Proyecto programación de Joan Olivan y Marc Badosa

Dataset: https://www.kaggle.com/c/histopathologic-cancer-detection/data

Test: 57.500 imagenes Train: 220.000 imagenes

Descripción del dataset:

El "dataset" consta de un gran número de imágenes con patologías para clasificar. Más específicamente consta de 220.000 imágenes para el conjunto de train y de 57.500 imágenes para el conjunto de test.

El nombre de las imágenes corresponde con su identificador especificado en el fichero de "train_labes.csv", con el cual, podemos saber la clase real de cada imagen del conjunto de Train. Las clases pueden ser, 0 o 1, en caso de que la imagen sea positiva (1) quiere decir que en la región central, de 32x32 píxeles, podemos encontrar, como mínimo, un pixel de tejido cancerígeno.

El tejido cancerígeno fuera de esta región central, no influye en la clasificación de la imagen. Esta región exterior está incluida para permitir la creación de modelos convolucionales completos, los cuales no usan zero-padding, para conseguir así un comportamiento estable cuando dicho modelo se aplica a una imagen completa.

Comentario importante

Para agilizar la corrección de esta actividad, todas las pruebas extras fuera del modelo con mejores resultado, serán añadidas al final de la actividad, en el último apartado llamado "Anexo".

Todos los modelos que se pueden encontrar en este documento, han sido entrenados durante 30 épocas, unas 5-6 horas por modelo. El entrenamiento ha sido realmente complicado debido a las limitaciones de la versión pro de Google colab. Por esto decidimos obtener la versión "pro" de Google colab, para poder trabajar de una manera más eficiente y cómoda, aparte esta versión pro, nos va a ser útil a ambos para la realización del TFM y de posteriores asignaturas.

Una vez terminada la práctica, nos dimos cuenta de que para la entrega se tenía que ver los outputs del entrenamiento de las redes neuronales principales, por lo que las volvimos a ejecutar, lo cual ha podido dar resultados con pequeñas diferencias debido a las inicializaciones aleatorias de los parámetros.

Enunciado

En esta actividad, el alumno debe evaluar y comparar dos estrategias para la clasificación de imágenes empleando el dataset asignado. El/La alumnx deberá resolver el reto proponiendo una solución válida basada en aprendizaje profundo, más concretamente en redes neuronales convolucionales (CNNs). Será indispensable que la solución propuesta siga el pipeline visto en clase para resolver este tipo de tareas de inteligencia artificial:

1. **Carga del conjunto de datos**
2. **Inspección del conjunto de datos**
3. **Acondicionamiento del conjunto de datos**
4. **Desarrollo de la arquitectura de red neuronal y entrenamiento de la solución**
5. **Monitorización del proceso de entrenamiento para la toma de decisiones**
6. **Evaluación del modelo predictivo y planteamiento de la siguiente prueba experimental**

La realización de esta actividad consta de dos grandes bloques, la estrategia 1 y la estrategia 2. Como ambas van a hacer uso del mismo "dataset", podemos agrupar los tres primeros puntos, de los seis totales que se deben realizar en cada estrategia, ya que al utilizar el mismo "dataset", sería redundante hacer los tres primeros pasos que son genéricos, en los dos apartados.

Primeramente, antes de empezar ningún apartado, vamos a definir una celda de código, donde concentrar todas las importaciones de librerías externas que vamos a utilizar para realizar la actividad:

In [ ]:

```
# !pip install --upgrade --force-reinstall --no-deps kaggle
import matplotlib.pyplot as plt
import numpy as np
import cv2, os, keras, shutil, glob
from PIL import Image
import tensorflow as tf
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense, Dropout,
BatchNormalization, MaxPooling2D
from tensorflow.keras.optimizers import Adam
import pandas as pd
import matplotlib.patches as patches
from sklearn.utils import shuffle
from google.colab import drive
from sklearn.metrics import classification_report
from tensorflow.keras.models import Model
from tensorflow.keras import backend as K
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG19, Xception
from tensorflow.keras.preprocessing import image
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau, ModelCheckpoint
from google.colab import drive
import pickle
from keras.models import load_model
import h5py
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
drive.mount('/content/drive')
BASE_FOLDER = "/content/drive/MyDrive/07MIAR_Proyecto_Programacion/"
```

```
Mounted at /content/drive
```

Ahora, una vez importadas todas las librerías externas, ya podemos empezar con los puntos del enunciado, que como hemos comentado en el bloque anterior, vamos a realizar solo una vez, ya que sería redundante realizarlo dos veces para cada uno de las estrategias, al estas, compartir el mismo "dataset".

1. Carga del conjunto de datos

In [ ]:

```
# Importamos de kaggle el dataset para realizar la actividad
from google.colab import files
files.upload()
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle competitions download -c histopathologic-cancer-detection

!mkdir histopathologic_cancer_detection_dataset
!unzip histopathologic-cancer-detection.zip -d histopathologic_cancer_detection_dataset
!unzip histopathologic_cancer_detection_dataset/test.zip

# El output de este bloque ha sido eliminado debido a que si no el pdf ocupaba unas 180 p
áginas y no aportaba ningún valor a la práctica al ser solo descargas de ficheros
```

Definimos los "paths" en los que vamos a ordenar las imágenes de nuestro "dataset", vamos a diferenciar tres directorios principales, "train", "validation" y "test". El directorio de "test" no va a ser utilizado, ya que no disponemos de las etiquetas de clase reales de dichas imágenes, por lo cual no podemos verificar que el modelo realice las predicciones correctamente.

Del directorio de "train", vamos a separar un 20%, aplicando una partición de tipo "Hold-out", de imágenes para crear el conjunto de validación, el cual vamos a utilizar para validar las predicciones de nuestros modelos. Este conjunto también va a ser utilizado para realizar las predicciones finales, de donde vamos a extraer las gráficas

para analizar los modelos.

Como se puede observar en el código a continuación, dentro de los directorios de "Train" y "Validation", vamos a generar dos carpetas, una para cada clase: "0_no_tumor" y "1_tumor", para poder ordenar nuestras imágenes en función de su clase, para a después, realizar el entrenamiento sin necesidad de cargar todas las imágenes en memoria RAM.

In [ ]:

```
# Directorio base
base_images_path = './histopathologic_cancer_detection_dataset/'

# Directorio para imágenes de test
test_images_path = base_images_path + 'test/'

# Directorio para imágenes de train
train_images_path = base_images_path + 'train/'
os.mkdir(train_images_path + '0_no_tumor')
os.mkdir(train_images_path + '1_tumor')

# Directorio para imágenes de train
validation_images_path = base_images_path + 'validation/'
os.mkdir(validation_images_path)
os.mkdir(validation_images_path + '0_no_tumor')
os.mkdir(validation_images_path + '1_tumor')

# Importamos también el fichero .csv que nos indica la clase a la que pertenecen nuestras imágenes del dataset, lo importamos en forma de dataframe de la librería de pandas
train_images_label_dataframe = pd.read_csv(base_images_path + 'train_labels.csv')

# Haciendo una búsqueda por internet y a posteriori comprobándolo, vimos que estas dos imágenes no eran correctas, por lo que decidimos eliminarlas al tener ya una importante cantidad de imágenes disponibles
train_images_label_dataframe[train_images_label_dataframe['id'] != 'dd6dfed324f9fcb6f93f46f32fc800f2ec196be2']
train_images_label_dataframe[train_images_label_dataframe['id'] != '9369c7278ec8bcc6c880d99194de09fc2bd4efbe']
```

Out[ ]:

	id	label
0	f38a6374c348f90b587e046aac6079959adf3835	0
1	c18f2d887b7ae4f6742ee445113fa1aef383ed77	1
2	755db6279dae599ebb4d39a9123cce439965282d	0
3	bc3f0c64fb968ff4a8bd33af6971ecae77c75e08	0
4	068aba587a4950175d04c680d38943fd488d6a9d	0
...	...	...
220020	53e9aa9d46e720bf3c6a7528d1fca3ba6e2e49f6	0
220021	d4b854fe38b07fe2831ad73892b3cec877689576	1
220022	3d046cead1a2a5cbe00b2b4847cfb7ba7cf5fe75	0
220023	f129691c13433f66e1e0671ff1fe80944816f5a2	0
220024	a81f84895ddcd522302ddf34be02eb1b3e5af1cb	1

220024 rows × 2 columns

2. Inspección del conjunto de datos

In [ ]:

```
print("Total Imagenes conjunto de train: " + str(len(os.listdir(train_images_path))))
print("Total Imagenes conjunto de test: " + str(len(os.listdir(test_images_path))))
```

```
Total Imagenes conjunto de train: 220027
```

```
Total Imagenes conjunto de test: 57458
```

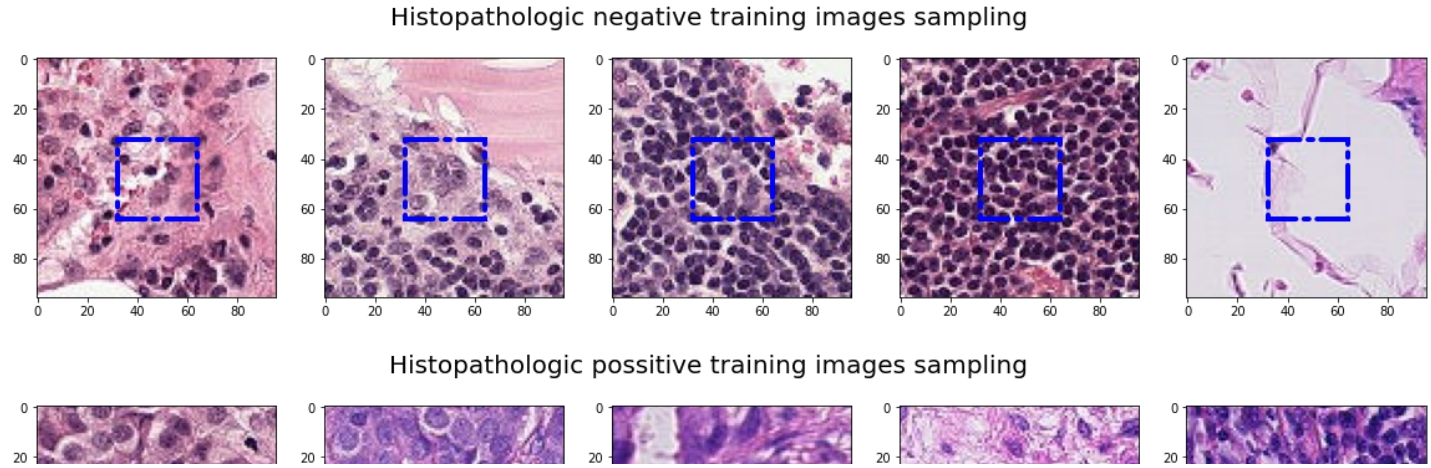
Como podemos observar, disponemos de, aproximadamente, 220000 imágenes en el conjunto de "train", pero para balancear las clases, solo vamos a usar 170.000 imágenes. Recordemos que el 20% lo vamos a mover a otro directorio para utilizarlo como validación, para comprobar las predicciones de nuestras redes entrenadas.

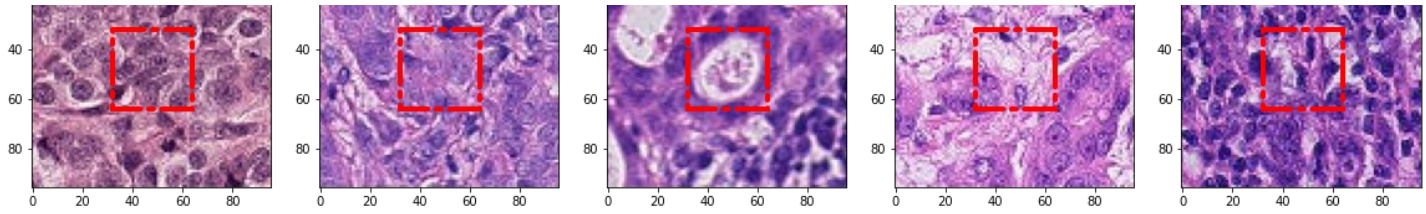
Aproximadamente disponemos de 57.500 imágenes en el conjunto de "test", las cuales no vamos a utilizar debido a que no tenemos la información sobre a que clase real pertenecen.

Vamos a realizar dos funciones, para poder leer imágenes desde la carpeta y poder visualizarlas, para hacer una inspección visual de las imágenes con las que vamos a tratar durante este ejercicio.

In [ ]:

```
# Función para leer una imagen a partir de su directorio.
def read_rgb_image_from_path(path):
    bgr_image = cv2.imread(path)
    b,g,r = cv2.split(bgr_image)
    rgb_image = cv2.merge([r,g,b])
    return rgb_image

# Función para mostrar visualmente imágenes de nuestro dataset, dado un título, un direct
orio, unas etiquetas aleatorias de las imágenes, la clase a la que perteneces y el color
del cuadrado de la región central
def sample_random_train_images(title, path, shuffled_labels, class_index, edge_color):
  fig, ax = plt.subplots(1,5, figsize=(20,4))
  fig.suptitle(title,fontsize=20)
  # Imágenes sin píxeles cancerígenos en la región centro
  for index, image in enumerate(shuffled_labels[shuffled_labels['label'] == class_index]
['id'][:5]):
    path = os.path.join(train_images_path, image)
    ax[index].imshow(read_rgb_image_from_path(path + '.tif'))
    # Marcamos la zona centro donde debemos detectar píxeles con tejido cancerígenos
    box = patches.Rectangle((32,32),32,32,linewidth=4,edgecolor=edge_color, facecolor='n
one', linestyle='-.', capstyle='round')
    ax[index].add_patch(box)

# Vamos a elegir 85.000 imágenes con tumor y 85.000 sin tumor
dataframe_no_tumor = train_images_label_dataframe[train_images_label_dataframe['label'] =
= 0].sample(85000, random_state = 42)
dataframe_tumor = train_images_label_dataframe[train_images_label_dataframe['label'] == 1
].sample(85000, random_state = 42)
train_images_label_dataframe = pd.concat([dataframe_no_tumor, dataframe_tumor], axis=0).
reset_index(drop=True)

train_images_label_dataframe['label'].value_counts()

# Vamos a mezclar las labels de las imágenes para mostrar aleatoriamente imágenes.
shuffled_train_images_label_dataframe = shuffle(train_images_label_dataframe)

sample_random_train_images('Histopathologic negative training images sampling',train_imag
es_path,shuffled_train_images_label_dataframe,0,'b')
sample_random_train_images('Histopathologic possitive training images sampling',train_ima
ges_path,shuffled_train_images_label_dataframe,1, 'r')
```





Explorando el fichero de "train_labels.csv", el cual nos va a indicar si la imagen consta de una región central que contiene píxeles cancerígenos o no, podemos observar que consta de 130.908 etiquetas negativas (0_no_tumor) y de 89.117 etiquetas positivas (1_tumor), por lo que sabemos que tenemos una proporción de 59,5% de imágenes sin tumor y 40,5% imágenes con tumor.

Vemos que el "dataset" no está muy desbalanceado, pero podría estar mejor balanceado, la proporción ideal sería de 50% - 50%, se podría conseguir esta proporción, por ejemplo entrenado una red "GAN" que nos permita crear imágenes de una clase en concreto, similares a las que ya tenemos.

También existen otros métodos para "data augmentation" con imágenes, como el "flipping", "cropping", "rotation", etc. Pero para este "dataset" en concreto, donde la información más importante se encuentra en la región central de las imágenes, no todos serian correctas, ya que con algunos métodos, como el Zoom, o la translación, se podría desplazar la región central de la imagen, pudiendo producir así un peor rendimiento de la red a la hora de clasificar las imágenes.

Así que para entrenar esta red neuronal, de la mejor manera vamos, a hacer uso de la función "ImageDataGenerator" de Keras, para aplicar las transformaciones geométricas, que no alteran una región central de las imágenes del "dataset".

Como hemos dicho, el "dataset" está desbalanceado, por lo que vamos a prescindir de algunas imágenes y vamos a crear un "dataset" con menos imágenes pero con una proporción entre clases del 50% 50%.

Observando las imágenes podemos describir sus características técnicas más a fondo:

- **Formato: .tif**
- **Tamaño: 96x96**
- **Canales: 3**
- **Bits por canal: 8**
- **Tipo de datos: Unsigned char**

3. Acondicionamiento del conjunto de datos

Vamos a ordenar, en los directorios que hemos creado anteriormente, los conjuntos de datos para poder comenzar con el entrenamiento de nuestras redes neuronales. Como hemos dicho antes, solo vamos a hacer uso del conjunto de "train", del cual, vamos a utilizar un 20% para validar el modelo.

Para generar esta partición nos vamos a ayudar de la función de "train_test_split" de la librería de sklearn, la cual nos permite realizar una partición de tipo "Hold-out" de nuestro conjunto de datos. A esta función le vamos a especificar el parámetro de "random_state" para que la semilla no cambie, pudiendo así reproducir la misma partición siempre que queramos.

Como hemos visto antes, el fichero .csv donde se encuentran las etiquetas de clase de nuestras imágenes, contiene dos columnas. La primera que nos indica el nombre de la imagen, y la segunda que nos indica a que clase pertenece (0 -> no tumor, 1 -> tumor)

In [ ]:

```
# Balanceamos las clases para el conjunto de validación
balancing_factor = shuffled_train_images_label_dataframe['label']

# utilizamos la función train_test_split con los parámetros de test_size = 0.2 (20%) y ra
ndom_state = 42
images_dataframe_train, images_dataframe_validation = train_test_split(train_images_label
_dataframe, test_size=0.2, random_state=42,stratify=balancing_factor)

print(images_dataframe_train.shape)
```

```
print(images_dataframe_validation.shape)

print(images_dataframe_train['label'].value_counts())
print(images_dataframe_validation['label'].value_counts())
```

```
(136000, 2)
(34000, 2)
1    68005
0    67995
Name: label, dtype: int64
0    17005
1    16995
Name: label, dtype: int64
```

Ahora que tenemos los identificadores (nombres) y la clase de las imágenes del conjunto de "validation" y "train". Vamos a ordenar, en los directorios previamente creados, las imágenes de ambos conjuntos, "train" y "validation".

In [ ]:

```
# Asignamos el índice del datafram a la columna de id
train_images_label_dataframe.set_index('id', inplace=True)

# Pasamos a listas los id del dataframe para poder iterar sobre ellas
train_image_list_id = list(images_dataframe_train['id'])
validation_image_list = list(images_dataframe_validation['id'])

for dataset_image in train_image_list_id:
    # El csv no incluye la extensión de la imagen
    fname = dataset_image + '.tif'
    # Nos guardamos el nombre de la imagen que queremos mover
    target = train_images_label_dataframe.loc[dataset_image,'label']

    # Cambiamos la clase de la imagen (0 | 1) por el nombre de las carpetas que hemos cre
ado antes
    if target == 0:
        label = '0_no_tumor'
    if target == 1:
        label = '1_tumor'

    # Path de origen de las imágenes
    src = os.path.join(train_images_path, fname)
    # Path destino de las imágenes
    dst = os.path.join(train_images_path, label, fname)
    # Copiamos las imágenes
    shutil.copyfile(src, dst)

# Ahora hacemos lo mismo para las imágenes de validación
for dataset_image in validation_image_list:
    # El csv no incluye la extensión de la imagen
    fname = dataset_image + '.tif'
    # Nos guardamos la label de la imagen que queremos mover
    target = train_images_label_dataframe.loc[dataset_image,'label']

    # Cambiamos la clase de la imagen (0 | 1) por el nombre de las carpetas que hemos cre
ado antes
    if target == 0:
        label = '0_no_tumor'
    if target == 1:
        label = '1_tumor'

    # Path de origen de las imágenes
    src = os.path.join(train_images_path, fname)
    # Path destino de las imágenes
    dst = os.path.join(validation_images_path, label, fname)
    # Copiamos las imágenes
    shutil.copyfile(src, dst)

# Una vez tenemos todas las imágenes en sus correspondientes carpetas, podemos ver la can
tidad de cada una de ellas de las que disponemos:
```

```
print("Imagenes Train sin tumor: " + str(len(os.listdir(train_images_path + '/0_no_tumor
'))))
print("Imagenes Train sin tumor: " + str(len(os.listdir(train_images_path + '/1_tumor'))
))
print("Imagenes Validation sin tumor: " + str(len(os.listdir(validation_images_path + '/
0_no_tumor'))))
print("Imagenes Validation sin tumor: " + str(len(os.listdir(validation_images_path + '/
1_tumor'))))
```

```
Imagenes Train sin tumor: 67995
Imagenes Train sin tumor: 68005
Imagenes Validation sin tumor: 17005
Imagenes Validation sin tumor: 16995
```

Una vez separadas las imágenes por propósito (Train y Validation) y clase (Tumor, No Tumor), vamos a inicializar las variables de entrenamiento comunes para el entrenamiento de nuestras redes neuronales. Así como también vamos a inicializar los generadores de imágenes, que como hemos indicado antes, vamos a utilizar la función de "ImageDataGenerator" de la librería de sklearn, que nos permite aplicar técnicas de "data augmentation", para conseguir más datos y un mejor resultado final.

