Memoria Técnica Práctica Deep Learning

Empezamos la práctica instalando el CUDA y las librerías necesarias para poder ejecutarlo en Visual Studio Code. (ver instalaciones en requirements.txt).

Una vez importamos todas las librerías que vamos a utilizar, creamos una semilla para poder reproducir los resultados y leemos el dataset.

Lo primero que hago es analizar el dataset y me doy cuenta que hay dos columnas que no son numéricas a pensar de ser números por lo que las transformo('categories', 'tags'). Observo que todas las columnas están rellenas pero no veo que sean relevantes las columnas 'id', 'name' y 'shortDescription' ya que se tratan de string y no son columnas que vaya a necesitar y hago un one-hot de la columna categories.

Defino una feature 'target' como: likes + bookmarks *2 + visits *0.5 - dislikes.

A su vez, clasifico los datos en la columna 'engagement_class' como 3 clases, el 33% del target más bajo como 0, el siguiente 33% como 1 y el ultimo 33% como 2, es decir, divido los datos en 3 grupos, bajo, medio, alto. Decido hacerlo así ya que me resulta más sencillo, pero podría haber definido en 5 grupos como se ven en algunos gráficos que muestro.

Una vez analizado y clasificado, divido los datos en train (80%), test (20%) y validación (20% de los datos de train).

Después de crear el Dataset POI y el transformador de imágenes, creo el modelo.

Recalco que esta memoria la estoy haciendo mucho después de hacer pruebas por lo que no tengo todo documentado.

Creo el modelo con tres capas convolucionales para las imágenes, 2 capas lineales para los metadatos y un clasificador con otras 2 capas lineales y creo una función para entrenar el modelo.

Al principio, no uso dropout ni batchnorm y tenía mucho overfitting.

Añado el dropout y el batchnorm y consigo un 81% en validación pero sigo probando manualmente y no documento los parámetros en ningun lado y no vuelvo a conseguirlo.

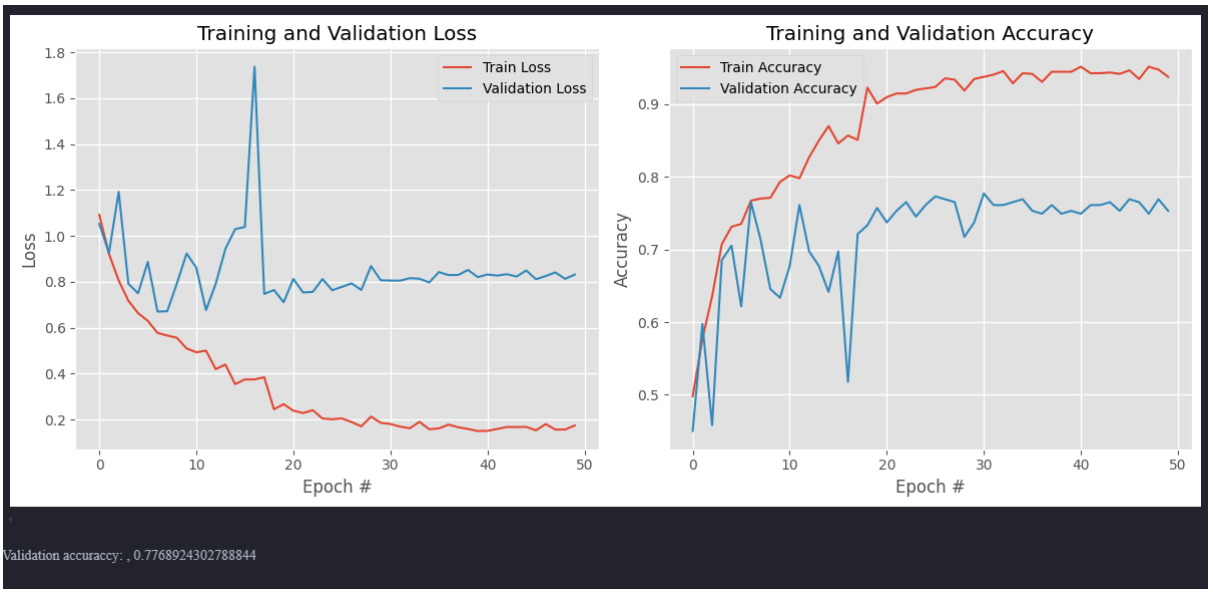
Tras muchas pruebas manuales con valores por debajo de 75% en validación, uso optuna para buscar hiperparámetros más rápido.

