

PRÁCTICA 2
(ENUNCIADO 3)

SEGMENTACIÓN SEMÁNTICA

Aprendizaje Automático
Curso 2022-2023

Crespí Valero, Maribel
Fortes Domínguez, Odilo

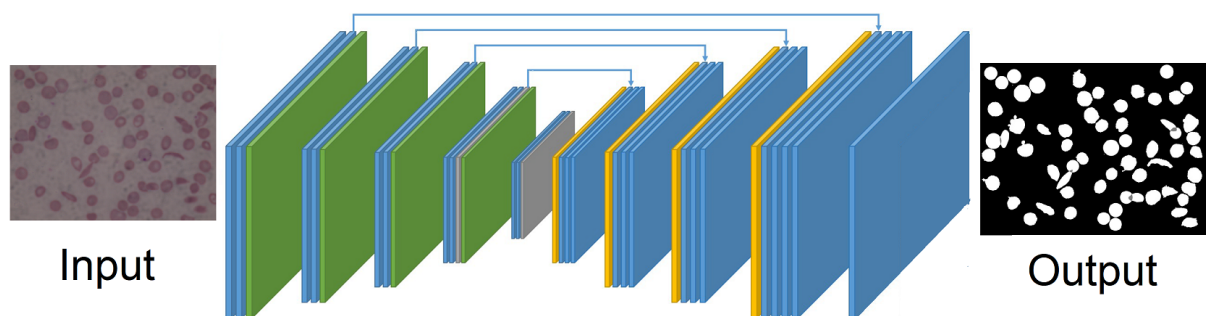
| | |
|---|-----------|
| Introducción al problema | 3 |
| Soluciones consideradas | 4 |
| Modelos | 4 |
| UNet | 4 |
| PotiNet | 4 |
| XopiNet | 5 |
| Tratamiento de los datos | 5 |
| Métricas | 7 |
| Intersection over Union | 7 |
| Dice Score | 7 |
| Función de pérdida | 8 |
| Optimizador | 8 |
| Experimentos realizados | 9 |
| UNet | 9 |
| PotiNet | 10 |
| XopiNet | 11 |
| Resultados de los experimentos | 12 |
| UNet vs PotiNet | 12 |
| UNet vs XopiNet | 15 |
| Conclusiones | 17 |
| Adjunción de los ficheros con los pesos + dataset modificado | 17 |

Introducción al problema

El [problema planteado en este ejercicio](#) es un problema de segmentación de imágenes médicas, en el que se deben separar las células que aparecen del fondo de la imagen.

El conjunto de datos original está formado por 50 pares de imágenes, donde cada par está en una carpeta diferente y todas están contenidas en el directorio principal '[Mabimi](#)'. Los pares de imágenes consisten en la imagen original y la máscara de la misma.

El objetivo es obtener modelos de redes neuronales que consigan segmentar imágenes médicas satisfactoriamente únicamente a partir de la imagen original, para posteriormente evaluarlos. Uno de los modelos estará basado en una red ya existente (elegiremos UNet), y los otros serán modificaciones de ésta (de este modo podremos hacer comparaciones más fácilmente).



*Los modelos deberán ser capaces de realizar segmentación a partir de una imagen original.

También habremos de seleccionar una función de pérdida adecuada al problema.

Para abordar el problema de segmentación de imágenes utilizaremos dos herramientas ya vistas previamente en la asignatura:

- **PyCharm:** entorno de desarrollo integrado (IDE) de Python que nos permite escribir, depurar y ejecutar código de manera eficiente.
- **Jupyter Notebook:** permite escribir y ejecutar código a la vez que nos permite visualizar los resultados de manera intuitiva y hacer anotaciones en el mismo.

