Instituto Tecnológico y de Estudios Superiores de Monterrey

Momento de Retroalimentación Individual: Implementación de un modelo de Deep Learning

Diego Alberto Baños Lopez A01275100

Monterrey, Nuevo León, México 04 de Noviembre del 2023

Inteligencia artificial avanzada para la ciencia de datos II (Gpo 501)

En este documento se busca aplicar los conocimientos de deep learning y con ello poder realizar en base a redes neuronales convolutivas (CNN) un modelo que pueda identificar caras de famosos, en este caso se usará el dataset de Labelled Faces in the Wild que es un dataset Fotografías de rostros diseñadas para el estudio del problema de reconocimiento de rostros sin restricciones. Esta base de datos fue creada y mantenida por investigadores de la Universidad de Massachusetts, Amherst y se encuentra disponible en la página de Kaggle

Empezaremos con la importación de las librerías, las cuales juegan con un papel indispensable para poder realizar el entrenamiento de dicho modelo

```
# %pip install opencv-python matplotlib pandas numpy Pillow mtcnn keras
scikit-learn
# %pip install tensorflow
import os
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from collections import OrderedDict

# Visualización
from PIL import Image

# Bounding Boxes
import matplotlib.patches as patches

# CNN
import keras
from sklearn.model_selection import train_test_split

# Directorios y Archivos
import shutil
from shutil import unpack_archive
from subprocess import check_output
import tensorflow as tf
```

Una vez hecho ello creamos nuestros valores en donde se indican en donde se ubicara los archivos necesarios para la designación de variables y la creacion de data frames necesarios para el modelo, se hará una explicación detallada de ello más adelante

```
# Ubicación de base
base_path = "./lfw-deepfunneled2/"

# Nuevas ubicaciones de archivos
lfw_allnames = pd.read_csv(base_path + "lfw_allnames.csv")
matchpairsDevTest = pd.read_csv(base_path + "matchpairsDevTest.csv")
matchpairsDevTrain = pd.read_csv(base_path + "matchpairsDevTrain.csv")
mismatchpairsDevTest = pd.read_csv(base_path +
"mismatchpairsDevTest.csv")
mismatchpairsDevTrain = pd.read_csv(base_path +
"mismatchpairsDevTrain.csv")
pairs = pd.read_csv(base_path + "pairs.csv")

# Datos de pares ordenados:
pairs = pairs.rename(columns={"name": "name1", "Unnamed: 3": "name2"})
matched_pairs = pairs[pairs["name2"].isnull()].drop("name2", axis=1)
mismatched_pairs = pairs[pairs["name2"].notnull()]

# Ubicación de personas
people_path = base_path + "people.csv"
people = pd.read_csv(people_path)

# Eliminación de Valores Nulos
people = people[people.name.notnull()]
peopleDevTest = pd.read_csv(base_path + "peopleDevTest.csv")
peopleDevTrain = pd.read_csv(base_path + "peopleDevTrain.csv")
```

Tras haber leído y obtenido los dataframes el siguiente paso es explorar y analizar los datos que en este caso serían los datos del dataset anteriormente mencionado. Se detectaron y centraron 13,233 imágenes de 5,749 personas mediante el detector de rostros Viola Jones y se recopilaron de la web. 1,680 de las personas retratadas tienen dos o más fotos distintas en el conjunto de datos. La base de datos original contiene cuatro conjuntos diferentes de imágenes de LFW y también tres tipos diferentes de imágenes "alineadas".

Es fundamental comprender la estructura de los datos antes de comenzar a desarrollar el modelo. A continuación, se dará un resumen de los archivos csv disponibles en el conjunto de datos LDW, junto con una breve descripción de su contenido:

- lfw_allnames.csv: Lista los nombres de los rostros del conjunto de datos y la cantidad de imágenes disponibles por rostro.
- pairs.csv: Proporciona conjuntos de datos para validación cruzada de 10 pliegues, formados por pares de imágenes, tanto coincidentes como no coincidentes, para configuraciones de entrenamiento restringidas.

