Práctica 30 de enero de 2024

1. **Refresca memoria de lo que hicimos la clase anterior**
2. **Este es el último ejercicio que os pedí**

```
import random
fig, axs = plt.subplots(6,2, figsize=(16,32))
count = 0
for x in range(6):
      i = random.randint(0, len(brain_df)) # Seleccionamos un índice aleatorio
      axs[count][0].title.set_text("MRI del Cerebro") # Configuramos el título
      axs[count][0].imshow(cv2.imread(brain_df.image_path[i])) # Mostramos la MRI
      axs[count][1].title.set_text("Máscara - " + str(brain_df['mask'][i])) # Colocámos el título en la máscara (0 o 1)
      axs[count][1].imshow(cv2.imread(brain_df.mask_path[i])) # Mostramos la máscara correspondiente
      count += 1

fig.show()
```

3. **Vamos a probar este código. ¿Qué hace?**

```
from skimage import io

i=445
img = io.imread(brain_df.image_path[i])
img2 = cv2.imread(brain_df.image_path[i])

mask = io.imread(brain_df.mask_path[i])
mask2 = cv2.imread(brain_df.mask_path[i])

mask2_gray = cv2.cvtColor(mask2, cv2.COLOR_BGR2GRAY)



img[mask2_gray == 255] = (255, 0, 0)  # Cambia el color de los píxeles en la imagen donde la máscara es 255

plt.imshow(img)
```

4. **[TAREA]Representar aleatoriamente 12 (imágenes de MRI seleccionadas aleatoriamente de entre los pacientes enfermos seguidas de su correspondiente máscara, tanto la imagen de la MRI junto con su máscara (de color rojo) una encima de la otra.**
5. **[TAREA] Entender la teoría y la intuición detrás de las redes neuronales y resnets convolucionales. Mira la teoría y resúmelo con tus palabras.**
6. **[TAREA] Entender la teoría y la intuición detrás del aprendizaje por transferencia. Mira la teoría y resúmelo con tus palabras.**

    - Excelente recurso sobre transferencia de aprendizaje por Dipanjan Sarkar: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a
    - Artículo de Jason Brownlee: https://machinelearningmastery.com/transfer-learning-for-deep-learning/

7. **[TAREA]Enumere los desafíos del aprendizaje por transferencia (se requiere investigación externa)**
8. Eliminamos la columna de id del paciente

```
# Eliminamos la columna de identificador del paciente
brain_df_train = brain_df.drop(columns = ['patient_id'])
brain_df_train.shape
```

9. Spiler. Convertir los datos en la columna de máscara a formato de string, para usar el modo categórico en flow_from_dataframe (Podemos no hacerlo, a ver si os da error luego. OJO! Tenedlo en cuenta)
10. **Dividimos datos train, test 0.15**
11. **Instalar todas las funcionalidades de Keras Preprocessing. Investiga y echa un vistazo…**

```
!pip install Keras-Preprocessing
```

12. **Importamos datagenerator**
13. **Vale, Creamos un generador de datos que escale los datos de 0 a 1 y haga una división de validación de 0,15.**

```
datagen = ImageDataGenerator(rescale=1./255., validation split = 0.15)

train_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x_col='image_path',
y_col='mask',
subset="training",
batch size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))



valid_generator=datagen.flow_from_dataframe(
dataframe=train,
directory= './',
x col='image path',
y_col='mask',
subset="validation",
batch_size=16,
shuffle=True,
class_mode="categorical",
target_size=(256,256))

# Creamos un generador de datos para imágenes de prueba
test_datagen=ImageDataGenerator(rescale=1./255.)

test_generator=test_datagen.flow_from_dataframe(
dataframe=test,
directory= './',
x_col='image_path',
y_col='mask',
batch size=16,
shuffle=False,
class_mode='categorical',
target_size=(256,256))
```

14. **Obtenemos el modelo de base. Haz summary**

```
basemodel = ResNet50(weights = 'imagenet', include_top = False, input_tensor = Input(shape=(256, 256, 3)))
```

15. **Congelamos los pesos. ¿Por qué es necesario?**

```
16.   for layer in basemodel.layers:
17.     layer.trainable = False
```

18. **Agregamos la cabecera de clasificación**

```
headmodel = basemodel.output
headmodel = AveragePooling2D(pool_size = (4,4))(headmodel)
headmodel = Flatten(name= 'flatten')(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
headmodel = Dense(256, activation = "relu")(headmodel)
headmodel = Dropout(0.3)(headmodel)
#headmodel = Dense(256, activation = "relu")(headmodel)
#headmodel = Dropout(0.3)(headmodel)
```

```
headmodel = Dense(2, activation = 'softmax')(headmodel)

model = Model(inputs = basemodel.input, outputs = headmodel)
```

19. **Compilamos el modelo**
20. **Utilizamos parada temprana**
21. **Guardamos el modelo con la menor pérdida**
22. **Hacemos el fit**

```
history = model.fit(train_generator, steps_per_epoch= train_generator.n
// 16,
                    epochs = 1, validation_data= valid_generator,
                    validation_steps= valid_generator.n // 16,
                    callbacks=[checkpointer, earlystopping])
```

23. **Guarda el archivo para un futuro**
24. **[RETO]** Cambia la arquitectura de la red agregando capas, neuronas o Dropouts más / menos densos.

Imprime el resumen del modelo y compara el número total de parámetros entrenables entre el modelo original y el nuevo

25. **Vamos a evaluar el rendimiento. Cargando los datos**
26. **Hacemos la predicción**
27. **Obtenemos tasa de acierto**
28. **Hacemos la matriz de confusión**
29. **[RETO]** Imprime el informe de clasificación y comenta sobre los resultados de precisión, recuperación y F1-Score