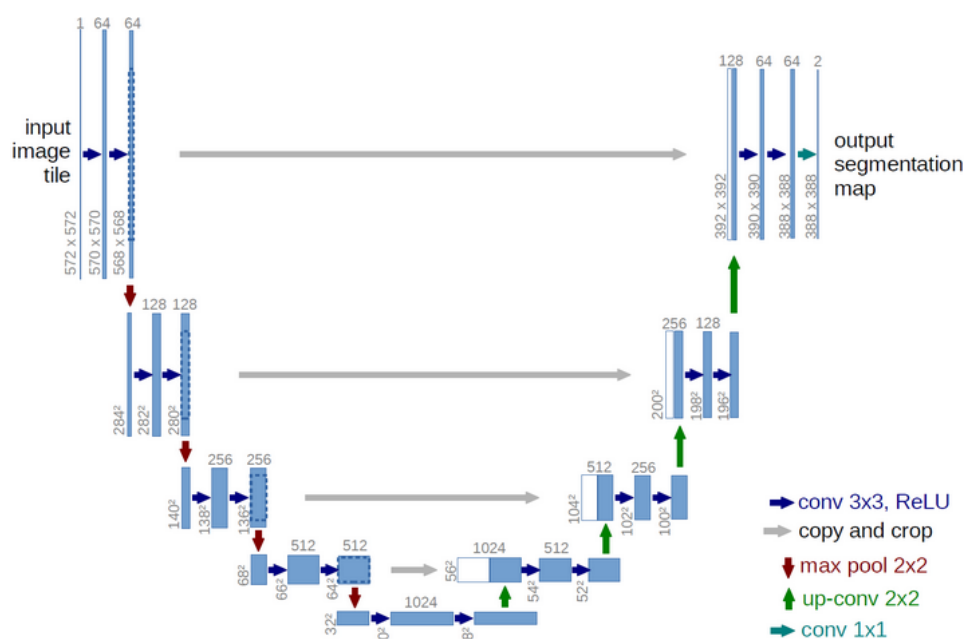
Soluciones consideradas

En todo problema de redes neuronales hay una serie de aspectos a tratar antes de pasar a la propia implementación de la solución.

Modelos

UNet

Para el caso de evaluar una arquitectura ya existente utilizaremos una [UNet](#). Esto es porque la UNet está especialmente diseñada para la segmentación de imágenes (tiene skip connections que retienen información de la imagen al reconstruirla tras extraer características), y porque además la hemos dado en clase y ya tenemos una [implementación](#) en el github del profesor.



* Las dimensiones de esta UNet son diferentes a la nuestra, pero la estructura y combinación de capas son iguales.

Para el caso de modelos propios, realizaremos distintos experimentos:

PotiNet

Variación de la UNet que se deshará de las skip connections. De este modo podremos evaluar cómo afecta el hecho de tener o no esta técnica implementada. Esperamos que los resultados sean peores, ya que ahora el decoder recibirá la mitad de features (las que ya no llegarán desde el encoder), e impactará

negativamente en la capacidad de la red para retener información a lo largo de las capas y resultará en una reducción de la precisión y un aumento del error.

XopiNet

Variación de la UNet en la que añadiremos más capas de profundidad. Quizás obtenga mejores resultados, pero corre riesgo de overfitting si se ajusta demasiado a los datos de entrenamiento. Además, será computacionalmente más costoso.

En la siguiente sección principal (Experimentos realizados) veremos la estructura de los modelos y de las propias capas.

Tratamiento de los datos

Debido a que utilizaremos la librería Pytorch, tendremos que aplicar una serie de cambios a las imágenes para así poder trabajar con ellas adecuadamente.

Al analizar los datos rápidamente nos damos cuenta de que las imágenes de las máscaras originales tienen varios detalles a tener en cuenta:

- El primero es que el espacio de color de las máscaras no es binario, es decir, hay un margen de grises que no nos interesan. Esto se puede apreciar muy claramente en aquellas células que se solapan, siendo la intersección de éstas de un color gris. A nosotros no nos interesa que haya grises, así que aplicamos un umbral donde los píxeles se vuelvan blancos completamente, y el resto que se vuelvan negros.
- Lo segundo es que había una imagen mask en concreto cuyo espacio de color era diferente al resto (mask de cell_14), concretamente RGB. Lo hemos notado ya que las operaciones con tensores fallaban, debido a que el tensor obtenido de esta imagen tenía dimensiones diferentes y causaba errores en las ejecuciones. Simplemente la hemos convertido a escala de grises y la hemos sustituido en el dataset. Esto quiere decir que para la ejecución del código proporcionado hay que descargar el dataset modificado (hay un link al final del documento que también incluye los pesos para descargar).

Una vez resueltas estas consideraciones iniciales, ya podemos proceder a crear nuestro dataset.

La idea es obtener dos listas: una que contenga los paths de las imágenes source y otra para las imágenes mask.