- people.csv: Similar a pairs.csv pero para configuraciones no restringidas, proporcionando conjuntos de datos basados en rostros individuales para la validación cruzada de 10 pliegues.
- matchpairsDevTest.csv: Contiene 500 pares de rostros coincidentes para el conjunto de pruebas en la configuración de pares.
- matchpairsDevTrain.csv: Incluye 1100 pares de rostros coincidentes para el conjunto de entrenamiento en la configuración de pares.
- mismatchpairsDevTest.csv: Contiene 500 pares de rostros no coincidentes para el conjunto de pruebas en la configuración de pares.
- mismatchpairsDevTrain.csv: Incluye 1100 pares de rostros no coincidentes para el conjunto de entrenamiento en la configuración de pares.
- peopleDevTest.csv: Ofrece un conjunto de pruebas con 1711 personas y 3708 imágenes para la configuración basada en personas.
- peopleDevTrain.csv: Proporciona un conjunto de entrenamiento con 4038 personas y 9525 imágenes para la configuración basada en personas

En el siguiente código se dividirá los conjuntos de datos en entrenamiento y prueba dependiendo de cada archivo csv mencionado anteriormente

```
# Reorganiza el marco de datos para que haya una fila por imagen,
relacionada con el archivo jpg correspondiente.
image_paths =
lfw_allnames.loc[lfw_allnames.index.repeat(lfw_allnames["images"])]
image_paths["image_path"] = 1 + image_paths.groupby("name").cumcount()
image_paths["image_path"] = image_paths.image_path.apply(lambda x:
"{0:0>4}".format(x))
image_paths["image_path"] = (
image_paths.name + "/" + image_paths.name + "_" +
image_paths.image_path + ".jpg"
)
image_paths = image_paths.drop("images", axis=1)

# Toma una muestra aleatoria: el 80% de los datos para el conjunto de
pruebas.
lfw_train, lfw_test = train_test_split(image_paths, test_size=0.2)
lfw_train = lfw_train.reset_index().drop("index", axis=1)
lfw_test = lfw_test.reset_index().drop("index", axis=1)

# Se verifica que haya una mezcla de individuos conocidos y
desconocidos en el conjunto de pruebas.
print(len(set(lfw_train.name).intersection(set(lfw_test.name))))
print(len(set(lfw_test.name) - set(lfw_train.name)))
```

Salida de la celda:
872
822

En la siguiente celda comprobamos que la resolución de las imágenes sea consistente, esto para evitar algún tipo de problema relacionado a ello, recordemos que para los modelos CNN es importante garantizar la uniformidad de los datos, en este caso la resolución, ya que de lo contrario podríamos obtener un entrenamiento del modelo bastante inestable y poco funcional.

```
# Verificar que la resolución de las imagenes sea consistente
widths = []
heights = []
files = image_paths.image_path
base_path = base_path + "lfw-deepfunneled/"
for file in files:
path = base_path + str(file)
im = Image.open(path)
widths.append(im.width)
heights.append(im.height)

pd.DataFrame({"height": heights, "width": widths}).describe()
```

Salida:

