Universidad Europea

Memoria Práctica 2

Martín de las Heras

Introducción

En esta práctica hemos aprendido los conceptos de *Transfer Learning* en *Computer Vision*, *Data Augmentation* y hemos realizado un modelo de clasificación completo con PyTorch. Todo esto se ha realizado bajo el objetivo de clasificar diversas imágenes de cáncer de piel en función de si tienen cáncer o no. La práctica se divide en los siguientes apartados:

- Leer el *dataset* de *Skin Cancer* y cargarlo como tensores.
- Realizar un *Data Augmentation* mediante ResizeCrop y HorizontalFlip.
- Usar el modelo predefinido y preentrenado de ResNet18 de PyTorch.
- Reemplazar la última capa por una capa Linear que tenga 2 clases a clasificar (benigno/maligno).
- Realizar un entrenamiento completo.
- Mostrar las gráficas de *Loss* y de *Accuracy*

Lectura del *dataset*

En primer lugar, leemos el *dataset* disponible en Canvas y lo guardamos como un array de tensores de cara a poder procesarlo. Las funciones que comienzan por *transform_* las veremos a continuación, ya que se encargan de la parte de *Data Augmentation*:

```
train_dir = 'skin_cancer/train'
test_dir = 'skin_cancer/test'
train_dataset = datasets.ImageFolder(train_dir, transforms_train)
test_dataset = datasets.ImageFolder(test_dir, transforms_test)
```

Data Augmentation

A continuación procedemos con el proceso de Data Augmentation. Para ello, definimos principalmente la función transforms_train, la cual mediante ResizeCrop y HorizontalFlip aumenta la cantidad de datos de entrenamiento del que será nuestro modelo. De la misma manera, redimensionamos las imágenes para que se puedan procesar por parte del modelo y las normalizamos:

```
transforms_train = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

ResNet18

A continuación, cargamos el modelo predefinido ResNet18 de PyTorch mediante la siguiente función:

```
model = torchvision.models.resnet18(pretrained=True)
```

Esto sin embargo tiene un problema. Debido a que dentro del nodo de nuestro grupo en LORCA no tenemos conexión a internet, fue necesario descargar el modelo preentrenado, pasarlo por *scp* al nodo y en dicho nodo cargarlo para poder entrenarlo. Esto se hace fácilmente con las siguientes líneas de código:

```
torch.save(model, 'resnet18')
model = torch.load('resnet18')
```

Reemplazar la última capa

La idea detrás del uso de este modelo es realizar *Transfer Learning*, es decir, aprovechar las capacidades adquiridas por este modelo durante su entrenamiento inicial, pero entrenando parte de él con nuestros datos para que pueda clasificar correctamente. Para ello, sustituimos la última capa (la capa densa que clasifica) por una capa que entrenaremos con nuestros datos. En primer lugar tenemos que definir los parámetros que vamos a necesitar que tenga dicha capa.