Crearemos nuestra propia clase dataset donde almacenaremos todas las imágenes junto a sus máscaras. Las ordenaremos para que se corresponda cada source con su mask. Además, es aquí donde les aplicaremos la transformación necesaria para poder tenerlas en un formato que se ajuste a nuestra manera de trabajar con las redes neuronales. Esta transformación consistirá en hacer resize a 224x224 (es un tamaño manejable computacionalmente y aceptable para obtener un resultado que se pueda visualizar correctamente). Posteriormente las convertimos en tensores. Finalmente normalizaremos las imágenes source, dividiendo entre el valor máximo de píxel.

Una vez hecho esto, dividimos el conjunto de datos en entrenamiento y test, otorgando un tamaño del 80% al entrenamiento (40 imágenes) y un 20% al test (10 imágenes). Lo hemos hecho así porque parece ser un split ratio común para datasets pequeños, aunque hay muchas opiniones variadas.

Luego realizaremos una división aleatoria de los datos de entrenamiento y los datos de test, para finalmente cargar los datos en lotes.

En este caso tan particular como apenas contamos con 50 muestras (es un número muy bajo) hemos decidido utilizar batches de tamaño 1. El tiempo de entrenamiento se alargará un poco, pero como no hay muchas muestras no supondrá demasiado coste computacional. Este tamaño de batch nos permitirá captar mejor las características de cada imagen y que no pasen desapercibidas.

Además, un tamaño de batch grande para el entrenamiento no hubiera sido correcto ya que por ejemplo con un tamaño de 10, sólo reajustaríamos los pesos 4 veces (teniendo en cuenta que nuestro conjunto de entrenamiento está formado por 40 imágenes).

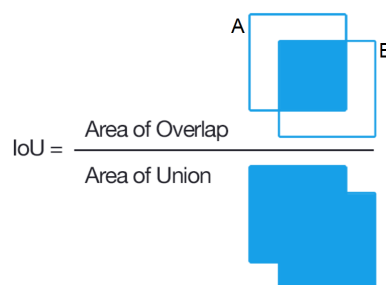
Métricas

Hemos seleccionado dos métricas para evaluar el rendimiento del modelo: IoU (Intersection over Union) y Dice Score. Utilizar dos nos proporcionará una visión más amplia del rendimiento de los modelos. Aunque en el caso de segmentación binaria, el Dice Score es la métrica más adecuada para evaluar la calidad de la segmentación. Esto es debido a que es una métrica que se ajusta bien a la evaluación de la similitud entre dos colecciones de píxeles, como los resultados de la segmentación y los datos de referencia, y de hecho es comúnmente utilizada en el contexto de segmentación binaria en imágenes médicas. Por otro lado, el IoU mide la similitud entre dos regiones y puede no ser la mejor opción para segmentación binaria, donde se clasifican los píxeles en lugar de identificar regiones completas.

Cabe mencionar que en un problema de este tipo, una medida como el Accuracy, que se podría calcular como el número de píxeles predichos de forma correcta sobre el total, no sería muy óptimo. Se podría dar el caso de que el objeto que queremos segmentar no ocupa mucho espacio de la imagen y que al realizar una clasificación entre el fondo y el objeto diera un resultado alto en esta métrica, a pesar de que no estaría realizando bien la segmentación.

Intersection over Union

Como su propio nombre indica, el cálculo de esta métrica consiste en la división de la intersección entre la unión de ambos conjuntos, siendo A el ground truth y B el resultado de la segmentación en la imagen.


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Dice Score

La [Dice Score](#) tiene en cuenta tanto los positivos encontrados como los que no se encontraron, penalizando si el algoritmo no los encuentra.

Esta métrica consiste en la siguiente fórmula:

$$\frac{2 \cdot \text{number of true positives}}{2 \cdot \text{number of true positives} + \text{number of false positives} + \text{number of false negatives}}$$

O de manera más gráfica, con operaciones entre conjuntos:

$$\frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Siendo de nuevo A el ground truth y B el resultado de la segmentación.

Función de pérdida

La función de pérdida es una forma de medir cuán lejos está el modelo de generar resultados esperados, o dicho de otra forma, calcula la diferencia entre la predicción y el resultado correcto. En conjunto con un optimizador será posible el entrenamiento del modelo, que tratará de minimizar el error.

Nosotros usaremos el Dice Loss, que como el propio nombre sugiere, está relacionado con el Dice Score, pues el Dice Loss se obtiene restando a 1 el propio Dice Score.

$$\text{DiceLoss} = 1 - \frac{2 \cdot |A \cap B|}{|A| + |B|}$$

Optimizador

El optimizador se encarga de actualizar los pesos y parámetros de la red neuronal en respuesta a los errores que se cometen en la predicción (información proporcionada por la función de pérdida).

