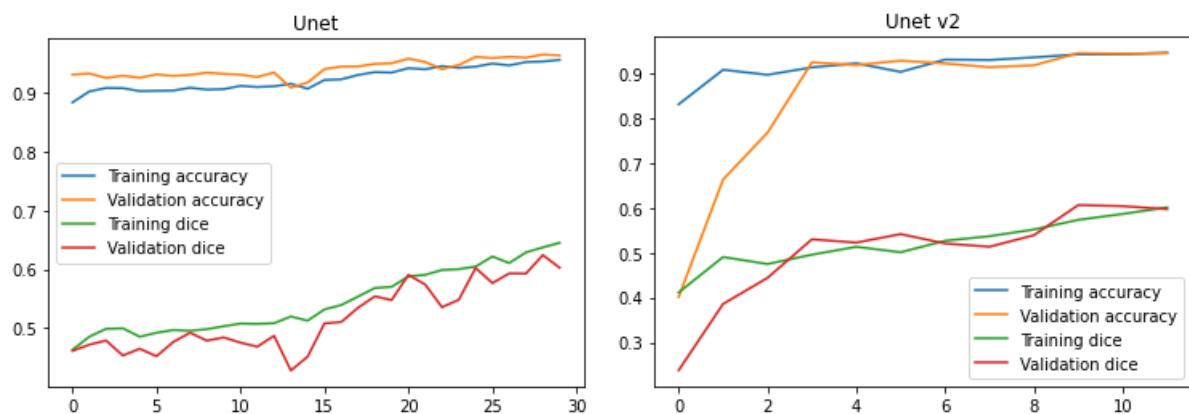
U-Net utiliza una capa intermedia de convolución entre el upsampling y las capas de concatenación que es con frecuencia obviada en otras implementaciones. Nosotros también hemos optado por ello debido a que su inclusión parecía indiferente para el rendimiento al realizar experimentos. De forma parecida, muchas versiones alternativas de UNet optan por reducir el número de filtros para así reducir el número de parámetros entrenables y agilizar el entrenamiento, a costa de perder potencia. De forma similar, nosotros hemos optado sacrificar anchura (número de filtros) por más profundidad. Concretamente, trabajamos con 5 niveles de encoder/decoder con tres convoluciones en cada uno, además de añadir una segunda convolución en el centro de la red.

Otra mejora que hemos propuesto es el uso de *Transpose Convolution* [16] o convoluciones transpuestas. En muchas variantes de U-Net [17] se utilizan estas capas en el decoder, tanto para sustituir a las convoluciones estándar, como para realizar el upsampling (utilizando strides de (2,2)). Debido a que parece mejorar experimentalmente los resultados, hemos optado por sustituir las convoluciones del decoder por convoluciones transpuestas, pero hemos mantenido la capa de upsampling.

En cuanto a la **regularización**, la U-Net original no utiliza nada. Es importante hacer notar que la red en ninguna de las variantes probadas parece sufrir un problema grave de overfitting, más bien lo contrario: el valor de *loss* y *dice* de entrenamiento se corresponde con el de validación (ver figura 3.12)

Sección 3 Métodos

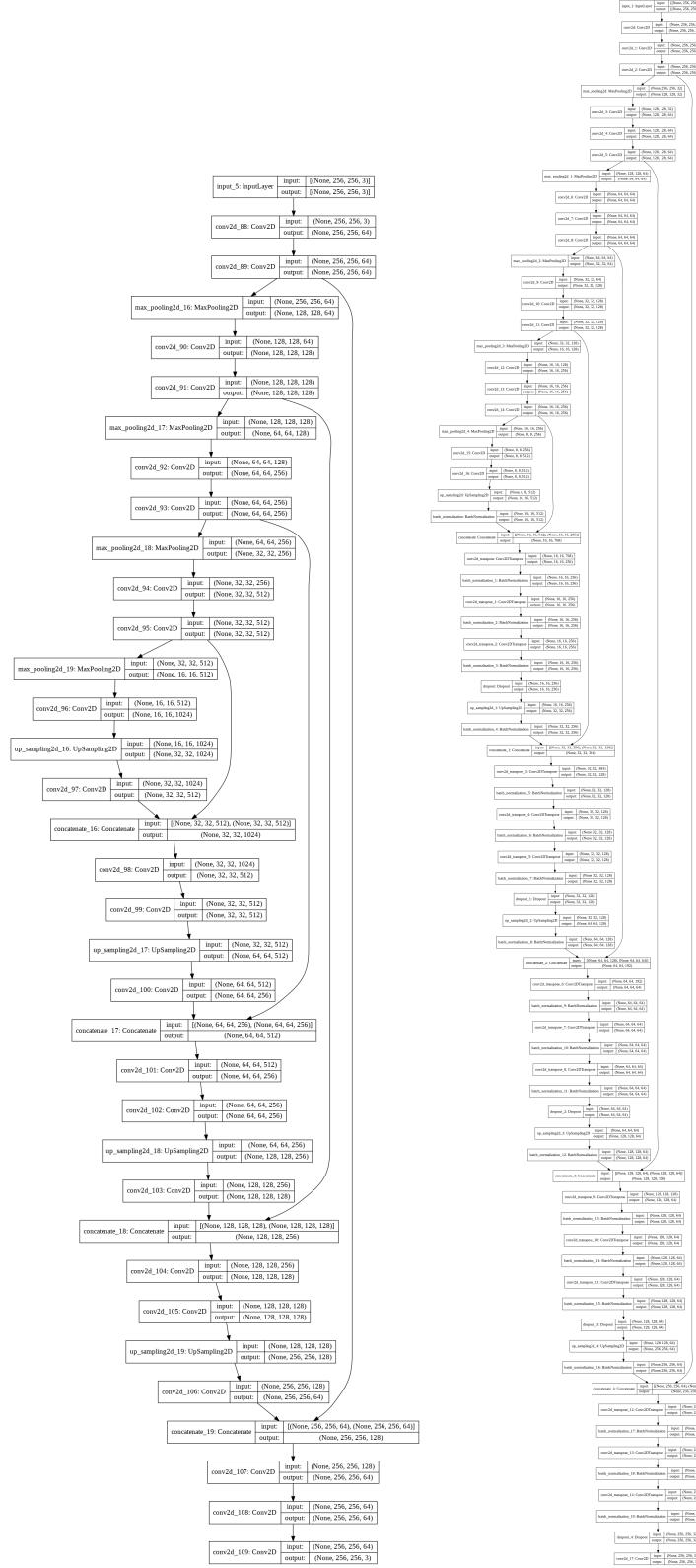
Figura 3.12: No hay apenas overftting



Aún así, siguiendo los ejemplos de algunas variantes de U-Net, hemos decidido probar a introducir *BatchNormalization* y *Dropout*. Distintas pruebas han demostrado que al introducir estas capas en el encoder, la red dejaba de converger y su resultado era paupérrimo. Sin embargo, utilizarlas en el decoder tiene un efecto positivo. Concretamente, hemos introducido una capa de *BatchNormalization* después de cada convolución traspuesta y una capa de *Dropout* con 0.2 antes de las capas de upsampling.

Sección 3 Métodos

Figura 3.14: Grafos de nuestras redes



(a) U-Net clásica

(b) U-Net v2

SECCIÓN 4

Experimentos

A continuación, vamos a ver una serie de experimentos realizados con el objetivo de tomar decisiones sobre nuestro modelo. Más concretamente nos ayudarán a decidir sobre el uso o no de aumento de datos, sobre si el uso de una Unet ya pre-entrenada nos aporta verdaderamente una mejora, así como qué función de pérdida es la mejor para nuestro problema.

Para decidir finalmente si vamos a adoptar alguna de estas decisiones, como ya dijimos vamos a utilizar 3-fold cross validation y decidiremos según el valor de la métrica que obtengamos.

Hay que destacar que la cantidad de experimentos que podríamos realizar es potencialmente infinita: selección del mejor optimizador (así como sus hiperparámetros), distintos tipos de aumento de datos, inicialización de los pesos, etc.... Sin embargo, muchos de estos parámetros los consideraremos fijos, ya que el tiempo de cómputo que sería necesario para realizar una búsqueda exhaustiva de estos hiperparámetros sería demasiado alto. Dos de estos parámetros que consideraremos fijos los siguientes:

- **Optimizador:** utilizaremos siempre Adam, ya que nos aporta la gran ventaja, entre otras, de que determinados parámetros del gradiente descendente estocástico como el learning rate será adaptativo. Se trata de un optimizador muy popular que ha sido muy probado y nos va asegurar buenos resultados en casi cualquier caso [21].
- **Parámetros de data augmentation:** aquí volvemos a tener muchos posibles parámetros a ajustar, zoom rate, flip, shift, etc..., pero como hemos dicho, tratar de hacer un gridSearch en búsqueda de los mejores sería demasiado costoso. Como ya se ha explicado en la sección 3.3, simplemente hemos tomado los valores más usados por el estado del arte.
- **Duración del entrenamiento:** como aún no estamos entrenando el modelo final, utilizaremos entrenamientos relativamente cortos por ahora (5 épocas con 100 steps si hay aumento de datos), ya que la intención es tener una idea rápida de aquello

Sección 4 Experimentos

que queremos introducir o quitar del modelo, y no queremos que el tiempo de entrenamiento sea demasiado largo.

