07MIAR - Redes Neuronales y Deep Learning: Proyecto de programación "*Deep Vision in classification tasks*"

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

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.).

Carga de datos desde la plataforma Kaggle

```
# Nos aseguramos que tenemos instalada la última versión de la API de Kaggle en Colab
!pip install --upgrade --force-reinstall --no-deps kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-whe
Collecting kaggle
  Downloading kaggle-1.5.13.tar.gz (63 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.3/63.3 kB 738.4 kB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
```

```
  Building wheels for collected packages: kaggle
    Building wheel for kaggle (setup.py) ... done
    Created wheel for kaggle: filename=kaggle-1.5.13-py3-none-any.whl size=77733 sh
    Stored in directory: /root/.cache/pip/wheels/f3/16/ff/34e7d368370d4fd68bb749a5
  Successfully built kaggle
  Installing collected packages: kaggle
    Attempting uninstall: kaggle
      Found existing installation: kaggle 1.5.13
      Uninstalling kaggle-1.5.13:
        Successfully uninstalled kaggle-1.5.13
  Successfully installed kaggle-1.5.13
```

```
# Creamos un directorio en el que copiamos el fichero kaggle.json
# el archivo kaggle.json debe existir para que el script funcione
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

```
# Verificar si la conexión con kaggle fue existosa
!kaggle datasets list
```

```
ref                                                  title
---------------------------------------------------  ----------------------------
arnabchaki/data-science-salaries-2023                Data Science Salaries 2023
salvatorerastelli/spotify-and-youtube                Spotify and Youtube
iammustafatz/diabetes-prediction-dataset             Diabetes prediction dataset
shawkyelgendy/furniture-price-prediction             Furniture Price Prediction
arnabchaki/indian-restaurants-2023                   Indian Restaurants 2023
erdemtaha/cancer-data                                Cancer Data
desalegngeb/students-exam-scores                     Students Exam Scores: Extend
lokeshparab/amazon-products-dataset                  Amazon Products Sales Datase
cuecacuela/england-football-matches                  England Football Matches
matarrgaye/uk-consumer-trends-current-price          UK Consumer Trends: 1997 - 2
harshsingh2209/supply-chain-analysis                 Supply Chain Analysis
ulrikthygepedersen/fastfood-nutrition                Fastfood Nutrition
priyanshusethi/minecraft-piracy-dataset              Minecraft Piracy Dataset
ppb00x/credit-risk-customers                         credit_risk_customers
arnabchaki/popular-video-games-1980-2023             Popular Video Games 1980 - 2
dansbecker/melbourne-housing-snapshot                Melbourne Housing Snapshot
shubhammeshram579/house                              Housing price prediction
r1chardson/the-world-university-rankings-2011-2023   THE World University Rankings
ashishraut64/internet-users                          Global Internet users
khalidryder777/500k-chatgpt-tweets-jan-mar-2023      500k ChatGPT-related Tweets
```

```
# descargar el data set
!kaggle datasets download -d gpiosenka/100-bird-species
```

```
Downloading 100-bird-species.zip to /content
100% 1.96G/1.96G [01:26<00:00, 24.4MB/s]
100% 1.96G/1.96G [01:26<00:00, 24.3MB/s]
```

```
# Creemos un directorio para descomprimir los datos
!mkdir my_dataset


# Descomprimimos los datos y los dejamos listos para trabajar
!unzip 100-bird-species.zip -d my_dataset
```

Streaming output truncated to the last 5000 lines.

```
  inflating: my_dataset/train/WHITE TAILED TROPIC/120.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/121.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/122.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/123.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/124.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/125.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/126.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/127.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/128.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/129.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/130.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/131.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/132.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/133.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/134.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/135.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/136.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/137.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/138.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/139.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/140.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/141.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/142.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/143.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/144.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/145.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/146.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/147.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/148.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/149.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/150.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/151.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/152.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/153.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/154.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/155.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/156.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/157.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/158.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/159.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/160.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/161.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/162.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/163.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/164.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/165.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/166.jpg
  inflating: my_dataset/train/WHITE TAILED TROPIC/167.jpg
```

```
        inflating: my_dataset/train/WHITE TAILED TROPIC/168.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/169.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/170.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/171.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/172.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/173.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/174.jpg
        inflating: my_dataset/train/WHITE TAILED TROPIC/175.jpg
        inflating: my_dataset/train/WHITE THROATED BEE EATER/001.jpg
```

```
# Realizar la conexión con drive para guardar las épocas
from google.colab import drive
drive.mount('/content/drive')
```

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

Escritura de datos tomando como referencia un BASE_FOLDER

```
# La variable BASE_FOLDER tendrá el path donde se guardarán las épocas del entrenamier
BASE_FOLDER = "/content/drive/MyDrive/07MIAR_Proyecto_Programacion/"
# Esta variable contiene el path con las imágenes en la instancia local del collab
# La carpeta se genera en el punto anterior
BASE_DATASET = 'my_dataset/'
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow import keras
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import Callback, EarlyStopping, ModelCheckpoint, Reduc
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import Model
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Input, Conv2D, Activation, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import SGD, Adam
import itertools
```

```
# System libraries
from google.colab import drive
from pathlib import Path
import os.path
import random
```

```
# Visualization Libraries
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import cv2
import seaborn as sns
sns.set_style('darkgrid')
```

Cargar datos

```
# Leemos el archivo CSV llamado 'birds.csv' y lo cargamos en un Dataframe:
# * Este archivo hace parte del set de datos.
# La ruta del archivo se construye utilizando la variable 'BASE_DATASET'

df = pd.read_csv(f'{BASE_DATASET}birds.csv')


# Una vez cargados los datos, creamos una nueva columna llamada 'filepaths' la cual
# utiliza una función lambda para concatenar el path de las imágenes a la ruta incluic
# en el achivo bird.csv.