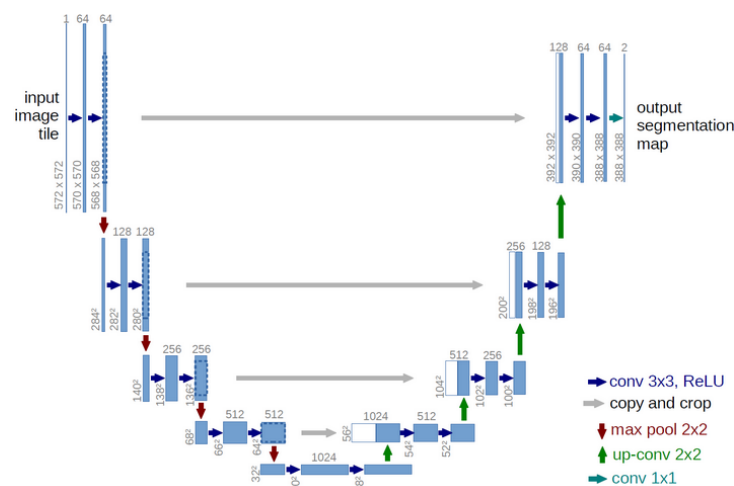
Para el problema usaremos el optimizador Adam, que es uno de los más populares en deep learning. Esto es debido a que tiene un rápido tiempo de computación y requiere afinar pocos parámetros. Los resultados de este optimizador son generalmente muy buenos, pues tiene una capacidad de convergencia rápida y efectiva.

Experimentos realizados

En este apartado vamos a explicar la estructura de capas de cada modelo brevemente, puesto que en el código proporcionado se puede observar más detalladamente.

UNet

En la UNet se sigue el esquema siguiente. Es importante mencionar que entre cada convolución y ReLU hay una capa de batch normalization, que ayudará a mejorar el entrenamiento y rendimiento del modelo. Además, después de la convolución final se utiliza la función sigmoide para obtener la probabilidad de que el píxel sea positivo o no (y más tarde lo binarizaremos para las métricas).



Para facilitar el proceso tenemos creado como bloque la secuencia de [{convolución, normalización, ReLU}, {convolución, normalización, ReLU}].

```
# Aplicamos la primera convolución, con su normalización pertinente y capa de activación ReLU
(name + "conv1", nn.Conv2d(in_channels=in_channels, out_channels=features, kernel_size=3, padding=1, bias=False)),
(name + "norm1", nn.BatchNorm2d(num_features=features)),
(name + "relu1", nn.ReLU(inplace=True)),
# Aplicamos la segunda convolución, con su normalización pertinente y capa de activación ReLU
(name + "conv2", nn.Conv2d(in_channels=features, out_channels=features, kernel_size=3, padding=1, bias=False)),
(name + "norm2", nn.BatchNorm2d(num_features=features)),
(name + "relu2", nn.ReLU(inplace=True))
```

PotiNet

Esta red es una variación de la UNet que únicamente omite las skip connections.

```
dec2 = self.upconv2(dec3)
dec2 = torch.cat((dec2, enc2), dim=1)
dec2 = self.decoder2(dec2)
```

Este fragmento de código es de la UNet, y se puede observar como el decodificador 2 no sólo procesa el resultado del decodificador 3, sino que además añade el del codificador 2. El código anterior pasará a ser el siguiente:

```
dec2 = self.upconv2(dec3)
dec2 = self.decoder2(dec2)
```

Simplemente omitimos el paso del codificador 2 (hemos puesto este como ejemplo, pero aplica a todas las skip connections). Además de esto hay que modificar el tamaño de los parámetros de las capas que se implementan. La parte omitida representaba la mitad de la entrada, por lo que deberemos “dividir” entre dos el tamaño del input. O dicho de otra forma, quitar la multiplicación por 2 adicional que había en la parte de los canales de entrada de cada uno de los decoder. Quedando el tamaño resultante igual que en la parte del encoder. Por eso ahora ya no es necesario realizar la concatenación del decoder con el encoder. Con estos cambios los tamaños cuadran y ya se puede utilizar el nuevo modelo para segmentar imágenes.