4.1 DATA AUGMENTATION

Para comprobar que nuestra aumentación de datos es beneficiosa, hemos realizado un experimento para contrastar, en igualdad de condiciones, nuestros datos aumentados. Los resultados no dejan lugar a dudas, incluso con una pequeña cantidad de épocas, la aumentación de datos mejora significativamente los resultados de U-Net.

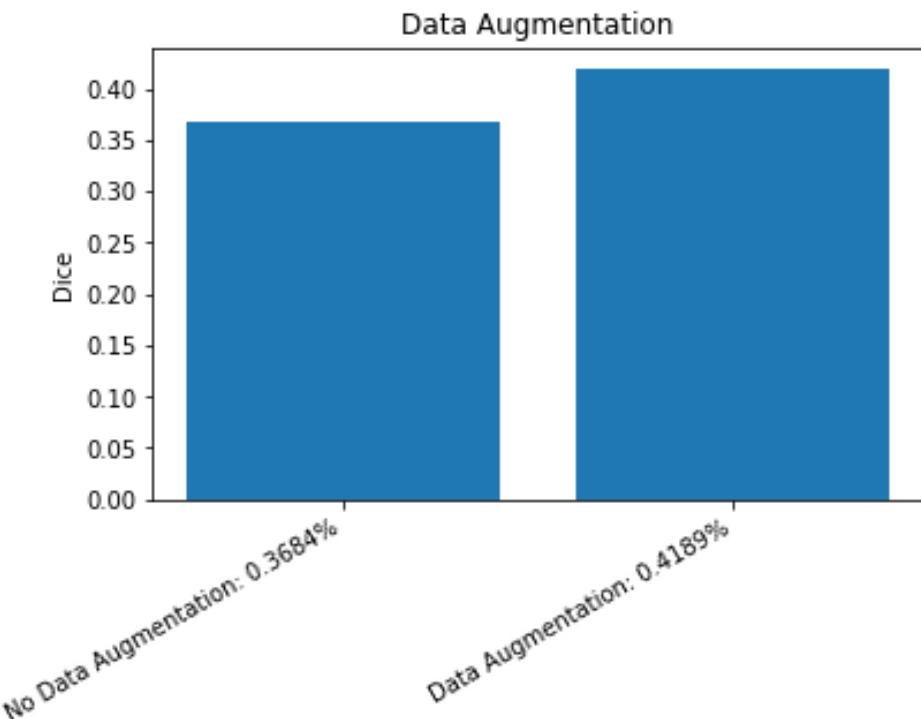


Figura 4.1: Data Augmentation

4.2 PRE-TRAINED UNET

Para hacernos una idea de cómo afecta el pre-entrenamiento al entrenamiento de nuestras redes, hemos diseñado un pequeño experimento, en el que compararemos el resultado de una red inicializada aleatoriamente con otra cuyos pesos hayan sido pre-entrenados como se indica en la sección 3.5.

Sección 4 Experimentos

Figura 4.2: Experimentos pre-entrenamiento

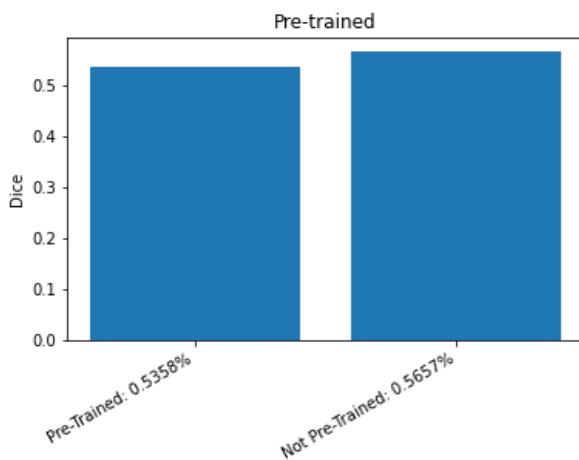
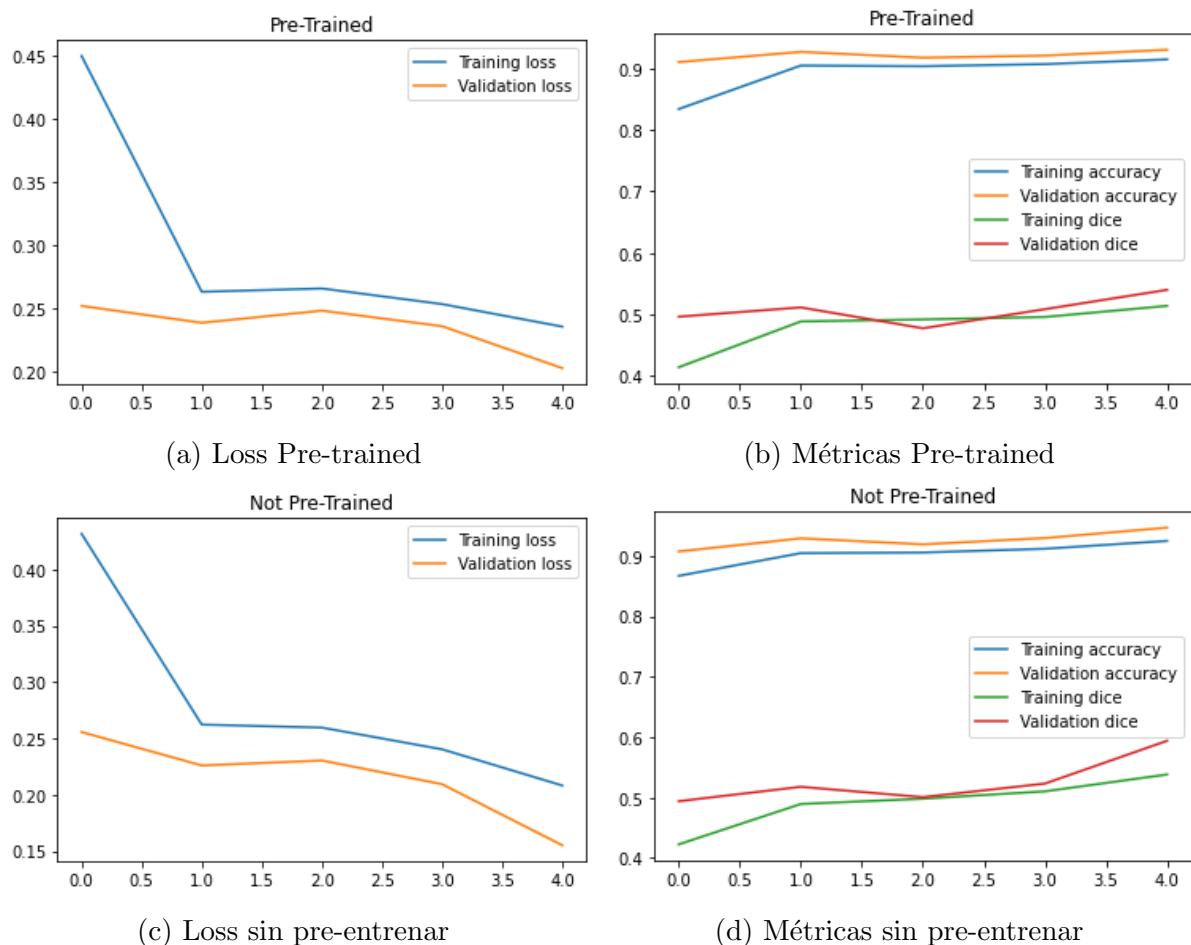


Figura 4.4: Comparativa de Dices

La conclusión que extraemos es que no parece haber diferencias significativas entre las dos opciones. La hipótesis que proponemos para explicarlo es el hecho de que U-Net, al menos en su versión clásica, es una red relativamente pequeña y no demasiado profunda, por lo que no se ve beneficiada en exceso del proceso de pre-entrenamiento.

Aún así, utilizaremos el pre-entrenamiento en todas las pruebas debido a que está definido como una condición en el enunciado.

4.3 FUNCIONES DE PÉRDIDA

El siguiente experimento que hemos realizado es el de tratar de encontrar la función de pérdida que mejor se adapte a nuestros datos. Como ya se explicó en la sección 3.6, vamos a comparar el rendimiento de **categorical cross entropy**, **weighted categorical cross entropy** y **dice loss**.