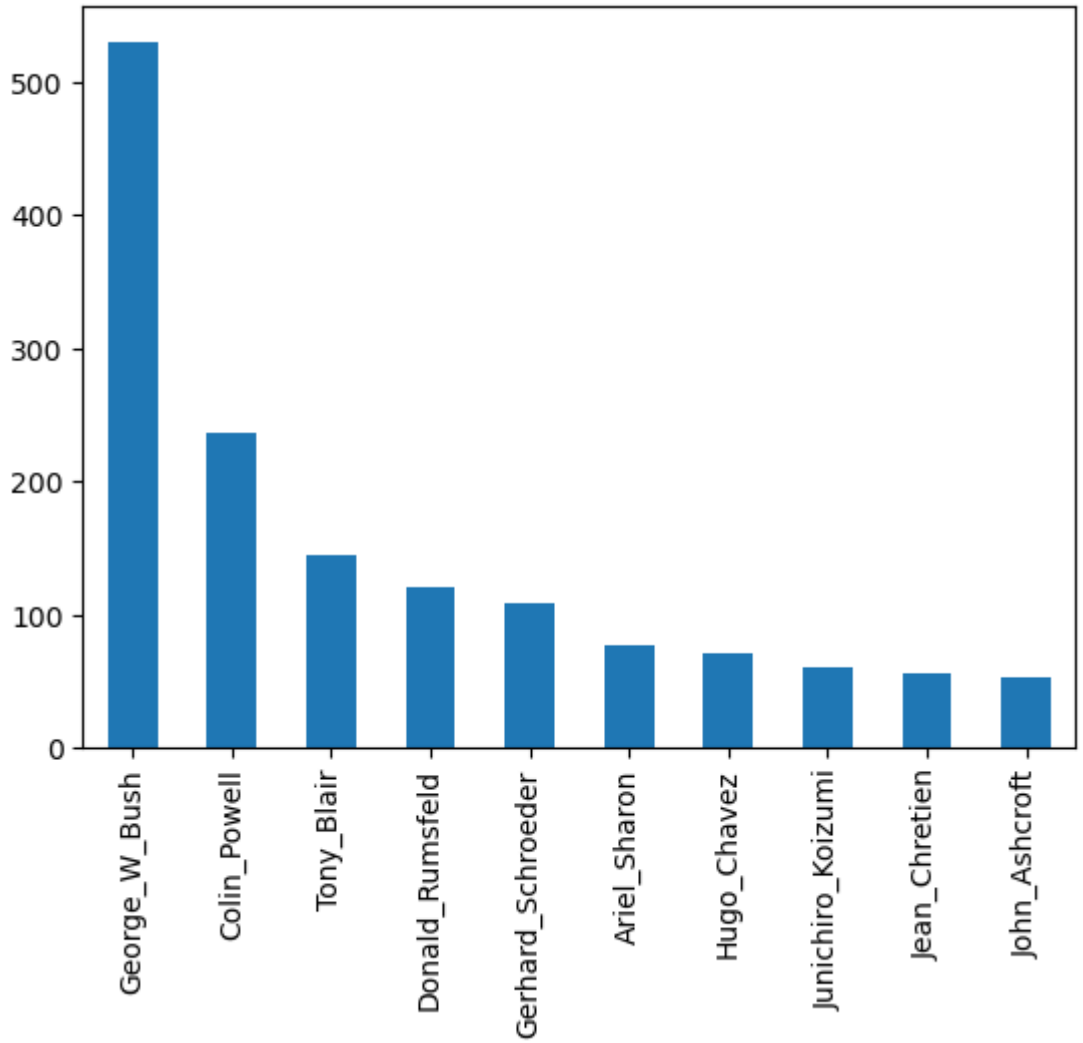
	height	width
count	13233.0	13233.0
mean	250.0	250.0
std	0.0	0.0
min	250.0	250.0
25%	250.0	250.0
50%	250.0	250.0
75%	250.0	250.0
max	250.0	250.0

Se puede observar que las imágenes del dataset son consistentes, teniendo una resolución de 250 x 250 pixeles.

Una vez observado la consistencia pasamos a hacer un análisis de las clases del dataset para poder observar potenciales sesgos que pueda ser generados bajo este dataset, debido a que no

hay la información más allá de la foto y de quien le pertenece se tendrán que hacer inferencias, primero observaremos los nombres que tienen más imágenes y en base a ello podremos asumir sesgos acorde a la distribución de los datos

```
image_paths["name"].value_counts()[:10].plot(kind="bar")
```



A primera vista podemos observar que dentro de los nombres con más imágenes se encuentran mayormente políticos del sexo masculino, por lo que puede indicar que el dataset se encuentra dominado por figuras publicas con este perfil, y por lo tanto haciendo que se pueda inferir de manera segura que hay sesgo en la parte del sexo de las personas, asi como la edad de estas, ya que las personas que se muestran que predominan en el dataset se nota que son personas que superan la edad de los 40 años.
Para continuar con ello se intenta inferir si existe algún posible subrepresentación de rostros con el siguiente codigo

```
ind_counts = image_paths.groupby("name").count().image_path
print(
str(sum(ind_counts[ind_counts == 1]))
+ " personas las cuales el "
```