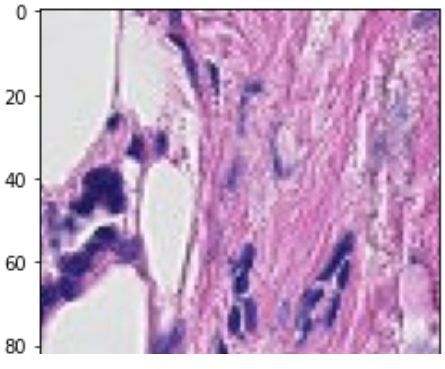
In [ ]:

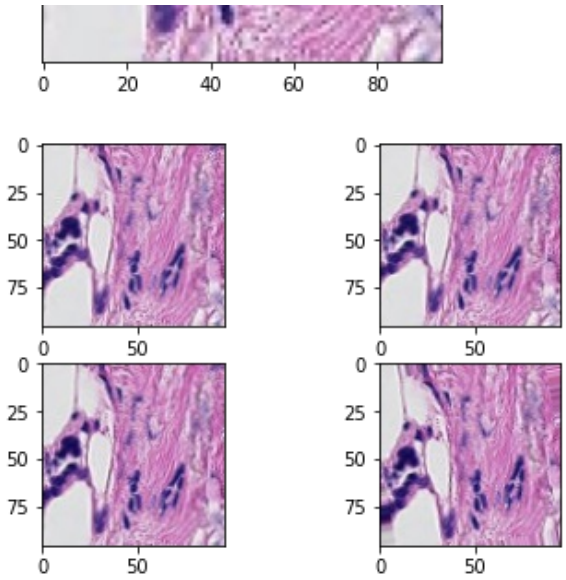
```
amount_images_validation = len(images_dataframe_validation)
amount_images_train = len(images_dataframe_train)
validation_batch_size = 16
train_batch_size = 16

amount_validation_steps = np.ceil(amount_images_validation / validation_batch_size)
amount_train_steps = np.ceil(amount_images_train / train_batch_size)

images_data_generation = ImageDataGenerator(
    rotation_range=15,
    horizontal_flip=True,
    channel_shift_range=0.2
)
```

Para comprobar que realmente la función de "data augmentation" está funcionado correctamente como queremos, vamos a visualizar una de las imágenes modificadas que nos devuelve dicha función.

In [ ]:

```
random_sample = 22

sample_to_plot = read_rgb_image_from_path( (train_images_path + '1_tumor/') + str(train_
image_list_id[random_sample]) + '.tif')
plt.imshow(sample_to_plot)
plt.show()

fig, axes = plt.subplots(2,2)
i = 0
for batch in images_data_generation.flow(sample_to_plot.reshape((1,96,96,3)),batch_size=
1):
  axes[i//2,i%2].imshow(image.array_to_img(batch[0]))
  i += 1
  if i == 4:
      break
plt.show()
```





Una vez tenemos la instancia de la clase para realizar la generación de datos con "data augmentation", tenemos que crear la instancia de la clase que nos va a permitir ir importando las imágenes directamente de sus directorios a la hora de entrenar, ya que como hemos indicado previamente debido al gran volumen de imágenes de las que disponemos nos es imposible cargar todas en memoria RAM. Para hacer este proceso vamos a utilizar el método "flow_from_directory" de la propia clase de "ImageDataGenerator" de la librería de sklearn.

Definimos tres instancias de la función de "flow_from_directory", una para el conjunto de "train", otra para el conjunto de validación y la última para "test" con el mismo conjunto de validación, con el tamaño de entrada de las imágenes, el "batch_size" y el tipo de clasificación de las imágenes.

In [ ]:

```
train_generator = images_data_generation.flow_from_directory(train_images_path,
                                          target_size=(96,96),
                                          batch_size=train_batch_size,
                                          class_mode='categorical')

validation_generator = images_data_generation.flow_from_directory(validation_images_path,
                                          target_size=(96,96),
                                          batch_size=validation_batch_size,
                                          class_mode='categorical')

# Shuffle false, para que no se mezclen los datos de validation
test_generator = images_data_generation.flow_from_directory(validation_images_path,
                                          target_size=(96,96),
                                          batch_size=1,
                                          class_mode='categorical',
                                          shuffle=False)
```

```
Found 136000 images belonging to 2 classes.
Found 34000 images belonging to 2 classes.
Found 34000 images belonging to 2 classes.
```

Una vez terminado los tres primeros puntos comunes del ejercicio, ya podemos empezar a diferenciar entre la estrategia 1 y la 2, ya que a partir de ahora los pasos son diferentes.

Estrategia 1: Entrenar desde cero o from scratch

La primera estrategia a comparar será una red neuronal profunda que el alumno debe diseñar, entrenar y optimizar. Se debe justificar empíricamente las decisiones que llevaron a la selección de la arquitectura e hiperparámetros final. Se espera que el alumno utilice todas las técnicas de regularización mostradas en clase de forma justificada para la mejora del rendimiento de la red neuronal (weight regularization, dropout, batch normalization, data augmentation, etc.).

4. Desarrollo de la arquitectura de red neuronal y entrenamiento de la solución

4. Desarrollo de la arquitectura de red neuronal y entrenamiento de la solución

Usando la API secuencial de Keras, vamos a definir una primera arquitectura de red convolucional.

Como en este ejercicio estamos tratando con imágenes, debemos hacer uso de redes convolucionales, ya que son las más óptimas, conocidas hasta el momento, para extraer características en imágenes.

También debemos de tener en cuenta que nuestras imágenes tienen un tamaño de 96 x 96 píxeles x 3 canales, pero la información realmente importante, que queremos explorar y que nos va a determinar la clase de la imagen, se encuentra en una región central de 32 x 32 píxeles, por lo que haciendo uso de las capas convolucionales junto con las capas de MaxPooling, buscamos precisamente que la imagen se vaya redimensionando, cada vez más pequeña para solo quedarnos finalmente con la información importante de dicha región central.

Las capas de Dropout al final de cada bloque de capas convolucionales, son simplemente porque haciendo pruebas hemos podido comprobar que se consiguen mejor resultados.

Como ya sabemos, estas capas dropout básicamente cambian el peso de algunas neuronas a 0 durante una iteración, con la intención de reducir el overfitting.

In [ ]:

```
# Definimos el tipo de modelo, en este caso sequencial
model = Sequential()

# BASE MODEL:

# Primer bloque de redes convulocionales, se trata de 3 redes Conv2D de 32 filtros, con f
unción de activación relu. Después Tenemos una cada de MaxPooling2D con un pool_size de (
2,2), para quedarnos con los píxeles más importantes
# de la región. Por último, una capa de Dropout con un valor de 0,3 para intentar reducir
el overfitting.
model.add(Conv2D(32, (3,3), activation = 'relu', input_shape = (96, 96, 3)))
model.add(Conv2D(32, (3,3), activation = 'relu'))
model.add(Conv2D(32, (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))

# Segundo bloque de redes convulocionales, se trata de 3 redes Conv2D de 64 filtros, con
función de activación relu. Después Tenemos una cada de MaxPooling2D con un pool_size de
(2,2), para quedarnos con los píxeles más importantes
# de la región. Por último, una capa de Dropout con un valor de 0,3 para intentar reducir
el overfitting.
model.add(Conv2D(64, (3,3), activation ='relu'))
model.add(Conv2D(64, (3,3), activation ='relu'))
model.add(Conv2D(64, (3,3), activation ='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))

# Tercer bloque de redes convulocionales, se trata de 3 redes Conv2D de 128 filtros, con
función de activación relu. Después Tenemos una cada de MaxPooling2D con un pool_size de
(2,2), para quedarnos con los píxeles más importantes
# de la región. Por último, una capa de Dropout con un valor de 0,3 para intentar reducir
el overfitting.
model.add(Conv2D(128, (3,3), activation ='relu'))
model.add(Conv2D(128, (3,3), activation ='relu'))
model.add(Conv2D(128, (3,3), activation ='relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.3))

#TOP MODEL:
# Una capa Flatten, para aplanar las imágenes, seguido de una capa Dense de 256 neuronas
con activación relu, una capa Dropout igual a las vistas en el base model. Por último, la
capa Dense de 2 neuronas ( = número de clases)
# con activación softmax para predecir la clase de la imagen de entrada. Recordemos que l
a función softmax, está acotada entre 0 y 1, y nos indica la probabilidad de pertenencia
a una clase
model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(0.3))
```

```
model.add(Dense(2, activation = "softmax"))

# Visualizamos la arquitectura de nuestra red
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 94, 94, 32)        896

 conv2d_1 (Conv2D)           (None, 92, 92, 32)        9248

 conv2d_2 (Conv2D)           (None, 90, 90, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 45, 45, 32)       0
 )

 dropout (Dropout)           (None, 45, 45, 32)        0

 conv2d_3 (Conv2D)           (None, 43, 43, 64)        18496

 conv2d_4 (Conv2D)           (None, 41, 41, 64)        36928

 conv2d_5 (Conv2D)           (None, 39, 39, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 19, 19, 64)       0
 2D)

 dropout_1 (Dropout)         (None, 19, 19, 64)        0

 conv2d_6 (Conv2D)           (None, 17, 17, 128)       73856

 conv2d_7 (Conv2D)           (None, 15, 15, 128)       147584

 conv2d_8 (Conv2D)           (None, 13, 13, 128)       147584

 max_pooling2d_2 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 6, 6, 128)         0

 flatten (Flatten)           (None, 4608)              0

 dense (Dense)               (None, 256)               1179904

 dropout_3 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 2)                 514

=================================================================
Total params: 1,661,186
Trainable params: 1,661,186
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
# Compilamos nuestra red convolucional
model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy',
              metrics=['accuracy'])

# Entrenamos nuestra red convolucionarl
history = model.fit(train_generator, steps_per_epoch=amount_train_steps,
                    validation_data=validation_generator,
                    validation_steps=amount_validation_steps,
                    epochs=30, verbose=1)
```

```
Epoch 1/30
8500/8500 [==============================] - 609s 70ms/step - loss: 0.4826 - accuracy: 0.
7768 - val_loss: 0.4245 - val_accuracy: 0.8237
Epoch 2/30
```

```
8500/8500 [==============================] - 581s 68ms/step - loss: 0.3718 - accuracy: 0.
8382 - val_loss: 0.3366 - val_accuracy: 0.8572
Epoch 3/30
8500/8500 [==============================] - 496s 58ms/step - loss: 0.3154 - accuracy: 0.
8664 - val_loss: 0.2830 - val_accuracy: 0.8847
Epoch 4/30
8500/8500 [==============================] - 488s 57ms/step - loss: 0.2900 - accuracy: 0.
8782 - val_loss: 0.2684 - val_accuracy: 0.8886
Epoch 5/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.2728 - accuracy: 0.
8868 - val_loss: 0.2549 - val_accuracy: 0.8977
Epoch 6/30
8500/8500 [==============================] - 577s 68ms/step - loss: 0.2616 - accuracy: 0.
8926 - val_loss: 0.2538 - val_accuracy: 0.8987
Epoch 7/30
8500/8500 [==============================] - 504s 59ms/step - loss: 0.2511 - accuracy: 0.
8979 - val_loss: 0.2484 - val_accuracy: 0.9019
Epoch 8/30
8500/8500 [==============================] - 487s 57ms/step - loss: 0.2426 - accuracy: 0.
9023 - val_loss: 0.2353 - val_accuracy: 0.9075
Epoch 9/30
8500/8500 [==============================] - 481s 57ms/step - loss: 0.2371 - accuracy: 0.
9047 - val_loss: 0.2266 - val_accuracy: 0.9121
Epoch 10/30
8500/8500 [==============================] - 481s 57ms/step - loss: 0.2309 - accuracy: 0.
9088 - val_loss: 0.2396 - val_accuracy: 0.9066
Epoch 11/30
8500/8500 [==============================] - 481s 57ms/step - loss: 0.2266 - accuracy: 0.
9099 - val_loss: 0.2377 - val_accuracy: 0.9101
Epoch 12/30
8500/8500 [==============================] - 482s 57ms/step - loss: 0.2225 - accuracy: 0.
9124 - val_loss: 0.2101 - val_accuracy: 0.9195
Epoch 13/30
8500/8500 [==============================] - 480s 56ms/step - loss: 0.2168 - accuracy: 0.
9143 - val_loss: 0.2050 - val_accuracy: 0.9206
Epoch 14/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.2132 - accuracy: 0.
9161 - val_loss: 0.2056 - val_accuracy: 0.9219
Epoch 15/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.2085 - accuracy: 0.
9180 - val_loss: 0.1996 - val_accuracy: 0.9246
Epoch 16/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.2066 - accuracy: 0.
9191 - val_loss: 0.1997 - val_accuracy: 0.9246
Epoch 17/30
8500/8500 [==============================] - 480s 56ms/step - loss: 0.2049 - accuracy: 0.
9200 - val_loss: 0.1941 - val_accuracy: 0.9278
Epoch 18/30
8500/8500 [==============================] - 480s 57ms/step - loss: 0.2014 - accuracy: 0.
9221 - val_loss: 0.1869 - val_accuracy: 0.9270
Epoch 19/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1968 - accuracy: 0.
9237 - val_loss: 0.1990 - val_accuracy: 0.9220
Epoch 20/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.1963 - accuracy: 0.
9245 - val_loss: 0.1948 - val_accuracy: 0.9253
Epoch 21/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.1920 - accuracy: 0.
9255 - val_loss: 0.1884 - val_accuracy: 0.9311
Epoch 22/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1920 - accuracy: 0.
9258 - val_loss: 0.2004 - val_accuracy: 0.9215
Epoch 23/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1904 - accuracy: 0.
9260 - val_loss: 0.1894 - val_accuracy: 0.9298
Epoch 24/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1870 - accuracy: 0.
9279 - val_loss: 0.1830 - val_accuracy: 0.9302
Epoch 25/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1858 - accuracy: 0.
9290 - val_loss: 0.2028 - val_accuracy: 0.9193
Epoch 26/30
```

```
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1847 - accuracy: 0.
9285 - val_loss: 0.1997 - val_accuracy: 0.9235
Epoch 27/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.1844 - accuracy: 0.
9293 - val_loss: 0.2144 - val_accuracy: 0.9158
Epoch 28/30
8500/8500 [==============================] - 480s 56ms/step - loss: 0.1811 - accuracy: 0.
9310 - val_loss: 0.2021 - val_accuracy: 0.9213
Epoch 29/30
8500/8500 [==============================] - 478s 56ms/step - loss: 0.1794 - accuracy: 0.
9305 - val_loss: 0.2006 - val_accuracy: 0.9205
Epoch 30/30
8500/8500 [==============================] - 479s 56ms/step - loss: 0.1786 - accuracy: 0.
9311 - val_loss: 0.1932 - val_accuracy: 0.9287
```

In [ ]:

```
# Guardamos el modelo entrenado en nuestro directorio de google drive
model.save(BASE_FOLDER+"CNN_from_scratch_26_from_scratch_30epochs.h5")

# Guardamos los datos sacados durante el entrenamiento para plotear las gráficas
with open(BASE_FOLDER + '/CNN_from_scratch_training_26_from_scratch_30epochs', 'wb') as f
ile_pi:
        pickle.dump(history.history, file_pi)
```

Una vez que tenemos el modelo entrenado, vamos a proceder a probarlo contra los datos de validación, ya que los de test no tienen etiqueta de clase. Para ello, como lo tenemos guardado en google drive, teniendo en cuenta que el ipynb se puede reiniciar y perder las variables ya inicializadas. Primero debemos cargarlo directamente desde google drive y después ya podemos realizar las predicciones.

Para ello vamos a utilizar la función "load_model" de keras que nos permite recuperar un modelo guardado en formato .h5. Para cargar el modelo guardado, primero debemos generar la arquitecta para después introducirle los pesos ya entrenados.

También vamos a aprovechar y con la librería de "pikcle" vamos a guardar los resultados de las predicciones, así podremos recuperar dichas predicciones y sus gráficas correspondientes una vez cerrado el notebook.

In [ ]:

```
# Recuperamos el modelo del fichero guardado en google drive
model = load_model(BASE_FOLDER+"CNN/best_CNN_model_30.h5")

# Ahora vamos a pasar a evaluar el modelo, para ello como los datos de test no estan etqi
uetados con sus classes, vamos a hacer uso del conjunto de validación que si esta etiquet
ado
labelNames = ["No Tumor", "Tumor"]

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en training)
predictions = model.predict(test_generator,steps=len(images_dataframe_validation), verbo
se=1)
```

```
34000/34000 [==============================] - 209s 6ms/step
```

In [ ]:

```
# Vamos a guardar el fichero con las predicciones realizadas por nuestro modelo
with open(BASE_FOLDER + 'CNN/best_CNN_predictions_30', 'wb') as file_pi:
        pickle.dump(predictions, file_pi)
```

Para evaluar el comportamiento de nuestra red entrenada, a partir de las predicciones realizadas, vamos a mostrar la métrica de "Receiver Operator Characteristic" -> "ROC", una métrica de evaluación para clasificación binaria. Dibujando dicha curva y calculando el área debajo de esta "AUC" nos da la capacidad de nuestra red de distinguir entre las clases.

Para conseguir los valores de dichas métricas vamos a utilizar la función de "roc_auc_score" de la librería de sklearn.

Junto con la curva "ROC" y su área debajo, vamos a crear y a mostrar por pantalla la matriz de confusión, que

de manera visual y rapida nos va a permitir visualizar la eficacia de nuestra red. Para ello vamos a emplear la función propia de la librería de sklearn ConfusionMatrixDisplay: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las predicciones que nos va a indicar la precisión, el "recall", el "f1-score" y el "support" conseguidos con las predicciones de nuestro modelo entrenado.

In [ ]:

```
# Primero de todo debemos cargar los datos de prediccion que hemos guardado en el bloque anterior
file = open(BASE_FOLDER + 'CNN/best_CNN_predictions_30','rb')
predictions = pickle.load(file)
file.close()

# Creamos un dataframe con las probabilidades de las predicciones y las clases
model_predictions_dataframe = pd.DataFrame(predictions, columns=['0_no_tumor', '1_tumor'])
display(model_predictions_dataframe.head())

# Guardamos las clases predichas
image_aviable_classes = test_generator.classes

# Para calcular la AUC necesitamos saber las predicciones de la clase positiva, para ello vamos a elegir la columna tumor del dataframe creado previamente "model_predictions_dataframe"
positive_model_predictions = model_predictions_dataframe['1_tumor']

# Ahora ya podemos mostrar la gráfica
roc_auc_score(image_aviable_classes, positive_model_predictions)
```

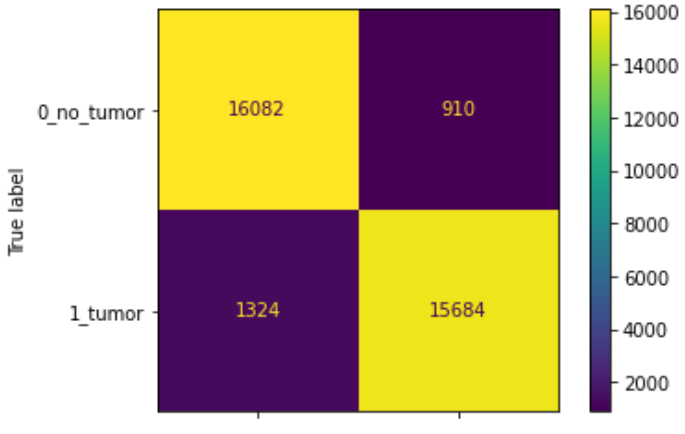
	0_no_tumor	1_tumor
0	0.999985	1.547547e-05
1	1.000000	1.291820e-07
2	0.840368	1.596321e-01
3	0.994794	5.205650e-03
4	0.999765	2.355191e-04

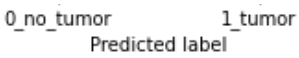
Out[ ]:

```
0.98153276407646
```

In [ ]:

```
# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```





In [ ]:

```
# Sacamos el reporte para el subconjunto de validación, utilizando la función de classification_report
# Como tenemos las predicciones de las clases como probabilidades debido a la función de activación de la última capa "softmax", debemos pasarlo a binario (0 o 1)
class_pred_binary = predictions.argmax(axis=1)

print(classification_report(image_aviable_classes, class_pred_binary, target_names=labelNames))
```

```
              precision    recall  f1-score   support

    No Tumor       0.92      0.95      0.94     16992
       Tumor       0.95      0.92      0.93     17008

    accuracy                           0.93     34000
   macro avg       0.93      0.93      0.93     34000
weighted avg       0.93      0.93      0.93     34000
```

Como también hemos guardado en google drive, los parámetros sacados durante el entrenamiento de nuestra red neuronal, ahora podemos recuperarlos y mostrar diferentes gráficas para discutir de una forma visual los resultados obtenidos época a época.

Los datos que vamos a mostrar en las gráficas son los siguientes:

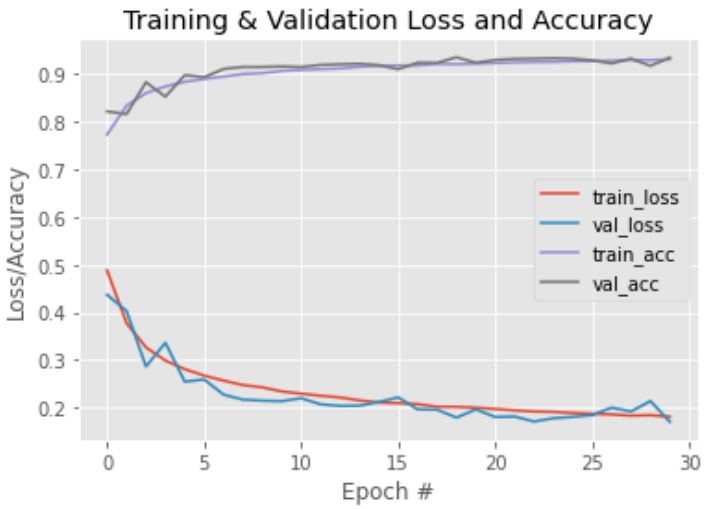
- **Train Loss: Perdidas con el dataset de train**
- **Validation Loss: Perdidas con el dataset de validation**
- **Train accuracy: Precisión con el dataset de train**
- **Validation accuracy: Precisión con el dataset de train Debemos tener en cuenta que tanto las perdidas como la precisión entre train y validation deben de ser similares, de no ser así, estaríamos delante de un problema de overfitting**

Como el entrenamiento lo hemos en 3 momentos separados, tenemos 3 ficheros guardados, por lo que los vamos a juntar para poder visualizar todo el entrenamiento desde la época 0 a la 30

In [ ]:

```
file = open(BASE_FOLDER + 'CNN/best_CNN_training_30','rb')
history_30 = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/best_CNN_training_20','rb')
history_20 = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/best_CNN_training','rb')
history = pickle.load(file)
file.close()