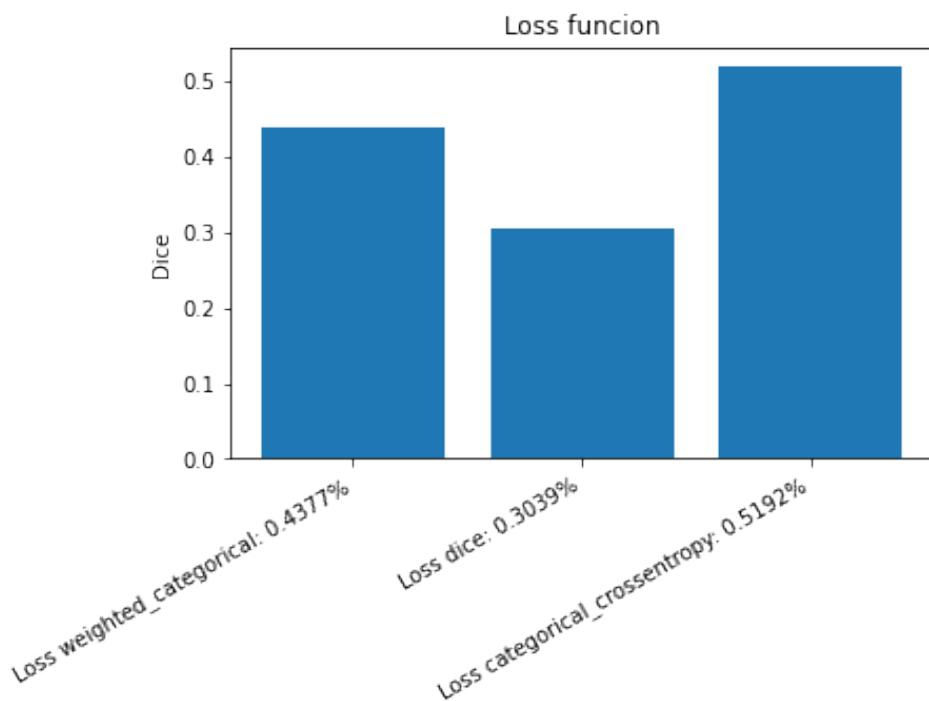


Figura 4.5: Comparativa de funciones de pérdida

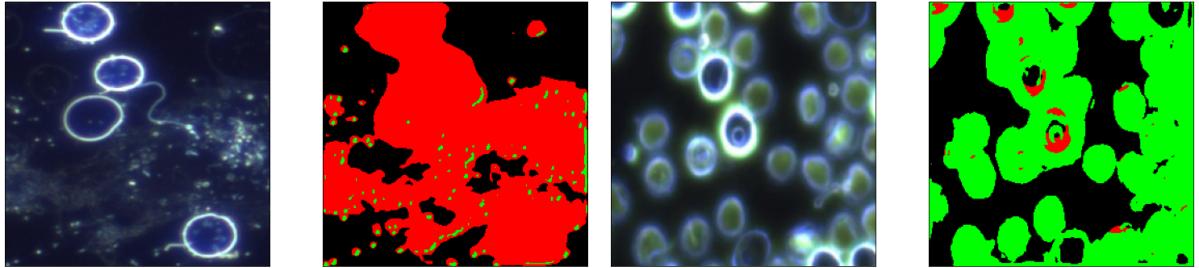
A pesar de que intuitivamente pensábamos que la función de pérdida que nos aportaría un mejor comportamiento con diferencia sería la que asigna pesos a las clases en función de la proporción en la que aparece, la que mejores resultados parece dar es la función de categorical cross-entropy “normal”.

Aunque no sabemos con certeza a qué se debe este hecho, una posible explicación es que el cálculo de los pesos realizado no sea el más adecuado, ya que asignamos los

Sección 4 Experimentos

pesos basándose en la proporción de clases en todo el conjunto de entrenamiento, pero luego existen grandes diferencias entre las imágenes del conjunto de datos, por lo que las proporciones calculadas en general no se cumplirán para cada imagen individualmente. Hemos realizado varias pruebas con otras formas de calcular los pesos, pero no hay una manera trivial de hacerlo y superar a categorical crossentropy.

Figura 4.6: Intentos de calcular pesos



(a) No dar suficiente peso al fondo causa esto (b) Problema de dar demasiado peso a las bacterias.

En cualquier caso, como los resultados que nos aporta la función de categorical crossentropy son razonablemente buenos y es ampliamente utilizada, no hemos considerado necesario seguir experimentando con los pesos de esta función.

Por otra parte, vemos que la función soft dice tiene un comportamiento muy inferior a las demás. Revisando trabajos previos que hayan utilizado esta función de pérdida, hemos podido obtener conclusiones sobre su mal funcionamiento, como se comentó en la sección 3.6.3.

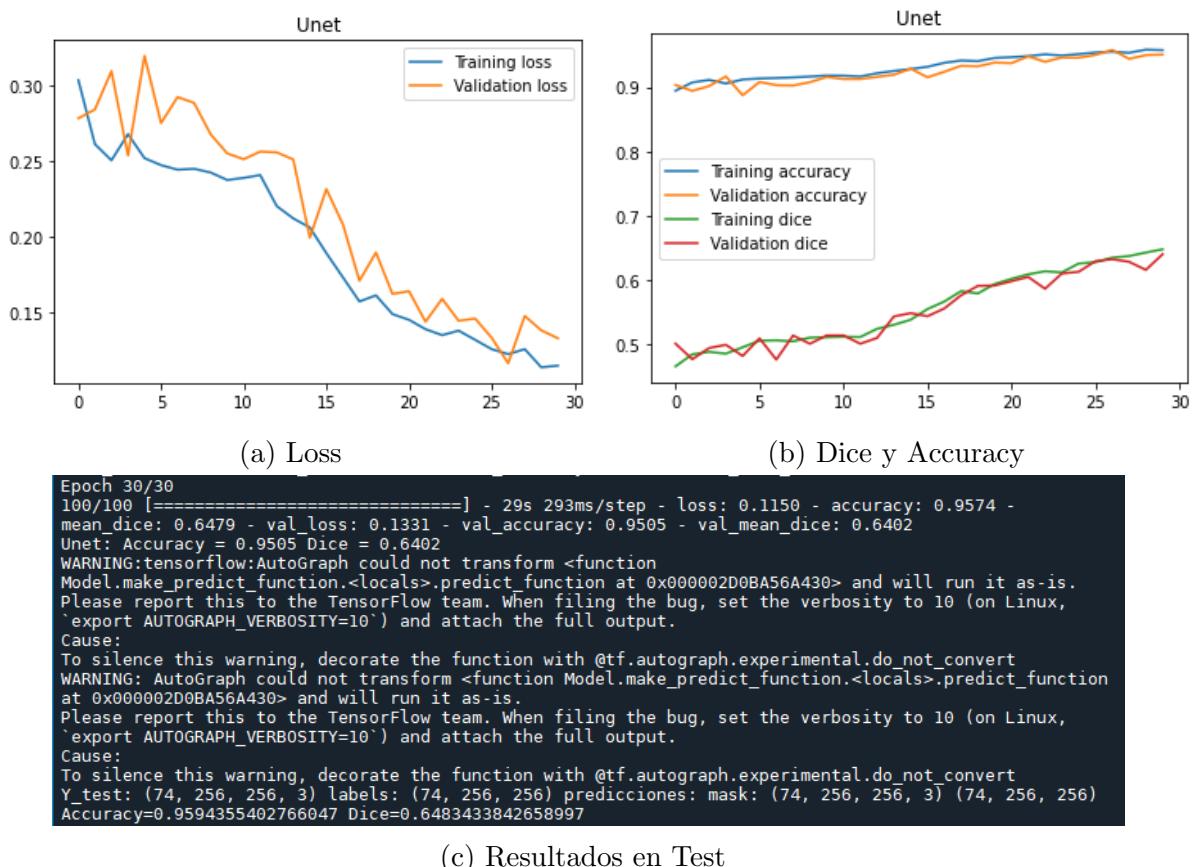
4.4 U-NET CLÁSICA

Ya decidido usar aumentación de datos y pre-entrenamiento, podemos probar la primera de nuestras arquitecturas a todo su potencial: la U-Net clásica.