Con este mismo modelo consigo un 0.7888 en validación:

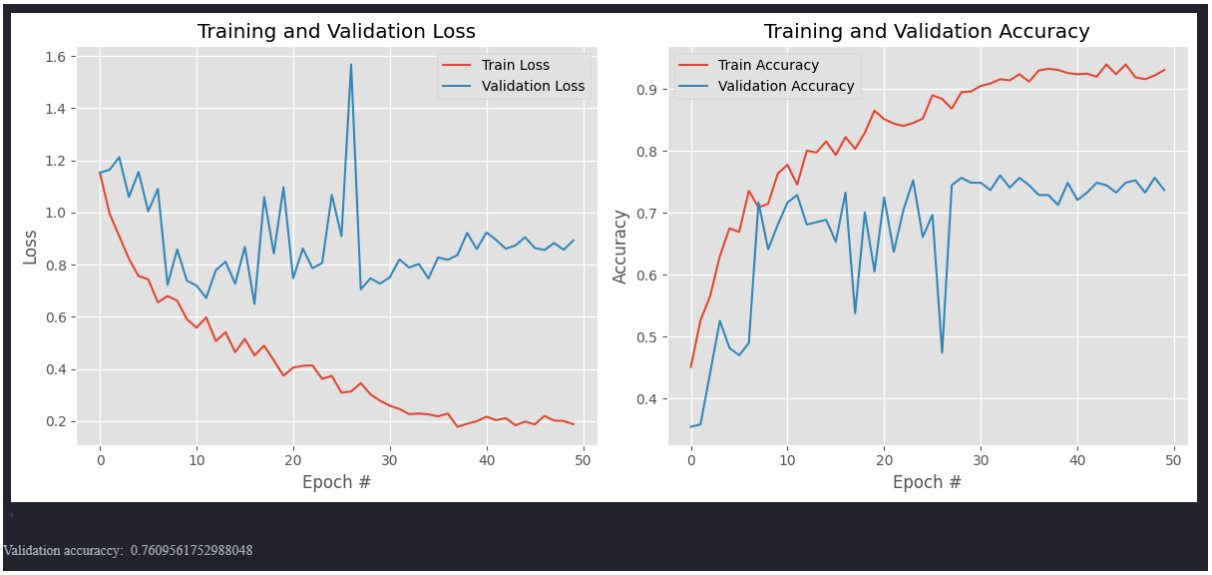
```python
self.cnn = nn.Sequential(
    nn.Conv2d(3, 16, kernel_size=3, padding=1),
    nn.BatchNorm2d(16),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.MaxPool2d(2),

    nn.Conv2d(16, 32, kernel_size=3, padding=1),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.MaxPool2d(2),

    nn.Conv2d(32, 64, kernel_size=3, padding=1),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.MaxPool2d(2),

)

self.cnn_output_dim = 64 * 16 * 16

self.metadata = nn.Sequential(
    nn.Linear(num_metadata_features, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.Linear(64, 64),
    nn.BatchNorm1d(64),
    nn.ReLU(),
)

self.classifier = nn.Sequential(
    nn.Linear(self.cnn_output_dim + 64, hidden_size),
    nn.BatchNorm1d(hidden_size),
    nn.ReLU(),
    nn.Dropout(dropout_rate),
    nn.Linear(hidden_size, num_classes)
)
```

```
--- Resultados de la optimización ---
Mejor trial: 0.7888 (Precisión de validación)
Mejores hiperparámetros:
  learning_rate: 0.003444938933182054
  batch_size: 16
  hidden_size: 128
  dropout_rate: 0.3691915136687273
  epochs: 50
```