```
model.fc = torch.nn.Linear(num_features, 2)
```

En el código, podemos ver que recibe *num_features* como entrada y como salida tiene dos neuronas (benigno/maligno). El primer parámetro tiene que ser el mismo que tenía la capa anterior, ya que conecta con el resto del modelo, y la segunda va en función de nuestro problema, en nuestro caso 2.

Entrenamiento

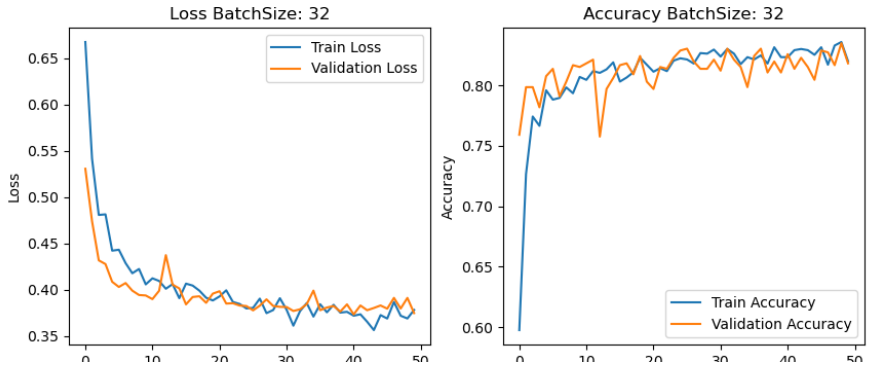
Por último entrenamos el modelo de manera normal, salvo que en los parámetros del optimizador, elegimos solamente esta última capa y el learning rate, codificado de la siguiente manera:

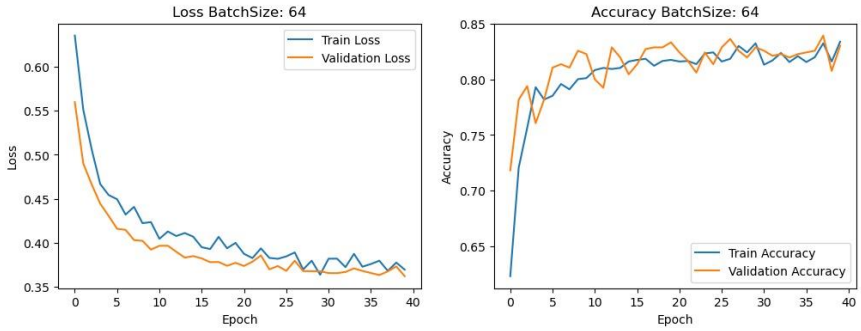
```
optimizer = torch.optim.Adadelta(model.fc.parameters(), lr=0.01)
```

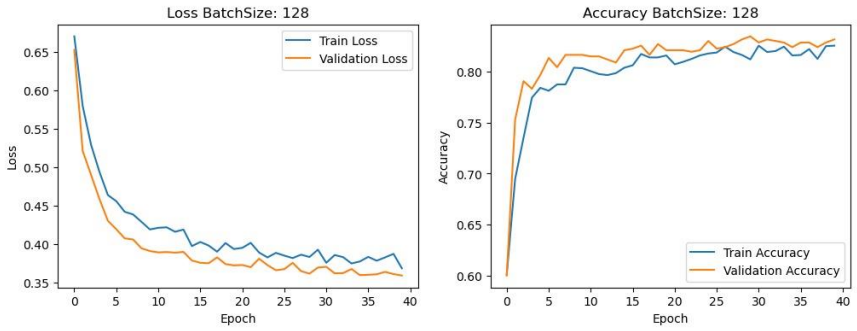
En esta línea, los parámetros a optimizar son los que se le pasan directamente al optimizador. El resto del entrenamiento se realiza de manera normal como hemos visto en clase.

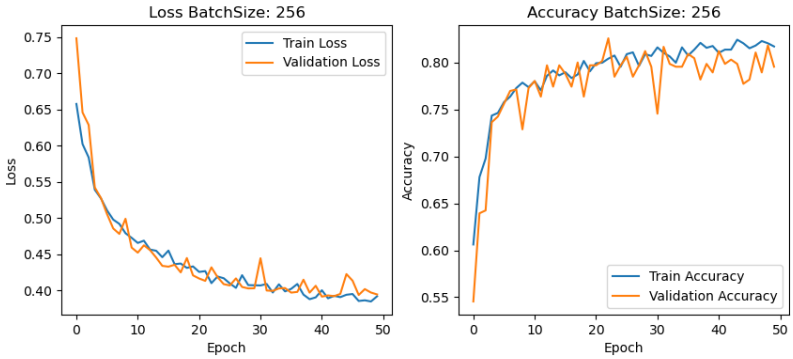
Gráficas de *Loss* y *Accuracy*

Una vez hemos entrenado el modelo, guardamos las métricas de *loss* y *accuracy* y las graficamos para poder determinar cómo ha funcionado el entrenamiento que hemos hecho. En nuestro caso hemos probado con diversas combinaciones de *batch size* y de épocas de entrenamiento, obteniendo los siguientes resultados:









Podemos ver que en el último caso (*batch size* 256), el *accuracy* es mucho más inestable, esto se puede deber a que, debido a que se trata de un *dataset* de unas 2000 imágenes, este tamaño de *batch* supone un porcentaje muy alto del total, por lo tanto es posible que esté cayendo en mínimos locales causados por los "saltos" grandes que tiene que dar el modelo. Esto por el contrario vemos que para los demás casos pasa en mucha menor medida.

Conclusiones

En esta práctica hemos visto cómo emplear *Transfer Learning* en un caso real de diagnóstico por imagen. De la misma manera hemos conseguido aumentar los datos mediante diversas técnicas, así como de manera transversal ejecutar modelos en un clúster. Esto nos ha aportado una experiencia práctica en cómo se trabaja en el mundo profesional, donde los datos son muy valiosos y escasos, por lo que conviene tener a mano este tipo de técnicas, en las que se reduce considerablemente el esfuerzo necesario, tanto computacional como técnico para resolver el problema en cuestión.