Este experimento ya se ha llevado a cabo utilizando todas las técnicas indicadas, con *Hold-Out* y 30 épocas. Los resultados son los siguientes:

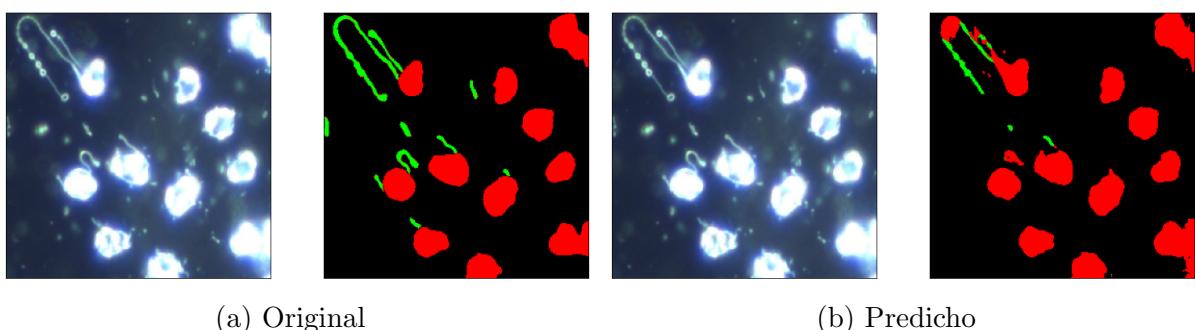
Sección 4 Experimentos

Figura 4.8: U-Net clásica

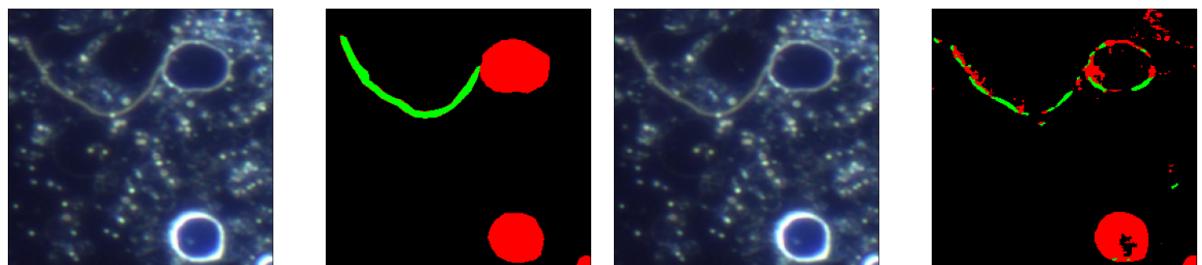


Dice: 64.83 % y Accuracy: 95.94 %

Como se ve, se alcanza valores altos de dice, pero la verdadera prueba está en hacer un reconocimiento visual de algunas segmentaciones.

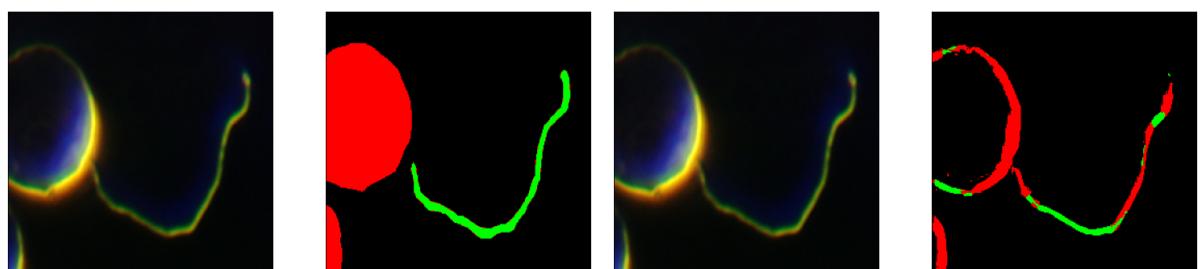


Sección 4 Experimentos



(a) Original

(b) Predicho



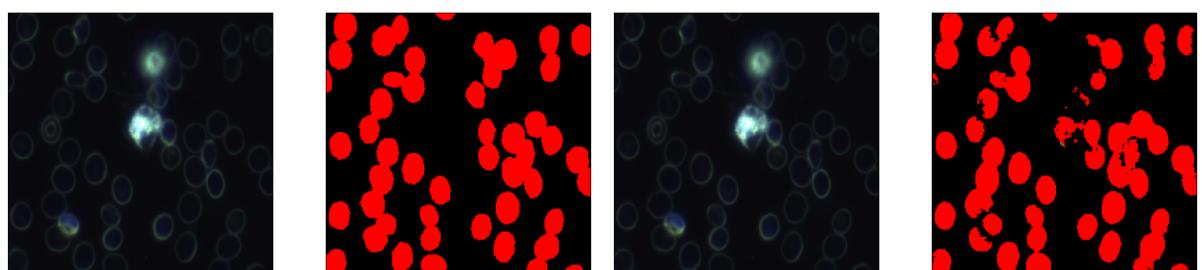
(a) Original

(b) Predicho



(a) Original

(b) Predicho



(a) Original

(b) Predicho

Como se puede observar, los resultados no son malos, pero hay un gran margen de mejora. Las bacterias se confunden con las células y el resultado suele ser una mezcla

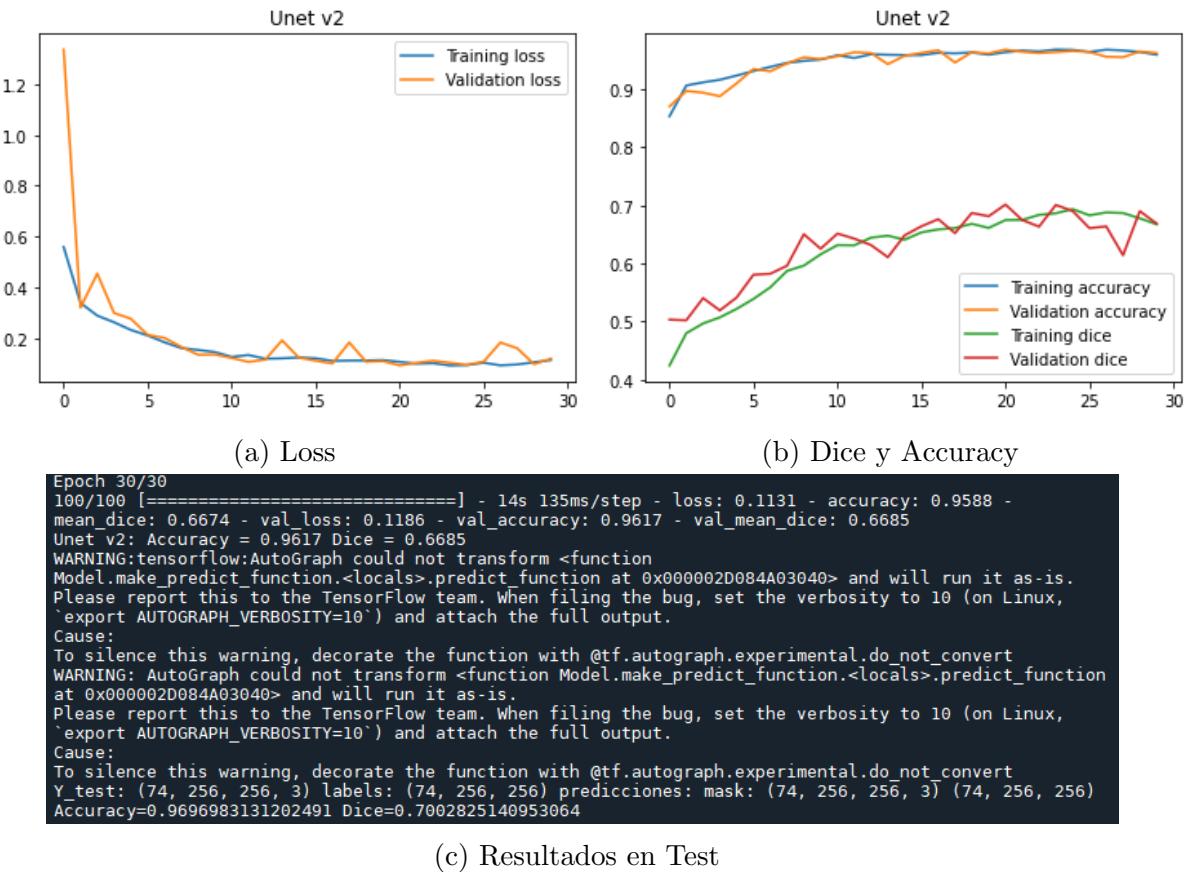
Sección 4 Experimentos

entre ambas. Es un problema común en este dataset que las células con más transparencia sean confundidas con bacterias.

4.5 U-NET v2

Y por último, llega la hora de probar nuestro modelo final. En igualdad de condiciones con U-Net clásico, se entrenará con aumentación de datos, pre-entrenamiento en *Hold-Out* y 30 épocas.