# Unimos los 3 ficheros
files = [history, history_20,history_30]
files_together = {}
for file in history.keys():
  files_together[file] = np.concatenate(list(files_together[file] for files_together in files))

plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), files_together["loss"], label="train_loss")
plt.plot(np.arange(0, 30), files_together["val_loss"], label="val_loss")
plt.plot(np.arange(0, 30), files_together["accuracy"], label="train_acc")
plt.plot(np.arange(0, 30), files_together["val_accuracy"], label="val_acc")
plt.title("Training & Validation Loss and Accuracy")
plt.xlabel("Epoch #")
```

```
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()
```



Como se puede observar en la gráfica, a medida que van pasando las épocas, se van aumentando a la par la accuracity de los datos de train y los de validation, así mismo pasa con las perdidas de train y validation que van disminuyendo a la par. Que los datos de train y de validation sigan la misma tendencia, es buena señal, ya que nos indica que el modelo se está comportando de manera similar con los dos conjuntos de datos. De no ser así podríamos estar delante de un problema de overfitting, cuando el modelo funciona mejor con los datos ya vistos o similares a los aprendidos.

Estrategia 2 Red pre-entrenada:

Para este apartado vamos a utilizar dos redes pre-entrenadas aplicando transfer learning y fine tuning para conseguir una clasificación de nuestro dataset. Estas redes se crearon para la competición anual de "imagenet". Las redes que hemos elegido son las:

- **Xception: https://arxiv.org/abs/1610.02357 Parámetros entrenables: 22.9M Top-5 accuracy: 94.5% Input Image size: 299x299**
- **VGG19: https://arxiv.org/abs/1409.1556 Parámetros entrenables: 143.7M Top-5 accuracy: 90% Input Image size: 224x224**

Como nuestras clases no son las mismas de imagenet, vamos solo a usar el base model, el top model lo vamos a crear nosotros mismos y vamos a utilizar el mismo top model para las dos redes neuronales.

Como hemos indicado en el parrafo anterior, en el primer entrenamiento solo vamos a utilizar transfer learning, lo que significa que vamos a utilizar los pesos previamente entrenados y a cambiar el top model para que se ajuste a nuestros datos y clases.

Para el segundo entrenamiento vamos a aplicar técnicas de fine tuning, lo que significa descongelar algunas capas o bloques de la red pre entrenada para que se reentrenen con nuestros datos de train.

En esta ocasión, vamos a descongelar los dos últimos bloques convolucionales, el bloque 4 y el 5.

VGG19 Transfer learning

In [ ]:

```
# Importamos el base model de la red VGG19, sin el top model y cambiando el input shape
vgg19_base_model = VGG19(weights='imagenet',
                 include_top=False,
                 input_shape=(96,96,3))

vgg19_base_model.summary()
# Evitar que los pesos se modifiquen en la parte convolucional -> TRANSFER LEARNING
vgg19_base_model.trainable = False
```

```
# Vamos a definir el top model para que termine dandonos la solución de nuestras dos clases
vgg19_pre_trained_model = Sequential()
vgg19_pre_trained_model.add(vgg19_base_model)
vgg19_pre_trained_model.add(Flatten())
vgg19_pre_trained_model.add(Dense(256, activation='relu'))
vgg19_pre_trained_model.add(Dense(2, activation='softmax'))

vgg19_pre_trained_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 96, 96, 3)]       0

 block1_conv1 (Conv2D)       (None, 96, 96, 64)        1792

 block1_conv2 (Conv2D)       (None, 96, 96, 64)        36928

 block1_pool (MaxPooling2D)  (None, 48, 48, 64)        0

 block2_conv1 (Conv2D)       (None, 48, 48, 128)       73856

 block2_conv2 (Conv2D)       (None, 48, 48, 128)       147584

 block2_pool (MaxPooling2D)  (None, 24, 24, 128)       0

 block3_conv1 (Conv2D)       (None, 24, 24, 256)       295168

 block3_conv2 (Conv2D)       (None, 24, 24, 256)       590080

 block3_conv3 (Conv2D)       (None, 24, 24, 256)       590080

 block3_conv4 (Conv2D)       (None, 24, 24, 256)       590080

 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0

 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160

 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv4 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0

 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 3, 3, 512)         20024384
```

```
 flatten_1 (Flatten)          (None, 4608)              0

 dense_2 (Dense)              (None, 256)               1179904

 dense_3 (Dense)              (None, 2)                 514

=================================================================
Total params: 21,204,802
Trainable params: 1,180,418
Non-trainable params: 20,024,384
_________________________________________________________________
```

In [ ]:

```
# Compilamos el modelo
vgg19_pre_trained_model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy',
              metrics=['accuracy'])
# Entrenamos el modelo compilado
vgg19_pre_trained_model_history = vgg19_pre_trained_model.fit(train_generator,
                    steps_per_epoch=amount_train_steps,
                    validation_data=validation_generator,
                    validation_steps=amount_validation_steps,
                    epochs=30, verbose=1)
```

```
Epoch 1/30
8500/8500 [==============================] - 507s 59ms/step - loss: 0.4768 - accuracy: 0.
8233 - val_loss: 0.3597 - val_accuracy: 0.8438
Epoch 2/30
8500/8500 [==============================] - 501s 59ms/step - loss: 0.3450 - accuracy: 0.
8484 - val_loss: 0.3412 - val_accuracy: 0.8523
Epoch 3/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.3297 - accuracy: 0.
8557 - val_loss: 0.3319 - val_accuracy: 0.8573
Epoch 4/30
8500/8500 [==============================] - 492s 58ms/step - loss: 0.3166 - accuracy: 0.
8635 - val_loss: 0.3352 - val_accuracy: 0.8579
Epoch 5/30
8500/8500 [==============================] - 495s 58ms/step - loss: 0.3086 - accuracy: 0.
8668 - val_loss: 0.3285 - val_accuracy: 0.8589
Epoch 6/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.3017 - accuracy: 0.
8706 - val_loss: 0.3245 - val_accuracy: 0.8640
Epoch 7/30
8500/8500 [==============================] - 494s 58ms/step - loss: 0.2958 - accuracy: 0.
8730 - val_loss: 0.3201 - val_accuracy: 0.8644
Epoch 8/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2894 - accuracy: 0.
8760 - val_loss: 0.3202 - val_accuracy: 0.8650
Epoch 9/30
8500/8500 [==============================] - 491s 58ms/step - loss: 0.2859 - accuracy: 0.
8773 - val_loss: 0.3136 - val_accuracy: 0.8674
Epoch 10/30
8500/8500 [==============================] - 492s 58ms/step - loss: 0.2782 - accuracy: 0.
8809 - val_loss: 0.3132 - val_accuracy: 0.8687
Epoch 11/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2743 - accuracy: 0.
8827 - val_loss: 0.3121 - val_accuracy: 0.8701
Epoch 12/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2712 - accuracy: 0.
8857 - val_loss: 0.3137 - val_accuracy: 0.8666
Epoch 13/30
8500/8500 [==============================] - 494s 58ms/step - loss: 0.2693 - accuracy: 0.
8855 - val_loss: 0.3190 - val_accuracy: 0.8675
Epoch 14/30
8500/8500 [==============================] - 494s 58ms/step - loss: 0.2647 - accuracy: 0.
8886 - val_loss: 0.3202 - val_accuracy: 0.8710
Epoch 15/30
8500/8500 [==============================] - 494s 58ms/step - loss: 0.2619 - accuracy: 0.
8891 - val_loss: 0.3336 - val_accuracy: 0.8642
Epoch 16/30
8500/8500 [==============================] - 492s 58ms/step - loss: 0.2590 - accuracy: 0.
```

```
8904 - val_loss: 0.3140 - val_accuracy: 0.8670
Epoch 17/30
8500/8500 [==============================] - 492s 58ms/step - loss: 0.2562 - accuracy: 0.
8921 - val_loss: 0.3146 - val_accuracy: 0.8728
Epoch 18/30
8500/8500 [==============================] - 489s 58ms/step - loss: 0.2544 - accuracy: 0.
8928 - val_loss: 0.3227 - val_accuracy: 0.8680
Epoch 19/30
8500/8500 [==============================] - 488s 57ms/step - loss: 0.2521 - accuracy: 0.
8935 - val_loss: 0.3221 - val_accuracy: 0.8717
Epoch 20/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2479 - accuracy: 0.
8956 - val_loss: 0.3220 - val_accuracy: 0.8726
Epoch 21/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2470 - accuracy: 0.
8967 - val_loss: 0.3237 - val_accuracy: 0.8709
Epoch 22/30
8500/8500 [==============================] - 492s 58ms/step - loss: 0.2460 - accuracy: 0.
8965 - val_loss: 0.3252 - val_accuracy: 0.8665
Epoch 23/30
8500/8500 [==============================] - 495s 58ms/step - loss: 0.2417 - accuracy: 0.
8991 - val_loss: 0.3269 - val_accuracy: 0.8725
Epoch 24/30
8500/8500 [==============================] - 497s 58ms/step - loss: 0.2420 - accuracy: 0.
8989 - val_loss: 0.3283 - val_accuracy: 0.8743
Epoch 25/30
8500/8500 [==============================] - 494s 58ms/step - loss: 0.2371 - accuracy: 0.
9011 - val_loss: 0.3283 - val_accuracy: 0.8709
Epoch 26/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2358 - accuracy: 0.
9017 - val_loss: 0.3239 - val_accuracy: 0.8738
Epoch 27/30
8500/8500 [==============================] - 493s 58ms/step - loss: 0.2343 - accuracy: 0.
9020 - val_loss: 0.3283 - val_accuracy: 0.8720
Epoch 28/30
8500/8500 [==============================] - 487s 57ms/step - loss: 0.2325 - accuracy: 0.
9025 - val_loss: 0.3333 - val_accuracy: 0.8738
Epoch 29/30
8500/8500 [==============================] - 483s 57ms/step - loss: 0.2302 - accuracy: 0.
9038 - val_loss: 0.3403 - val_accuracy: 0.8699
Epoch 30/30
8500/8500 [==============================] - 483s 57ms/step - loss: 0.2286 - accuracy: 0.
9051 - val_loss: 0.3469 - val_accuracy: 0.8691
```

In [ ]:

```
# Guardamos el modelo entrenado en nuestro directorio de google drive
vgg19_pre_trained_model.save(BASE_FOLDER+"VGG/best_fine_tunning_VGG_model.h5")

# Guardamos los datos sacados durante el entrenamiento para plotear las graficas
with open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_training', 'wb') as file_pi:
        pickle.dump(vgg19_pre_trained_model_history.history, file_pi)
```

Como hemos hecho en el apartado anterior, una vez que tenemos el modelo entrenado, vamos a proceder a probar-lo contra los datos de validación, ya que los de test no tienen etiqueta de clase. Para ello, como lo tenemos guardado en google drive, teniendo en cuenta que el ipynb se puede reiniciar y perder las variables ya inicializadas. Primero debemos cargarlo directamente desde google drive y después ya podemos realizar las predicciones.

Para ello vamos a utilizar la función "load_model" de keras que nos permite recuperar un modelo guardado en formato .h5. Para cargar el modelo guardado, primero debemos generar la arquitecta para después introducirle los pesos ya entrenados.

También vamos a aprovechar y con la librería de "pikcle" vamos a guardar los resultados de las predicciones, así podremos recuperar dichas predicciones y sus gráficas correspondientes una vez cerrado el notebook.

In [ ]:

```
# Recuperamos el modelo del fichero guardado en google drive
```

```
# model = load_model(BASE_FOLDER+"CNN_from_scratch_26_from_scratch_20epochs.h5")

# Ahora vamos a pasar a evaluar el modelo, para ello como los datos de test no estan etqi
uetados con sus classes, vamos a hacer uso del conjunto de validación que si esta etiquet
ado
labelNames = ["No Tumor", "Tumor"]

# Efectuamos la predicción (empleamos el mismo valor de batch_size que en training)
predictions = vgg19_pre_trained_model.predict(test_generator,steps=len(images_dataframe_v
alidation), verbose=1)
```

```
34000/34000 [==============================] - 282s 8ms/step
```

In [ ]:

```
# Vamos a guardar el fichero con las predicciones realizadas por nuestro modelo
with open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_predictions', 'wb') as file_pi:
        pickle.dump(predictions, file_pi)
```

Al igual que en el apartado anterior, para evaluar el comportamiento de nuestra red entrenada, a partir de las predicciones realizadas, vamos a mostrar la métrica de "Receiver Operator Characteristic" -> "ROC", una métrica de evaluación para clasificación binaria. Dibujando dicha curva y calculando el área debajo de esta "AUC" nos da la capacidad de nuestra red de distinguir entre las clases.

Para conseguir los valores de dichas métricas vamos a utilizar la función de "roc_auc_score" de la librería de sklearn.

Junto con la curva "ROC" y su área debajo, vamos a crear y a mostrar por pantalla la matriz de confusión, que de manera visual y rápida nos va a permitir visualizar la eficacia de nuestra red. Para ello vamos a emplear la función propia de la librería de sklearn ConfusionMatrixDisplay: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las predicciones que nos va a indicar la precisión, el "recall", el "f1-score" y el "support" conseguidos con las predicciones de nuestro modelo entrenado.

In [ ]:

```
# Primero de todo debemos cargar los datos de prediccion que hemos guardado en el bloque
anterior
file = open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_predictions','rb')
predictions = pickle.load(file)
file.close()

# Creamos un dataframe con las probabilidades de las predicciones y las clases
model_predictions_dataframe = pd.DataFrame(predictions, columns=['0_no_tumor', '1_tumor'
])
display(model_predictions_dataframe.head())

# Guardamos las clases predichas
image_aviable_classes = test_generator.classes

# Para calcular la AUC necesitamos saber las predicciones de la clase positiva, para ello
vamos a elegir la columna tumor del dataframe creado previamente "model_predictions_dataf
rame"
positive_model_predictions = model_predictions_dataframe['1_tumor']

# Ahora ya podemos mostrar la gráfica
roc_auc_score(image_aviable_classes, positive_model_predictions)
```

	0_no_tumor	1_tumor
0	0.999928	0.000072
1	0.996689	0.003311
2	0.970010	0.029991
3	0.996878	0.003122
4	0.999989	0.000011

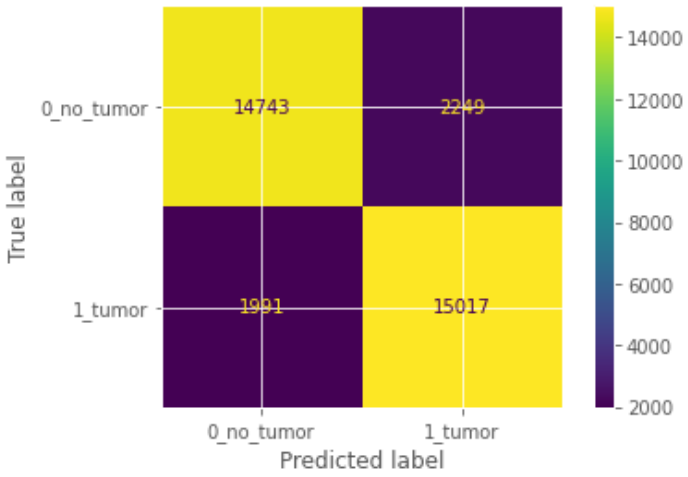
0 no tumor 1 tumor

Out[ ]:

0.943748151902705

In [ ]:

```
# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argma
x(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```



In [ ]:

```
# Sacamos el reporte para el subconjunto de validación, utilizando la función de classifi
cation_report
# Como tenemos las predicciones de las clases como probabilidades debido a la función de
activación de la última capa "softmax", debemos pasarlo a binario (0 o 1)
class_pred_binary = predictions.argmax(axis=1)

print(classification_report(image_aviable_classes, class_pred_binary, target_names=label
Names))
```

```
              precision    recall  f1-score   support

    No Tumor       0.88      0.87      0.87     16992
       Tumor       0.87      0.88      0.88     17008

    accuracy                           0.88     34000
   macro avg       0.88      0.88      0.88     34000
weighted avg       0.88      0.88      0.88     34000
```

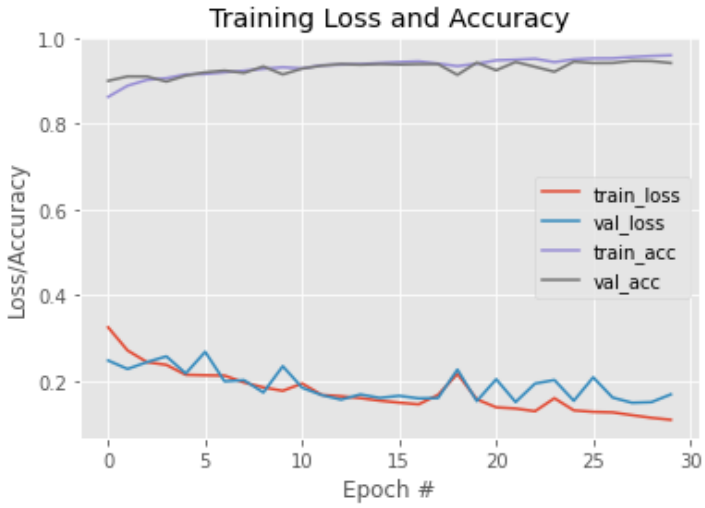
Repitiendo lo mismo que en apartado anterior, como también hemos guardado en google drive, los parámetros sacados durante el entrenamiento de nuestra red neuronal, ahora podemos recuperarlos y mostrar diferentes gráficas para discutir de una forma visual los resultados obtenidos época a época.

Los datos que vamos a mostrar en las gráficas son los siguientes:

- **Train Loss: Perdidas con el dataset de train**
- **Validation Loss: Perdidas con el dataset de validation**
- **Train accuracy: Precisión con el dataset de train**
- **Validation accuracy: Precisión con el dataset de train Debemos tener en cuenta que tanto las perdidas como la precisión entre train y validation deben de ser similares, de no ser así, estaríamos delante de un problema de overfitting.**

In [ ]:

```
file = open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_training','rb')
history = pickle.load(file)

# Gráficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), history["loss"], label="train_loss")
plt.plot(np.arange(0, 30), history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 30), history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 30), history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

file.close()
```



Como podemos ver con solo transfer leanring, básicamente substituyendo el top model de la red VGG19 por uno simple, hecho por nosotros conseguimos unos resultados bastante decentes, pero vamos a intentar mejorar estos resultados aplicando aparte de trasnfer learning, tecnicas de fine tuning que significa descongelando alguna de las capas de la red pre-entrenada para que dichas capas se reentrenen junto con el top model que nosotros le pasamos. Para ello primero vamos a ver las capas de las que dispone la red VGG19 y a decidir cuales de ellas vamos a descongelar.

VGG19 Fine tunning

Como hemos indicado en el párrafo anterior, ahora aparte de importar el modelo de la red de VGG19 pre entrenado, vamos a descongelar algunas de las capas convolucionales de este, para que se reentrenen con nuestros datos, a ver si así conseguimos mejores resultados.

En concreto, vamos a descongelar los dos últimos bloques convolucionales, el número 4 y el número 5.

Para evitar ningún posible problema con ejecuciones pasadas, vamos a reimportar la red VGG19 pre entrenada.