image_df = df.copy()
image_df['filepaths'] = image_df['filepaths'].map(lambda x: BASE_DATASET + x)
image_df
```

	class id	filepaths	labels	data set	scientific name
0	0.0	my_dataset/train/ABBOTTS BABBLER/001.jpg	ABBOTTS BABBLER	train	MALACOCINCLA ABBOTTI
1	0.0	my_dataset/train/ABBOTTS BABBLER/007.jpg	ABBOTTS BABBLER	train	MALACOCINCLA ABBOTTI
2	0.0	my_dataset/train/ABBOTTS BABBLER/008.jpg	ABBOTTS BABBLER	train	MALACOCINCLA ABBOTTI
3	0.0	my_dataset/train/ABBOTTS BABBLER/009.jpg	ABBOTTS BABBLER	train	MALACOCINCLA ABBOTTI
4	0.0	my_dataset/train/ABBOTTS BABBLER/002.jpg	ABBOTTS BABBLER	train	MALACOCINCLA ABBOTTI
...	...	...	...	...	...
89880	524.0	my_dataset/valid/BLACK BREASTED PUFFBIRD/3.jpg	BLACK BREASTED PUFFBIRD	valid	NOTHARCHUS PECTORALIS
		my_dataset/valid/BLACK	BLACK		NOTHARCHUS

```
# Se crean los queries para separa los data set de test, validación y entrenamiento
# df_test = datos para probar el modelo
```

```
# df_val = datos de validación durante el entrenamiento
# df_train = datos para entrenar el modelo

df_test = image_df[image_df['data set'] == 'test']
df_val = image_df[image_df['data set'] == 'valid']
df_train = image_df[image_df['data set'] == 'train']
```

Explorar datos

```
# Generamos una visualización de la distribución de las 20 etiquetas
# más comunes en el conjunto de entrenamiento ("df_train") del conjunto de datos de ir

label_counts = df_train['labels'].value_counts()[:20]
plt.figure(figsize=(20, 6))
sns.barplot(x=label_counts.index, y=label_counts.values, alpha=0.8, palette='dark:salr
plt.title('Distribution of Top 20 Train Labels in Image Dataset', fontsize=16)
plt.xlabel('Label', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45)
plt.show()
```

```
# Imprimimos en la consola el tamaño de los conjuntos de datos de prueba, validación y
```

```
print('data def:')
print(df_test.shape)

print('data val:')
print(df_val.shape)

print('data train:')
print(df_train.shape)
```

```
data def:
(2625, 5)
data val:
(2625, 5)
data train:
(84635, 5)
```

```
# Como parte de la exploración de datos, mostramos 16 imagenes del Dataset con sus lab
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(10, 10),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
  series_img = df_train.sample()
  img_path = series_img['filepaths'].iloc[0]
  ax.imshow(plt.imread(img_path))
  ax.set_title(series_img['labels'].iloc[0])
plt.tight_layout()
plt.show()
```

Carga de imágenes en memoria

A continuación generamos la primera parte del pipeline para el entrenamiento del modelo. Esto incluye una capa de Data Agmentation utilizando la clase `ImageDataGenerator` disponible en la biblioteca keras.

Cabe destacar que para el entrenamiento de la red convolucional, se utilizó un `class_mode="spare"`, dato que será relevante a la hora de seleccionar el método para el cálculo de la pérdida.

Una vez ejecutadas el código, se evidencia un total de `84635` imagenes disponibles para el entrenamiento, y `2625` imagenes para el proceso de validación y pruebas. Esto es coherente con la información disponible en el archivo `birds.csv`.

```
import tensorflow as tf
import tensorflow_datasets as tfds

# Tamaño de las imagenes
```

```
SEED = 42
BATCH_SIZE = 32
TARGET_SIZE = (128, 128)

# Se define los path de los directorios TRAIN VALIDATION Y TEST
TRAIN_PATH = BASE_DATASET + '/train'
VAL_PATH   = BASE_DATASET + '/valid'
TEST_PATH  = BASE_DATASET + '/test'

# Generador de imagenes para entrenamiento
train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale = 1./ 255,
    zoom_range = 0.2,
    rotation_range = 5,
    horizontal_flip = True
)

train = train_datagen.flow_from_directory(
  TRAIN_PATH,
  seed=SEED,
  target_size=TARGET_SIZE,
  batch_size=BATCH_SIZE,
  class_mode="sparse"
)

valid_datagen = ImageDataGenerator(rescale = 1. / 255)
valid = valid_datagen.flow_from_directory(
  VAL_PATH,
  seed=SEED,
  target_size=TARGET_SIZE,
  batch_size=BATCH_SIZE,
  class_mode="sparse"
)

test_datagen = ImageDataGenerator(rescale = 1. / 255)
test = test_datagen.flow_from_directory(
    TEST_PATH,
    seed=SEED,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode='sparse'
)
```

```
Found 84635 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
```

Arquitectura de la CNN

La arquitectura seleccionada para esta red convolucional, corresponde a un encoder con dos bloques convolucionales. Cada bloque incluye una capa convolucional seguida de una capa de normalización. Al final del bloque se hace un max polloing y se desactiva el 50% de las neuronas durante cada paso de la etapa de entrenamiento.

El bloque final se encarga de la clasificación del modelo, realizando un aplanamiento de los datos y finalizando con una capa densa con los 525 clases que puede tomar una imágen de entrada.

```
########################################
###### Definimos la arquitectura ########
########################################
from keras import regularizers

#BASE MODEL
# Definimos entradas
inputs = Input(shape=(TARGET_SIZE[0], TARGET_SIZE[1], 3))

# Primer set de capas CONV => RELU => CONV => RELU => POOL
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(inputs)
x1 = BatchNormalization()(x1)
x1 = Conv2D(32, (3, 3), padding="same", activation="relu")(x1)
x1 = BatchNormalization()(x1)
x1 = MaxPooling2D(pool_size=(2, 2))(x1)
x1 = Dropout(0.25)(x1)

# Segundo set de capas CONV => RELU => CONV => RELU => POOL
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x1)
x2 = BatchNormalization()(x2)
x2 = Conv2D(64, (3, 3), padding="same", activation="relu")(x2)
x2 = BatchNormalization()(x2)
x2 = MaxPooling2D(pool_size=(2, 2))(x2)
x2 = Dropout(0.25)(x2)

# TOP MODEL
# Primer (y único) set de capas FC => RELU
xfc = Flatten()(x2)
xfc = Dense(256, activation="relu")(xfc)
xfc = BatchNormalization()(xfc)
xfc = Dropout(0.5)(xfc)
# Clasificador softmax
predictions = Dense(525, activation="softmax")(xfc)

# Unimos las entradas y el modelo mediante la función Model con parámetros inputs y ou
model_cnn = Model(inputs=inputs, outputs=predictions)
```

Antes de compilar el modelo, es importante recordar que los datos fueron cargados utilizando el parámetro `class_mode="sparse"`, por lo tanto el parámetro `loss` del método `compile` debe tomar el valor `"sparse_categorical_crossentropy"`.

```
model_cnn.compile(
  loss="sparse_categorical_crossentropy",
  optimizer=Adam(learning_rate=1e-2, beta_1=0.9, beta_2=0.999, epsilon=1e-08),
  metrics=["accuracy"]
)
```

Finalmente, se muestra el resumen de la red utilizando el método `summary()`, en donde se puede evidenciar que el total de parámetros entrenables es de `16,979,757`. Un dato importante a mencionar, es que se hizo un reescalado de las imágenes de 224 a 128 con el objetivo de reducir el tiempo de entrenamiento.

```
model_cnn.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]     0

 conv2d_4 (Conv2D)           (None, 128, 128, 32)      896

 batch_normalization_5 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 conv2d_5 (Conv2D)           (None, 128, 128, 32)      9248

 batch_normalization_6 (Batc  (None, 128, 128, 32)     128
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 dropout_3 (Dropout)         (None, 64, 64, 32)        0

 conv2d_6 (Conv2D)           (None, 64, 64, 64)        18496

 batch_normalization_7 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 conv2d_7 (Conv2D)           (None, 64, 64, 64)        36928

 batch_normalization_8 (Batc  (None, 64, 64, 64)       256
 hNormalization)

 max_pooling2d_3 (MaxPooling  (None, 32, 32, 64)       0
 2D)

 dropout_4 (Dropout)         (None, 32, 32, 64)        0

 flatten_1 (Flatten)         (None, 65536)             0
```

```
 dense_2 (Dense)             (None, 256)               16777472

 batch_normalization_9 (Batc  (None, 256)              1024
 hNormalization)

 dropout_5 (Dropout)         (None, 256)               0

 dense_3 (Dense)             (None, 525)               134925

=================================================================
Total params: 16,979,757
Trainable params: 16,978,861
Non-trainable params: 896
_________________________________________________________________
```

Con el objetivo de mejorar el entrenamiento, A continuación se definen una serie de callbacks para serán utilizados durante la fase de entrenamiento de la CNN. Los callbacks utilizados fueron:

1. **LearningRateScheduler**: Actualiza la tasa de aprendizaje del modelo en función del número de épocas a entrenar.
2. **ReduceLROnPlateau**: Reduce la tasa de aprendizaje del modelo si no mejora el `val_accuracy` durante el número de épocas a entrenar.
3. **EarlyStopping**: Detiene el entrenamiento si no mejora el `val_accuracy` durante un número determinado de épocas.
4. **ModelCheckpoint**: Guarda el module después de cada época si el `accuracy` que se optiene es mejor comparado con las iteraciones anteriores.

```
from tensorflow.keras.models import load_model

# Save model after each epoch
checkpoint_path = "/content/drive/MyDrive/07MIAR/checkpoint-cnn-{epoch:02d}.h5"

lr_start   = 0.0012
lr_max     = 0.00015 * BATCH_SIZE
lr_min     = 1e-3
lr_ramp_ep = 3
lr_sus_ep  = 0
lr_decay   = 0.7

def lrfn(epoch):
    if epoch < lr_ramp_ep:
        lr = (lr_max - lr_start) / lr_ramp_ep * epoch + lr_start

    elif epoch < lr_ramp_ep + lr_sus_ep:
        lr = lr_max

    else:
```

```
        lr = (lr_max - lr_min) * lr_decay**(epoch - lr_ramp_ep - lr_sus_ep) + lr_min

    return lr

checkpoint_callback = [
  tf.keras.callbacks.LearningRateScheduler(lrfn),
  tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_accuracy",
    factor = 0.2,
    patience = 2,
    mode = "auto",
    cooldown = 0,
    min_lr = 0.001,
  ),
  tf.keras.callbacks.EarlyStopping(
    monitor = "val_accuracy",
    patience = 5,
    verbose = 1,
    mode = "auto",
  ),
  tf.keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_path,
    save_weights_only = False,
    monitor = 'val_accuracy',
    mode = 'auto',
    save_best_only = True
  )
]
```

Entrenamiento de la CNN

El entrenamiento de la red convolucional utiliza los set de datos de entrenamiento y validación, y se definió un número de 20 épocas. Cada época duro más de 500s, con un total aproximado de 2.7 horas en total. El mejor accuracy obtenido durante la etapa de entrenamiento, fue durante la época 18, y puede ser descargada del siguiente enlace: https://drive.google.com/file/d/1-ZbwxylltRSDQGhvzaxs-zOZDGLFSwU0/view?usp=sharing

```
# latest with 128, batch 32
epochs = 20

history = model_cnn.fit(
    train,
    validation_data=valid,
    epochs=epochs,
    shuffle=True,
    verbose=1,
```

```
    callbacks=[checkpoint_callback]
)
```

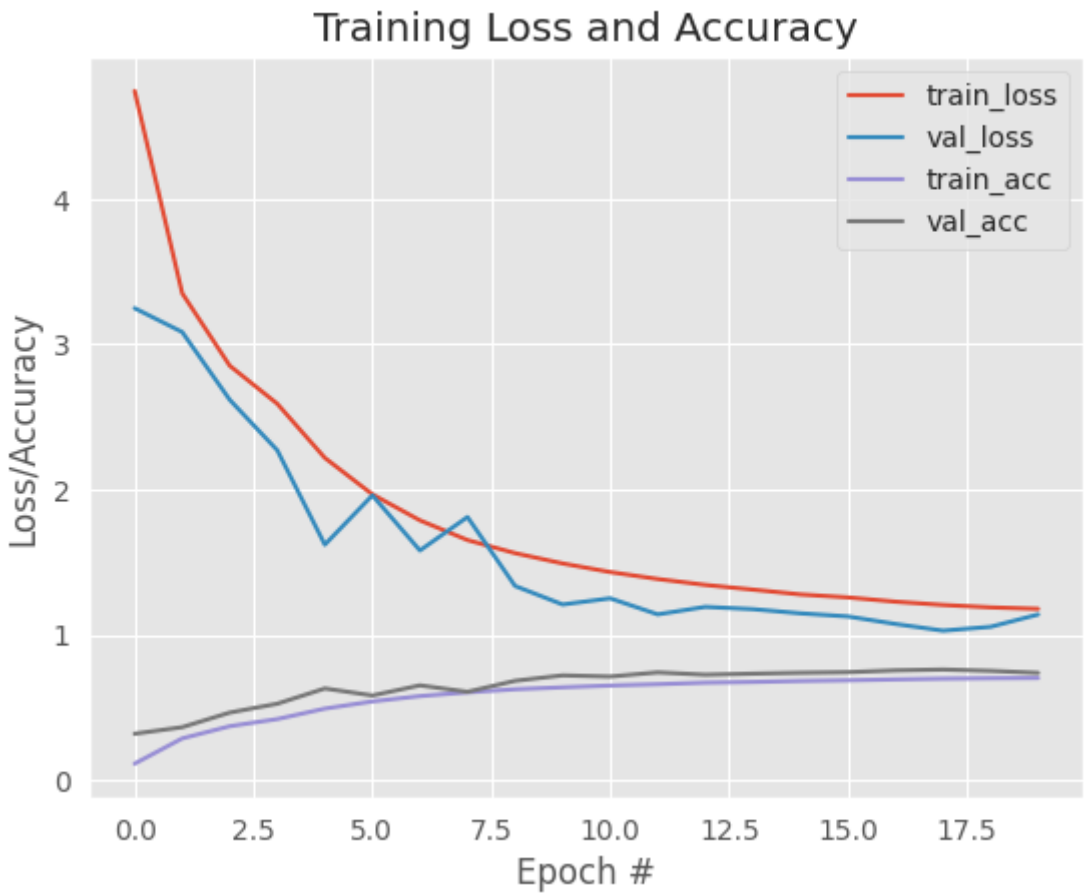
```
Epoch 1/20
2645/2645 [==============================] - 525s 193ms/step - loss: 4.7434 - ac
Epoch 2/20
2645/2645 [==============================] - 507s 192ms/step - loss: 3.3510 - ac
Epoch 3/20
2645/2645 [==============================] - 508s 192ms/step - loss: 2.8523 - ac
Epoch 4/20
2645/2645 [==============================] - 506s 191ms/step - loss: 2.5905 - ac
Epoch 5/20
2645/2645 [==============================] - 507s 192ms/step - loss: 2.2181 - ac
Epoch 6/20
2645/2645 [==============================] - 508s 192ms/step - loss: 1.9670 - ac
Epoch 7/20
2645/2645 [==============================] - 505s 191ms/step - loss: 1.7882 - ac
Epoch 8/20
2645/2645 [==============================] - 498s 188ms/step - loss: 1.6525 - ac
Epoch 9/20
2645/2645 [==============================] - 498s 188ms/step - loss: 1.5625 - ac
Epoch 10/20
2645/2645 [==============================] - 495s 187ms/step - loss: 1.4915 - ac
Epoch 11/20
2645/2645 [==============================] - 493s 186ms/step - loss: 1.4322 - ac
Epoch 12/20
2645/2645 [==============================] - 492s 186ms/step - loss: 1.3844 - ac
Epoch 13/20
2645/2645 [==============================] - 494s 187ms/step - loss: 1.3437 - ac
Epoch 14/20
2645/2645 [==============================] - 492s 186ms/step - loss: 1.3115 - ac
Epoch 15/20
2645/2645 [==============================] - 491s 186ms/step - loss: 1.2783 - ac
Epoch 16/20
2645/2645 [==============================] - 492s 186ms/step - loss: 1.2578 - ac
Epoch 17/20
2645/2645 [==============================] - 503s 190ms/step - loss: 1.2284 - ac
Epoch 18/20
2645/2645 [==============================] - 495s 187ms/step - loss: 1.2059 - ac
Epoch 19/20
2645/2645 [==============================] - 494s 187ms/step - loss: 1.1892 - ac
Epoch 20/20
2645/2645 [==============================] - 490s 185ms/step - loss: 1.1791 - ac
```

Métricas

A continuación se grafica la evolución del entrenamiento y la validación de la red neuronal durante las 20 épocas de entrenamiento.

Se utilizó la biblioteca Matplotlib para crear el gráfico. El gráfico incluye cuatro líneas: la pérdida de entrenamiento, la pérdida de validación, la precisión de entrenamiento y la precisión de validación.

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




```
# Calculamos la pérdida y la precisión del modelo en el conjunto
# de datos de prueba (test)

loss, Accuracy = model_cnn.evaluate(test)
```

```
83/83 [==============================] - 5s 60ms/step - loss: 1.0268 - accuracy:
```

```
# Hacemos predicciones sobre el conjunto de datos de prueba

predictions = model_cnn.predict(test, batch_size=BATCH_SIZE)
predicted_categories = np.argmax(predictions, axis=1)
```

```
83/83 [==============================] - 5s 53ms/step
```

```
# Finalmente se muestra un reporte de clasificación para las imagenes de prueba

print(classification_report(df_test['class id'], predicted_categories))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.20      0.33         5
         1.0       0.00      0.00      0.00         5
         2.0       0.67      0.40      0.50         5
         3.0       0.40      0.80      0.53         5
         4.0       0.83      1.00      0.91         5
         5.0       1.00      0.80      0.89         5
         6.0       1.00      1.00      1.00         5
         7.0       1.00      0.20      0.33         5
         8.0       1.00      1.00      1.00         5
         9.0       0.80      0.80      0.80         5
        10.0       1.00      0.40      0.57         5
        11.0       1.00      1.00      1.00         5
        12.0       0.00      0.00      0.00         5
        13.0       1.00      0.20      0.33         5
        14.0       1.00      0.80      0.89         5
        15.0       1.00      0.60      0.75         5
        16.0       0.57      0.80      0.67         5
        17.0       0.71      1.00      0.83         5
        18.0       0.83      1.00      0.91         5
        19.0       1.00      1.00      1.00         5
        20.0       1.00      1.00      1.00         5
        21.0       1.00      0.60      0.75         5
        22.0       1.00      0.80      0.89         5
        23.0       1.00      0.80      0.89         5
        24.0       0.67      0.40      0.50         5
        25.0       0.75      0.60      0.67         5
        26.0       1.00      0.60      0.75         5
        27.0       1.00      1.00      1.00         5
        28.0       1.00      0.80      0.89         5
        29.0       1.00      1.00      1.00         5
        30.0       0.75      0.60      0.67         5
        31.0       0.83      1.00      0.91         5
        32.0       1.00      0.80      0.89         5
        33.0       1.00      0.60      0.75         5
        34.0       1.00      0.80      0.89         5
        35.0       0.44      0.80      0.57         5
        36.0       1.00      1.00      1.00         5
        37.0       0.00      0.00      0.00         5
        38.0       1.00      0.80      0.89         5
        39.0       0.80      0.80      0.80         5
        40.0       1.00      0.80      0.89         5
```

41.0	1.00	1.00	1.00	5
42.0	0.56	1.00	0.71	5
43.0	0.83	1.00	0.91	5
44.0	0.20	0.20	0.20	5
45.0	0.71	1.00	0.83	5
46.0	0.80	0.80	0.80	5
47.0	0.80	0.80	0.80	5
48.0	1.00	1.00	1.00	5
49.0	1.00	0.80	0.89	5
50.0	1.00	0.40	0.57	5
51.0	0.80	0.80	0.80	5
52.0	1.00	0.80	0.89	5
53.0	1.00	0.80	0.89	5
54.0	0.56	1.00	0.71	5

Para concluir, el modelo se entrenó durante más de 2 horas y media, y logró una buena precisión del `76.15%` utilizando los datos de validación y `76%` utilizando los datos de prueba. Estos resultados demuestran que el modelo tiene una buena capacidad para generalización a la hora de categorizar imagenes desconocidas, sin embargo, el tiempo requerido para alcanzar niveles altos de precisión es elevado.

Estrategia 2: Red pre-entrenada

La segunda estrategia a comparar debe incluir la utilización de una **red preentrenada** con el dataset ImageNet, llevando a cabo tareas de ***transfer learning*** y ***fine-tuning*** para resolver la tarea de clasificación asignada. Deben **compararse al menos dos tipos de arquitecturas** (VGGs, ResNet50, Xception, InceptionV3, InceptionResNetV2, MobileNetV2, DenseNet, ResNet) y se debe **seleccionar la que mayor precisión proporcione** (información sobre las arquitecturas disponibles en https://keras.io/applications/). Se espera que el/la alumnx 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.).

Contexto

Para realizar la comparación del segundo punto, se seleccionaron las redes MobileNetV2 y DenseNet201.

MobileNetV2

Carga de datos para: MobileNetV2

Para realiar el aprendizaje por refuerzo, los datos fueron cargados utilizando One-Hot-Encoding, esto se realiza al configurar el parámetro `class_mode` con un valor de `'categorical'`.

```
BASE_DATASET = 'my_dataset/'

# Tamaño de las imagenes
SEED = 42
BATCH_SIZE = 32
TARGET_SIZE = (128, 128)

# Se define los path de los directorios TRAIN VALIDATION Y TEST
TRAIN_PATH = BASE_DATASET + '/train'
VAL_PATH   = BASE_DATASET + '/valid'
TEST_PATH  = BASE_DATASET + '/test'

train_datagen = ImageDataGenerator(
    rescale=1/255.,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2
)
val_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    VAL_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode='categorical'
)
```

```
Found 84635 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
```

Arquitectura de la red: MobileNetV2

Para la creación de la arquitectura, primero importamos de red MobileNetV2 disponible en la biblioteca Keras y la asignamos a la variable base_model.

La instancia del modelo incluye:

1. El parámetro `include_top` con un valor de `False` para indicar que se excluya la capa fully-connected (FC) en la parte superior de la red.
2. El parámetro, `weights` con un valor de `"imagenet"` para indicar que se carguen los pesos pre-entrenados en el conjunto de datos `ImageNet`.
3. El parámetro `input_shape` con un valor de `(TARGET_SIZE[0], TARGET_SIZE[1], 3)` que especifica la forma de los datos de entrada. El valor `3` en la tupla se utiliza para indicar que que los datos de entrada son en 3 dimensiones, en este caso imágenes a color RGB.

```
from tensorflow.keras.applications import MobileNetV2

base_model = MobileNetV2(
    include_top = False, # Indica si se incluye o no la capa completamente conectada e
    weights = 'imagenet', # Pesos pre-entrenados utilizados para inicializar el modelo
    input_shape = (TARGET_SIZE[0], TARGET_SIZE[1], 3) # Tamaño de entrada de las imáge
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
9406464/9406464 [==============================] - 0s 0us/step
```

```
base_model.summary() # muestra información sobre las capas, sus formas de salida y el
```

```
Model: "mobilenetv2_1.00_128"
__________________________________________________________________________________
 Layer (type)                Output Shape              Param #     Connected to
==================================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3       0           []
                             )]

 Conv1 (Conv2D)              (None, 64, 64, 32)        864         ['input_2[0][0]

 bn_Conv1 (BatchNormalization)  (None, 64, 64, 32)     128         ['Conv1[0][0]']

 Conv1_relu (ReLU)           (None, 64, 64, 32)        0           ['bn_Conv1[0][0

 expanded_conv_depthwise (Depth  (None, 64, 64, 32)    288         ['Conv1_relu[0]
 wiseConv2D)
```

```
expanded_conv_depthwise_BN (Ba  (None, 64, 64, 32)  128         ['expanded_conv_
tchNormalization)

expanded_conv_depthwise_relu (  (None, 64, 64, 32)  0           ['expanded_conv_
ReLU)                                                           ]']

expanded_conv_project (Conv2D)  (None, 64, 64, 16)  512         ['expanded_conv_
                                                                [0]']

expanded_conv_project_BN (Batc  (None, 64, 64, 16)  64          ['expanded_conv_
hNormalization)

block_1_expand (Conv2D)        (None, 64, 64, 96)   1536        ['expanded_conv_
                                                                ]

block_1_expand_BN (BatchNormal  (None, 64, 64, 96)  384         ['block_1_expan
ization)

block_1_expand_relu (ReLU)     (None, 64, 64, 96)   0           ['block_1_expan

block_1_pad (ZeroPadding2D)    (None, 65, 65, 96)   0           ['block_1_expan

block_1_depthwise (DepthwiseCo  (None, 32, 32, 96)  864         ['block_1_pad[0
nv2D)

block_1_depthwise_BN (BatchNor  (None, 32, 32, 96)  384         ['block_1_depth
malization)

block_1_depthwise_relu (ReLU)  (None, 32, 32, 96)   0           ['block_1_depth

block_1_project (Conv2D)       (None, 32, 32, 24)   2304        ['block_1_depth

block_1_project_BN (BatchNorma  (None, 32, 32, 24)  96          ['block_1_proje
lization)

block_2_expand (Conv2D)        (None, 32, 32, 144)  3456        ['block_1_proje

block_2_expand_BN (BatchNormal  (None, 32, 32, 144)  576        ['block_2_expan
ization)

block_2_expand_relu (ReLU)     (None, 32, 32, 144)  0           ['block_2_expan
```

Para este ejercicio, se congelaron todas las capas del modelo base, sin embargo, es posible congelar alguno de los layers de la siguiente manera:

```
# Obtener el número de capas del modelo base
num_layers = len(base_model.layers)

# Ej: congelar la mitad de las capas
for layer in base_model.layers[:num_layers//2]:
    layer.trainable = False
```

```
# Deshabilitar el entrenamiento del modelo base
base_model.trainable = False
```

```
    number of layers: 154
```

```
from tensorflow.keras.models import load_model

# Ruta para guardar el modelo después de cada época
checkpoint_path = "/content/drive/MyDrive/07MIAR/checkpoint-MobileNetV2-{epoch:02d}.h5


checkpoint_callback = [
  # Callback para reducir la tasa de aprendizaje cuando la métrica de validación se es
  tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_accuracy",
    factor = 0.2,
    patience = 2,
    mode = "auto",
    cooldown = 0,
    min_lr = 0.001,
  ),
  # Callback para detener el entrenamiento si la métrica de validación deja de mejora
  tf.keras.callbacks.EarlyStopping(
    monitor = "val_accuracy",
    patience = 5,
    verbose = 1,
    mode = "auto",
  ),
  # Callback para guardar el mejor modelo durante el entrenamiento
  tf.keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_path,
    save_weights_only = False,
    monitor = 'val_accuracy',
    mode = 'auto',
    save_best_only = True
  )
]
```

A continuación se utiliza el API secuencial para crear la arquitectura de la red. El modelo cuenta con un "top modelling" que incluye una capa de aplanamiento, seguida de una capa densa con 256 neuronas y una normalización. Además se incluyó una capa de Dropout para desahabilitar el 50% de las neuronas en cada paso. La última capa de salida cuenta con 525 neuronas que se activaran para diferenciar los valores de las clases incluidas en el set datos.

```
inputs = Input(shape =(TARGET_SIZE[0], TARGET_SIZE[1], 3), name = "input-layer")
```

```
x = base_model(inputs)

# TOP model:
xfc = Flatten()(x) # Capa de aplanamiento
xfc = Dense(256, activation="relu")(xfc) # Capa densa con activación ReLU
xfc = BatchNormalization()(xfc) # Capa de normalización
xfc = Dropout(0.5)(xfc) # Capa de regularización Dropout
predictions = Dense(525, activation="softmax")(xfc) # Capa de salida con activación so

# Unimos las entradas y el modelo mediante la función Model con parámetros inputs y ou
model_MobileNetV2 = Model(inputs=inputs, outputs=predictions) #(X)
model_MobileNetV2.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input-layer (InputLayer)    [(None, 128, 128, 3)]     0

 mobilenetv2_1.00_128 (Funct  (None, 4, 4, 1280)       2257984
 ional)

 flatten (Flatten)           (None, 20480)             0

 dense (Dense)               (None, 256)               5243136

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 525)               134925

=================================================================
Total params: 7,637,069
Trainable params: 5,378,573
Non-trainable params: 2,258,496
_________________________________________________________________
```

Para compilar el modelo, se utilizo una perdida de tipo `categorical_crossentropy`, debido a que las imágenes fueron cargadas utilizando one-hot-encoding. Adicional se utilizó un learning rate de `0.0001` el cual produjo los mejores resultados a la hora de entrenar la red convolucional del punto 1.

```
from tensorflow.keras.optimizers import Adam

model_MobileNetV2.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Entrenamiento: MobileNetV2

El entrenamiento se corrió por solo 10 épocas, la mitad de las épocas utilizadas para entrenar la red convolucional entrenada en el primer punto. Adicional, en los callbacks no se utilizó un `learning rate scheduler`, en su lugar, se utilizó "learning rate" que dio mejores resultados en el primer punto.

[ ] *5 celdas ocultas*

Conclusiones: MobileNetV2

Con los valores de prueba se obtuvo una una de pérdida de `69.79%` y una presición de `82.78`. Al evaluar el modelo sobre los datos de prueba, se obtuvo una perdida de `60.86%` y una precisión de `85.33%`. Se puede ver como el modelo pudo obtener una mejor precisión, en menos épocas y con un tiempo de entrenamiento menor si lo comparamos con la red convolucional que se entrenón en el primer punto.

DenseNet201

Para el entrenamiento de la siguiente red, se repitieron los pasos del punto anterior.

Carga de datos: DenseNet201

```
BASE_DATASET = 'my_dataset/'

# Tamaño de las imagenes
SEED = 42
BATCH_SIZE = 28
TARGET_SIZE = (112, 112)

# Se define los path de los directorios TRAIN VALIDATION Y TEST
TRAIN_PATH = BASE_DATASET + '/train'
VAL_PATH   = BASE_DATASET + '/valid'
TEST_PATH  = BASE_DATASET + '/test'

train_datagen = ImageDataGenerator(
```

```
    rescale=1/255.,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2
)
val_datagen = ImageDataGenerator(rescale=1/255.)
test_datagen = ImageDataGenerator(rescale=1/255.)

train_generator = train_datagen.flow_from_directory(
    TRAIN_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=True,
    class_mode='categorical'
)

validation_generator = val_datagen.flow_from_directory(
    VAL_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    TEST_PATH,
    target_size=TARGET_SIZE,
    batch_size=BATCH_SIZE,
    shuffle=False,
    class_mode='categorical'
)
```

```
Found 84635 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
Found 2625 images belonging to 525 classes.
```

Arquitectura de la red: DenseNet201

```
from tensorflow.keras.applications import DenseNet201

base_model = DenseNet201(
    include_top = False,
    weights = 'imagenet',
    input_shape = (TARGET_SIZE[0], TARGET_SIZE[1], 3)
)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
74836368/74836368 [==============================] - 4s 0us/step
```

```
base_model.summary()
```

```
Model: "densenet201"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 112, 112, 3  0           []
                                )]

 zero_padding2d (ZeroPadding2D)  (None, 118, 118, 3)  0          ['input_1[0][0]

 conv1/conv (Conv2D)            (None, 56, 56, 64)   9408        ['zero_padding2d

 conv1/bn (BatchNormalization)  (None, 56, 56, 64)   256         ['conv1/conv[0]

 conv1/relu (Activation)        (None, 56, 56, 64)   0           ['conv1/bn[0][0

 zero_padding2d_1 (ZeroPadding2  (None, 58, 58, 64)  0          ['conv1/relu[0]
 D)

 pool1 (MaxPooling2D)           (None, 28, 28, 64)   0           ['zero_padding2d

 conv2_block1_0_bn (BatchNormal  (None, 28, 28, 64)  256         ['pool1[0][0]']
 ization)

 conv2_block1_0_relu (Activatio  (None, 28, 28, 64)  0           ['conv2_block1_0
 n)

 conv2_block1_1_conv (Conv2D)   (None, 28, 28, 128)  8192        ['conv2_block1_0

 conv2_block1_1_bn (BatchNormal  (None, 28, 28, 128)  512        ['conv2_block1_1
 ization)

 conv2_block1_1_relu (Activatio  (None, 28, 28, 128)  0          ['conv2_block1_1
 n)

 conv2_block1_2_conv (Conv2D)   (None, 28, 28, 32)   36864       ['conv2_block1_1

 conv2_block1_concat (Concatena  (None, 28, 28, 96)  0           ['pool1[0][0]',
 te)                                                              'conv2_block1_2

 conv2_block2_0_bn (BatchNormal  (None, 28, 28, 96)  384         ['conv2_block1_c
 ization)

 conv2_block2_0_relu (Activatio  (None, 28, 28, 96)  0           ['conv2_block2_0
 n)

 conv2_block2_1_conv (Conv2D)   (None, 28, 28, 128)  12288       ['conv2_block2_0

 conv2_block2_1_bn (BatchNormal  (None, 28, 28, 128)  512        ['conv2_block2_1
 ization)

 conv2_block2_1_relu (Activatio  (None, 28, 28, 128)  0          ['conv2_block2_1
 n)

 conv2_block2_2_conv (Conv2D)   (None, 28, 28, 32)   36864       ['conv2_block2_1
```

```
conv2_block2_concat (Concatena  (None, 28, 28, 128)  0          ['conv2_block1_c
te)                                                              'conv2_block2_
```

```
num_layers = len(base_model.layers)
print(f'number of layers: {num_layers}')

# Freeze the first half of the layers
for layer in base_model.layers[:num_layers//2]:
    layer.trainable = False

base_model.trainable = False
```

```
number of layers: 707
```

```
from tensorflow.keras.models import load_model

# Save model after each epoch
checkpoint_path = "/content/drive/MyDrive/07MIAR/checkpoint-DenseNet201-{epoch:02d}.h5

checkpoint_callback = [
  tf.keras.callbacks.ReduceLROnPlateau(
    monitor = "val_accuracy",
    factor = 0.2,
    patience = 2,
    mode = "auto",
    cooldown = 0,
    min_lr = 0.001,
  ),
  tf.keras.callbacks.EarlyStopping(
    monitor = "val_accuracy",
    patience = 5,
    verbose = 1,
    mode = "auto",
  ),
  tf.keras.callbacks.ModelCheckpoint(
    filepath = checkpoint_path,
    save_weights_only = False,
    monitor = 'val_accuracy',
    mode = 'auto',
    save_best_only = True
  )
]
```

```
inputs = Input(shape =(TARGET_SIZE[0], TARGET_SIZE[1], 3), name = "input-layer")

x = base_model(inputs)
xfc = Flatten()(x) #(X)
xfc = Dense(256, activation="relu")(xfc) #(X)
```

```
xfc = BatchNormalization()(xfc) #(X)
xfc = Dropout(0.5)(xfc) #(X)
# Clasificador softmax
predictions = Dense(525, activation="softmax")(xfc) #(X)

# Unimos las entradas y el modelo mediante la función Model con parámetros inputs y ou
model_DenseNet201 = Model(inputs=inputs, outputs=predictions) #(X)
model_DenseNet201.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input-layer (InputLayer)    [(None, 112, 112, 3)]     0

 densenet201 (Functional)    (None, 3, 3, 1920)        18321984

 flatten (Flatten)           (None, 17280)             0

 dense (Dense)               (None, 256)               4423936

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 dropout (Dropout)           (None, 256)               0

 dense_1 (Dense)             (None, 525)               134925

=================================================================
Total params: 22,881,869
Trainable params: 4,559,373
Non-trainable params: 18,322,496
_________________________________________________________________
```

```
from tensorflow.keras.optimizers import Adam

model_DenseNet201.compile(
    optimizer=Adam(learning_rate=0.00012),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Entrenamiento: DenseNet201