Figura 4.15: U-Netv2

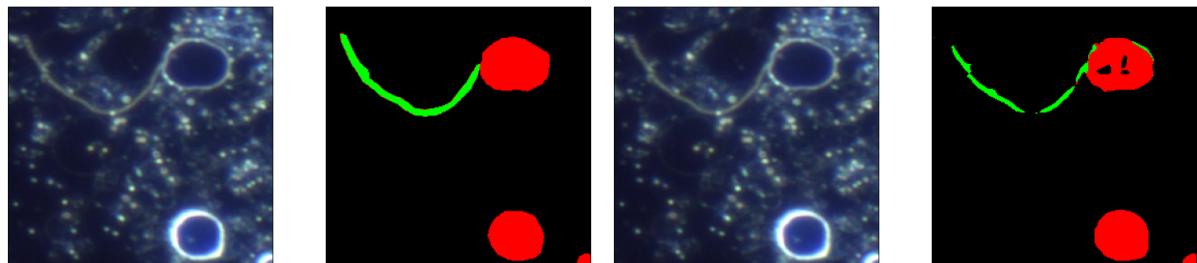


(c) Resultados en Test

Dice: 70.02 % y Accuracy: 96.96 %

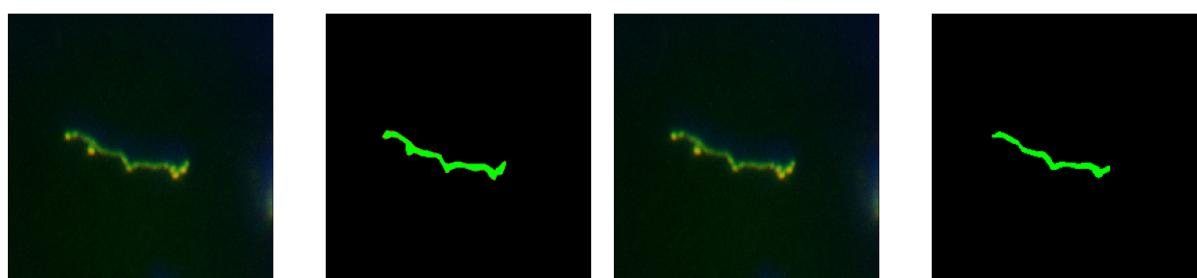
Los resultados son muy positivos, al mismo nivel que el estado del arte. De nuevo, mostramos a continuación algunos resultados visuales.