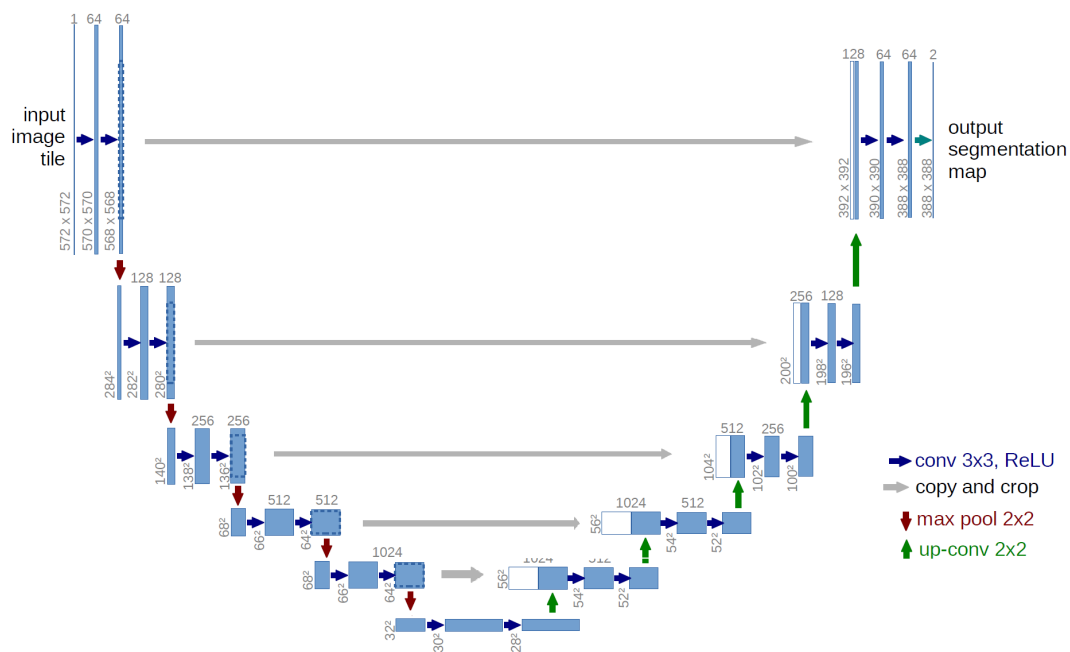
De este nuevo modelo podemos esperar un rendimiento algo menor respecto a la arquitectura original, pues ahora perdemos esa información extra que nos venía desde las skip connections.

XopiNet

Para XopiNet, de nuevo hemos utilizado de base la UNet. En este caso, la principal diferencia es que se ha agregado una capa adicional de profundidad. Esto ha sido tan simple como añadir una capa más de encoder y posteriormente una capa más de decoder equivalente. En este modelo sí que preservamos las skip connections.

En general se podría esperar un mejor rendimiento del nuevo modelo teniendo en cuenta que al agregar una capa extra, debería tener una mayor capacidad de aprendizaje de detección y segmentación de imágenes, pues al tener un mayor número de neuronas, puede capturar características más complejas de los datos de entrada.

Sin embargo esto quedará por ver, ya que se podría decir que en nuestro caso los objetos a segmentar (las células) no poseen formas demasiado complejas.

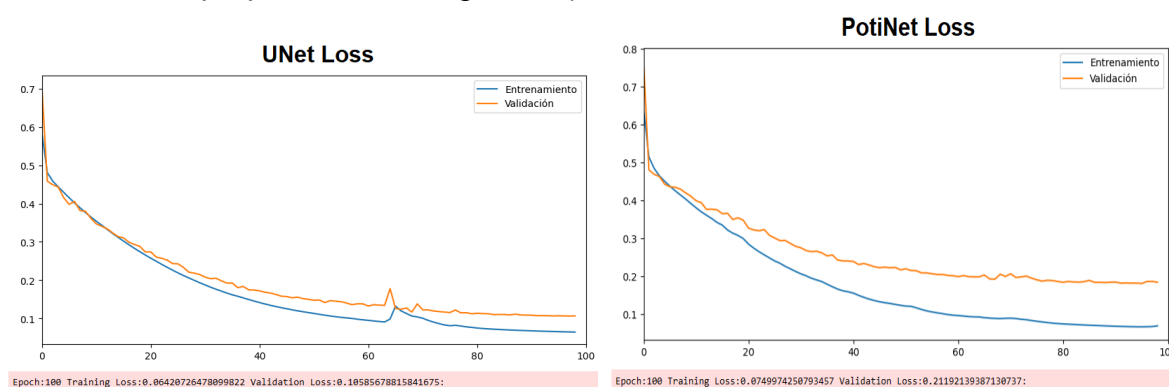


Resultados de los experimentos

Dado que ambos modelos propios parten de la UNet, tiene sentido compararlos con ésta. Cabe añadir que hemos escogido ejecuciones representativas y que no son las mismas que las que se hallan en los ficheros de jupyter notebook (aunque los modelos sean los mismos).

UNet vs PotiNet

Vamos a observar la evolución de la pérdida de cada modelo para compararlos. Cabe destacar que la pérdida de cada uno inicia desde diferentes valores (0.7 vs 0.8), por lo que estiramos la gráfica de PotiNet para que correspondan las marcas (lo hacemos así porque sólo queremos echar un vistazo rápido, de otro modo hubiéramos superpuesto ambas gráficas).



Como se puede observar, el error de PotiNet disminuye a una tasa más lenta en comparación a la UNet. Esto se debe a que la red está limitada en su capacidad para aprender y mejorar con cada iteración de entrenamiento, pues carece de la información adicional proporcionada por las skip connections.

| | Training Loss | Validation Loss |
|----------------|---------------|-----------------|
| UNet | 0.06421 | 0.10586 |
| PotiNet | 0.07500 | 0.21192 |

En la tabla anterior podemos ver la media aritmética de cada tipo de pérdida para cada modelo. El training loss parece no diferir mucho entre los dos modelos, pero el validation loss de PotiNet es alto y de hecho es el doble que el de la UNet. Esto indica que hay algo de overfitting, ya que el modelo se ha ajustado demasiado a las muestras de entrenamiento (y por eso muestra buenos resultados de entrenamiento), pero no es capaz de generalizar del todo ese conocimiento a nuevos datos.

En cuanto a las métricas de evaluación, podemos observar lo siguiente:

| | DiceScore | Iou |
|----------------|------------------|------------|
| UNet | 92.95% | 86.84% |
| PotiNet | 90.08% | 82.13% |

Los resultados obtenidos en la UNet son bastante buenos, y lógicamente mejores que los de PotiNet. Como ya hemos explicado anteriormente, esto es debido a que al quitar las skip connections se pierde información y capacidad de abstracción, y por tanto tendrá una mayor dificultad para mantener la integridad espacial de los objetos segmentados. Por lo tanto al realizar una peor segmentación de la imagen, al realizar la evaluación de métricas también se obtendrán resultados más bajos.

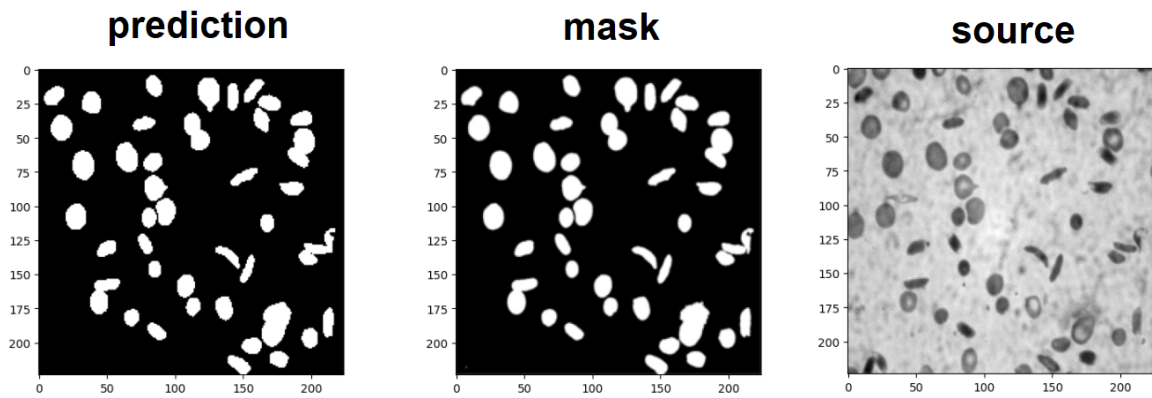
Es importante mencionar que en las máscaras originales hay ciertas células sin resaltar (hecho adrede por quién etiquetó las fotos, ya que representan células enfermas). Sin embargo nuestro modelo no las diferencia y las acaba resaltando en su predicción. Este hecho provoca que de por sí ya haya diferencias entre el ground truth y las predicciones (y se obtendrá un menor rendimiento según las métricas), a pesar de que el modelo podría decirse que segmenta correctamente.

Recordemos que DiceScore está más enfocado hacia segmentación binaria, así que tomando más en cuenta esta métrica, podemos afirmar que realizan una segmentación muy buena.

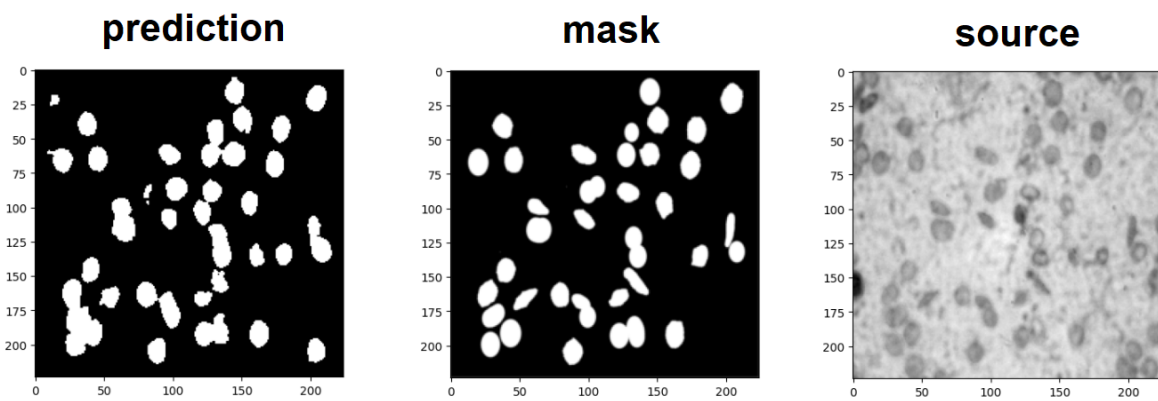
Finalmente hay que aclarar un hecho muy importante, y es que los resultados son relativamente buenos para el tamaño de datos que tenemos. Si tuviéramos más muestras podríamos habernos permitido un entrenamiento más extenso y una validación más fidedigna.

A continuación se mostrará un ejemplo de imagen del resultado de la segmentación (para UNet y PotiNet):

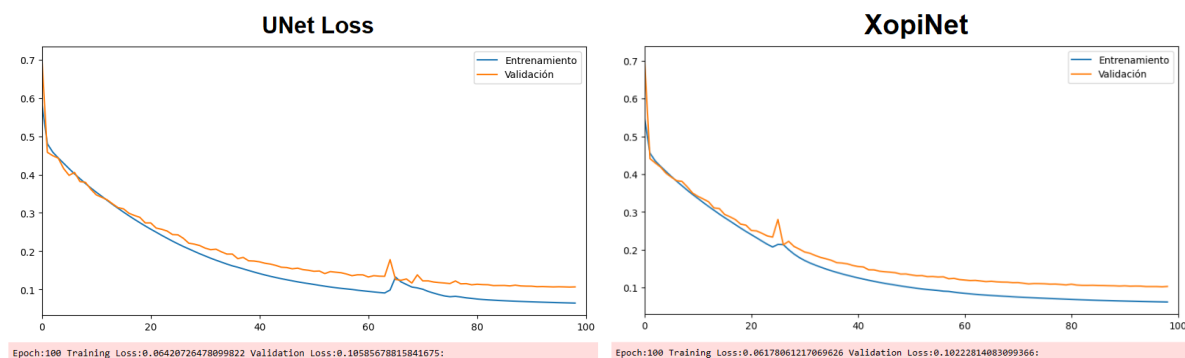
Resultados Segmentación UNet



Resultados Segmentación PotiNet



UNet vs XopiNet



Se puede observar que la evolución de ambas pérdidas son muy similares, y que los resultados finales son prácticamente idénticos.

Además hay presente un pico extraño en cada gráfica. No sabemos exactamente por qué ocurre esto, pero pensamos que es posible que en el proceso de optimización de la función de pérdida, la red se quede atrapada en un mínimo local en lugar de encontrar el mínimo global, lo que resultaría en un aumento temporal de la pérdida.