```
history = model_DenseNet201.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,
    verbose=1,
    callbacks=[checkpoint_callback]
)
```

```
Epoch 1/10
3023/3023 [==============================] - 467s 154ms/step - loss: 4.2206 - ac
Epoch 2/10
3023/3023 [==============================] - 452s 150ms/step - loss: 2.6416 - ac
Epoch 3/10
3023/3023 [==============================] - 449s 148ms/step - loss: 2.0788 - ac
Epoch 4/10
3023/3023 [==============================] - 443s 147ms/step - loss: 1.8249 - ac
Epoch 5/10
3023/3023 [==============================] - 441s 146ms/step - loss: 1.6732 - ac
Epoch 6/10
3023/3023 [==============================] - 440s 145ms/step - loss: 1.5715 - ac
Epoch 7/10
3023/3023 [==============================] - 442s 146ms/step - loss: 1.4996 - ac
Epoch 8/10
3023/3023 [==============================] - 443s 146ms/step - loss: 1.4420 - ac
Epoch 9/10
3023/3023 [==============================] - 444s 147ms/step - loss: 1.4009 - ac
Epoch 10/10
3023/3023 [==============================] - 442s 146ms/step - loss: 1.3658 - ac
```

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


```
loss, Accuracy = model_DenseNet201.evaluate(test_generator)
print(f'real loss: {loss}, real accuracy: {Accuracy}')
```

```
94/94 [==============================] - 8s 89ms/step - loss: 0.4662 - accuracy:
real loss: 0.46615225076675415, real accuracy: 0.8761904835700989
```

```
predictions = model_DenseNet201.predict(test_generator, batch_size=BATCH_SIZE)
predicted_categories = np.argmax(predictions, axis=1)
```

```
94/94 [==============================] - 9s 67ms/step
```

```
from sklearn.metrics import classification_report

print(classification_report(df_test['class id'], predicted_categories))
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.40      0.57         5
         1.0       1.00      0.40      0.57         5
         2.0       1.00      1.00      1.00         5
         3.0       0.71      1.00      0.83         5
         4.0       1.00      0.80      0.89         5
         5.0       0.67      0.80      0.73         5
         6.0       1.00      1.00      1.00         5
         7.0       1.00      0.80      0.89         5
         8.0       0.80      0.80      0.80         5
         9.0       0.80      0.80      0.80         5
        10.0       1.00      0.60      0.75         5
        11.0       1.00      1.00      1.00         5
        12.0       0.83      1.00      0.91         5
        13.0       1.00      0.40      0.57         5
        14.0       1.00      1.00      1.00         5
        15.0       1.00      0.80      0.89         5
        16.0       0.83      1.00      0.91         5
        17.0       0.80      0.80      0.80         5
        18.0       1.00      1.00      1.00         5
```

19.0	0.62	1.00	0.77	5
20.0	1.00	1.00	1.00	5
21.0	1.00	0.80	0.89	5
22.0	1.00	0.80	0.89	5
23.0	0.83	1.00	0.91	5
24.0	0.67	0.40	0.50	5
25.0	1.00	0.60	0.75	5
26.0	1.00	0.80	0.89	5
27.0	1.00	1.00	1.00	5
28.0	1.00	0.60	0.75	5
29.0	1.00	1.00	1.00	5
30.0	0.80	0.80	0.80	5
31.0	0.80	0.80	0.80	5
32.0	1.00	0.80	0.89	5
33.0	0.75	0.60	0.67	5
34.0	1.00	1.00	1.00	5
35.0	0.42	1.00	0.59	5
36.0	1.00	1.00	1.00	5
37.0	1.00	0.60	0.75	5
38.0	0.83	1.00	0.91	5
39.0	1.00	1.00	1.00	5
40.0	1.00	0.80	0.89	5
41.0	1.00	0.80	0.89	5
42.0	1.00	1.00	1.00	5
43.0	0.62	1.00	0.77	5
44.0	0.36	0.80	0.50	5
45.0	0.56	1.00	0.71	5
46.0	0.83	1.00	0.91	5
47.0	0.83	1.00	0.91	5
48.0	1.00	1.00	1.00	5
49.0	1.00	0.40	0.57	5
50.0	1.00	0.40	0.57	5
51.0	1.00	0.80	0.89	5
52.0	0.67	0.80	0.73	5
53.0	0.71	1.00	0.83	5
54.0	1.00	0.80	0.89	5
55.0	0.83	1.00	0.91	5

Conclusiones: DenseNet201

Con los valores de prueba se obtuvo una una de pérdida de `59.69%` y una presición de `84.42`. Al evaluar el modelo sobre los datos de prueba, se obtuvo una perdida de `46.61%` y una precisión de `87.61%`. Se puede ver como el modelo pudo obtener una mejor precisión, en menos épocas y con un tiempo de entrenamiento menor si lo comparamos con la red convolucional que se entrenón en el primer punto.

Comparación de las soluciones

En la siguiente gráfica se puede ver un resumen de las métricas de los tres modelos entrenados:

Modelo	*accuracy*	*loss*
CNN	76.00	1.0268
*MobileNetV*2	85.33	60.86
*DenseNet*201	87.61	46.61

De la tabla inferir que los mejores resultados se obtuvieron al entrenar una red con un modelo base que implemente la arquitectura `DenseNet201`. Este modelo presentó las menores pérdidas y la mayor precisión.

A partir de estos resultados, se puede inferir que el modelo basado en la arquitectura `DenseNet201`