Sección 4 Experimentos



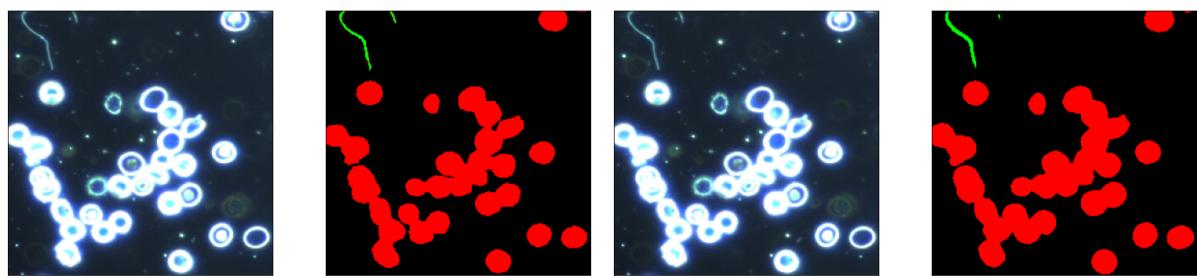
(a) Original

(b) Predicho



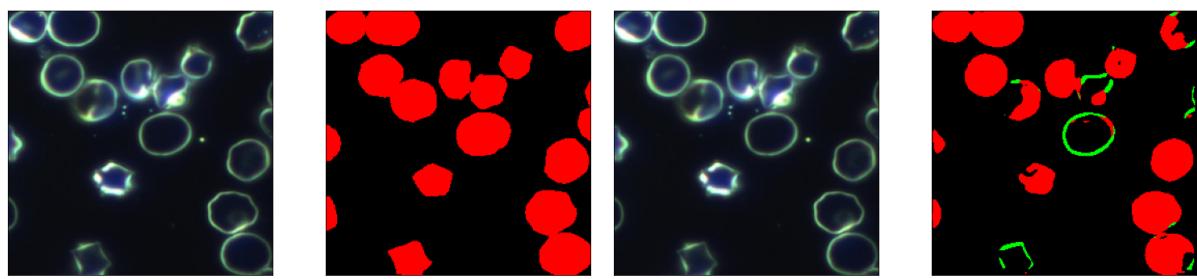
(a) Original

(b) Predicho



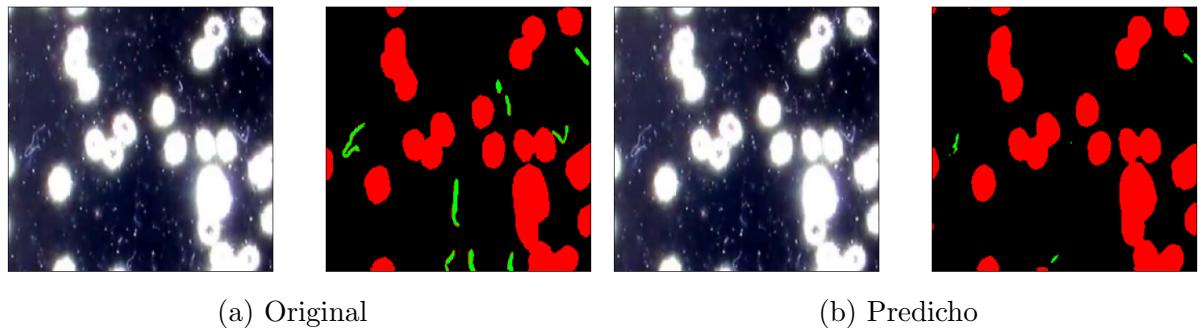
(a) Original

(b) Predicho



(a) Original

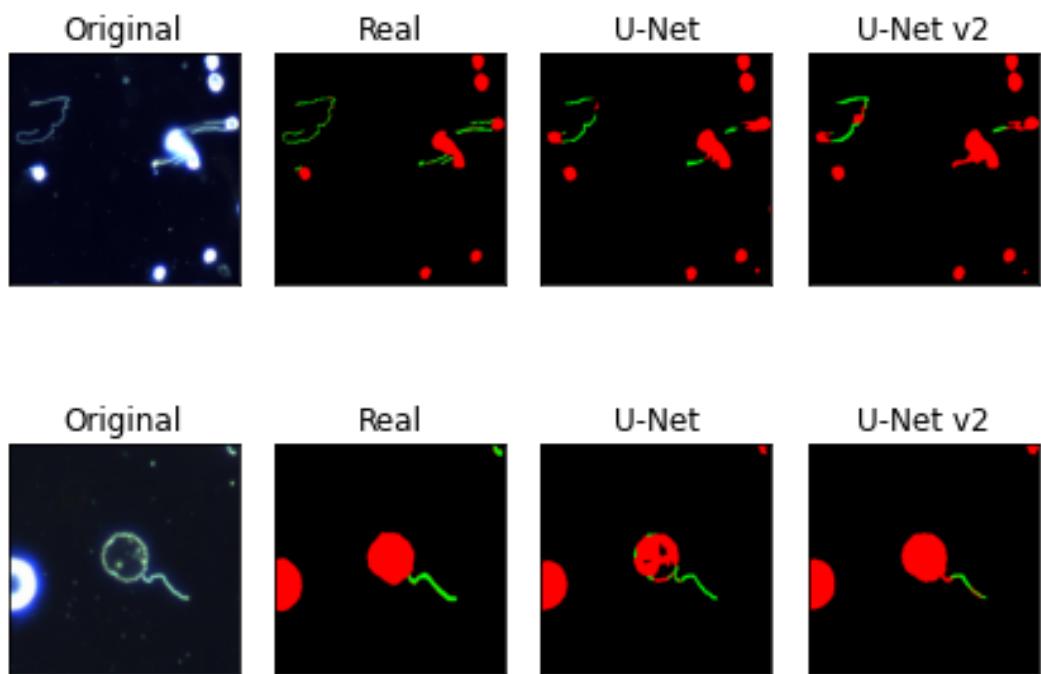
(b) Predicho



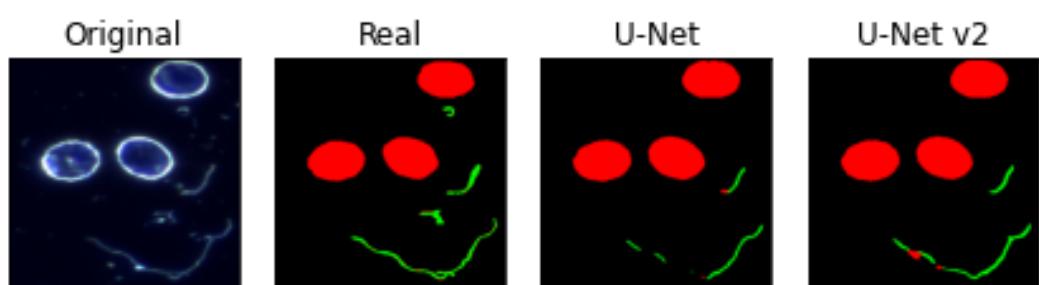
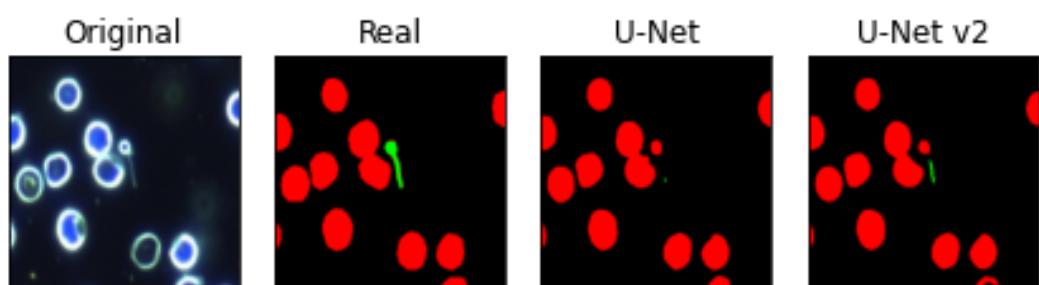
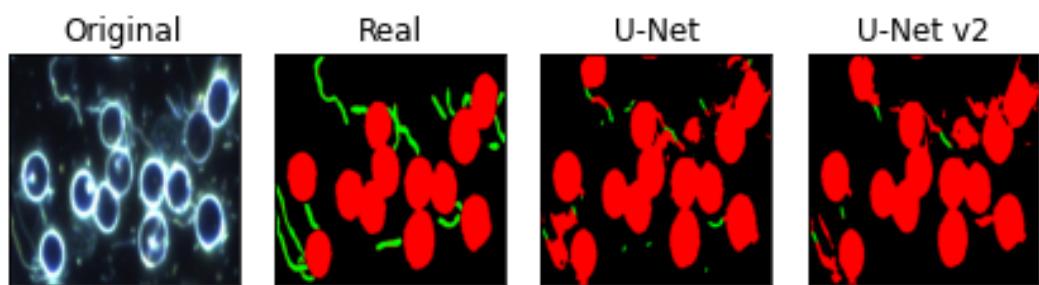
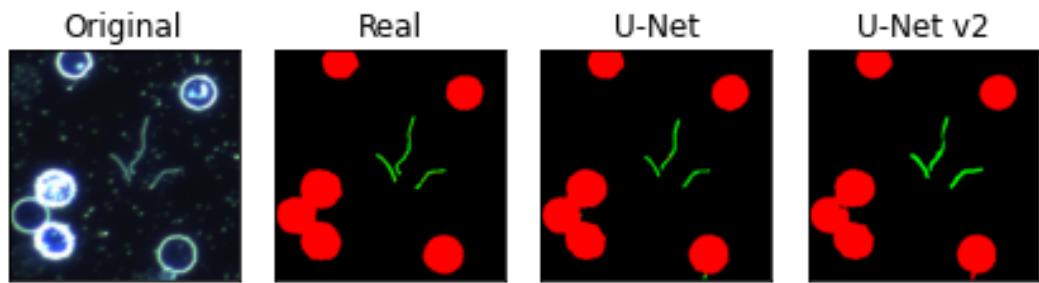
Los resultados parecen ser mejores que los anteriores. La red sigue teniendo problemas para distinguir células demasiado transparentes, pero es capaz de detectar bacterias con más precisión.

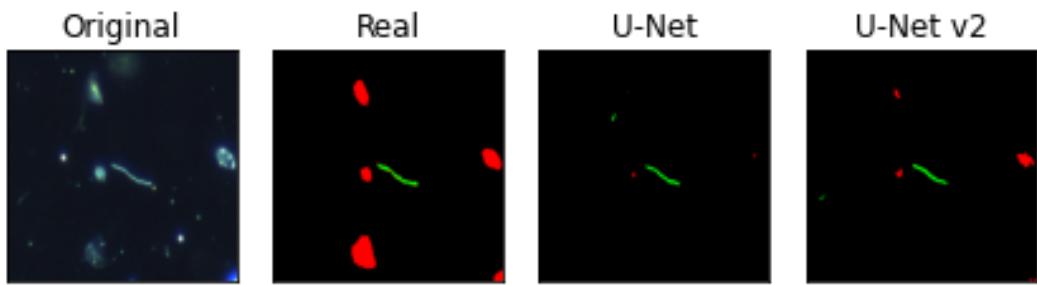
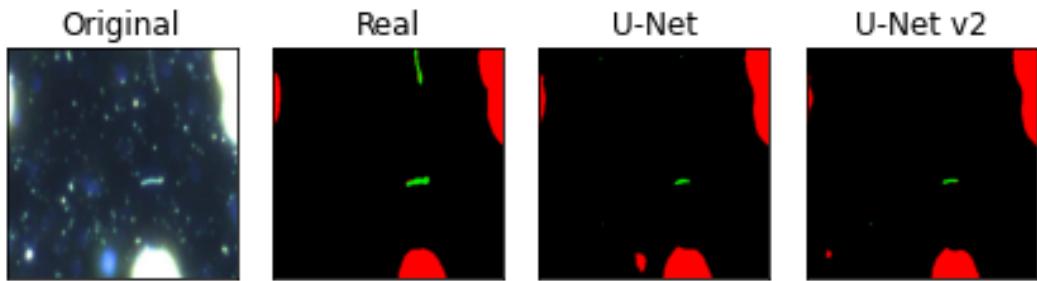
4.6 COMPARACIÓN U-NET VS U-NETv2

Para observar más cuidadosamente las diferencias entre ambas redes, a continuación se exponen algunos resultados con imágenes de test.



Sección 4 Experimentos



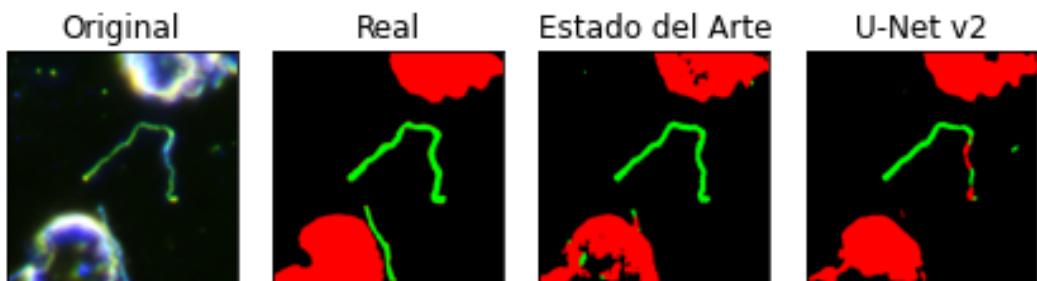


Se puede ver a simple vista que, aunque son redes bastante parecidas, la versión 2, en general, obtiene mejores resultados. Concretamente, U-Net v2 es más proclive a dibujar las células completamente rellenas, mientras que U-Net tiene algunas dificultades para ello (ejemplo 2). Sin embargo, en algunos casos concretos (ejemplo 4), la U-Net clásica obtiene mejores resultados detectando bacterias que su competidora.

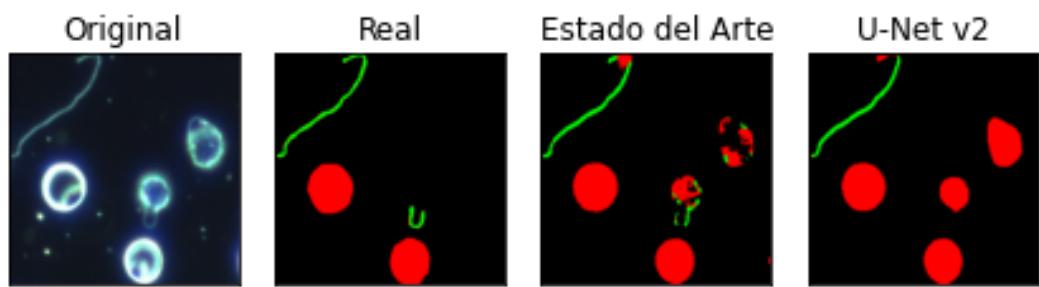
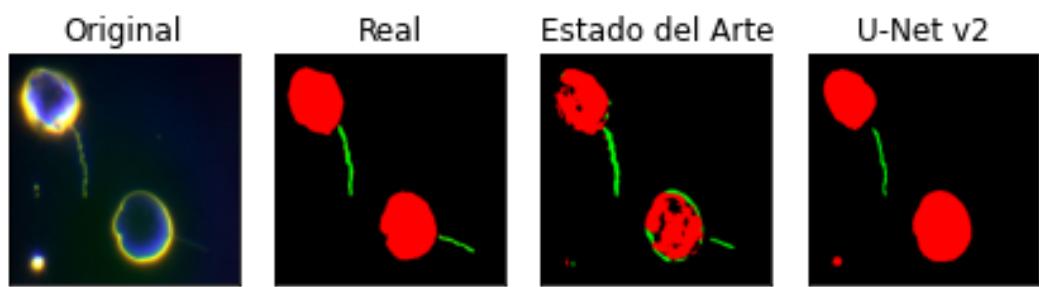
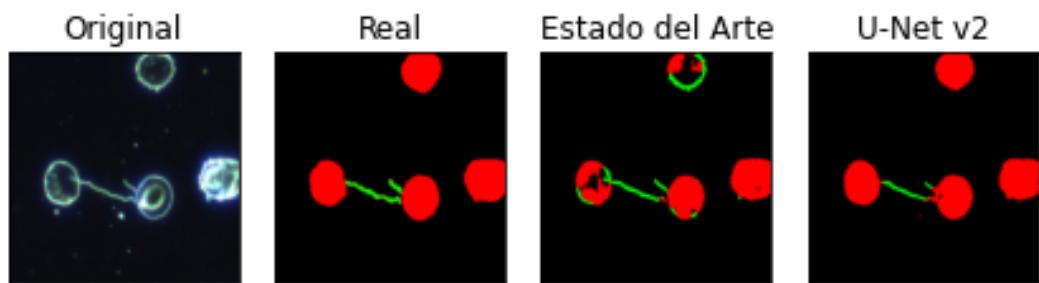
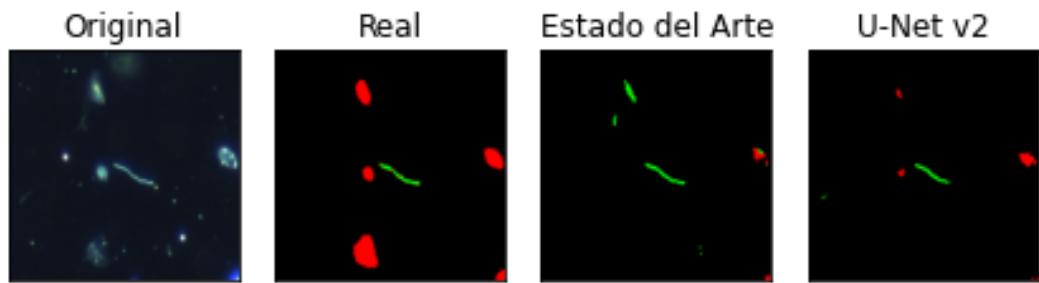
Aún así, podemos afirmar experimentalmente que, U-Netv2 es una red superior a la versión clásica en este problema.