In [ ]:

```
# Importamos el base model de la red VGG19, sin el top model y cambiando el input shape
vgg19_base_model = VGG19(weights='imagenet',
                 include_top=False,
                 input_shape=(96,96,3))

vgg19_base_model.summary()

# Congelamos todos los bloques y capas, menos el bloque 4 y el 5 como hemos índicado prev
iamente
for layer in vgg19_base_model.layers:
  if 'block4_' in layer.name:
    print('Bloque ' + layer.name + ' no congelado...')
    break
```

```
  layer.trainable = False
  print('Capa ' + layer.name + ' congelada...')

# Definimos el top model para que termine dandonos la solución de nuestras dos clases
vgg19_pre_trained_with_fine_tunning_model = Sequential()
# Juntamos el base model importado con el nuevo top model creado por nosotros
vgg19_pre_trained_with_fine_tunning_model.add(vgg19_base_model)
vgg19_pre_trained_with_fine_tunning_model.add(Flatten())
vgg19_pre_trained_with_fine_tunning_model.add(Dense(256, activation='relu'))
vgg19_pre_trained_with_fine_tunning_model.add(Dense(2, activation='softmax'))

# Vemos como és por dentro nuestra nueva red neuronal
vgg19_pre_trained_with_fine_tunning_model.summary()
```

```
Model: "vgg19"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 96, 96, 3)]       0

 block1_conv1 (Conv2D)       (None, 96, 96, 64)        1792

 block1_conv2 (Conv2D)       (None, 96, 96, 64)        36928

 block1_pool (MaxPooling2D)  (None, 48, 48, 64)        0

 block2_conv1 (Conv2D)       (None, 48, 48, 128)       73856

 block2_conv2 (Conv2D)       (None, 48, 48, 128)       147584

 block2_pool (MaxPooling2D)  (None, 24, 24, 128)       0

 block3_conv1 (Conv2D)       (None, 24, 24, 256)       295168

 block3_conv2 (Conv2D)       (None, 24, 24, 256)       590080

 block3_conv3 (Conv2D)       (None, 24, 24, 256)       590080

 block3_conv4 (Conv2D)       (None, 24, 24, 256)       590080

 block3_pool (MaxPooling2D)  (None, 12, 12, 256)       0

 block4_conv1 (Conv2D)       (None, 12, 12, 512)       1180160

 block4_conv2 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv3 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_conv4 (Conv2D)       (None, 12, 12, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 6, 6, 512)         0

 block5_conv1 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv2 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv3 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_conv4 (Conv2D)       (None, 6, 6, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 3, 3, 512)         0

=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
Non-trainable params: 0
_________________________________________________________________
Capa input_2 congelada...
Capa block1_conv1 congelada...
Capa block1_conv2 congelada...
Capa block1_pool congelada...
Capa block2_conv1 congelada...
```

```
Capa block2_conv2 congelada...
Capa block2_pool congelada...
Capa block3_conv1 congelada...
Capa block3_conv2 congelada...
Capa block3_conv3 congelada...
Capa block3_conv4 congelada...
Capa block3_pool congelada...
Bloque block4_conv1 no congelado...
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 3, 3, 512)         20024384

 flatten_2 (Flatten)         (None, 4608)              0

 dense_4 (Dense)             (None, 256)               1179904

 dense_5 (Dense)             (None, 2)                 514

=================================================================
Total params: 21,204,802
Trainable params: 18,879,234
Non-trainable params: 2,325,568
_________________________________________________________________
```

In [ ]:

```
# Compilamos el modelo
vgg19_pre_trained_with_fine_tunning_model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy',
              metrics=['accuracy'])

# Entrenamos el modelo
vgg19_pre_trained_with_fine_tunning_model_history = vgg19_pre_trained_with_fine_tunning_model.fit(train_generator,
                    steps_per_epoch=amount_train_steps,
                    validation_data=validation_generator,
                    validation_steps=amount_validation_steps,
                    epochs=30, verbose=1)
```

```
Epoch 1/30
8500/8500 [==============================] - 534s 63ms/step - loss: 0.3396 - accuracy: 0.8577 - val_loss: 0.3649 - val_accuracy: 0.8445
Epoch 2/30
8500/8500 [==============================] - 532s 63ms/step - loss: 0.2905 - accuracy: 0.8804 - val_loss: 0.2656 - val_accuracy: 0.8935
Epoch 3/30
8500/8500 [==============================] - 530s 62ms/step - loss: 0.2746 - accuracy: 0.8898 - val_loss: 0.2601 - val_accuracy: 0.8939
Epoch 4/30
8500/8500 [==============================] - 537s 63ms/step - loss: 0.2546 - accuracy: 0.8985 - val_loss: 0.2766 - val_accuracy: 0.8929
Epoch 5/30
8500/8500 [==============================] - 539s 63ms/step - loss: 0.2446 - accuracy: 0.9031 - val_loss: 0.2161 - val_accuracy: 0.9138
Epoch 6/30
8500/8500 [==============================] - 544s 64ms/step - loss: 0.2217 - accuracy: 0.9125 - val_loss: 0.3961 - val_accuracy: 0.8397
Epoch 7/30
8500/8500 [==============================] - 547s 64ms/step - loss: 0.2147 - accuracy: 0.9148 - val_loss: 0.2281 - val_accuracy: 0.9114
Epoch 8/30
8500/8500 [==============================] - 538s 63ms/step - loss: 0.2080 - accuracy: 0.9191 - val_loss: 0.2054 - val_accuracy: 0.9247
Epoch 9/30
8500/8500 [==============================] - 538s 63ms/step - loss: 0.2037 - accuracy: 0.9220 - val_loss: 0.1929 - val_accuracy: 0.9246
Epoch 10/30
8500/8500 [==============================] - 537s 63ms/step - loss: 0.1924 - accuracy: 0.9261 - val_loss: 0.1854 - val_accuracy: 0.9276
Epoch 11/30
```

```
8500/8500 [==============================] - 532s 63ms/step - loss: 0.1843 - accuracy: 0.
9290 - val_loss: 0.1848 - val_accuracy: 0.9284
Epoch 12/30
8500/8500 [==============================] - 542s 64ms/step - loss: 0.3180 - accuracy: 0.
9260 - val_loss: 0.1930 - val_accuracy: 0.9259
Epoch 13/30
8500/8500 [==============================] - 545s 64ms/step - loss: 0.1793 - accuracy: 0.
9305 - val_loss: 0.1825 - val_accuracy: 0.9329
Epoch 14/30
8500/8500 [==============================] - 539s 63ms/step - loss: 0.1901 - accuracy: 0.
9285 - val_loss: 0.1781 - val_accuracy: 0.9319
Epoch 15/30
8500/8500 [==============================] - 546s 64ms/step - loss: 0.1684 - accuracy: 0.
9356 - val_loss: 0.1881 - val_accuracy: 0.9233
Epoch 16/30
8500/8500 [==============================] - 537s 63ms/step - loss: 0.2044 - accuracy: 0.
9318 - val_loss: 0.1778 - val_accuracy: 0.9340
Epoch 17/30
8500/8500 [==============================] - 553s 65ms/step - loss: 0.1853 - accuracy: 0.
9301 - val_loss: 0.1823 - val_accuracy: 0.9279
Epoch 18/30
8500/8500 [==============================] - 536s 63ms/step - loss: 0.1631 - accuracy: 0.
9379 - val_loss: 0.1743 - val_accuracy: 0.9333
Epoch 19/30
8500/8500 [==============================] - 533s 63ms/step - loss: 0.1849 - accuracy: 0.
9364 - val_loss: 0.1640 - val_accuracy: 0.9377
Epoch 20/30
8500/8500 [==============================] - 532s 63ms/step - loss: 0.1570 - accuracy: 0.
9402 - val_loss: 0.1721 - val_accuracy: 0.9347
Epoch 21/30
8500/8500 [==============================] - 535s 63ms/step - loss: 0.1539 - accuracy: 0.
9417 - val_loss: 0.1674 - val_accuracy: 0.9376
Epoch 22/30
8500/8500 [==============================] - 535s 63ms/step - loss: 0.1463 - accuracy: 0.
9447 - val_loss: 0.1615 - val_accuracy: 0.9390
Epoch 23/30
8500/8500 [==============================] - 538s 63ms/step - loss: 0.1412 - accuracy: 0.
9469 - val_loss: 0.1770 - val_accuracy: 0.9329
Epoch 24/30
8500/8500 [==============================] - 537s 63ms/step - loss: 0.1702 - accuracy: 0.
9397 - val_loss: 0.1676 - val_accuracy: 0.9357
Epoch 25/30
8500/8500 [==============================] - 537s 63ms/step - loss: 0.1501 - accuracy: 0.
9443 - val_loss: 0.1707 - val_accuracy: 0.9351
Epoch 26/30
8500/8500 [==============================] - 538s 63ms/step - loss: 0.1431 - accuracy: 0.
9457 - val_loss: 0.1542 - val_accuracy: 0.9434
Epoch 27/30
8500/8500 [==============================] - 539s 63ms/step - loss: 0.1404 - accuracy: 0.
9473 - val_loss: 0.1953 - val_accuracy: 0.9229
Epoch 28/30
8500/8500 [==============================] - 543s 64ms/step - loss: 0.1337 - accuracy: 0.
9504 - val_loss: 0.1515 - val_accuracy: 0.9440
Epoch 29/30
8500/8500 [==============================] - 540s 64ms/step - loss: 0.1372 - accuracy: 0.
9495 - val_loss: 0.1574 - val_accuracy: 0.9433
Epoch 30/30
8500/8500 [==============================] - 540s 64ms/step - loss: 0.1240 - accuracy: 0.
9534 - val_loss: 0.1511 - val_accuracy: 0.9437
```

In [ ]:

```
# Guardamos el modelo entrenado en nuestro directorio de google drive
vgg19_pre_trained_with_fine_tunning_model.save(BASE_FOLDER+"/VGG/best_fine_tunning_VGG_mo
del.h5")

# Guardamos los datos sacados durante el entrenamiento para plotear las graficas
with open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_training', 'wb') as file_pi:
        pickle.dump(vgg19_pre_trained_with_fine_tunning_model_history.history, file_pi)
```

Como hemos hecho en el apartado anterior, una vez que tenemos el modelo entrenado, vamos a proceder a

probar-lo contra los datos de validación, ya que los de test no tienen etiqueta de clase. Para ello, como lo tenemos guardado en google drive, teniendo en cuenta que el ipynb se puede reiniciar y perder las variables ya inicializadas. Primero debemos cargarlo directamente desde google drive y después ya podemos realizar las predicciones.

Para ello vamos a utilizar la función "load_model" de keras que nos permite recuperar un modelo guardado en formato .h5. Para cargar el modelo guardado, primero debemos generar la arquitecta para después introducirle los pesos ya entrenados.

También vamos a aprovechar y con la librería de "pikcle" vamos a guardar los resultados de las predicciones, así podremos recuperar dichas predicciones y sus gráficas correspondientes una vez cerrado el notebook.

In [ ]:

```
# Recuperamos el modelo del fichero guardado en google drive
vgg19_pre_trained_with_fine_tunning_model = load_model(BASE_FOLDER+"/VGG/best_fine_tunning_VGG_model.h5")

# Ahora vamos a pasar a evaluar el modelo, para ello como los datos de test no estan etquetados con sus classes, vamos a hacer uso del conjunto de validación que si esta etiquetado
labelNames = ["No Tumor", "Tumor"]

# Efectuamos las predicciónes (empleamos el mismo valor de batch_size que en training)
predictions = vgg19_pre_trained_with_fine_tunning_model.predict(test_generator,steps=len(images_dataframe_validation), verbose=1)
```

```
34000/34000 [==============================] - 264s 8ms/step
```

In [ ]:

```
# Vamos a guardar el fichero con las predicciones realizadas por nuestro modelo
with open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_predictions', 'wb') as file_pi:
        pickle.dump(predictions, file_pi)
```

Al igual que en el apartado anterior, para evaluar el comportamiento de nuestra red entrenada, a partir de las predicciones realizadas, vamos a mostrar la métrica de "Receiver Operator Characteristic" -> "ROC", una métrica de evaluación para clasificación binaria. Dibujando dicha curva y calculando el área debajo de esta "AUC" nos da la capacidad de nuestra red de distinguir entre las clases.

Para conseguir los valores de dichas métricas vamos a utilizar la función de "roc_auc_score" de la librería de sklearn.

Junto con la curva "ROC" y su área debajo, vamos a crear y a mostrar por pantalla la matriz de confusión, que de manera visual y rápida nos va a permitir visualizar la eficacia de nuestra red. Para ello vamos a emplear la función propia de la librería de sklearn ConfusionMatrixDisplay: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las predicciones que nos va a indicar la precisión, el "recall", el "f1-score" y el "support" conseguidos con las predicciones de nuestro modelo entrenado.

In [ ]:

```
# Primero de todo debemos cargar los datos de prediccion que hemos guardado en el bloque anterior
file = open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_predictions','rb')
predictions = pickle.load(file)
file.close()

# Creamos un dataframe con las probabilidades de las predicciones y las clases
model_predictions_dataframe = pd.DataFrame(predictions, columns=['0_no_tumor', '1_tumor'])
display(model_predictions_dataframe.head())

# Guardamos las clases predichas
image_aviable_classes = test_generator.classes
```

```
# Para calcular la AUC necesitamos saber las predicciones de la clase positiva, para ello vamos a elegir la columna tumor del dataframe creado previamente "model_predictions_dataframe"
positive_model_predictions = model_predictions_dataframe['1_tumor']

# Ahora ya podemos mostrar la gráfica
roc_auc_score(image_aviable_classes, positive_model_predictions)
```

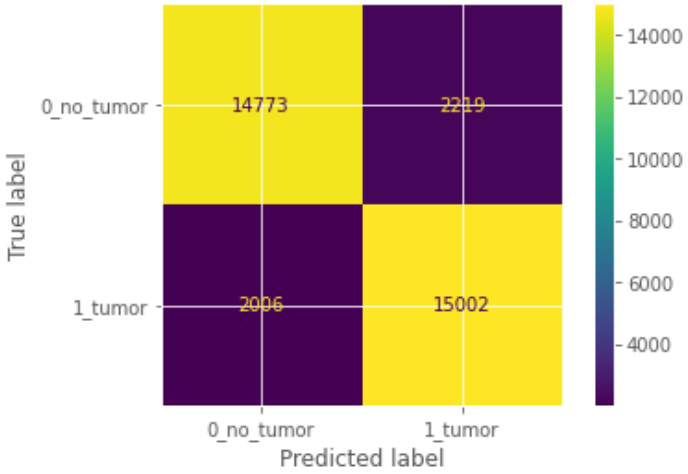
	0_no_tumor	1_tumor
0	**0.999899**	**0.000101**
1	**0.913458**	**0.086542**
2	**0.975086**	**0.024914**
3	**0.996202**	**0.003798**
4	**0.999998**	**0.000002**

Out[ ]:

```
0.9448280604463524
```

In [ ]:

```
# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```



In [ ]:

```
# Sacamos el reporte para el subconjunto de validación, utilizando la función de classification_report
# Como tenemos las predicciones de las clases como probabilidades debido a la función de activación de la última capa "softmax", debemos pasarlo a binario (0 o 1)
class_pred_binary = predictions.argmax(axis=1)

print(classification_report(image_aviable_classes, class_pred_binary, target_names=labelNames))
```

```
              precision    recall  f1-score   support

    No Tumor       0.88      0.87      0.87     16992
       Tumor       0.87      0.88      0.88     17008

    accuracy                           0.88     34000
   macro avg       0.88      0.88      0.88     34000
weighted avg       0.88      0.88      0.88     34000
```

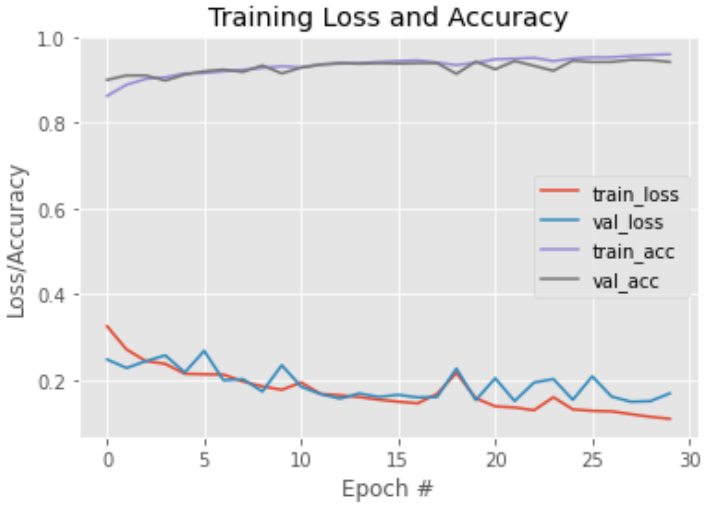
Repitiendo lo mismo que en apartado anterior, como también hemos guardado en google drive, los parámetros sacados durante el entrenamiento de nuestra red neuronal, ahora podemos recuperarlos y mostrar diferentes gráficas para discutir de una forma visual los resultados obtenidos época a época.

Los datos que vamos a mostrar en las gráficas son los siguientes:

- **Train Loss: Perdidas con el dataset de train**
- **Validation Loss: Perdidas con el dataset de validation**
- **Train accuracy: Precisión con el dataset de train**
- **Validation accuracy: Precisión con el dataset de train Debemos tener en cuenta que tanto las perdidas como la precisión entre train y validation deben de ser similares, de no ser así, estaríamos delante de un problema de overfitting.**

In [ ]:

```
file = open(BASE_FOLDER + '/VGG/best_fine_tunning_VGG_training','rb')
history = pickle.load(file)

# Gráficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), history["loss"], label="train_loss")
plt.plot(np.arange(0, 30), history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 30), history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 30), history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

file.close()
```



Como podemos observar a través de las gráficas y de las métricas extraídas de las predicciones del modelo, usando fine tunning y descongelando los dos últimos bloques covolucionales de la red de vgg conseguimos unos resultados aún mejores.

La red de VGG19, es una de las mejores arquitecturas del concurso de imageNet, por lo que es natural que usando esta arquitectura y reentrenándola con nuestros datos consigamos unos muy buenos resultados.

Xception Transfer learning

Al igual que en el párrafo anterior de transfer learning de la red VGG, vamos a realizar el entrenamiento usando transfer learning, lo que significa que vamos a utilizar los pesos previamente entrenados y a cambiar el top model para que se ajuste a nuestros datos y clases.

In [ ]:

```
# Importamos el base model de la red Xception, sin el top model y cambiando el input shap
```

```
e
xception_base_model = Xception(weights='imagenet',
                 include_top=False,
                 input_shape=(96,96,3))

xception_base_model.summary()
# Evitar que los pesos se modifiquen en la parte convolucional -> TRANSFER LEARNING
xception_base_model.trainable = False

# Vamos a definir el top model para que termine dandonos la solución de nuestras dos clas
es
xception_pre_trained_model = Sequential()
xception_pre_trained_model.add(xception_base_model)
xception_pre_trained_model.add(Flatten())
xception_pre_trained_model.add(Dense(256, activation='relu'))
xception_pre_trained_model.add(Dense(2, activation='softmax'))

xception_pre_trained_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/xcepti
on/xception_weights_tf_dim_ordering_tf_kernels_notop.h5
83689472/83683744 [==============================] - 1s 0us/step
83697664/83683744 [==============================] - 1s 0us/step
Model: "xception"
__________________________________________________________________________________________
_________
 Layer (type)                   Output Shape         Param #     Connected to

==========================================================================================
=========
 input_3 (InputLayer)           [(None, 96, 96, 3)]  0           []


 block1_conv1 (Conv2D)          (None, 47, 47, 32)   864         ['input_3[0][0]']


 block1_conv1_bn (BatchNormaliz  (None, 47, 47, 32)  128         ['block1_conv1[0][0]']

 ation)


 block1_conv1_act (Activation)  (None, 47, 47, 32)   0           ['block1_conv1_bn[0][0]'
]


 block1_conv2 (Conv2D)          (None, 45, 45, 64)   18432       ['block1_conv1_act[0][0]
']


 block1_conv2_bn (BatchNormaliz  (None, 45, 45, 64)  256         ['block1_conv2[0][0]']

 ation)


 block1_conv2_act (Activation)  (None, 45, 45, 64)   0           ['block1_conv2_bn[0][0]'
]


 block2_sepconv1 (SeparableConv  (None, 45, 45, 128)  8768       ['block1_conv2_act[0][0]
']
 2D)


 block2_sepconv1_bn (BatchNorma  (None, 45, 45, 128)  512        ['block2_sepconv1[0][0]'
]
 lization)
```

```
 block2_sepconv2_act (Activatio  (None, 45, 45, 128)  0          ['block2_sepconv1_bn[0][
0]']
 n)

 block2_sepconv2 (SeparableConv  (None, 45, 45, 128)  17536      ['block2_sepconv2_act[0]
[0]']
 2D)

 block2_sepconv2_bn (BatchNorma  (None, 45, 45, 128)  512        ['block2_sepconv2[0][0]'
]
 lization)

 conv2d_9 (Conv2D)              (None, 23, 23, 128)  8192        ['block1_conv2_act[0][0]
']

 block2_pool (MaxPooling2D)     (None, 23, 23, 128)  0          ['block2_sepconv2_bn[0][
0]']

 batch_normalization (BatchNorm  (None, 23, 23, 128)  512        ['conv2d_9[0][0]']

 alization)

 add (Add)                      (None, 23, 23, 128)  0          ['block2_pool[0][0]',

                                                                  'batch_normalization[
0][0]']

 block3_sepconv1_act (Activatio  (None, 23, 23, 128)  0          ['add[0][0]']

 n)

 block3_sepconv1 (SeparableConv  (None, 23, 23, 256)  33920      ['block3_sepconv1_act[0]
[0]']
 2D)

 block3_sepconv1_bn (BatchNorma  (None, 23, 23, 256)  1024       ['block3_sepconv1[0][0]'
]
 lization)

 block3_sepconv2_act (Activatio  (None, 23, 23, 256)  0          ['block3_sepconv1_bn[0][
0]']
 n)

 block3_sepconv2 (SeparableConv  (None, 23, 23, 256)  67840      ['block3_sepconv2_act[0]
[0]']
 2D)

 block3_sepconv2_bn (BatchNorma  (None, 23, 23, 256)  1024       ['block3_sepconv2[0][0]'
]
```

```
 lization)

 conv2d_10 (Conv2D)             (None, 12, 12, 256)  32768       ['add[0][0]']

 block3_pool (MaxPooling2D)     (None, 12, 12, 256)  0           ['block3_sepconv2_bn[0][
0]']

 batch_normalization_1 (BatchNo  (None, 12, 12, 256)  1024       ['conv2d_10[0][0]']
 rmalization)

 add_1 (Add)                    (None, 12, 12, 256)  0           ['block3_pool[0][0]',
                                                                  'batch_normalization_
1[0][0]']

 block4_sepconv1_act (Activatio  (None, 12, 12, 256)  0          ['add_1[0][0]']
 n)

 block4_sepconv1 (SeparableConv  (None, 12, 12, 728)  188672     ['block4_sepconv1_act[0]
[0]']
 2D)

 block4_sepconv1_bn (BatchNorma  (None, 12, 12, 728)  2912       ['block4_sepconv1[0][0]'
]
 lization)

 block4_sepconv2_act (Activatio  (None, 12, 12, 728)  0          ['block4_sepconv1_bn[0][
0]']
 n)

 block4_sepconv2 (SeparableConv  (None, 12, 12, 728)  536536     ['block4_sepconv2_act[0]
[0]']
 2D)

 block4_sepconv2_bn (BatchNorma  (None, 12, 12, 728)  2912       ['block4_sepconv2[0][0]'
]
 lization)

 conv2d_11 (Conv2D)             (None, 6, 6, 728)    186368      ['add_1[0][0]']

 block4_pool (MaxPooling2D)     (None, 6, 6, 728)    0           ['block4_sepconv2_bn[0][
0]']

 batch_normalization_2 (BatchNo  (None, 6, 6, 728)   2912        ['conv2d_11[0][0]']
 rmalization)
```

```
 add_2 (Add)                    (None, 6, 6, 728)    0           ['block4_pool[0][0]',
                                                                  'batch_normalization_
2[0][0]']

 block5_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_2[0][0]']
 n)

 block5_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv1_act[0]
[0]']
 2D)

 block5_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv1[0][0]'
]
 lization)

 block5_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block5_sepconv1_bn[0][
0]']
 n)

 block5_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv2_act[0]
[0]']
 2D)

 block5_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv2[0][0]'
]
 lization)

 block5_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block5_sepconv2_bn[0][
0]']
 n)

 block5_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv3_act[0]
[0]']
 2D)

 block5_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv3[0][0]'
]
 lization)

 add_3 (Add)                    (None, 6, 6, 728)    0           ['block5_sepconv3_bn[0]
[0]',
                                                                  'add_2[0][0]']

 block6_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_3[0][0]']
 n)
```

```
 block6_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv1_act[0]
[0]']
 2D)


 block6_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv1[0][0]'
]
 lization)


 block6_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block6_sepconv1_bn[0][
0]']
 n)


 block6_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv2_act[0]
[0]']
 2D)


 block6_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv2[0][0]'
]
 lization)


 block6_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block6_sepconv2_bn[0][
0]']
 n)


 block6_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv3_act[0]
[0]']
 2D)


 block6_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv3[0][0]'
]
 lization)


 add_4 (Add)                    (None, 6, 6, 728)    0           ['block6_sepconv3_bn[0]
[0]',
                                                                 'add_3[0][0]']


 block7_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_4[0][0]']

 n)


 block7_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv1_act[0]
[0]']
 2D)


 block7_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv1[0][0]'
]
 lization)
```

```
 block7_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block7_sepconv1_bn[0][
0]']
 n)

 block7_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv2_act[0]
[0]']
 2D)

 block7_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv2[0][0]'
]
 lization)

 block7_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block7_sepconv2_bn[0][
0]']
 n)

 block7_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv3_act[0]
[0]']
 2D)

 block7_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv3[0][0]'
]
 lization)

 add_5 (Add)                    (None, 6, 6, 728)    0           ['block7_sepconv3_bn[0]
[0]',
                                                                  'add_4[0][0]']

 block8_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_5[0][0]']

 n)

 block8_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv1_act[0]
[0]']
 2D)

 block8_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv1[0][0]'
]
 lization)

 block8_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block8_sepconv1_bn[0][
0]']
 n)

 block8_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv2_act[0]
[0]']
 2D)
```

```
 block8_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv2[0][0]'
]
 lization)

 block8_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block8_sepconv2_bn[0][
0]']
 n)

 block8_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv3_act[0]
[0]']
 2D)

 block8_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv3[0][0]'
]
 lization)

 add_6 (Add)                    (None, 6, 6, 728)    0           ['block8_sepconv3_bn[0]
[0]',
                                                                  'add_5[0][0]']

 block9_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_6[0][0]']

 n)

 block9_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv1_act[0]
[0]']
 2D)

 block9_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv1[0][0]'
]
 lization)

 block9_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block9_sepconv1_bn[0][
0]']
 n)

 block9_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv2_act[0]
[0]']
 2D)

 block9_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv2[0][0]'
]
 lization)

 block9_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block9_sepconv2_bn[0][
0]']
 n)
```

```
 block9_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv3_act[0]
[0]']
 2D)

 block9_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv3[0][0]'
]
 lization)

 add_7 (Add)                    (None, 6, 6, 728)    0           ['block9_sepconv3_bn[0]
[0]',
                                                                  'add_6[0][0]']

 block10_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_7[0][0]']

 on)

 block10_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv1_act[0
][0]']
 v2D)

 block10_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv1[0][0]
']
 alization)

 block10_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block10_sepconv1_bn[0]
[0]']
 on)

 block10_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv2_act[0
][0]']
 v2D)

 block10_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv2[0][0]
']
 alization)

 block10_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block10_sepconv2_bn[0]
[0]']
 on)

 block10_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv3_act[0
][0]']
 v2D)

 block10_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv3[0][0]
']
 alization)
```

```
 add_8 (Add)                    (None, 6, 6, 728)    0           ['block10_sepconv3_bn[0
][0]',
                                                                  'add_7[0][0]']

 block11_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_8[0][0]']
 on)

 block11_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv1_act[0
][0]']
 v2D)

 block11_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv1[0][0]
']
 alization)

 block11_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block11_sepconv1_bn[0]
[0]']
 on)

 block11_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv2_act[0
][0]']
 v2D)

 block11_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv2[0][0]
']
 alization)

 block11_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block11_sepconv2_bn[0]
[0]']
 on)

 block11_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv3_act[0
][0]']
 v2D)

 block11_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv3[0][0]
']
 alization)

 add_9 (Add)                    (None, 6, 6, 728)    0           ['block11_sepconv3_bn[0
][0]',
                                                                  'add_8[0][0]']

 block12_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_9[0][0]']
 on)
```

```
 block12_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv1_act[0
][0]']
 v2D)

 block12_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv1[0][0]
']
 alization)

 block12_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block12_sepconv1_bn[0]
[0]']
 on)

 block12_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv2_act[0
][0]']
 v2D)

 block12_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv2[0][0]
']
 alization)

 block12_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block12_sepconv2_bn[0]
[0]']
 on)

 block12_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv3_act[0
][0]']
 v2D)

 block12_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv3[0][0]
']
 alization)

 add_10 (Add)                   (None, 6, 6, 728)    0           ['block12_sepconv3_bn[0
][0]',
                                                                  'add_9[0][0]']

 block13_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_10[0][0]']

 on)

 block13_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block13_sepconv1_act[0
][0]']
 v2D)

 block13_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block13_sepconv1[0][0]
']
 alization)
```

```
 block13_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block13_sepconv1_bn[0]
[0]']
 on)

 block13_sepconv2 (SeparableCon  (None, 6, 6, 1024)  752024      ['block13_sepconv2_act[0
][0]']
 v2D)

 block13_sepconv2_bn (BatchNorm  (None, 6, 6, 1024)  4096        ['block13_sepconv2[0][0]
']
 alization)

 conv2d_12 (Conv2D)             (None, 3, 3, 1024)   745472      ['add_10[0][0]']

 block13_pool (MaxPooling2D)    (None, 3, 3, 1024)   0           ['block13_sepconv2_bn[0]
[0]']

 batch_normalization_3 (BatchNo  (None, 3, 3, 1024)  4096        ['conv2d_12[0][0]']

 rmalization)

 add_11 (Add)                   (None, 3, 3, 1024)   0           ['block13_pool[0][0]',

                                                                  'batch_normalization_
3[0][0]']

 block14_sepconv1 (SeparableCon  (None, 3, 3, 1536)  1582080     ['add_11[0][0]']

 v2D)

 block14_sepconv1_bn (BatchNorm  (None, 3, 3, 1536)  6144        ['block14_sepconv1[0][0]
']
 alization)

 block14_sepconv1_act (Activati  (None, 3, 3, 1536)  0           ['block14_sepconv1_bn[0]
[0]']
 on)

 block14_sepconv2 (SeparableCon  (None, 3, 3, 2048)  3159552     ['block14_sepconv1_act[0
][0]']
 v2D)

 block14_sepconv2_bn (BatchNorm  (None, 3, 3, 2048)  8192        ['block14_sepconv2[0][0]
']
 alization)

 block14_sepconv2_act (Activati  (None, 3, 3, 2048)  0           ['block14_sepconv2_bn[0]
[0]']
```

on)

```
=================================================================================================
=========
Total params: 20,861,480
Trainable params: 20,806,952
Non-trainable params: 54,528
_________________________________________________________________________________________________
_________
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 xception (Functional)       (None, 3, 3, 2048)        20861480

 flatten_3 (Flatten)         (None, 18432)             0

 dense_6 (Dense)             (None, 256)               4718848

 dense_7 (Dense)             (None, 2)                 514

=================================================================
Total params: 25,580,842
Trainable params: 4,719,362
Non-trainable params: 20,861,480
_________________________________________________________________
```

In [ ]:

```
xception_pre_trained_model.compile(Adam(learning_rate=0.0001), loss='binary_crossentropy'
,
              metrics=['accuracy'])

xception_pre_trained_model_history = xception_pre_trained_model.fit_generator(train_gener
ator,
                    steps_per_epoch=amount_train_steps,
                    validation_data=validation_generator,
                    validation_steps=amount_validation_steps,
                    epochs=30, verbose=1)
```

```
Epoch 1/30
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:8: UserWarning: `Model.fit_g
enerator` is deprecated and will be removed in a future version. Please use `Model.fit`,
which supports generators.

8500/8500 [==============================] - 502s 59ms/step - loss: 0.5454 - accuracy: 0.
7577 - val_loss: 0.4832 - val_accuracy: 0.7731
Epoch 2/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.4813 - accuracy: 0.
7723 - val_loss: 0.5011 - val_accuracy: 0.7590
Epoch 3/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4745 - accuracy: 0.
7757 - val_loss: 0.4839 - val_accuracy: 0.7726
Epoch 4/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.4697 - accuracy: 0.
7775 - val_loss: 0.4664 - val_accuracy: 0.7824
Epoch 5/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.4651 - accuracy: 0.
7801 - val_loss: 0.4653 - val_accuracy: 0.7829
Epoch 6/30
8500/8500 [==============================] - 497s 58ms/step - loss: 0.4633 - accuracy: 0.
7817 - val_loss: 0.4716 - val_accuracy: 0.7773
Epoch 7/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4603 - accuracy: 0.
7826 - val_loss: 0.4637 - val_accuracy: 0.7849
Epoch 8/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.4583 - accuracy: 0.
7838 - val_loss: 0.4571 - val_accuracy: 0.7877
Epoch 9/30
```

```
Epoch 9/30
8500/8500 [==============================] - 498s 59ms/step - loss: 0.4572 - accuracy: 0.
7840 - val_loss: 0.4588 - val_accuracy: 0.7848
Epoch 10/30
8500/8500 [==============================] - 497s 58ms/step - loss: 0.4551 - accuracy: 0.
7858 - val_loss: 0.4581 - val_accuracy: 0.7839
Epoch 11/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4545 - accuracy: 0.
7854 - val_loss: 0.4550 - val_accuracy: 0.7885
Epoch 12/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4529 - accuracy: 0.
7880 - val_loss: 0.4570 - val_accuracy: 0.7886
Epoch 13/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4528 - accuracy: 0.
7869 - val_loss: 0.4563 - val_accuracy: 0.7856
Epoch 14/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4511 - accuracy: 0.
7868 - val_loss: 0.4566 - val_accuracy: 0.7876
Epoch 15/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4501 - accuracy: 0.
7890 - val_loss: 0.4524 - val_accuracy: 0.7891
Epoch 16/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4493 - accuracy: 0.
7882 - val_loss: 0.4514 - val_accuracy: 0.7902
Epoch 17/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4470 - accuracy: 0.
7900 - val_loss: 0.4661 - val_accuracy: 0.7794
Epoch 18/30
8500/8500 [==============================] - 502s 59ms/step - loss: 0.4482 - accuracy: 0.
7900 - val_loss: 0.4489 - val_accuracy: 0.7924
Epoch 19/30
8500/8500 [==============================] - 501s 59ms/step - loss: 0.4471 - accuracy: 0.
7891 - val_loss: 0.4489 - val_accuracy: 0.7888
Epoch 20/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4454 - accuracy: 0.
7909 - val_loss: 0.4517 - val_accuracy: 0.7879
Epoch 21/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4449 - accuracy: 0.
7908 - val_loss: 0.4545 - val_accuracy: 0.7892
Epoch 22/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4441 - accuracy: 0.
7913 - val_loss: 0.4533 - val_accuracy: 0.7866
Epoch 23/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4454 - accuracy: 0.
7911 - val_loss: 0.4489 - val_accuracy: 0.7894
Epoch 24/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4440 - accuracy: 0.
7925 - val_loss: 0.4473 - val_accuracy: 0.7945
Epoch 25/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4437 - accuracy: 0.
7918 - val_loss: 0.4598 - val_accuracy: 0.7888
Epoch 26/30
8500/8500 [==============================] - 501s 59ms/step - loss: 0.4439 - accuracy: 0.
7916 - val_loss: 0.4528 - val_accuracy: 0.7899
Epoch 27/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4428 - accuracy: 0.
7919 - val_loss: 0.4453 - val_accuracy: 0.7904
Epoch 28/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4423 - accuracy: 0.
7925 - val_loss: 0.4480 - val_accuracy: 0.7905
Epoch 29/30
8500/8500 [==============================] - 499s 59ms/step - loss: 0.4415 - accuracy: 0.
7935 - val_loss: 0.4558 - val_accuracy: 0.7892
Epoch 30/30
8500/8500 [==============================] - 500s 59ms/step - loss: 0.4412 - accuracy: 0.
7939 - val_loss: 0.4611 - val_accuracy: 0.7840
```

In [ ]:

```
# Guardamos el modelo entrenado en nuestro directorio de google drive
xception_pre_trained_model.save(BASE_FOLDER+"XCEPTION/best_transfer_learning_xception_mod
el.h5")
```

```
# Guardamos los datos sacados durante el entrenamiento para plotear las graficas
with open(BASE_FOLDER + 'XCEPTION/best_transfer_learning_xception_training', 'wb') as fil
e_pi:
        pickle.dump(xception_pre_trained_model_history.history, file_pi)
```

Como hemos hecho en el resto de apartados anteriores, una vez que tenemos el modelo entrenado, vamos a proceder a probar-lo contra los datos de validación, ya que los de test no tienen etiqueta de clase. Para ello, como tenemos el model guardado en google drive, teniendo en cuenta que el ipynb se puede reiniciar y perder las variables ya inicializadas. Primero debemos cargarlo directamente desde google drive y después ya podemos realizar las predicciones.

Vamos a utilizar la función "load_model" de keras que nos permite recuperar un modelo guardado en formato .h5. Para cargar el modelo guardado, primero debemos generar la arquitecta para después introducirle los pesos ya entrenados.

También vamos a aprovechar y con la librería de "pickle" vamos a guardar los resultados de las predicciones, así podremos recuperar dichas predicciones y sus gráficas correspondientes una vez cerrado el notebook.

In [ ]:

```
# Recuperamos el modelo del fichero guardado en google drive
xception_pre_trained_model = load_model(BASE_FOLDER+"XCEPTION/best_transfer_learning_xcep
tion_model.h5")

# Ahora vamos a pasar a evaluar el modelo, para ello como los datos de test no estan etqi
uetados con sus classes, vamos a hacer uso del conjunto de validación que si esta etiquet
ado
labelNames = ["No Tumor", "Tumor"]

# Efectuamos las predicciónes (empleamos el mismo valor de batch_size que en training)
predictions = xception_pre_trained_model.predict(test_generator,steps=len(images_datafram
e_validation), verbose=1)
```

```
34000/34000 [==============================] - 307s 9ms/step
```

In [ ]:

```
# Vamos a guardar el fichero con las predicciones realizadas por nuestro modelo
with open(BASE_FOLDER + 'XCEPTION/best_transfer_learning_xception_predictions', 'wb') as
file_pi:
        pickle.dump(predictions, file_pi)
```

Al igual que en el resto de apartados anteriores, para evaluar el comportamiento de nuestra red entrenada, a partir de las predicciones realizadas. Vamos a mostrar la métrica de "Receiver Operator Characteristic" -> "ROC", una métrica de evaluación para clasificación binaria. Dibujando dicha curva y calculando el área debajo de esta conocida como "AUC" podremos saber la capacidad de nuestra red para distinguir entre las clases.

Para conseguir los valores de dichas métricas vamos a utilizar la función de "roc_auc_score" de la librería de sklearn.

Junto con la curva "ROC" y su área debajo "AUC", vamos a crear y a mostrar por pantalla la matriz de confusión, que de manera visual y rápida nos va a permitir visualizar la eficacia de nuestra red. Para ello vamos a emplear la función propia de la librería de sklearn ConfusionMatrixDisplay: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las predicciones que nos va a indicar la precisión, el "recall", el "f1-score" y el "support" conseguidos con las predicciones de nuestro modelo entrenado.

In [ ]:

```
# Primero de todo debemos cargar los datos de prediccion que hemos guardado en el bloque
anterior
file = open(BASE_FOLDER + 'XCEPTION/best_transfer_learning_xception_predictions','rb')
predictions = pickle.load(file)
file.close()
```

```
# Creamos un dataframe con las probabilidades de las predicciones y las clases
model_predictions_dataframe = pd.DataFrame(predictions, columns=['0_no_tumor', '1_tumor'
])
print("Sample de los valores sobre algunas predicciones:")
display(model_predictions_dataframe.head())
# Guardamos las clases predichas
image_aviable_classes = test_generator.classes

# Para calcular la AUC necesitamos saber las predicciones de la clase positiva, para ello
vamos a elegir la columna tumor del dataframe creado previamente "model_predictions_dataf
rame"
positive_model_predictions = model_predictions_dataframe['1_tumor']

# Ahora ya podemos mostrar la gráfica
print("Área bajo la curva ROC:"+ str(roc_auc_score(image_aviable_classes, positive_model_
predictions)))
```

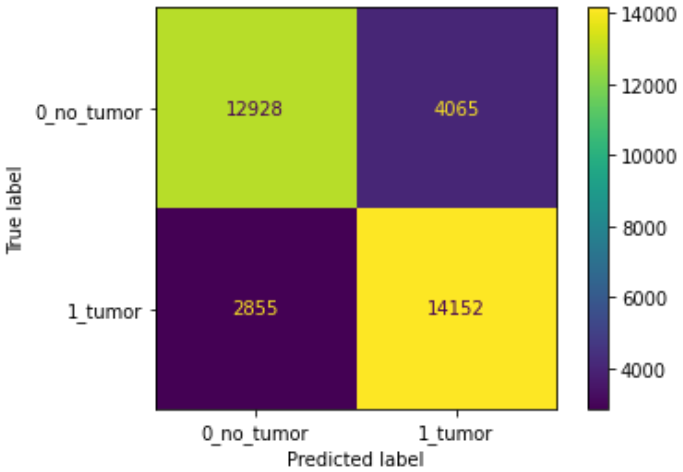
Sample de los valores sobre algunas predicciones:

	0_no_tumor	1_tumor
0	**0.991394**	**0.008606**
1	**0.693595**	**0.306405**
2	**0.777439**	**0.222561**
3	**0.994712**	**0.005288**
4	**0.395255**	**0.604745**

Área bajo la curva ROC:0.8829489818840833

In [ ]:

```
# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argma
x(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```



In [ ]:

```
# Sacamos el reporte para el subconjunto de validación, utilizando la función de classifi
cation_report
# Como tenemos las predicciones de las clases como probabilidades debido a la función de
activación de la última capa "softmax", debemos pasarlo a binario (0 o 1)
class_pred_binary = predictions.argmax(axis=1)

print(classification_report(image_aviable_classes, class_pred_binary, target_names=label
Names))
```

```
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

    No Tumor       0.82      0.76      0.79     16993
       Tumor       0.78      0.83      0.80     17007

    accuracy                           0.80     34000
   macro avg       0.80      0.80      0.80     34000
weighted avg       0.80      0.80      0.80     34000
```

Repitiendo, al igual que en todos los apartados anteriores, como también hemos guardado en google drive, los parámetros sacados durante el entrenamiento de nuestra red neuronal. Ahora podemos a recuperarlos y a mostrar diferentes gráficas para discutir de una forma visual los resultados obtenidos época a época.

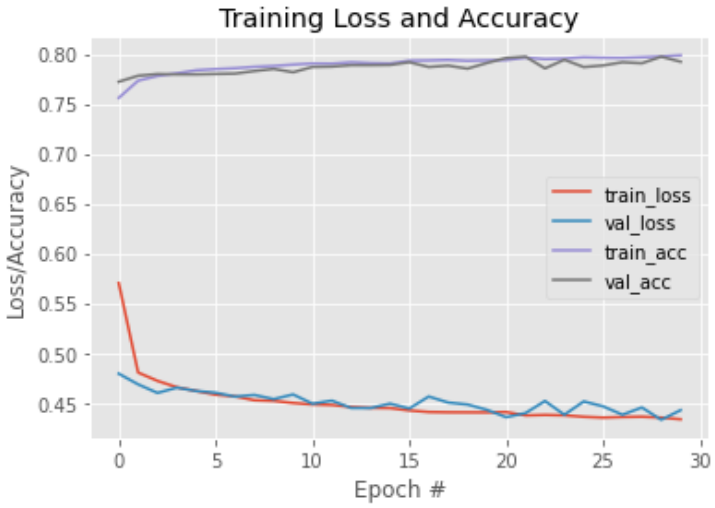
Los datos que vamos a mostrar en las gráficas son los siguientes:

- **Train Loss: Perdidas con el dataset de train**
- **Validation Loss: Perdidas con el dataset de validation**
- **Train accuracy: Precisión con el dataset de train**
- **Validation accuracy: Precisión con el dataset de train**

Debemos tener en cuenta que tanto las perdidas como la precisión entre train y validation deben de ser similares, de no ser así, estaríamos delante de un problema de overfitting.

In [ ]:

```
file = open(BASE_FOLDER + 'XCEPTION/best_transfer_learning_xception_training','rb')
history = pickle.load(file)

# Gráficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), history["loss"], label="train_loss")
plt.plot(np.arange(0, 30), history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 30), history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 30), history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

file.close()
```



Como podemos observar a través de las gráficas y de las métricas extraídas de las predicciones del modelo, usando solamente transfer learning sin descongelar ninguna capa ni bloque de la red Xception, conseguimos unos resultados decentes tirando a normales, aproximadamente un 0.798% de precisión, pero que no son ni de lejos los mejores dentro de las redes que hemos entrenado.

Por ejemplo, el mismo procedimiento con la red VGG19, nos da mejores resultados, cosa obvia debido a que en el concurso de Imagenet la red VGG19 ya obtuvo mejores resultados que Xcpetion.

Xception Fine tunning

Como podemos ver con solo transfer learning, básicamente substituyendo el top model de la red de Xception, por uno simple hecho por nosotros. Conseguimos unos resultados decentes, pero vamos a intentar mejorar estos resultados aplicando aparte de trasnfer learning, tecnicas de fine tuning que significa descongelando alguna de las capas de la red pre-entrenada para que dichas capas se reentrenen junto con el top model que nosotros le pasamos.

Para ello primero vamos a ver las capas de las que dispone la red de Xception y a decidir cuáles de ellas vamos a descongelar.

Primero ello vamos a reimportar la red de Xception limpia para no tener ningún conflicto con la usada en el apartado anterior.

