

Organización de la Práctica:

1- Procesamiento de datos

Revisar datos, hacer las comprobaciones oportunas para dejar los datos listos para entrar en la red.

- Carga de csv
- Paso a tipos nativos de Python porque los datos estas como objetos
- Comprobación de nulos y Na
- Codificación de variables categóricas para poder usarlas en la red
- Generación de la variable de engagement: En este caso empecé por una fórmula

inicial sencilla del tipo `df['target'] = (df['Likes']*df['Bookmarks']*df['Bookmarks'] - df['Dislikes'])/df['Visits']`

Es muy importante definir bien esta métrica para ser efectivos. Decidí darle más peso a los guardados en favoritos elevando al cuadrado y dividiendo entre las visitas para relativizar los datos. Al final me decidí por esta fórmula:

```
df['target'] = ((df['Likes']*df['Bookmarks']*df['Bookmarks'] - df['Dislikes'])/df['Visits'])*(1/df['Tier'])
```

Como el tier está en número enteros y supongo que el número 1 es de más nivel el 4 el que menos, es lógico pensar que va a generar más engagement puntos de interés de un alto nivel, por tanto al multiplicar el por 1/ Tier nos garantizamos que pesen más las de tier alto que las de bajo. No es lo mismo la torre de Pizza que una escultura en un parque.

- Dividir la variable objetivo en 3 grupos para que se una problema de clasificación de engagement: bajo -medio-alto

2- Analisis de imagenes

Realizó un análisis exploratorio de las imágenes, viendo los tamaños, cuantas hay los canales a tratar, una exploración para saber que tenemos.

Al final no los puedo procesar como los metadatos, es la red la que se encarga de sacar las características y de optimizar los pesos para su procesamiento.

3- Dataset

Creo un dataset personalizado que le diga al modelo cuántos elementos hay y cómo acceder a ellos.

- Dividir los datos en dos grupos metadatos e imágenes y en train, validación y test
- Escaló los datos cada uno en función de su división
- Definir las transformaciones aplicar a la clase para que se hagan las transformaciones según se instancie

4- Modelo

Hago una primera red sencilla para ir mejorando. POIClassifier que hereda de nn.Module:

- Capas convolucionales con un bloque más pequeño de 32 para las características más genéricas a las que luego vamos doblando los filtros para extracción de características y patrones más específicos.
- Max Pool para reducir la resolución a la mitad que nos deja un total de $128 \times 16 \times 16$ parámetros
- Metadatos: Capas lineales y funcion de activacion relu
- Clasificador Concateno justo antes del clasificador las imágenes con los metadatos para que tenga en cuenta ambas fuentes de información. Última salida con 3 clases para clasificación en función del engagement.

Esta estructura es eficiente para los datos que tenemos que son muy pocos, tanto de imágenes como por metadatos.

```
Epoch 1, Loss: 1.1218255096011691, Acc: 37.64813126709207, Val Loss: 1.079649657011032, Val Acc: 31.779661016949152
Epoch 2, Loss: 1.0307245420085058, Acc: 49.22515952597995, Val Loss: 0.9326535016298294, Val Acc: 62.28813559322034
Epoch 3, Loss: 0.898385634024938, Acc: 61.71376481312671, Val Loss: 0.8213520050048828, Val Acc: 67.79661016949153
Epoch 4, Loss: 0.7536472280820211, Acc: 69.73564266180492, Val Loss: 0.7599957138299942, Val Acc: 68.64406779661017
Epoch 5, Loss: 0.7152528133657243, Acc: 68.64175022789426, Val Loss: 0.7285967618227005, Val Acc: 68.22033898305085
Epoch 6, Loss: 0.69756304886606, Acc: 70.00911577028259, Val Loss: 0.7302047312259674, Val Acc: 68.22033898305085
Epoch 7, Loss: 0.6849264767434862, Acc: 72.01458523245215, Val Loss: 0.692778080701828, Val Acc: 69.91525423728814
Epoch 8, Loss: 0.656890932056639, Acc: 72.47037374658159, Val Loss: 0.7293376177549362, Val Acc: 68.22033898305085
Epoch 9, Loss: 0.6352980815702014, Acc: 71.83226982680037, Val Loss: 0.7245961725711823, Val Acc: 68.22033898305085
Epoch 10, Loss: 0.6174286239677005, Acc: 72.92616226071102, Val Loss: 0.7353823035955429, Val Acc: 68.22033898305085
```

Test accuracy: 67.373



5- Modelo pre-entrenado con ResNet: A Diferencia con el modelo anterior:

- Al estar entrenada con 11 millones de imágenes las capas que tiene pueden detectar patrones más complejos que en el modelo anterior ya que tiene muchos menos datos y solo 3 capas.
- Hay transfer Learning → aprovechó el conocimientos previo de ResNet
- Se puede entrenar las capas que tu quieras, o solo el clasificador y la última capa.

La verdad que accuracy no sube mucho con respecto a el clasificador sencillo, pero es capaz de extraer información de mucha más calidad.

Es curioso porque me dan peores datos con Resnet que con mi red. Esto se debe a que la red está entrenada con sus propias imágenes y quizás los filtros pre entrenados esperan ciertos tipo de detecciones (texturas, colores, patrones) que quizás por calidad o lo que sea nuestra imágenes no tienen. También puede ser por el tamaño del batch, ya que en otras pruebas me ha dado algo mejor .

(aquí ha pasado que de últimas lo compile sin coger los datos y por estar tocando a última hora es lo que puedo presentar)

```
Epoch 1, Train Loss: 0.6433, Train Acc: 69.0064, Val Loss: 1.2502, Val Acc: 61.0169, LR: 0.010000
Epoch 2, Train Loss: 0.5825, Train Acc: 73.9289, Val Loss: 0.7158, Val Acc: 65.2542, LR: 0.010000
Epoch 3, Train Loss: 0.5947, Train Acc: 72.0146, Val Loss: 0.6868, Val Acc: 70.3390, LR: 0.010000
Epoch 4, Train Loss: 0.5610, Train Acc: 73.3820, Val Loss: 0.6835, Val Acc: 71.6102, LR: 0.010000
```