En general, lo primero que observamos es que el entrenamiento se ha realizado correctamente y a la hora de validar las muestras, el modelo ha funcionado bien. Al añadir una capa extra al modelo, resultaría en una mayor extracción de características y una mayor abstracción para mejorar el aprendizaje. Esto podría tener el problema de provocar overfitting. Sin embargo vemos que no ha sido el caso.

| | Training Loss | Validation Loss |
|---------|---------------|-----------------|
| UNet | 0.06421 | 0.10586 |
| XopiNet | 0.06178 | 0.10222 |

Como podemos observar, ambas pérdidas son prácticamente idénticas.

Ahora analicemos las métricas de evaluación:

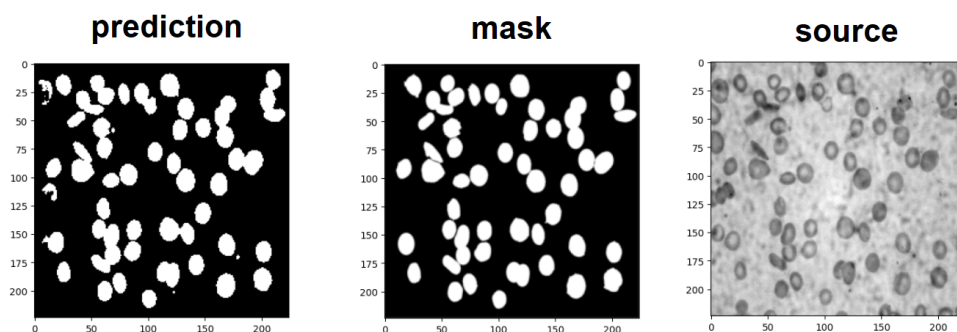
| | DiceScore | Iou |
|---------|-----------|--------|
| UNet | 92.95% | 86.84% |
| XopiNet | 89.77% | 81.53% |

Al añadir un nivel más de profundidad y por tanto más neuronas a nuestra red, podríamos esperar una mejora en los resultados de aprendizaje. Sin embargo, podemos ver mediante las métricas de evaluación que éstos incluso han empeorado. Evidentemente también se ha sufrido una penalización en cuanto al tiempo de ejecución del entrenamiento.

Por lo tanto, concluimos que añadir una capa extra no aporta mejoras sustanciales a nuestro modelo. Podría decirse que el nivel de capas de la UNet es lo suficientemente adecuado para poder resolver este tipo de problemas de segmentación. Añadir una, dos o n capas más al modelo no repercutiría en un impacto positivo real del desempeño final.

Nuestra interpretación es que los objetos a segmentar presentan formas sencillas, por lo que no es necesario añadir tantas características extras para poder realizar la segmentación de forma efectiva. Y es que precisamente la UNet se diseñó originalmente para la segmentación de imágenes médicas.

Resultados Segmentación XopiNet



Conclusiones

En resumen, la presencia de conexiones de saltos en la UNet es un factor clave en el rendimiento de la red y su capacidad para aprender y mejorar. Al eliminar estas conexiones, el error disminuye a una tasa más lenta y el error final es mayor en comparación con la UNet original. Este decremento de rendimiento se debe a que la red carece de la información adicional proporcionada por las skip connections y, por lo tanto, tiene dificultades para aprender, capturar características complejas y lograr una precisión de segmentación mayor.

También podemos concluir que añadir una capa más no aporta información relevante para nuestro modelo a la hora de realizar predicciones. Las formas de nuestros objetos a segmentar son relativamente sencillas por lo que no requiere de tanta abstracción en el aprendizaje de características. Así que en este caso, la UNet ya es una arquitectura ideal para este tipo de problemas.

En cuanto al proyecto en sí, estamos contentos de haber entendido todo el contenido, aplicando los conocimientos de la teoría para poder darle solución a un problema específico. Ha resultado muy satisfactorio ir viendo cómo íbamos completando los objetivos propuestos y sobre todo, ver las imágenes de las máscaras generadas.

También queremos agradecer al profesor por su ayuda y paciencia dada la situación en la que nos encontrábamos. Ha sido un factor determinante que ha contribuido a que pudiéramos comprender el contexto del problema y la propia resolución del mismo.

Adjunción de los ficheros con los pesos + dataset modificado

<https://drive.google.com/drive/folders/1jBZgbxT2wDI6CKrsh8tQCUdw8IMGGurF?usp=sharing>

Para cargar los pesos:

```
model.load_state_dict(torch.load('path/to/file.pth'))
```