Universidad Europea

Memoria Práctica 3

Martín de las Heras

Introducción

En esta práctica hemos aprendido el concepto de *Semantic Segmentation* en *Computer Vision* y conceptos subyacentes como el de Máscara mediante la realización de un modelo de clasificación con la arquitectura UNet. Todo esto se ha realizado bajo el objetivo de diferenciar en varias imágenes de inundaciones los píxeles que forman parte de la inundación y cuáles no. La práctica se divide en los siguientes apartados:

- Leer el *dataset* de *Flood Segmentation* y cargarlo como tensores.
- Realizar una estandarización del tamaño de los tensores (tanto en las imágenes como en la Máscara).
- Cargar el modelo U-Net predefinido en clase por el profesor.
- Realizar un entrenamiento completo de la U-Net con el *dataset* de *Flood Segmentation*.
- Mostrar las gráficas de *Loss* y del Jaccard *Index*.

Lectura del *dataset* como tensores

En primer lugar, iteramos por el *dataset* disponible en Canvas, transformando las imágenes y sus máscaras por parejas en tensores:

```
dd = PIL.Image.open(f'data/Image/{image}')
tt = F.pil_to_tensor(dd)
```

```
dd = PIL.Image.open(f'data/Mask/{mask}')
mm = F.pil_to_tensor(dd)
```

Estandarización del tamaño de los tensores

Durante la iteración definida antes sobre el *dataset*, redimensionamos las imágenes de manera que sean ingestables por el modelo U-Net. De la misma manera, las normalizamos para que todas estén en el mismo rango de valores.

```
tt = F.resize(tt, (100, 100))

tt = tt[None, :, :, :]
tt = torch.tensor(tt, dtype=torch.float) / 255.
```

En el caso de las máscaras, las modificaciones son mayores, debido a la necesidad que tenemos de procesarla más adelante, la redimensionamos para adaptarla a la salida del modelo U-Net. Para ello las redimensionamos y modificamos de manera que cada píxel represente la clase a la que pertenece. Por último permutamos las dimensiones para que coincidan con las de la salida del modelo.

```
mm = mm.repeat(3, 1, 1)
mm = F.resize(mm, (100, 100))
mm = mm[:1, :, :]

mm = torch.tensor((mm > 0.).detach().numpy(), dtype=torch.long)
mm = torch.nn.functional.one_hot(mm)
mm = torch.permute(mm, (0, 3, 1, 2))
mm = torch.tensor(mm, dtype=torch.float)
```

Cargar el modelo U-Net

A continuación, cargamos el modelo predefinido en clase U-Net con 3 canales (canales de color de la imagen) y 2 clases (clases de salida para saber si es agua o no). Para ello simplemente importamos del fichero presente en Canvas:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
unet = UNet(n_channels=3, n_classes=2).to(device)
```

Realizar el entrenamiento completo

De cara al entrenamiento nos encontramos con dos posibilidades:

- Separar *train* y *validation*
- No separar

Debido a que se trata de un ejercicio docente, hemos decidido probar las dos opciones. En primer lugar, definimos el optimizador, el cual tendrá todos los parámetros del modelo para ajustar, y la función de pérdida, en este caso la entropía cruzada:

```
optim = torch.optim.Adam(unet.parameters(), lr=1e-3)
cross_entropy = torch.nn.CrossEntropyLoss()
```

El resto del entrenamiento es casi idéntico al que realizamos la práctica pasada, salvo que, en el lugar de la precisión, definimos el índice de Jaccard o IoU (*Intersection over Union*):