Pruebo con todo igual y cambio las capas de activación SELU:



Hay overfitting por lo que subo el dropout de 0.37 a 0.6:



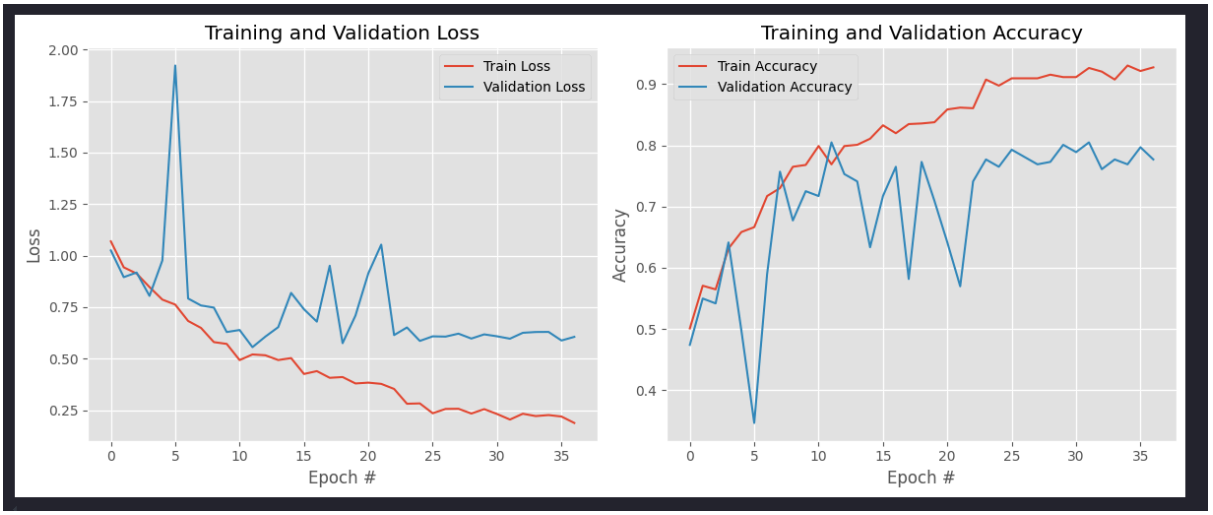
Sigue habiendo overfitting, entiendo que es porque el dataset no es suficientemente grande por lo que creo un transformador de imagenes y añado imagenes como en Redes_Convolucionales_II:

```python
image_transforms_cv2_train = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize((128, 128), antialias=True), #Añado esta linea como segu
    transforms.RandomHorizontalFlip(p=0.5),
    transforms.RandomRotation(10),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])
```

```python
def train_model(model, learning_rate, epochs, batch_size, trial=None):

    train_dataset = POI(train_targets, train_paths, train_features, transform=image_transforms_cv2_train)
    val_dataset = POI(val_targets, val_paths, val_features, transform=image_transforms_cv2_train)

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, worker_init_fn=seed_worker, generator=g)
    val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, worker_init_fn=seed_worker, generator=g)

    optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-5)
    criterion = nn.CrossEntropyLoss()

    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', factor=0.1)
```

Ejecuto optuna y el mejor resultado en 30 intentos diferentes es este:

```
--- Resultados de la optimización ---
Mejor trial: 0.8127 (Precisión de validación)
Mejores hiperparámetros:
  learning_rate: 0.0011744876025409567
  batch_size: 16
  hidden_size: 512
  dropout_rate: 0.2582685099719334
  epochs: 37
```



Con estos resultado pruebo en test: