Alumnos:

- **Frida Cano Falcón**
  Matrícula: A01752953
- **Jorge Javier Sosa Briseño**
  Matrícula: A01749489
- **Guillermo Romeo Cepeda Medina**
  Matrícula: A01284015
- **Daniel Saldaña Rodríguez**
  Matrícula: A00829752

# Convolutional Neural Networks y Transfer Learning para el modelo y refinamiento

En este código se entrena un modelo de red neuronal convolucional (CNN) utilizando Transfer Learning (TL) con la arquitectura VGG16.

El objetivo de este documento es mostrar el uso de la técnica de Transfer Learning mediante la implementación de modelos pre-entrenados para posteriormente entrenarlos con nuevos datos y ser usados para clasficiar nuevos objetivos. En este caso su objetivo es clasificar rostros en diferentes categorías.

## Importar dependencias

```
# @title **Importar dependencias**
import numpy as np
import matplotlib.pyplot as plt
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from keras import layers
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.models import Sequential, load_model
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint


import numpy as np
from PIL import Image
import os
import zipfile as zf

# Load the data
data = np.load("/content/olivetti_faces.npy.zip")
target = np.load("/content/olivetti_faces_target.npy")

# Extract the .npy file from the .zip
with zf.ZipFile("/content/olivetti_faces.npy.zip", 'r') as zip_ref:
    zip_ref.extractall('images')

data1 = np.load("/content/images/olivetti_faces.npy")

# Ensure the directory to save images exists
output_dir = "output_images_first_3_classes"
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Save images
for index, image_array in enumerate(data1):
    # Only save images for the first 3 classes (0, 1, 2)
    if target[index] > 3:
       break

    # Convert the numpy array to an image
    img = Image.fromarray((image_array * 255).astype(np.uint8))

    # Create a directory for the class if it doesn't exist
    class_dir = os.path.join(output_dir, str(target[index]))
    if not os.path.exists(class_dir):
        os.makedirs(class_dir)

    # Save the image to the corresponding class folder
    img_path = os.path.join(class_dir, f"image_{index}.png")
```

```
    img.save(img_path)

print("Images from the first 3 classes saved!")
```

```
Images from the first 3 classes saved!
```

```
from keras.preprocessing.image import ImageDataGenerator

img_size = (256, 256)
batch_size = 8
data_dir = "/content/output_images_first_3_classes"

# Data augmentation
train_datagen = ImageDataGenerator(
        validation_split=0.2,
        shear_range=0.4,
        zoom_range=0.4,
        rotation_range=60,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(
        validation_split=0.2)

# Data split Train and Validation
train_ds = train_datagen.flow_from_directory(
        data_dir,
        subset='training',
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        seed=42)

valid_ds = test_datagen.flow_from_directory(
        data_dir,
        subset='validation',
        target_size=img_size,
        batch_size=batch_size,
        class_mode='categorical',
        seed=42)

class_names = list(train_ds.class_indices.keys())
print('Class names:', class_names)
```

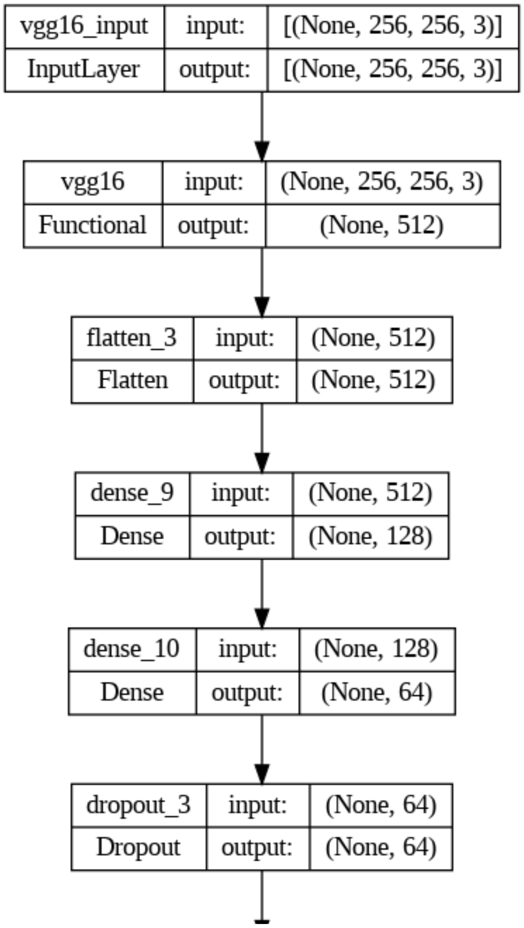
```
Found 32 images belonging to 4 classes.
Found 8 images belonging to 4 classes.
Class names: ['0', '1', '2', '3']
```

```
import numpy as np
import matplotlib.pyplot as plt
import zipfile as zf

# Load the data
with zf.ZipFile("/content/olivetti_faces.npy.zip", 'r') as zip_ref:
    zip_ref.extractall('/content/')

data = np.load("/content/olivetti_faces.npy")
target = np.load("/content/olivetti_faces_target.npy")

# Set how many images per class you want to display
images_per_class = 10

# Find unique classes in the dataset
classes = np.unique(target)

# Set the figure size
plt.figure(figsize=(15, 15))

# We will only take the first 3 classes and display 10 images per class
for class_index in classes:
    if class_index > 2:  # We only want the first 3 classes
        break
    # Get the indices of images that belong to the current class
    indices = np.where(target == class_index)[0][:images_per_class]
    for i, img_index in enumerate(indices, start=1):
        # Calculate the position of the subplot
        plt.subplot(3, images_per_class, class_index * images_per_class + i)
        plt.imshow(data[img_index], cmap='gray')
        plt.axis('off')

# Adjust the layout of the plots
plt.tight_layout()
```

```
plt.show()
```

## Importar modelo VGG16

```
# @title **Importar modelo VGG16**

pretrained_model = keras.applications.VGG16(
  include_top=False,
  input_shape=img_size+(3,),  # Corrected input shape to match ResNet-50
  pooling='avg',
  classes=len(class_names),
  weights='imagenet')

# Transfer Learning
for layer in pretrained_model.layers:
  layer.trainable = False

model = Sequential()
model.add(pretrained_model)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.15))
model.add(Dense(len(class_names), activation='softmax'))
model.build(input_shape=img_size+(3,))

from keras.utils import plot_model
from keras.models import load_model  # Import your model

plot_model(model, show_shapes=True, show_layer_names=True)
```



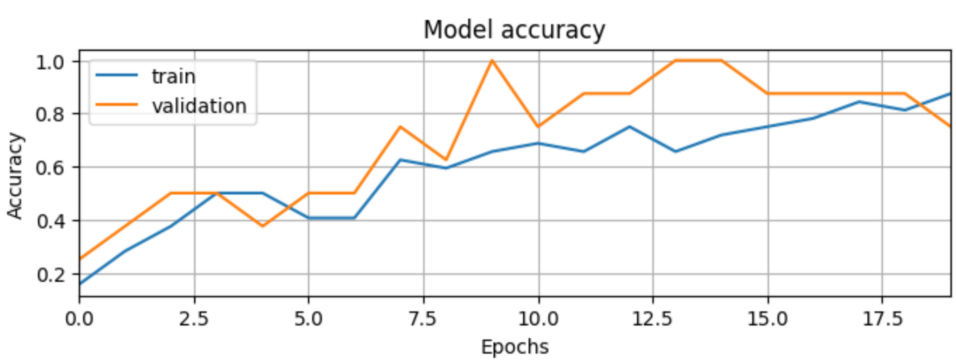

```
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy'])


epochs = 20
checkpoint = ModelCheckpoint(
    "best_model.h5",
    monitor="val_accuracy",
    save_best_only=True,
    mode="max")

# Train the model with the callback
history = model.fit(
    train_ds,
    validation_data=valid_ds,
    epochs=epochs,
    callbacks=[checkpoint])

# Load the best model weights
model.load_weights("best_model.h5")
```

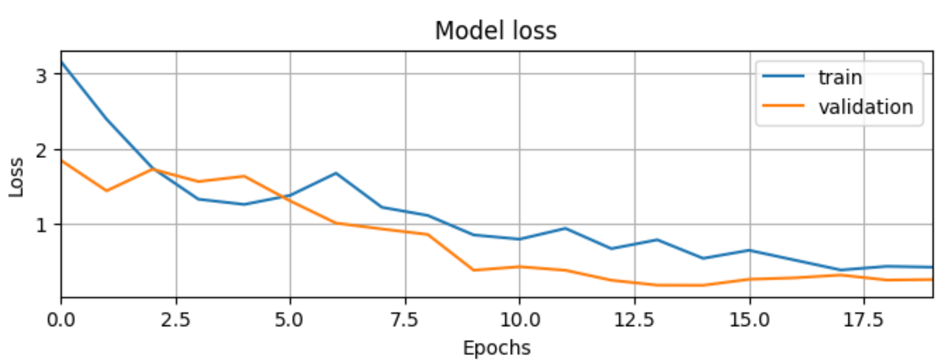
```
Epoch 1/20
4/4 [==============================] - 2s 260ms/step - loss: 3.1656 - accuracy: 0.1562 - val_loss: 1.8455 - val_accuracy: 0.2500
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079: UserWarning: You are saving your model as an HDF5 file
  saving_api.save_model(
Epoch 2/20
4/4 [==============================] - 1s 262ms/step - loss: 2.3929 - accuracy: 0.2812 - val_loss: 1.4370 - val_accuracy: 0.3750
Epoch 3/20
4/4 [==============================] - 1s 213ms/step - loss: 1.7414 - accuracy: 0.3750 - val_loss: 1.7254 - val_accuracy: 0.5000
Epoch 4/20
4/4 [==============================] - 1s 162ms/step - loss: 1.3250 - accuracy: 0.5000 - val_loss: 1.5610 - val_accuracy: 0.5000
Epoch 5/20
4/4 [==============================] - 1s 160ms/step - loss: 1.2552 - accuracy: 0.5000 - val_loss: 1.6317 - val_accuracy: 0.3750
Epoch 6/20
4/4 [==============================] - 1s 181ms/step - loss: 1.3761 - accuracy: 0.4062 - val_loss: 1.3022 - val_accuracy: 0.5000
Epoch 7/20
4/4 [==============================] - 1s 162ms/step - loss: 1.6722 - accuracy: 0.4062 - val_loss: 1.0057 - val_accuracy: 0.5000
Epoch 8/20
4/4 [==============================] - 1s 233ms/step - loss: 1.2169 - accuracy: 0.6250 - val_loss: 0.9278 - val_accuracy: 0.7500
Epoch 9/20
4/4 [==============================] - 1s 163ms/step - loss: 1.1082 - accuracy: 0.5938 - val_loss: 0.8539 - val_accuracy: 0.6250
Epoch 10/20
4/4 [==============================] - 1s 203ms/step - loss: 0.8477 - accuracy: 0.6562 - val_loss: 0.3768 - val_accuracy: 1.0000
```

```
Epoch 11/20
4/4 [==============================] - 1s 155ms/step - loss: 0.7915 - accuracy: 0.6875 - val_loss: 0.4238 - val_accuracy: 0.7500
Epoch 12/20
4/4 [==============================] - 1s 155ms/step - loss: 0.9354 - accuracy: 0.6562 - val_loss: 0.3761 - val_accuracy: 0.8750
Epoch 13/20
4/4 [==============================] - 1s 159ms/step - loss: 0.6646 - accuracy: 0.7500 - val_loss: 0.2430 - val_accuracy: 0.8750
Epoch 14/20
4/4 [==============================] - 1s 263ms/step - loss: 0.7831 - accuracy: 0.6562 - val_loss: 0.1777 - val_accuracy: 1.0000
Epoch 15/20
4/4 [==============================] - 1s 193ms/step - loss: 0.5352 - accuracy: 0.7188 - val_loss: 0.1756 - val_accuracy: 1.0000
Epoch 16/20
4/4 [==============================] - 1s 160ms/step - loss: 0.6431 - accuracy: 0.7500 - val_loss: 0.2556 - val_accuracy: 0.8750
Epoch 17/20
4/4 [==============================] - 1s 170ms/step - loss: 0.5114 - accuracy: 0.7812 - val_loss: 0.2754 - val_accuracy: 0.8750
Epoch 18/20
4/4 [==============================] - 1s 161ms/step - loss: 0.3798 - accuracy: 0.8438 - val_loss: 0.3128 - val_accuracy: 0.8750
Epoch 19/20
4/4 [==============================] - 1s 156ms/step - loss: 0.4288 - accuracy: 0.8125 - val_loss: 0.2458 - val_accuracy: 0.8750
Epoch 20/20
4/4 [==============================] - 1s 158ms/step - loss: 0.4179 - accuracy: 0.8750 - val_loss: 0.2524 - val_accuracy: 0.7500
```

## Visualización de resultados

```
# @title **Visualización de resultados**

fig1 = plt.figure(figsize=(8, 5))
plt.subplot(2,1,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(['train', 'validation'])
plt.axis(xmin=0,xmax=epochs-1)
plt.grid()
plt.show()

fig2 = plt.figure(figsize=(8, 5))
plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['train', 'validation'])
plt.axis(xmin=0,xmax=epochs-1)
plt.grid()
plt.show()
```



LINK:

https://colab.research.google.com/drive/1509oT-an-lB_noAxdEGnhrQQ6XqnRK1V#revisionId=0Bz4GkKYXD_MjU25wbmh3eFFQSWo4NndDdEwzbjZ1bFdzTElBPQ&scrollTo=YKraoC5_53qw