```
intersection = torch.sum(pred_unflatten & target_unflatten, dim=(1,2))/torch.sum(pred_unflatten | target_unflatten, dim=(1,2))
jaccard_epoch_train.append(torch.mean(intersection).detach())
```

Podemos ver que realmente lo que se está calculando es la media normalizada de las intersecciones de la predicción sobre la unión. Recordemos que ambas son tensores donde se indica solamente la pertenencia a la clase determinada, de manera que se divide por el número de píxeles de la imagen para normalizar los resultados.

Gráficas de *Loss* y Jaccard *Index*

Una vez hemos entrenado el modelo, guardamos las métricas de *loss* y Jaccard y las graficamos para poder determinar cómo ha funcionado el entrenamiento que hemos hecho. Como especificamos anteriormente, probamos con y sin división entre entrenamiento y validación. Para un entrenamiento de 100 épocas, hemos obtenido las siguientes gráficas:

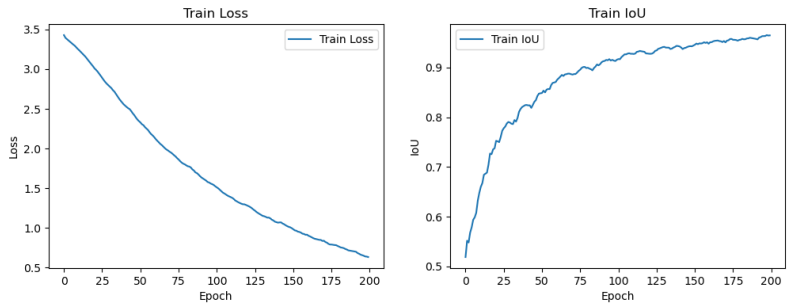

Ilustración 0-1 Gráficas sin división

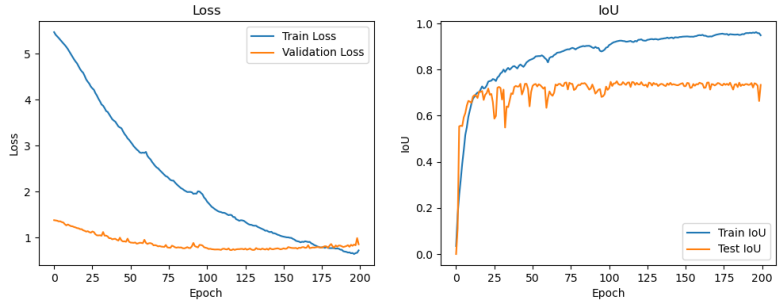

Ilustración 0-2 Gráficas con división

Podemos ver que en ambos casos conseguimos un IoU (o índice de Jaccard) cercano al 95% para el entrenamiento, pero que en el caso de la división, el índice se estanca justo debajo al 75% en menos de 20 épocas y va oscilando mientras el valor del entrenamiento sigue subiendo.

Esto se trata de un caso claro de sobreajuste, donde el modelo se aprende los datos de entrenamiento, pero llega a una cota superior de poder de generalización. Esto podría haberse evitado mediante la implantación de un sistema de *early stopping*, en el cual se pararía el entrenamiento si el conjunto de prueba no mejora más de un umbral definido después de un número de épocas que elijamos.

Como ejemplo, podemos imprimir por pantalla una imagen para ver cómo de bien es capaz de ajustar la máscara:

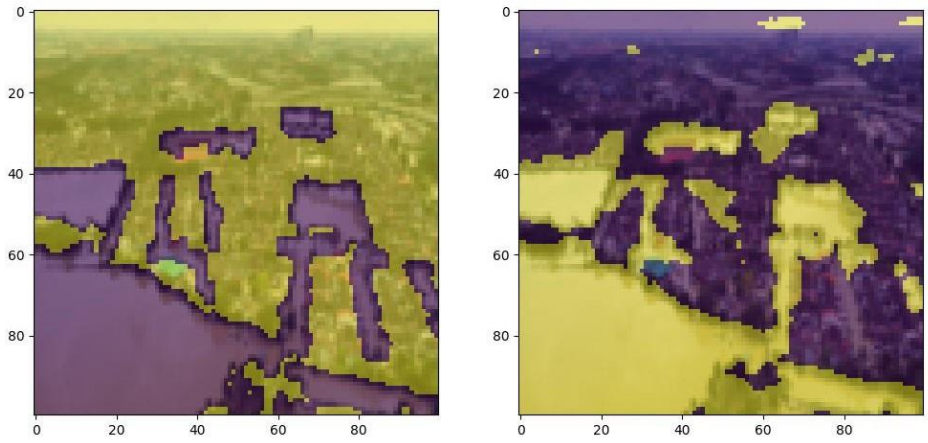

Ilustración 0-3 Máscara Original (izq.) vs Predicción (dcha.)

Conclusiones

En esta práctica hemos visto cómo emplear una arquitectura U-Net para poder realizar segmentación de imágenes, en este caso para diferenciar en imágenes agua de tierra y poder detectar inundaciones. De la misma manera hemos aprendido a procesar imágenes de manera correcta de cara a un algoritmo predefinido de *Computer Vision.* De esta manera, podríamos modificar la estructura interna de la red sin que esto implicase un cambio en los datos (salvo si cambiamos algo que tenga que ver con los tensores de entrada o de salida). Esto nos ha aportado una experiencia práctica en cómo se trabaja con datos reales, donde el procesado de datos te puede tanto solucionar como generar nuevos problemas, así como tener experiencia práctica en un caso de uso en el que las técnicas estudiadas en clase pueden ser aplicadas a un problema real.