In [ ]:

```
# Importamos el base model de la red Xception, sin el top model y cambiando el input shap
e
xception_base_model = Xception(weights='imagenet',
                 include_top=False,
                 input_shape=(96,96,3))

xception_base_model.summary()

# Como podemos observar el base model de la red de Xception consta de 14 bloques 11 de el
los Convolucionales, entre ellos el bloque 14 el cual nosotros vamos a dejar libre para q
ue pueda ser re entrenado y asi comparar
# el resultado con el obtenido sin aplicar ningún tipo de fine tunning
for layer in xception_base_model.layers:
  if 'block14_sepconv' in layer.name:
    print('Bloque ' + layer.name + ' no congelado...')
    break
  layer.trainable = False
  print('Capa ' + layer.name + ' congelada...')

# Ahora una vez ya tenemos el base model con los bloques congelados que queremos, simplem
ente tenemos que juntarlo con neustro top model y volver a entrenar para ver si realmente
ha funcionadi y conseguiremos un mejor
# resultado o no

# Vamos a definir el top model para que termine dandonos la solución de nuestras dos clas
es
xception_pre_trained_with_fine_tunning_model = Sequential()
xception_pre_trained_with_fine_tunning_model.add(xception_base_model)
xception_pre_trained_with_fine_tunning_model.add(Flatten())
xception_pre_trained_with_fine_tunning_model.add(Dense(256, activation='relu'))
xception_pre_trained_with_fine_tunning_model.add(Dense(2, activation='softmax'))

xception_pre_trained_with_fine_tunning_model.summary()
```

```
Model: "xception"
__________________________________________________________________________________________
_________
 Layer (type)                Output Shape              Param #     Connected to

==========================================================================================
=========
 input_4 (InputLayer)        [(None, 96, 96, 3)]       0           []


 block1_conv1 (Conv2D)       (None, 47, 47, 32)        864         ['input_4[0][0]']


 block1_conv1_bn (BatchNormaliz  (None, 47, 47, 32)    128         ['block1_conv1[0][0]']

 ation)
```

```
 block1_conv1_act (Activation)  (None, 47, 47, 32)   0           ['block1_conv1_bn[0][0]'
]

 block1_conv2 (Conv2D)          (None, 45, 45, 64)   18432       ['block1_conv1_act[0][0]
']

 block1_conv2_bn (BatchNormaliz  (None, 45, 45, 64)  256         ['block1_conv2[0][0]']
 ation)

 block1_conv2_act (Activation)  (None, 45, 45, 64)   0           ['block1_conv2_bn[0][0]'
]

 block2_sepconv1 (SeparableConv  (None, 45, 45, 128)  8768       ['block1_conv2_act[0][0]
']
 2D)

 block2_sepconv1_bn (BatchNorma  (None, 45, 45, 128)  512        ['block2_sepconv1[0][0]'
]
 lization)

 block2_sepconv2_act (Activatio  (None, 45, 45, 128)  0          ['block2_sepconv1_bn[0][
0]']
 n)

 block2_sepconv2 (SeparableConv  (None, 45, 45, 128)  17536      ['block2_sepconv2_act[0]
[0]']
 2D)

 block2_sepconv2_bn (BatchNorma  (None, 45, 45, 128)  512        ['block2_sepconv2[0][0]'
]
 lization)

 conv2d_13 (Conv2D)             (None, 23, 23, 128)  8192        ['block1_conv2_act[0][0]
']

 block2_pool (MaxPooling2D)     (None, 23, 23, 128)  0           ['block2_sepconv2_bn[0][
0]']

 batch_normalization_4 (BatchNo  (None, 23, 23, 128)  512        ['conv2d_13[0][0]']
 rmalization)

 add_12 (Add)                   (None, 23, 23, 128)  0           ['block2_pool[0][0]',
                                                                  'batch_normalization_
4[0][0]']

 block3_sepconv1_act (Activatio  (None, 23, 23, 128)  0          ['add_12[0][0]']
```

```
 n)


 block3_sepconv1 (SeparableConv  (None, 23, 23, 256)  33920      ['block3_sepconv1_act[0]
[0]']
 2D)


 block3_sepconv1_bn (BatchNorma  (None, 23, 23, 256)  1024       ['block3_sepconv1[0][0]'
]
 lization)


 block3_sepconv2_act (Activatio  (None, 23, 23, 256)  0          ['block3_sepconv1_bn[0][
0]']
 n)


 block3_sepconv2 (SeparableConv  (None, 23, 23, 256)  67840      ['block3_sepconv2_act[0]
[0]']
 2D)


 block3_sepconv2_bn (BatchNorma  (None, 23, 23, 256)  1024       ['block3_sepconv2[0][0]'
]
 lization)


 conv2d_14 (Conv2D)             (None, 12, 12, 256)  32768      ['add_12[0][0]']


 block3_pool (MaxPooling2D)     (None, 12, 12, 256)  0          ['block3_sepconv2_bn[0][
0]']


 batch_normalization_5 (BatchNo  (None, 12, 12, 256)  1024       ['conv2d_14[0][0]']

 rmalization)


 add_13 (Add)                   (None, 12, 12, 256)  0          ['block3_pool[0][0]',

                                                                  'batch_normalization_
5[0][0]']


 block4_sepconv1_act (Activatio  (None, 12, 12, 256)  0          ['add_13[0][0]']

 n)


 block4_sepconv1 (SeparableConv  (None, 12, 12, 728)  188672     ['block4_sepconv1_act[0]
[0]']
 2D)


 block4_sepconv1_bn (BatchNorma  (None, 12, 12, 728)  2912       ['block4_sepconv1[0][0]'
]
 lization)
```

```
 block4_sepconv2_act (Activatio  (None, 12, 12, 728)  0          ['block4_sepconv1_bn[0][
0]']
 n)

 block4_sepconv2 (SeparableConv  (None, 12, 12, 728)  536536     ['block4_sepconv2_act[0]
[0]']
 2D)

 block4_sepconv2_bn (BatchNorma  (None, 12, 12, 728)  2912       ['block4_sepconv2[0][0]'
]
 lization)

 conv2d_15 (Conv2D)             (None, 6, 6, 728)    186368      ['add_13[0][0]']

 block4_pool (MaxPooling2D)     (None, 6, 6, 728)    0           ['block4_sepconv2_bn[0][
0]']

 batch_normalization_6 (BatchNo  (None, 6, 6, 728)   2912        ['conv2d_15[0][0]']
 rmalization)

 add_14 (Add)                   (None, 6, 6, 728)    0           ['block4_pool[0][0]',
                                                                  'batch_normalization_
6[0][0]']

 block5_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_14[0][0]']
 n)

 block5_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv1_act[0]
[0]']
 2D)

 block5_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv1[0][0]'
]
 lization)

 block5_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block5_sepconv1_bn[0][
0]']
 n)

 block5_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv2_act[0]
[0]']
 2D)

 block5_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv2[0][0]'
]
 lization)
```

```
 block5_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block5_sepconv2_bn[0][
0]']
 n)

 block5_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block5_sepconv3_act[0]
[0]']
 2D)

 block5_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block5_sepconv3[0][0]'
]
 lization)

 add_15 (Add)                   (None, 6, 6, 728)    0           ['block5_sepconv3_bn[0]
[0]',
                                                                  'add_14[0][0]']

 block6_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_15[0][0]']

 n)

 block6_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv1_act[0]
[0]']
 2D)

 block6_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv1[0][0]'
]
 lization)

 block6_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block6_sepconv1_bn[0][
0]']
 n)

 block6_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv2_act[0]
[0]']
 2D)

 block6_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv2[0][0]'
]
 lization)

 block6_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block6_sepconv2_bn[0][
0]']
 n)

 block6_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block6_sepconv3_act[0]
[0]']
 2D)
```

```
 block6_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block6_sepconv3[0][0]'
]
 lization)

 add_16 (Add)                    (None, 6, 6, 728)    0           ['block6_sepconv3_bn[0]
[0]',
                                                                  'add_15[0][0]']

 block7_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_16[0][0]']

 n)

 block7_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv1_act[0]
[0]']
 2D)

 block7_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv1[0][0]'
]
 lization)

 block7_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block7_sepconv1_bn[0][
0]']
 n)

 block7_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv2_act[0]
[0]']
 2D)

 block7_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv2[0][0]'
]
 lization)

 block7_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block7_sepconv2_bn[0][
0]']
 n)

 block7_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block7_sepconv3_act[0]
[0]']
 2D)

 block7_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block7_sepconv3[0][0]'
]
 lization)

 add_17 (Add)                    (None, 6, 6, 728)    0           ['block7_sepconv3_bn[0]
[0]',
                                                                  'add_16[0][0]']
```

```
 block8_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_17[0][0]']
 n)

 block8_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv1_act[0]
[0]']
 2D)

 block8_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv1[0][0]'
]
 lization)

 block8_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block8_sepconv1_bn[0][
0]']
 n)

 block8_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv2_act[0]
[0]']
 2D)

 block8_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv2[0][0]'
]
 lization)

 block8_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block8_sepconv2_bn[0][
0]']
 n)

 block8_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block8_sepconv3_act[0]
[0]']
 2D)

 block8_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block8_sepconv3[0][0]'
]
 lization)

 add_18 (Add)                   (None, 6, 6, 728)    0           ['block8_sepconv3_bn[0]
[0]',
                                                                 'add_17[0][0]']

 block9_sepconv1_act (Activatio  (None, 6, 6, 728)   0           ['add_18[0][0]']
 n)

 block9_sepconv1 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv1_act[0]
[0]']
 2D)
```

```
 block9_sepconv1_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv1[0][0]'
]
 lization)

 block9_sepconv2_act (Activatio  (None, 6, 6, 728)   0           ['block9_sepconv1_bn[0][
0]']
 n)

 block9_sepconv2 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv2_act[0]
[0]']
 2D)

 block9_sepconv2_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv2[0][0]'
]
 lization)

 block9_sepconv3_act (Activatio  (None, 6, 6, 728)   0           ['block9_sepconv2_bn[0][
0]']
 n)

 block9_sepconv3 (SeparableConv  (None, 6, 6, 728)   536536      ['block9_sepconv3_act[0]
[0]']
 2D)

 block9_sepconv3_bn (BatchNorma  (None, 6, 6, 728)   2912        ['block9_sepconv3[0][0]'
]
 lization)

 add_19 (Add)                   (None, 6, 6, 728)    0           ['block9_sepconv3_bn[0]
[0]',
                                                                  'add_18[0][0]']

 block10_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_19[0][0]']

 on)

 block10_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv1_act[0
][0]']
 v2D)

 block10_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv1[0][0]
']
 alization)

 block10_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block10_sepconv1_bn[0]
[0]']
 on)
```

```
 block10_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv2_act[0
][0]']
 v2D)

 block10_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv2[0][0]
']
 alization)

 block10_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block10_sepconv2_bn[0]
[0]']
 on)

 block10_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block10_sepconv3_act[0
][0]']
 v2D)

 block10_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block10_sepconv3[0][0]
']
 alization)

 add_20 (Add)                   (None, 6, 6, 728)    0           ['block10_sepconv3_bn[0
][0]',
                                                                  'add_19[0][0]']

 block11_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_20[0][0]']

 on)

 block11_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv1_act[0
][0]']
 v2D)

 block11_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv1[0][0]
']
 alization)

 block11_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block11_sepconv1_bn[0]
[0]']
 on)

 block11_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv2_act[0
][0]']
 v2D)

 block11_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv2[0][0]
']
 alization)
```

```
 block11_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block11_sepconv2_bn[0]
[0]']
 on)

 block11_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block11_sepconv3_act[0
][0]']
 v2D)

 block11_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block11_sepconv3[0][0]
']
 alization)

 add_21 (Add)                   (None, 6, 6, 728)    0           ['block11_sepconv3_bn[0
][0]',
                                                                  'add_20[0][0]']

 block12_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_21[0][0]']

 on)

 block12_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv1_act[0
][0]']
 v2D)

 block12_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv1[0][0]
']
 alization)

 block12_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block12_sepconv1_bn[0]
[0]']
 on)

 block12_sepconv2 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv2_act[0
][0]']
 v2D)

 block12_sepconv2_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv2[0][0]
']
 alization)

 block12_sepconv3_act (Activati  (None, 6, 6, 728)   0           ['block12_sepconv2_bn[0]
[0]']
 on)

 block12_sepconv3 (SeparableCon  (None, 6, 6, 728)   536536      ['block12_sepconv3_act[0
][0]']
 v2D)
```

```
 block12_sepconv3_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block12_sepconv3[0][0]
']
 alization)

 add_22 (Add)                   (None, 6, 6, 728)    0           ['block12_sepconv3_bn[0
][0]',
                                                                  'add_21[0][0]']

 block13_sepconv1_act (Activati  (None, 6, 6, 728)   0           ['add_22[0][0]']
 on)

 block13_sepconv1 (SeparableCon  (None, 6, 6, 728)   536536      ['block13_sepconv1_act[0
][0]']
 v2D)

 block13_sepconv1_bn (BatchNorm  (None, 6, 6, 728)   2912        ['block13_sepconv1[0][0]
']
 alization)

 block13_sepconv2_act (Activati  (None, 6, 6, 728)   0           ['block13_sepconv1_bn[0]
[0]']
 on)

 block13_sepconv2 (SeparableCon  (None, 6, 6, 1024)  752024      ['block13_sepconv2_act[0
][0]']
 v2D)

 block13_sepconv2_bn (BatchNorm  (None, 6, 6, 1024)  4096        ['block13_sepconv2[0][0]
']
 alization)

 conv2d_16 (Conv2D)             (None, 3, 3, 1024)   745472      ['add_22[0][0]']

 block13_pool (MaxPooling2D)    (None, 3, 3, 1024)   0           ['block13_sepconv2_bn[0]
[0]']

 batch_normalization_7 (BatchNo  (None, 3, 3, 1024)  4096        ['conv2d_16[0][0]']
 rmalization)

 add_23 (Add)                   (None, 3, 3, 1024)   0           ['block13_pool[0][0]',
                                                                  'batch_normalization_
7[0][0]']

 block14_sepconv1 (SeparableCon  (None, 3, 3, 1536)  1582080     ['add_23[0][0]']
```

```
 v2D)


 block14_sepconv1_bn (BatchNorm  (None, 3, 3, 1536)  6144        ['block14_sepconv1[0][0]
']
 alization)


 block14_sepconv1_act (Activati  (None, 3, 3, 1536)  0           ['block14_sepconv1_bn[0]
[0]']
 on)


 block14_sepconv2 (SeparableCon  (None, 3, 3, 2048)  3159552     ['block14_sepconv1_act[0
][0]']
 v2D)


 block14_sepconv2_bn (BatchNorm  (None, 3, 3, 2048)  8192        ['block14_sepconv2[0][0]
']
 alization)


 block14_sepconv2_act (Activati  (None, 3, 3, 2048)  0           ['block14_sepconv2_bn[0]
[0]']
 on)


==================================================================================================
=========
Total params: 20,861,480
Trainable params: 20,806,952
Non-trainable params: 54,528
__________________________________________________________________________________________________
_________
Capa input_4 congelada...
Capa block1_conv1 congelada...
Capa block1_conv1_bn congelada...
Capa block1_conv1_act congelada...
Capa block1_conv2 congelada...
Capa block1_conv2_bn congelada...
Capa block1_conv2_act congelada...
Capa block2_sepconv1 congelada...
Capa block2_sepconv1_bn congelada...
Capa block2_sepconv2_act congelada...
Capa block2_sepconv2 congelada...
Capa block2_sepconv2_bn congelada...
Capa conv2d_13 congelada...
Capa block2_pool congelada...
Capa batch_normalization_4 congelada...
Capa add_12 congelada...
Capa block3_sepconv1_act congelada...
Capa block3_sepconv1 congelada...
Capa block3_sepconv1_bn congelada...
Capa block3_sepconv2_act congelada...
Capa block3_sepconv2 congelada...
Capa block3_sepconv2_bn congelada...
Capa conv2d_14 congelada...
Capa block3_pool congelada...
Capa batch_normalization_5 congelada...
Capa add_13 congelada...
Capa block4_sepconv1_act congelada...
Capa block4_sepconv1 congelada...
Capa block4_sepconv1_bn congelada...
Capa block4_sepconv2_act congelada...
```

```
Capa block4_sepconv2 congelada...
Capa block4_sepconv2_bn congelada...
Capa conv2d_15 congelada...
Capa block4_pool congelada...
Capa batch_normalization_6 congelada...
Capa add_14 congelada...
Capa block5_sepconv1_act congelada...
Capa block5_sepconv1 congelada...
Capa block5_sepconv1_bn congelada...
Capa block5_sepconv2_act congelada...
Capa block5_sepconv2 congelada...
Capa block5_sepconv2_bn congelada...
Capa block5_sepconv3_act congelada...
Capa block5_sepconv3 congelada...
Capa block5_sepconv3_bn congelada...
Capa add_15 congelada...
Capa block6_sepconv1_act congelada...
Capa block6_sepconv1 congelada...
Capa block6_sepconv1_bn congelada...
Capa block6_sepconv2_act congelada...
Capa block6_sepconv2 congelada...
Capa block6_sepconv2_bn congelada...
Capa block6_sepconv3_act congelada...
Capa block6_sepconv3 congelada...
Capa block6_sepconv3_bn congelada...
Capa add_16 congelada...
Capa block7_sepconv1_act congelada...
Capa block7_sepconv1 congelada...
Capa block7_sepconv1_bn congelada...
Capa block7_sepconv2_act congelada...
Capa block7_sepconv2 congelada...
Capa block7_sepconv2_bn congelada...
Capa block7_sepconv3_act congelada...
Capa block7_sepconv3 congelada...
Capa block7_sepconv3_bn congelada...
Capa add_17 congelada...
Capa block8_sepconv1_act congelada...
Capa block8_sepconv1 congelada...
Capa block8_sepconv1_bn congelada...
Capa block8_sepconv2_act congelada...
Capa block8_sepconv2 congelada...
Capa block8_sepconv2_bn congelada...
Capa block8_sepconv3_act congelada...
Capa block8_sepconv3 congelada...
Capa block8_sepconv3_bn congelada...
Capa add_18 congelada...
Capa block9_sepconv1_act congelada...
Capa block9_sepconv1 congelada...
Capa block9_sepconv1_bn congelada...
Capa block9_sepconv2_act congelada...
Capa block9_sepconv2 congelada...
Capa block9_sepconv2_bn congelada...
Capa block9_sepconv3_act congelada...
Capa block9_sepconv3 congelada...
Capa block9_sepconv3_bn congelada...
Capa add_19 congelada...
Capa block10_sepconv1_act congelada...
Capa block10_sepconv1 congelada...
Capa block10_sepconv1_bn congelada...
Capa block10_sepconv2_act congelada...
Capa block10_sepconv2 congelada...
Capa block10_sepconv2_bn congelada...
Capa block10_sepconv3_act congelada...
Capa block10_sepconv3 congelada...
Capa block10_sepconv3_bn congelada...
Capa add_20 congelada...
Capa block11_sepconv1_act congelada...
Capa block11_sepconv1 congelada...
Capa block11_sepconv1_bn congelada...
Capa block11_sepconv2_act congelada...
Capa block11_sepconv2 congelada...
Capa block11_sepconv2_bn congelada...
```

```
Capa block11_sepconv3_act congelada...
Capa block11_sepconv3 congelada...
Capa block11_sepconv3_bn congelada...
Capa add_21 congelada...
Capa block12_sepconv1_act congelada...
Capa block12_sepconv1 congelada...
Capa block12_sepconv1_bn congelada...
Capa block12_sepconv2_act congelada...
Capa block12_sepconv2 congelada...
Capa block12_sepconv2_bn congelada...
Capa block12_sepconv3_act congelada...
Capa block12_sepconv3 congelada...
Capa block12_sepconv3_bn congelada...
Capa add_22 congelada...
Capa block13_sepconv1_act congelada...
Capa block13_sepconv1 congelada...
Capa block13_sepconv1_bn congelada...
Capa block13_sepconv2_act congelada...
Capa block13_sepconv2 congelada...
Capa block13_sepconv2_bn congelada...
Capa conv2d_16 congelada...
Capa block13_pool congelada...
Capa batch_normalization_7 congelada...
Capa add_23 congelada...
Bloque block14_sepconv1 no congelado...
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 xception (Functional)        (None, 3, 3, 2048)        20861480

 flatten_4 (Flatten)          (None, 18432)             0

 dense_8 (Dense)              (None, 256)               4718848

 dense_9 (Dense)              (None, 2)                 514

=================================================================
Total params: 25,580,842
Trainable params: 9,468,162
Non-trainable params: 16,112,680
_________________________________________________________________
```