4.7 COMPARACIÓN ESTADO DEL ARTE VS U-NET v2

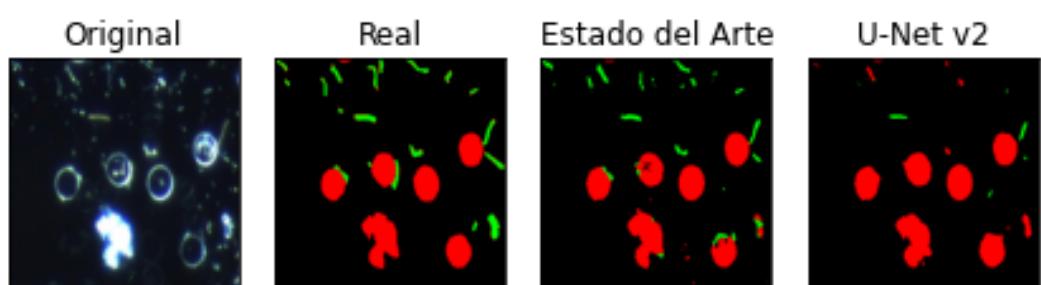
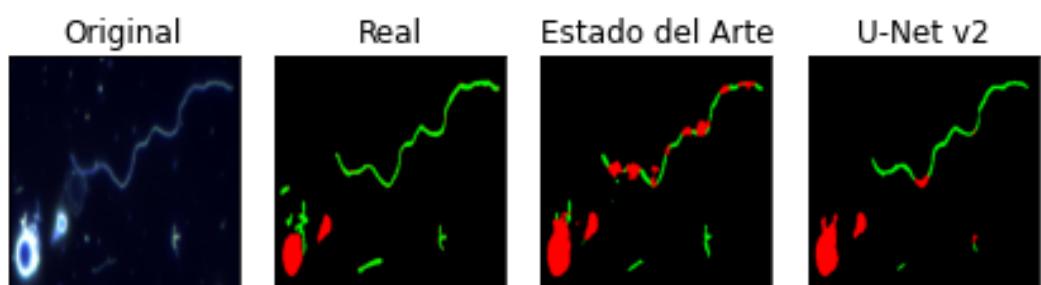
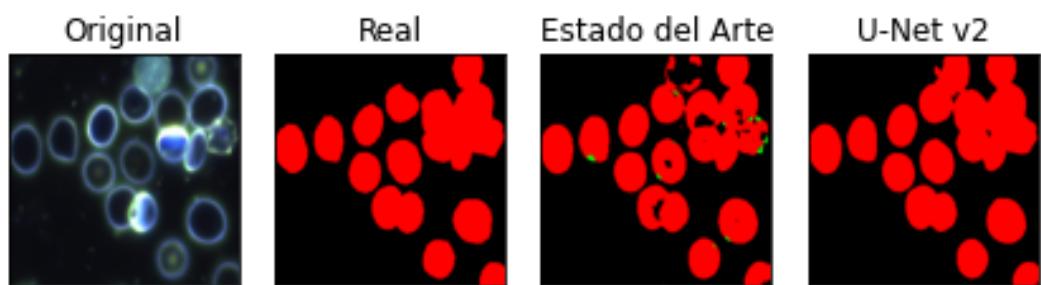
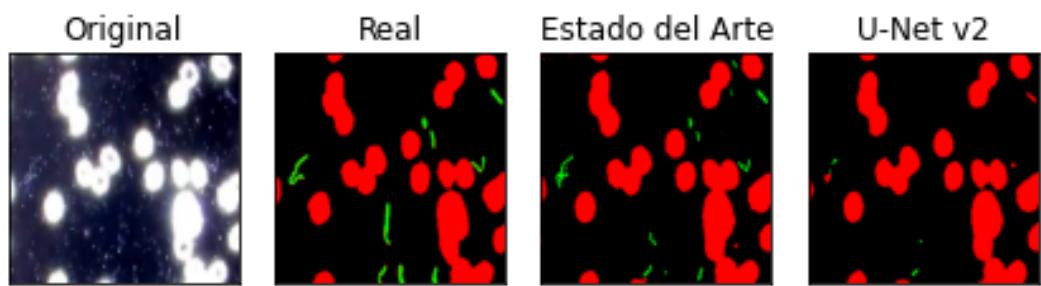
Como último experimento, podemos comparar nuestra red con el estado del arte actual del problema, la red propuesta por A. Le [10]. Su código ha sido ejecutado tal y como está publicado y, a continuación, se expondrán ejemplos de su comparativa con U-Net v2.



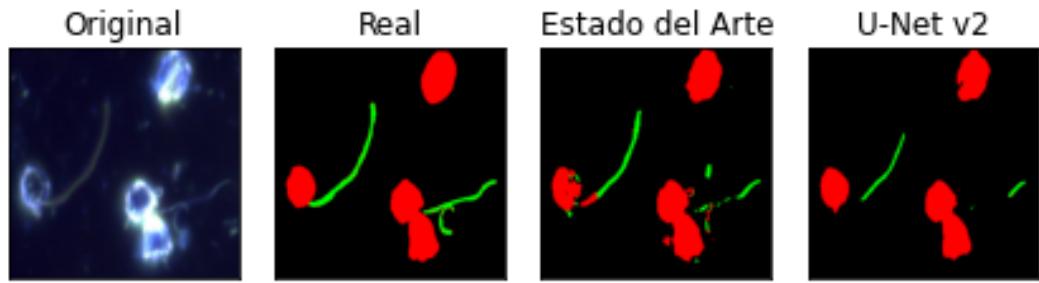
Sección 4 Experimentos



Sección 4 Experimentos



Sección 4 Experimentos



En este caso, vemos que ambos modelos son bastante buenos aunque presentan pequeñas diferencias. Por un lado podemos ver que U-Net v2 realiza una segmentación más precisa de las células sanguíneas, rellenando perfectamente su interior, en lo que falla el modelo del estado del arte (ejemplo 3). U-Net v2 también segmenta mejor imágenes donde las bacterias son de gran tamaño (ejemplo 8). También, nuestro modelo es más conservador a la hora de detectar bacterias, con lo cual suele tener una imagen más “ limpia” sin detectar bacterias en el fondo, pero esto le hace pasar por alto cuando las bacterias son muy pequeñas y el fondo muy turbio (ejemplo 9). En estos casos, el estado del arte funciona mejor.

Puede ser debatible si nuestra red mejora el estado del arte, pero, lo que está claro es que, al menos, se encuentra a su nivel.

SECCIÓN 5

Cota sobre el error

Ya que tenemos el modelo final entrenado con todos los datos de training, siempre es interesante el hecho de calcular una cota superior sobre el error que cometerá nuestro modelo ante la llegada de nuevos datos [22].