In [ ]:

```
# Compilamos el modelo
xception_pre_trained_with_fine_tunning_model.compile(Adam(learning_rate=0.0001), loss='bi
nary_crossentropy',
              metrics=['accuracy'])
# Entrenamos el modelo
xception_pre_trained_with_fine_tunning_model_history = xception_pre_trained_with_fine_tun
ning_model.fit(train_generator,
                    steps_per_epoch=amount_train_steps,
                    validation_data=validation_generator,
                    validation_steps=amount_validation_steps,
                    epochs=30, verbose=1)
```

```
Epoch 1/30
8500/8500 [==============================] - 513s 60ms/step - loss: 0.4735 - accuracy: 0.
7769 - val_loss: 0.5071 - val_accuracy: 0.7449
Epoch 2/30
8500/8500 [==============================] - 511s 60ms/step - loss: 0.4361 - accuracy: 0.
7974 - val_loss: 0.4202 - val_accuracy: 0.8056
Epoch 3/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.4200 - accuracy: 0.
8062 - val_loss: 0.4215 - val_accuracy: 0.8051
Epoch 4/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.4088 - accuracy: 0.
8117 - val_loss: 0.4245 - val_accuracy: 0.8106
Epoch 5/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.3989 - accuracy: 0.
8168 - val_loss: 0.4263 - val_accuracy: 0.8073
```

```
Epoch 6/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3921 - accuracy: 0.
8208 - val_loss: 0.4453 - val_accuracy: 0.7937
Epoch 7/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3875 - accuracy: 0.
8233 - val_loss: 0.3775 - val_accuracy: 0.8288
Epoch 8/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3822 - accuracy: 0.
8255 - val_loss: 0.4049 - val_accuracy: 0.8199
Epoch 9/30
8500/8500 [==============================] - 508s 60ms/step - loss: 0.3779 - accuracy: 0.
8290 - val_loss: 0.3793 - val_accuracy: 0.8298
Epoch 10/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.3737 - accuracy: 0.
8311 - val_loss: 0.3763 - val_accuracy: 0.8311
Epoch 11/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.3692 - accuracy: 0.
8325 - val_loss: 0.3630 - val_accuracy: 0.8376
Epoch 12/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3668 - accuracy: 0.
8347 - val_loss: 0.4436 - val_accuracy: 0.7932
Epoch 13/30
8500/8500 [==============================] - 511s 60ms/step - loss: 0.3630 - accuracy: 0.
8372 - val_loss: 0.4493 - val_accuracy: 0.7915
Epoch 14/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3593 - accuracy: 0.
8394 - val_loss: 0.4382 - val_accuracy: 0.8042
Epoch 15/30
8500/8500 [==============================] - 509s 60ms/step - loss: 0.3575 - accuracy: 0.
8404 - val_loss: 0.4397 - val_accuracy: 0.7896
Epoch 16/30
8500/8500 [==============================] - 508s 60ms/step - loss: 0.3536 - accuracy: 0.
8421 - val_loss: 0.3761 - val_accuracy: 0.8326
Epoch 17/30
8500/8500 [==============================] - 510s 60ms/step - loss: 0.3535 - accuracy: 0.
8422 - val_loss: 0.3561 - val_accuracy: 0.8439
Epoch 18/30
8500/8500 [==============================] - 505s 59ms/step - loss: 0.3480 - accuracy: 0.
8445 - val_loss: 0.3742 - val_accuracy: 0.8331
Epoch 19/30
8500/8500 [==============================] - 505s 59ms/step - loss: 0.3468 - accuracy: 0.
8453 - val_loss: 0.3496 - val_accuracy: 0.8471
Epoch 20/30
8500/8500 [==============================] - 504s 59ms/step - loss: 0.3445 - accuracy: 0.
8466 - val_loss: 0.3861 - val_accuracy: 0.8359
Epoch 21/30
8500/8500 [==============================] - 505s 59ms/step - loss: 0.3412 - accuracy: 0.
8491 - val_loss: 0.3798 - val_accuracy: 0.8360
Epoch 22/30
8500/8500 [==============================] - 505s 59ms/step - loss: 0.3402 - accuracy: 0.
8485 - val_loss: 0.3665 - val_accuracy: 0.8411
Epoch 23/30
8500/8500 [==============================] - 506s 59ms/step - loss: 0.3378 - accuracy: 0.
8498 - val_loss: 0.3562 - val_accuracy: 0.8429
Epoch 24/30
8500/8500 [==============================] - 504s 59ms/step - loss: 0.3366 - accuracy: 0.
8507 - val_loss: 0.3442 - val_accuracy: 0.8494
Epoch 25/30
8500/8500 [==============================] - 518s 61ms/step - loss: 0.3346 - accuracy: 0.
8525 - val_loss: 0.3397 - val_accuracy: 0.8509
Epoch 26/30
8500/8500 [==============================] - 512s 60ms/step - loss: 0.3308 - accuracy: 0.
8539 - val_loss: 0.3583 - val_accuracy: 0.8458
Epoch 27/30
8500/8500 [==============================] - 511s 60ms/step - loss: 0.3290 - accuracy: 0.
8552 - val_loss: 0.3606 - val_accuracy: 0.8394
Epoch 28/30
8500/8500 [==============================] - 513s 60ms/step - loss: 0.3290 - accuracy: 0.
8549 - val_loss: 0.3864 - val_accuracy: 0.8276
Epoch 29/30
8500/8500 [==============================] - 514s 60ms/step - loss: 0.3254 - accuracy: 0.
8578 - val_loss: 0.3495 - val_accuracy: 0.8450
```

```
Epoch 30/30
8500/8500 [==============================] - 513s 60ms/step - loss: 0.3250 - accuracy: 0.
8568 - val_loss: 0.3553 - val_accuracy: 0.8461
```

In [ ]:

```
# Guardamos el modelo entrenado en nuestro directorio de google drive
xception_pre_trained_with_fine_tunning_model.save(BASE_FOLDER+"XCEPTION/best_fine_tunning
_xception_model.h5")

# Guardamos los datos sacados durante el entrenamiento para plotear las graficas
with open(BASE_FOLDER + 'XCEPTION/best_fine_tunning_xception_training', 'wb') as file_pi
:
        pickle.dump(xception_pre_trained_with_fine_tunning_model_history.history, file_p
i)
```

Por último, como hemos hecho en el resto de apartados anteriores, una vez que tenemos el modelo entrenado, vamos a proceder a probar-lo contra los datos de validación, ya que los de test no tienen etiqueta de clase. Para ello, como tenemos el modelo guardado en google drive, teniendo en cuenta que el ipynb se puede reiniciar y perder las variables ya inicializadas. Primero debemos cargarlo directamente desde google drive y después ya podemos realizar las predicciones.

Vamos a utilizar la función "load_model" de keras que nos permite recuperar un modelo guardado en formato .h5. Para cargar el modelo guardado, primero debemos generar la arquitecta para después introducirle los pesos ya entrenados.

También vamos a aprovechar y con la librería de "pickle" vamos a guardar los resultados de las predicciones, así podremos recuperar dichas predicciones y sus gráficas correspondientes una vez cerrado el notebook.

In [ ]:

```
# Recuperamos el modelo del fichero guardado en google drive
xception_pre_trained_with_fine_tunning_model = load_model(BASE_FOLDER+"XCEPTION/best_fine
_tunning_xception_model.h5")

# Ahora vamos a pasar a evaluar el modelo, para ello como los datos de test no estan etqi
uetados con sus classes, vamos a hacer uso del conjunto de validación que si esta etiquet
ado
labelNames = ["No Tumor", "Tumor"]

# Efectuamos las predicciónes (empleamos el mismo valor de batch_size que en training)
predictions = xception_pre_trained_with_fine_tunning_model.predict(test_generator,steps=l
en(images_dataframe_validation), verbose=1)
```

```
34000/34000 [==============================] - 275s 8ms/step
```

In [ ]:

```
# Vamos a guardar el fichero con las predicciones realizadas por nuestro modelo
with open(BASE_FOLDER + 'XCEPTION/best_fine_tunning_xception_predictions', 'wb') as file_
pi:
        pickle.dump(predictions, file_pi)
```

Por último, al igual que en el resto de apartados anteriores, para evaluar el comportamiento de nuestra red entrenada, a partir de las predicciones realizadas. Vamos a mostrar la métrica de "Receiver Operator Characteristic" -> "ROC", una métrica de evaluación para clasificación binaria. Dibujando dicha curva y calculando el área debajo de esta conocida como "AUC" podremos saber la capacidad de nuestra red para distinguir entre las clases.

Para conseguir los valores de dichas métricas vamos a utilizar la función de "roc_auc_score" de la librería de sklearn.

Junto con la curva "ROC" y su área debajo "AUC", vamos a crear y a mostrar por pantalla la matriz de confusión, que de manera visual y rápida nos va a permitir visualizar la eficacia de nuestra red. Para ello vamos a emplear la función propia de la librería de sklearn ConfusionMatrixDisplay: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las

Por último vamos a aprovechar más aún la librería de sklearn y vamos a mostrar por pantalla el reporte de las predicciones que nos va a indicar la precisión, el "recall", el "f1-score" y el "support" conseguidos con las predicciones de nuestro modelo entrenado.

In [ ]:

```
# Primero de todo debemos cargar los datos de prediccion que hemos guardado en el bloque anterior
file = open(BASE_FOLDER + 'XCEPTION/best_fine_tunning_xception_predictions','rb')
predictions = pickle.load(file)
file.close()

# Creamos un dataframe con las probabilidades de las predicciones y las clases
model_predictions_dataframe = pd.DataFrame(predictions, columns=['0_no_tumor', '1_tumor'])
print("Sample de los valores sobre algunas predicciones:")
display(model_predictions_dataframe.head())

# Guardamos las clases predichas
image_aviable_classes = test_generator.classes

# Para calcular la AUC necesitamos saber las predicciones de la clase positiva, para ello vamos a elegir la columna tumor del dataframe creado previamente "model_predictions_dataframe"
positive_model_predictions = model_predictions_dataframe['1_tumor']

# Ahora ya podemos mostrar la gráfica
print("Área bajo la curva ROC:"+ str(roc_auc_score(image_aviable_classes, positive_model_predictions)))
```

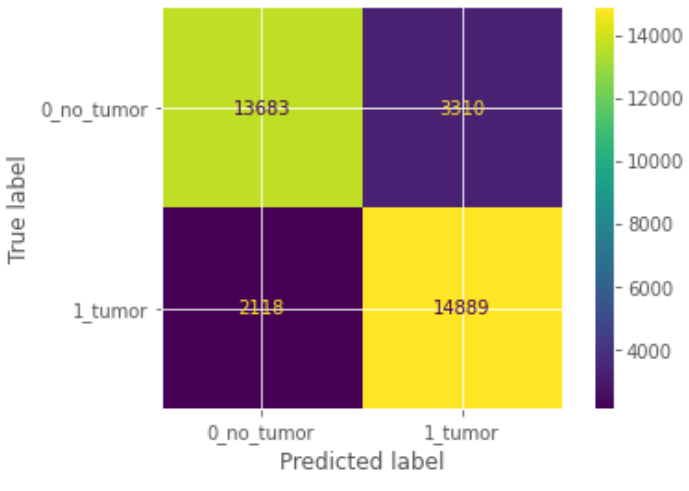
```
Sample de los valores sobre algunas predicciones:
```

	0_no_tumor	1_tumor
0	**0.878724**	**0.121276**
1	**0.430445**	**0.569555**
2	**0.965688**	**0.034312**
3	**0.505911**	**0.494089**
4	**0.124054**	**0.875946**

```
Área bajo la curva ROC:0.923982599567984
```

In [ ]:

```
# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```



In [ ]:

```
# Sacamos el reporte para el subconjunto de validación, utilizando la función de classification_report
# Como tenemos las predicciones de las clases como probabilidades debido a la función de activación de la última capa "softmax", debemos pasarlo a binario (0 o 1)
class_pred_binary = predictions.argmax(axis=1)

print(classification_report(image_aviable_classes, class_pred_binary, target_names=labelNames))
```

```
              precision    recall  f1-score   support

    No Tumor       0.87      0.81      0.83     16993
       Tumor       0.82      0.88      0.85     17007

    accuracy                           0.84     34000
   macro avg       0.84      0.84      0.84     34000
weighted avg       0.84      0.84      0.84     34000
```

Por último, repitiendo, al igual que en todos los apartados anteriores, como también hemos guardado en google drive, los parámetros sacados durante el entrenamiento de nuestra red neuronal. Ahora podemos a recuperarlos y a mostrar diferentes gráficas para discutir de una forma visual los resultados obtenidos época a época.

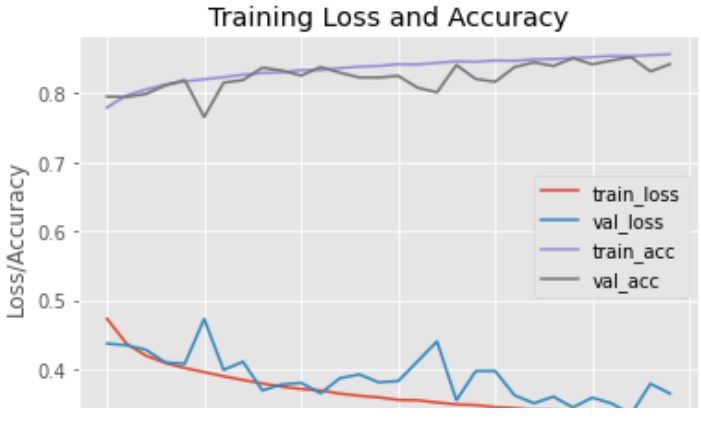
Los datos que vamos a mostrar en las gráficas son los siguientes:

- **Train Loss: Perdidas con el dataset de train**
- **Validation Loss: Perdidas con el dataset de validation**
- **Train accuracy: Precisión con el dataset de train**
- **Validation accuracy: Precisión con el dataset de train**

Debemos tener en cuenta que tanto las perdidas como la precisión entre train y validation deben de ser similares, de no ser así, estaríamos delante de un problema de overfitting.

In [ ]:

```
file = open(BASE_FOLDER + 'XCEPTION/best_fine_tunning_xception_training','rb')
history = pickle.load(file)

# Gráficas
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, 30), history["loss"], label="train_loss")
plt.plot(np.arange(0, 30), history["val_loss"], label="val_loss")
plt.plot(np.arange(0, 30), history["accuracy"], label="train_acc")
plt.plot(np.arange(0, 30), history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend()
plt.show()

file.close()
```



Como podemos observar a través de las gráficas y de las métricas extraídas de las predicciones del modelo, usando técnicas de fine tunning, descongelando el último bloque convolucional, el número 15, de la red Xception. Conseguimos unos resultados mejores que con solamente transfer learning, ya que adaptamos una capa más de la red pre entrenada a nuestros datos.

Los resultados que obtenemos son de aproximadamente un 0.85% de precisión, una precisión decente, pero que no son está entre las mejores de los otros modelos que hemos entrenado.

Por ejemplo, el mismo procedimiento de fine tunning con la red VGG19, nos da los mejores resultados de las redes que hemos entrenado, cosa esperada, debido a que en el concurso de Imagenet la red VGG19 ya obtuvo mejores resultados que Xcpetion e incluso los mejores resultados de todo el concurso.

Conclusión

Después de realizar todos los apartados requeridos para la realización de la práctica, hemos podido experimentar como es la creación de un end to end pipeline the deep learning.

Aunque para un experto dentro del campo del deep learning, esta práctica no sea de mucha dificultad, para nuestros, ambos, alumnos novatos en el campo del deep learning, nos ha resultado todo un reto. Después de revisar el material dado en clase, buscar por internet y muchas horas de probar, finalmente podemos concluir que hemos alcanzado el objetivo de la práctica.

Los modelos que hemos presentado todos tienen resultados decentes, sobre todo el modelo que mayor precisión nos ha reportado, ha sido el de la actividad 1, el modelo build form scratch, lo cual nos enorgullece, ya que es un modelo decidido por nosotros, buscando por internet y probando soluciones. A priori pensábamos que los resultados de transfer learning serian insuperables, pero hemos sido capaces de superarlos.

Para finalizar vamos a proceder a listar los resultados que hemos obtenido de todos los modelos entrenados durante la práctica:

Porcentajes de precisión de las predicciones realizadas:

- **CNN propia: 0.93 % de precisión**
- **VGG19 Transfer learning: 0.88 % de precisión**
- **VGG19 Fine Tunning: 0.88 % de precisión**
- **Xception Transfer learning: 0.80 % de precisión**
- **Xception Fine Tunning: 0.84 % de precisión**

Anexo

En este apartado final, vamos a mostrar todos los modelos entrenados no tan buenos como los vistos en los apartados pertinentes. Este apartado contiene modelos de las dos estrategias, entrenamiento from scratch y uso de transfer leanring junto con fine tunning.

También vamos a comentar el porqué estos modelos no han sido seleccionados como los mejores, y que modificaciones hemos realizado para conseguir modelos mejores.

Estrategia 1: Entrenar desde cero o from scratch

Los tres primeros modelos que vamos a mostrar a continuación, son el mismo pero con 10 épocas de entrenamiento de diferencia, el primero de la época 0 a la 10, el segundo de la 10 a la 20 y el tercero de la 20 a la 30.

Como se puede observar en las métricas conseguidas durante los entrenamientos, estos modelos prometían mucho, todos con más de 0.9% de precisión y una función de perdidas muy optimizada.

Porque están aquí entonces?, cuando empezamos a realizar el ejercicio, caímos en el error de utilizar todas las

imágenes sin mirar si, realmente, las clases estaban balanceados o no, error de novato, por lo que estos modelos tenían más tendencia a predecir una clase que otra, justamente con la misma desproporción de las clases de las imágenes.

Aún así los resultados eran muy prometedores, por lo que intuimos que íbamos muy bien con la arquitectura de red elegida.

De cada modelo, vamos a visualizar las mismas métricas que hemos analizado para los mejores modelos.

Primero vamos a crear una función auxiliar para poder mostrar las gráficas de entrenamiento

In [ ]:

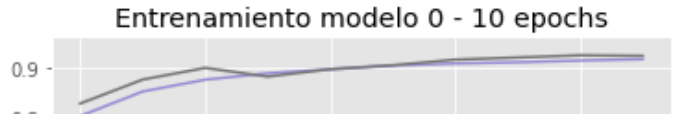
```
def plot_training_graphs(training_history, title):
  plt.style.use("ggplot")
  plt.figure()
  plt.plot(np.arange(0, 10), training_history["loss"], label="train_loss")
  plt.plot(np.arange(0, 10), training_history["val_loss"], label="val_loss")
  plt.plot(np.arange(0, 10), training_history["accuracy"], label="train_acc")
  plt.plot(np.arange(0, 10), training_history["val_accuracy"], label="val_acc")
  plt.title(title)
  plt.xlabel("Epoch #")
  plt.ylabel("Loss/Accuracy")
  plt.legend()
  plt.show()
```

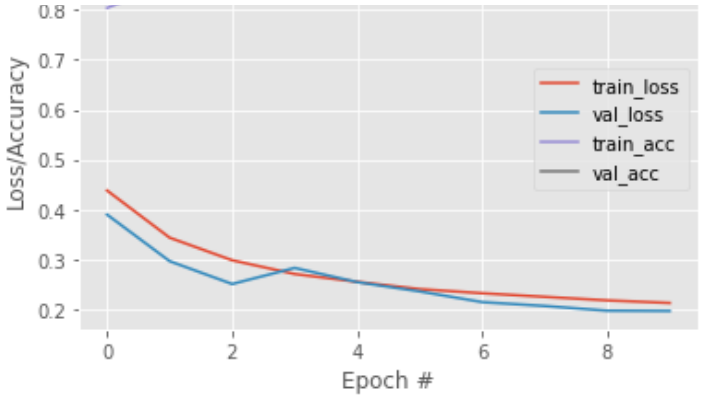
In [ ]:

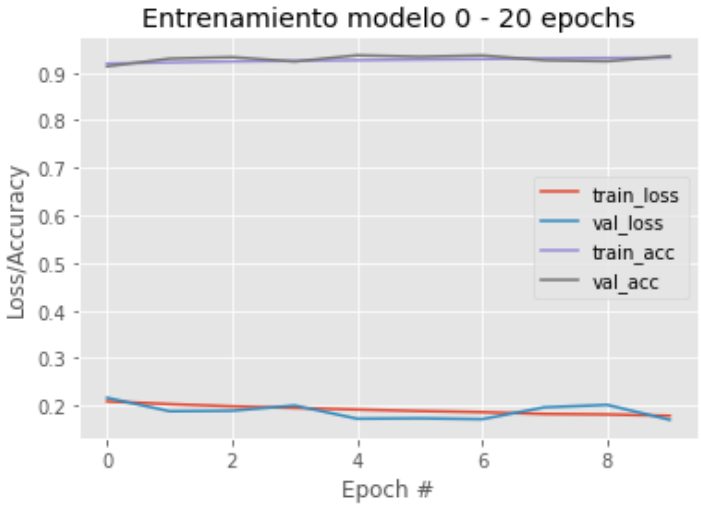
```
# Importamos los 3 modelos, sus métricas de entrenamiento y sus predicciones
model_10_epochs = load_model(BASE_FOLDER+"CNN/old_CNN_model.h5")
model_20_epochs = load_model(BASE_FOLDER+"CNN/old_CNN_model_20.h5")
model_30_epochs = load_model(BASE_FOLDER+"CNN/old_CNN_model_30.h5")

# Métricas entrenamientos
file = open(BASE_FOLDER + 'CNN/old_CNN_training','rb')
training_model_10_epochs = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/old_CNN_training_20','rb')
training_model_20_epochs = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/old_CNN_training_30','rb')
training_model_30_epochs = pickle.load(file)
file.close()

# Predicciones
file = open(BASE_FOLDER + 'CNN/old_CNN_predictions','rb')
predictions_model_10_epochs = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/old_CNN_predictions_20','rb')
predictions_model_20_epochs = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'CNN/old_CNN_predictions_30','rb')
predictions_model_30_epochs = pickle.load(file)
file.close()
```

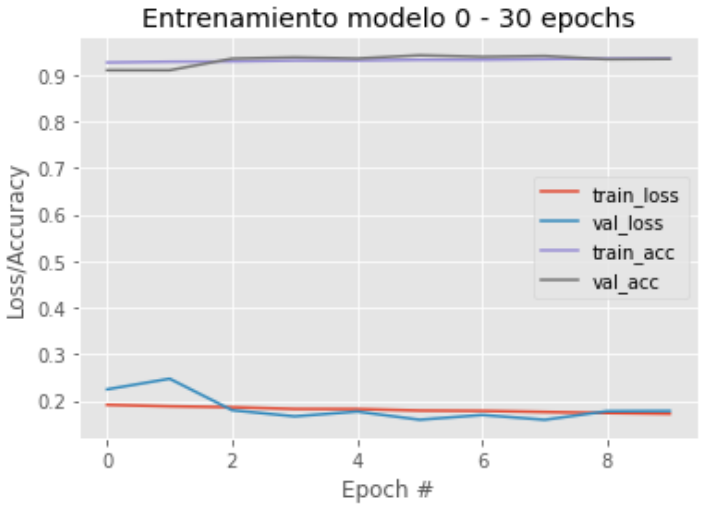
In [ ]:

```
# Vamos a mostrar las gráficas de la información extraída durante el entrenamiento
plot_training_graphs(training_model_10_epochs, "Entrenamiento modelo 0 - 10 epochs")
plot_training_graphs(training_model_20_epochs, "Entrenamiento modelo 0 - 20 epochs")
plot_training_graphs(training_model_30_epochs, "Entrenamiento modelo 0 - 30 epochs")
```









Ahora vamos a mostrar las métricas sobre las predicciones realizadas

In [ ]:

```
class_pred_binary_model_1 = predictions_model_10_epochs.argmax(axis=1)
class_pred_binary_model_2 = predictions_model_20_epochs.argmax(axis=1)
class_pred_binary_model_3 = predictions_model_30_epochs.argmax(axis=1)

# Ahora vamos a mostrar el classification report de nuestras predicciones
print("Modelo 1")
print(classification_report(image_aviable_classes, class_pred_binary_model_1, target_name
s=labelNames))
print("Modelo 2")
print(classification_report(image_aviable_classes, class_pred_binary_model_2, target_name
s=labelNames))
print("Modelo 3")
print(classification_report(image_aviable_classes, class_pred_binary_model_3, target_name
s=labelNames))

# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions.argma
x(axis=1))
```

```
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()

# Ahora mostramos las matrizes de confusión
predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions_model
_10_epochs.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()

predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions_model
_20_epochs.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()

predictions_confusion_matrix = confusion_matrix(test_generator.classes, predictions_model
_30_epochs.argmax(axis=1))
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```

Modelo 1

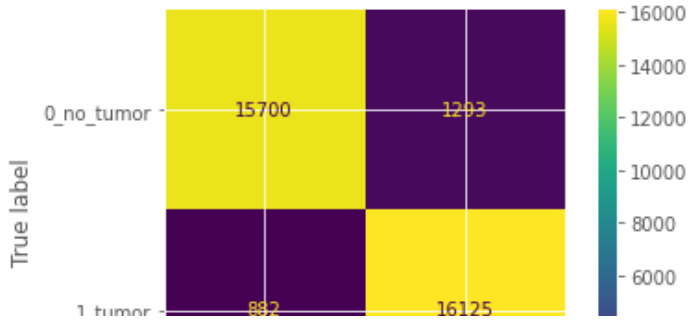
	precision	recall	f1-score	support
No Tumor	0.87	0.97	0.92	16993
Tumor	0.97	0.86	0.91	17007
accuracy			0.91	34000
macro avg	0.92	0.91	0.91	34000
weighted avg	0.92	0.91	0.91	34000

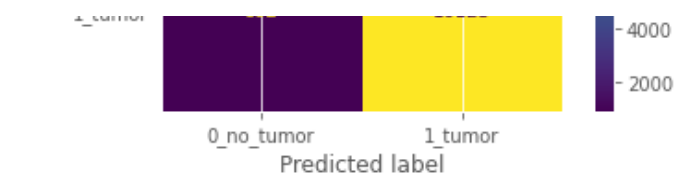
Modelo 2

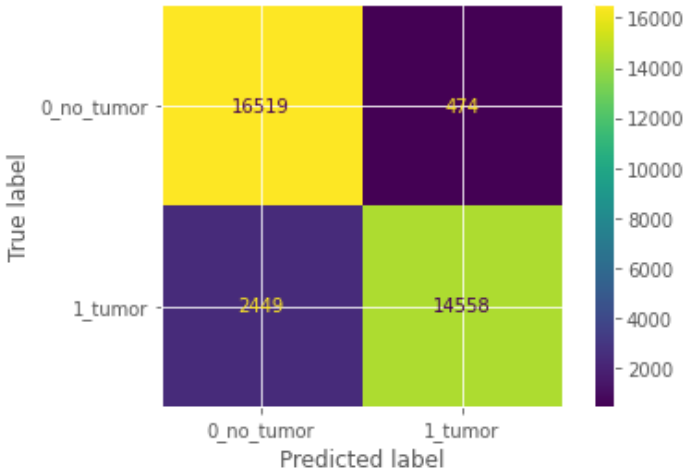
	precision	recall	f1-score	support
No Tumor	0.90	0.97	0.93	16993
Tumor	0.97	0.89	0.93	17007
accuracy			0.93	34000
macro avg	0.93	0.93	0.93	34000
weighted avg	0.93	0.93	0.93	34000

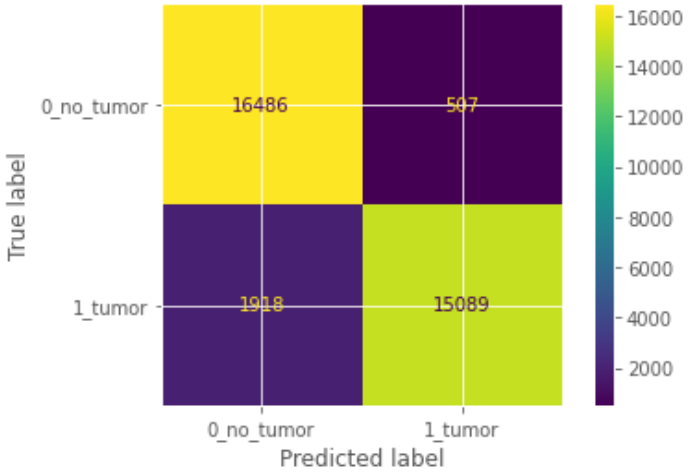
Modelo 3

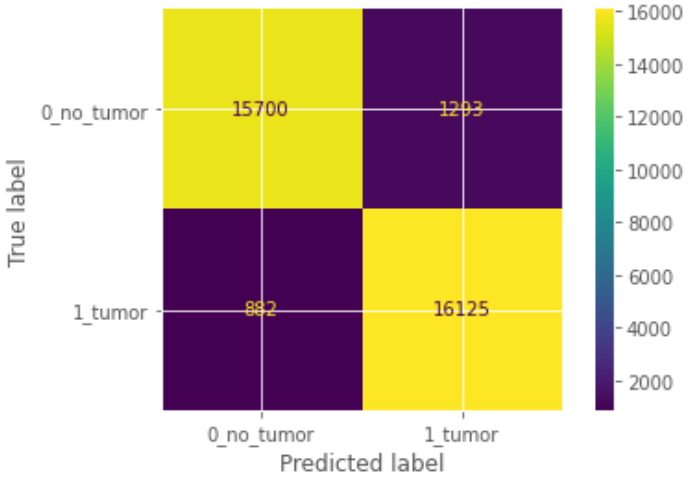
	precision	recall	f1-score	support
No Tumor	0.95	0.92	0.94	16993
Tumor	0.93	0.95	0.94	17007
accuracy			0.94	34000
macro avg	0.94	0.94	0.94	34000
weighted avg	0.94	0.94	0.94	34000











Estrategia 2: Red pre-entrenada VGG19:

En este apartado del anexo, vamos a mostrar 4 modelos de redes neuronales entrenados.

Primeramente, vamos a mostrar dos modelos de la red VGG19, el primero fue entrenado solo haciendo uso de transfer learning, lo que significa que importamos la red pre entrenada, le cambiamos el top model y la entrenamos con nuestros datos.

Después vamos a mostrar la misma red VGG19, pero esta vez aparte de transfer learning , le aplicamos técnicas de fine tunning, descongelando los bloques convolucionales 4 y 5.

Como se puede observar en las métricas conseguidas durante los entrenamientos, estos modelos prometían mucho, todos con más de 0.9% de precisión y una función de perdidas bastante optimizada y a la par los de

train con los de validación.

Igual que las del apartado anterior de la estrategia 1, con esta red no obtuvimos los resultados esperados, ya que fue entrenada con un dataset desbalanceado, produciendo así un pésimo resultado.

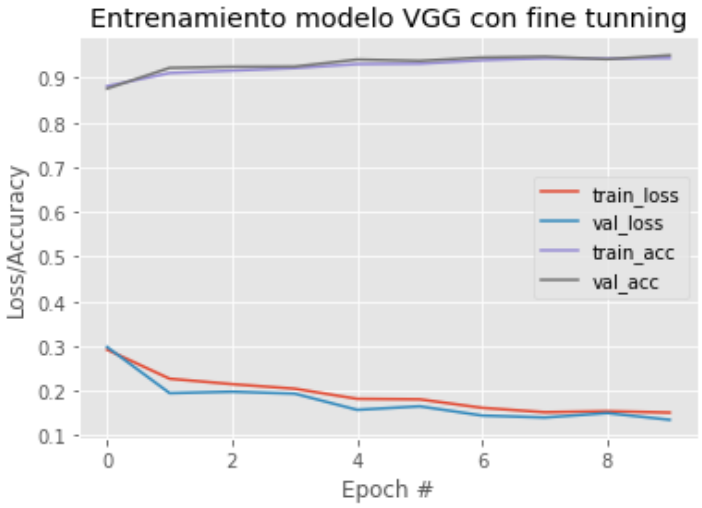
De cada modelo, vamos a visualizar las métricas para comprobar su eficacia.

In [ ]:

```
# Importamos los 2 modelos, sus métricas de entrenamiento y sus predicciones
model_transfer_learning_vgg = load_model(BASE_FOLDER+"VGG/old_transfer_learning_vgg_model
.h5")
model_fine_tunning_vgg = load_model(BASE_FOLDER+"VGG/old_fine_tunning_vgg_model.h5")

file = open(BASE_FOLDER + 'VGG/old_fine_tunning_vgg_training','rb')
training_model_fine_tunning_vgg = pickle.load(file)
file.close()

# Predicciones
file = open(BASE_FOLDER + 'VGG/old_transfer_learning_vgg_predictions','rb')
predictions_model_transfer_learning_vgg = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'VGG/old_fine_tunning_vgg_predictions','rb')
predictions_model_fine_tunning_vgg = pickle.load(file)
file.close()
```

In [ ]:

```
# Vamos a mostrar las gráficas de la información extraída durante el entrenamiento
# Por desgracia no pudimos guardar la información de training de la primera iteración de
la red VGG con transfer_learning
#plot_training_graphs(training_model_transfer_learning_vgg, "Entrenamiento modelo VGG con
transfer learning")
plot_training_graphs(training_model_fine_tunning_vgg, "Entrenamiento modelo VGG con fine
tunning")
```



In [ ]:

```
class_pred_binary_model_transfer_learning_vgg = predictions_model_transfer_learning_vgg.a
rgmax(axis=1)
class_pred_binary_model_fine_tunning_vgg = predictions_model_fine_tunning_vgg.argmax(axis
=1)

# Ahora vamos a mostrar el classification report de nuestras predicciones
print("Transfer Learning")
print(classification_report(image_aviable_classes, class_pred_binary_model_transfer_learn
ing_vgg, target_names=labelNames))
print("Fine Tunning")
print(classification_report(image_aviable_classes, class_pred_binary_model_fine_tunning_v
gg, target_names=labelNames))

# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
```

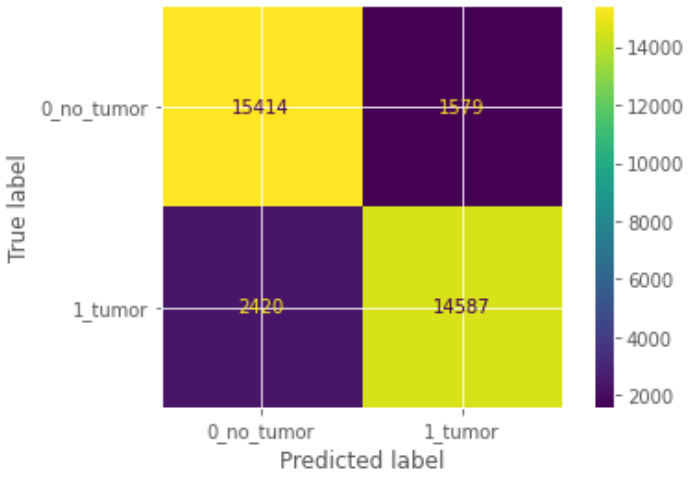
```
predictions_confusion_matrix = confusion_matrix(test_generator.classes, class_pred_binary
_model_transfer_learning_vgg)
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()

# Ahora mostramos las matrizes de confusión
predictions_confusion_matrix = confusion_matrix(test_generator.classes, class_pred_binary
_model_fine_tunning_vgg)
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```

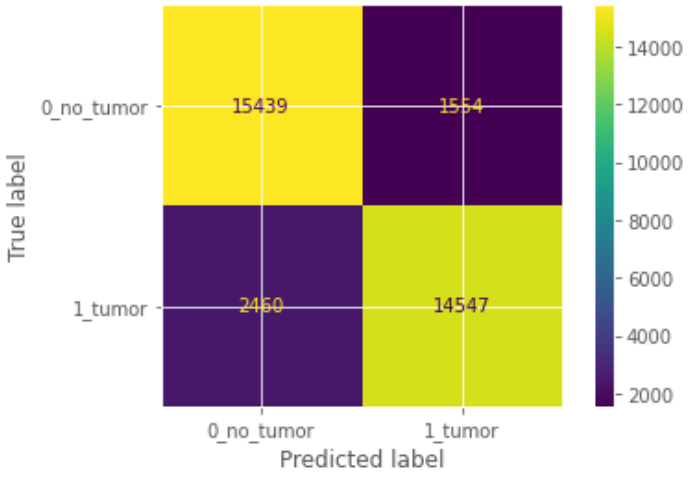
Transfer Learning

	precision	recall	f1-score	support
No Tumor	0.86	0.91	0.89	16993
Tumor	0.90	0.86	0.88	17007
accuracy			0.88	34000
macro avg	0.88	0.88	0.88	34000
weighted avg	0.88	0.88	0.88	34000

Fine Tunning

	precision	recall	f1-score	support
No Tumor	0.86	0.91	0.88	16993
Tumor	0.90	0.86	0.88	17007
accuracy			0.88	34000
macro avg	0.88	0.88	0.88	34000
weighted avg	0.88	0.88	0.88	34000





Estrategia 2: Red pre-entrenada Xception:

En este último apartado de la práctica, vamos a mostrar dos modelos de la red Xception, al igual que hemos comentado en el apartado anterior de la red VGG19, el primer modelo fue entrenado solo haciendo uso de transfer learning, lo que significa que importamos la red pre entrenada, le cambiamos el top model y la entrenamos con nuestros datos.

Después vamos a mostrar la misma red Xception, pero esta vez, aparte de transfer learning, le aplicamos técnicas de fine tunning, descongelando el último bloque convolucional el número 15.

Como se puede observar en las métricas conseguidas durante los entrenamientos, estos modelos prometían mucho, todos con más de 0.9% de precisión y una función de perdidas bastante optimizada y a la par los de train con los de validación.

Igual que las del apartado anterior de la estrategia 2 con la red pre entrenada VGG19, con la red Xception red no obtuvimos los resultados esperados, ya que fue entrenada al igual que las anteriores con un dataset desbalanceado, produciendo así un pésimo resultado a la hora de predecir nuevas imágenes.

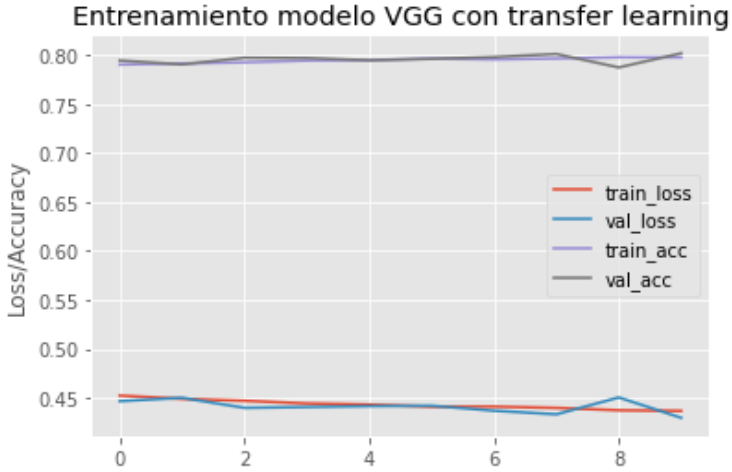
De cada modelo, vamos a mostrar algunas métricas para poder evaluarla.

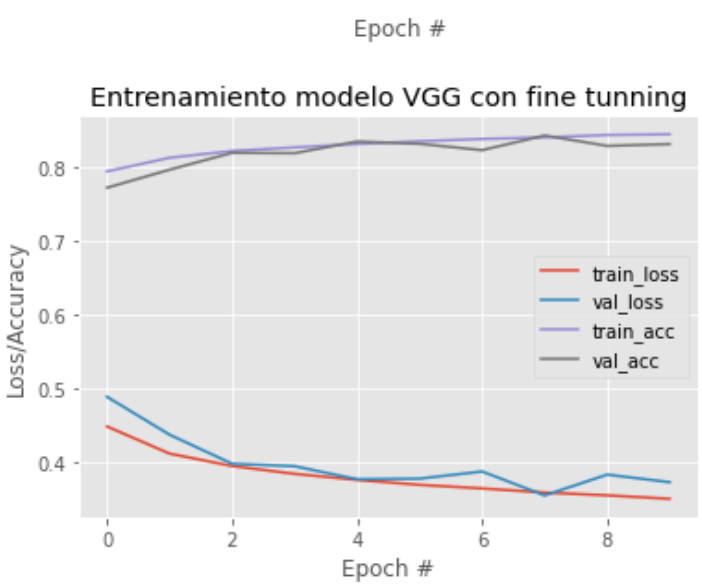
In [ ]:

```
# Importamos los 2 modelos, sus métricas de entrenamiento y sus predicciones
model_transfer_learning_xception = load_model(BASE_FOLDER+"XCEPTION/old_transfer_learning_xception_model.h5")
model_fine_tunning_xception = load_model(BASE_FOLDER+"XCEPTION/old_fine_tunning_xception_model.h5")

# Métricas entrenamientos
file = open(BASE_FOLDER + 'XCEPTION/old_transfer_learning_xception_training','rb')
training_model_transfer_learning_xception = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'XCEPTION/old_fine_tunning_xception_training','rb')
training_model_fine_tunning_xception = pickle.load(file)
file.close()

# Predicciones
file = open(BASE_FOLDER + 'XCEPTION/old_transfer_learning_xception_predictions','rb')
predictions_model_transfer_learning_xception = pickle.load(file)
file.close()

file = open(BASE_FOLDER + 'XCEPTION/old_fine_tunning_learning_predictions','rb')
predictions_model_fine_tunning_xception = pickle.load(file)
file.close()
```

In [ ]:

```
# Vamos a mostrar las gráficas de la información extraída durante el entrenamiento
plot_training_graphs(training_model_transfer_learning_xception, "Entrenamiento modelo VGG con transfer learning")
plot_training_graphs(training_model_fine_tunning_xception, "Entrenamiento modelo VGG con fine tunning")
```

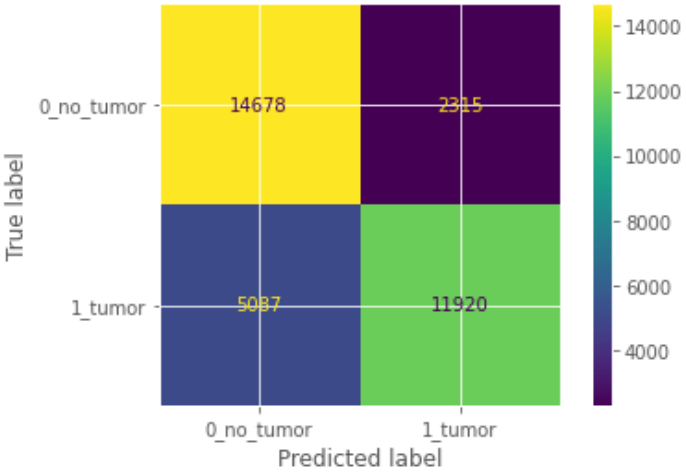


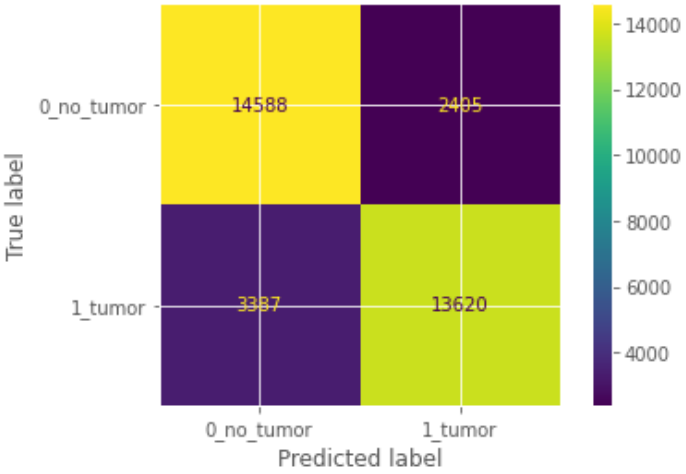


In [ ]:

```
class_pred_binary_model_transfer_learning_xception = predictions_model_transfer_learning_
xception.argmax(axis=1)
class_pred_binary_model_fine_tunning_xception = predictions_model_fine_tunning_xception.a
rgmax(axis=1)

# Ahora vamos a mostrar el classification report de nuestras predicciones
print("Transfer Learning")
print(classification_report(image_aviable_classes, class_pred_binary_model_transfer_learn
ing_xception, target_names=labelNames))
print("Fine Tunning")
print(classification_report(image_aviable_classes, class_pred_binary_model_fine_tunning_x
ception, target_names=labelNames))

# Ahora vamos a mostrar la matriz de confusión de nuestras predicciones
predictions_confusion_matrix = confusion_matrix(test_generator.classes, class_pred_binary
_model_transfer_learning_xception)
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()

# Ahora mostramos las matrizes de confusión
predictions_confusion_matrix = confusion_matrix(test_generator.classes, class_pred_binary
_model_fine_tunning_xception)
confusion_matrix_display = ConfusionMatrixDisplay(confusion_matrix=predictions_confusion_
matrix,
                                  display_labels=['0_no_tumor', '1_tumor'])
confusion_matrix_display.plot()
plt.show()
```

```
Transfer Learning
              precision    recall  f1-score   support

    No Tumor       0.74      0.86      0.80     16993
       Tumor       0.84      0.70      0.76     17007

    accuracy                           0.78     34000
   macro avg       0.79      0.78      0.78     34000
weighted avg       0.79      0.78      0.78     34000

Fine Tunning
              precision    recall  f1-score   support

    No Tumor       0.81      0.86      0.83     16993
       Tumor       0.85      0.80      0.82     17007

    accuracy                           0.83     34000
   macro avg       0.83      0.83      0.83     34000
weighted avg       0.83      0.83      0.83     34000
```


0_no_tumor
1_tumor
True label
14678
2315
5037
11920
0_no_tumor
1_tumor
Predicted label
14000
12000
10000
8000
6000
4000


0_no_tumor
1_tumor
True label
14588
2405
3387
13620
0_no_tumor
1_tumor
Predicted label
14000
12000
10000
8000
6000
4000