Aunque no explicaremos en detalle este paso, ya que se explicó en profundidad en la asignatura de Aprendizaje automático, podemos recordar que la cota superior sobre error que cometerá nuestro modelo fuera de la muestra E_{out} a partir del error cometido en test E_{test} viene dada por la desigualdad de Hoeffding:

$$E_{out}(h) \leq E_{test}(h) + \sqrt{\frac{1}{2N} \log \frac{2}{\delta}} \text{ con probabilidad } 1 - \delta$$

Siendo N el número de ejemplos de test.

Aplicando esta expresión sobre el resultado del mejor modelo obtenido, 0.701 en el coeficiente de Dice, y teniendo en cuenta que el conjunto de test está formado por 74 ejemplos, podemos calcular la cota superior sobre el error cometido con una confianza del 95 % de la siguiente forma:

$$E_{out} \geq 0,701 - \sqrt{\frac{1}{74 \times 2} \log \frac{2}{0,05}} \approx 0,543$$

Notar que la medida del error utilizada se trata de una métrica a maximizar, por ello la diferencia sobre la expresión anterior.

Por tanto, podríamos afirmar que con una confianza del 95 %, que nuestro modelo obtendrá valores de Dice superiores a 0.54 ante nuevos datos.

SECCIÓN 6

Conclusiones

En este proyecto hemos conseguido adaptar con éxito una red U-Net ya entrenada al problema de clasificación de bacterias, obteniendo resultados del estado del arte de esta base de datos, por lo que consideramos los objetivos cumplidos.

Como posibles trabajos futuros, apuntamos a la idea de investigar más funciones de loss o combinaciones entre ellas para este problema concreto, además de diseñar una función más sofisticada para calcular los pesos de las clases.

Otros posibles experimentos sería utilizar una red más avanzada que U-Net e incorporar más mejoras incrementales.

Bibliografía

- [1] The Editors of Encyclopaedia Britannica, *spirochete — Definition, Examples, Diseases, and Facts*, Encyclopedia Britannica. [Online]. Available: <https://www.britannica.com/science/spirochete>. [Accessed: 21 - Jan- 2021]. 1.3
- [2] A. Hafiz and G. Bhat, *A survey on instance segmentation: state of the art*, International Journal of Multimedia Information Retrieval, vol. 9, no. 3, pp. 171-189, 2020. Available: 10.1007/s13735-020-00195-x. 2.1
- [3] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz and D. Terzopoulos, *Image Segmentation Using Deep Learning: A Survey*, arXiv, 2020. Available: <https://arxiv.org/abs/2001.05566>. [Accessed 14 January 2021]. 2.1
- [4] S. Wiewman and H. de Villiers, *Semantic segmentation of bioimages using convolutional neural networks*, 2016 International Joint Conference on Neural Networks (IJCNN), Vancouver, BC, 2016, pp. 624-631, doi: 10.1109/IJCNN.2016.7727258. 2.1
- [5] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in International Conference on Medical image computing and computer-assisted intervention. Springer, 2015, pp. 234–241. 2.1, 3.7, 3.10
- [6] F. Milletari, N. Navab, and S.-A. Ahmadi, *V-net: Fully convolutional neural networks for volumetric medical image segmentation*, in 2016 Fourth International Conference on 3D Vision (3DV). IEEE, 2016, pp. 565–571. 2.1
- [7] Z. Zhou, M. Siddiquee, N. Tajbakhsh and J. Liang, *UNet++: Redesigning Skip Connections to Exploit Multiscale Features in Image Segmentation*, IEEE Transactions on Medical Imaging, vol. 39, no. 6, pp. 1856-1867, 2020. Available: 10.1109/tmi.2019.2959609. 2.1, 2.2
- [8] S. Seal, *U-NET Custom_Segmentation kernel 3, 5, and 7*, Kaggle.com, 2020. [Online]. Available: <https://www.kaggle.com/sankarsanseal/u-net-custom-segmentation-kernel-3-5-and-7>. [Accessed: 21- Jan- 2021]. 2.2, 3.3
- [9] S. Seal, *U-NET Custom_Squeeze_and_Excitation*, Kaggle.com, 2020. [Online]. Available: <https://www.kaggle.com/sankarsanseal/u-net-custom-squeeze-and-excitation>. [Accessed: 21- Jan- 2021]. 2.2

Bibliografía

- [10] A. Le, *Bacteria Segmentation*, Kaggle.com, 2020. [Online]. Available: <https://www.kaggle.com/anhvle/bacteria-segmentation>. [Accessed: 21- Jan- 2021]. 2.2, 3.3, 3.7.2, 4.7
- [11] L. Nguyen, *Unet Plus Plus for image segmentation*, Kaggle.com, 2020. [Online]. Available: <https://www.kaggle.com/longnguyen2306/unet-plus-plus-for-image-segmentation>. [Accessed: 21- Jan- 2021]. 2.2, 3.4, 3.7.2
- [12] J. Chen, *neural network converges too fast and predicts blank results*, Stack Overflow, 2018. [Online]. Available: <https://stackoverflow.com/a/52125590>. [Accessed: 21- Jan- 2021]. 3.4
- [13] A. Monteux, *Metrics for semantic segmentation*, Excursions in data, 2019. [Online]. Available: <https://ilmonteux.github.io/2019/05/10/segmentation-metrics.html>. [Accessed: 21- Jan- 2021]. 3.4
- [14] *DRIVE - Grand Challenge*, grand-challenge.org. [Online]. Available: <https://drive.grand-challenge.org/>. [Accessed: 29- Jan- 2021]. 3.5, 3.5
- [15] J. Shruti, *A survey of loss functions for semantic segmentation*, IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), 2020. [Accessed 21- Jan- 2021]. 3.6
- [16] D. Mishra, *Transposed Convolution Demystified*, Medium, 2020. [Online]. Available: <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>. [Accessed: 21- Jan- 2021]. 3.7.2
- [17] H. Lamba, *Understanding Semantic Segmentation with UNET*, Medium, 2019. [Online]. Available: <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>. [Accessed: 28- Jan- 2021]. 3.7.2
- [18] Lei Huang, *Dice-coefficient loss function vs cross-entropy*, Cross Validated, 2021. [Online]. Available: <https://stats.stackexchange.com/questions/321460/dice-coefficient-loss-function-vs-cross-entropy>. [Accessed: 27- Jan- 2021]. 3.6.3
- [19] R. Gómez, *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*, Gombru.github.io, 2021. [Online]. Available: https://gombru.github.io/2018/05/23/cross_entropy_loss/. [Accessed: 23- Jan- 2021]. 3.6.1
- [20] M. Kazemi, *How is total loss calculated over multiple classes in Keras?*, Stack Overflow, 2019. [Online]. Available: <https://stackoverflow.com/questions/52034983/how-is-total-loss-calculated-over-multiple-classes-in-keras>. [Accessed: 24- Jan- 2021]. 3.6.1
- [21] V. Bushaev, *Adam—latest trends in deep learning optimization.*, Medium, 2021. [Online]. Available: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>. [Accessed: 21- Jan- 2021]. 4

Bibliografía

- [22] Y. Abu-Mostafa, Y. Magdon-Ismail and M. Lin H, *Learning from data: A short course.* 2012. 5

Anexo: Instrucciones de ejecución

Para ejecutar nuestro código es necesario seguir los siguientes pasos:

1. Descomprimir la carpeta **Codigo** que está entregada junto a esta memoria.
2. Descargarse los datos de **Bacteria detection with darkfield microscopy** de esta página de kaggle:
<https://www.kaggle.com/longnguyen2306/bacteria-detection-with-darkfield-microscopy>
3. Descomprimir los datos descargados en el directorio **Codigo/data/**. Los datos deben de estar accesibles con los paths **Codigo/data/images/** y **Codigo/data/masks/**.
4. Descargar los datos de pre-entrenamiento desde la página:
<https://drive.grand-challenge.org/>
5. Para poder descargar los datos de DRIVE, primero es necesario registrarse, haciendo click en el apartado “join”. Luego, podemos pulsar en Download para acceder a los datos subidos en dropbox.
6. Descomprimir los datos descargados en el directorio **Codigo/data_pretraining/**. Los datos deben de estar accesibles con los paths **Codigo/data_pretraining/-test/** y **Codigo/data_pretraining/train/**.
7. Por último, ejecutar el archivo **Codigo/proyecto/main.py**

La estructura final de directorios debe ser esta:

- BacteriaDetectionMemoria.pdf
- Codigo/
 - data/
 - images/
 - ...
 - masks/
 - ...
 - data_pretraining/

Bibliografía

```
- test/
  - images/
    - ...
  - masks/
    - ...
- training/
  - images/
    - ...
  - masks/
    - ...
- proyecto/
  - saves/
  - loss.py
  - main.py
  - visualization.py
```

El código main.py está preparado para realizar un pre-entrenamiento y posterior entrenamiento de U-Net y U-Net v2.

Si se quiere ejecutar alguno de los otros experimentos, basta con descomentar la llamada a su respectiva función en el main.

Si tienen cualquier problema, puede contactar con nosotros escribiendo a *alekpinel@correo.ugr.es* o a *angelvij@correo.ugr.es*.