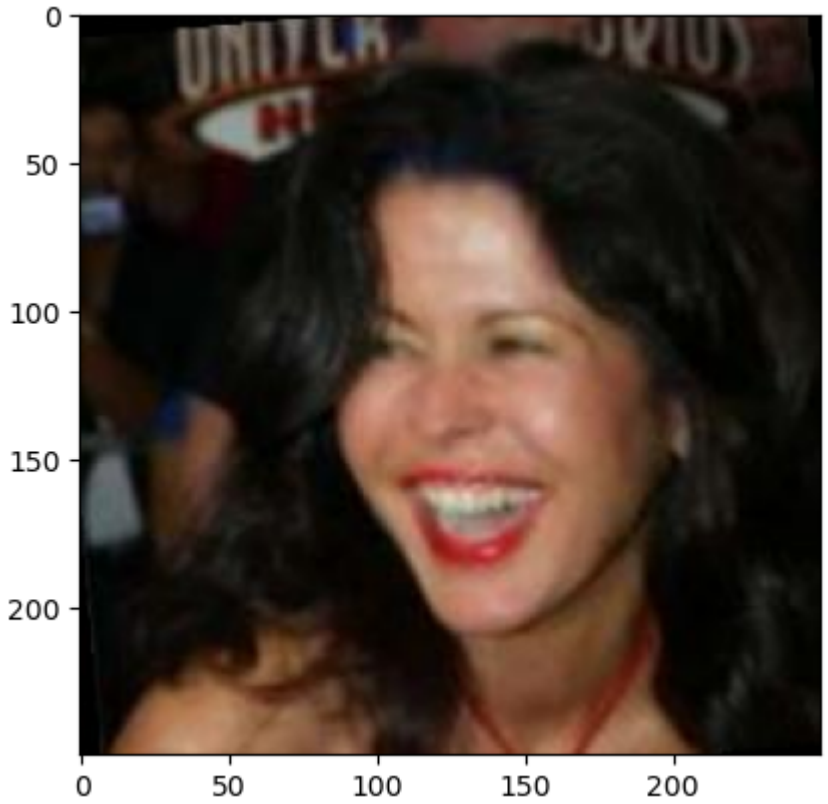
```
+ str(round(100 * (sum(ind_counts[ind_counts == 1]) /
sum(ind_counts))))
+ "% de las personas consideradas en total, sólo están representadas
por una única imagen en este conjunto de datos."
)
```

Salida:
4069 personas las cuales el 31% de las personas consideradas en total, sólo están representadas por una única imagen en este conjunto de datos.

Este código nos indica que potencialmente el 31% está subrepresentado y a la hora de que el dataset sea usado para entrenar modelos, puedan que no sean tan identificables a diferencia de las personas que posean una mayor cantidad de imágenes, por lo que podemos inferir que muy probablemente el dataset contiene sesgos relacionados tanto al género como a la edad de una persona, haciendo que posiblemente el modelo a la hora de ser entrenado pueda tener problemas en identificar a personas que no contengan las características de las personas con mayor cantidad de datos de este último. Esto no puede ser confirmado al 100% ya que se requeriría de un análisis más extenso para ello pero esto sale del foco de este documento.

Una vez realizado el análisis cenamos que efectivamente el dataset está cargado en el notebook

```
im = Image.open(base_path + str(lfw_train.image_path[0]))
plt.imshow(im)
```



Una vez comprobado que las imágenes cargan procedemos con la construcción del modelo CNN, en donde primeramente se empieza a cargar el dataset, así como se empieza a cargar el framework de Keras en donde se cargará todo lo necesario para entrenar nuestro modelo CNN, este modelo será entrenado con el top 6 personas que cuentan con la mayor cantidad de datos en el dataset.

```
# inicializar red secuencial
from keras.models import Sequential

# incluir capas convolucionales
from keras.layers import Conv2D

# Pooling
from keras.layers import MaxPooling2D

#Aplana las capas en un único vector
from keras.layers import Flatten
from keras.layers import Dense

import keras.utils as image

from tensorflow.keras.preprocessing import image

top_base = "./"
```

```
#Funcion para copiar datos y poder usarlos de manera mas sencilla
def directory_mover(data, dir_name):
co = 0
for image in data.image_path:

if not os.path.exists(os.path.join(dir_name)):
shutil.os.mkdir(os.path.join(top_base, dir_name))
data_type = data[data["image_path"] == image]["name"]
data_type = str(list(data_type)[0])
if not os.path.exists(os.path.join(top_base, dir_name, data_type)):
shutil.os.mkdir(os.path.join(top_base, dir_name, data_type))
path_from = os.path.join(base_path, image)
path_to = os.path.join(top_base, dir_name, data_type)
shutil.copy(path_from, path_to)
co += 1

print("Se movieron {} imagenes a la carpeta {}.".format(co, dir_name))
```

```
image_paths["name"].value_counts()[:6]
```

Salida:
George_W_Bush 530
Colin_Powell 236
Tony_Blair 144
Donald_Rumsfeld 121
Gerhard_Schroeder 109
Ariel_Sharon 77
Name: name, dtype: int64

iniciaremos el proceso de modelado utilizando el modelo secuencial de Keras. A este modelo base, le agregaremos capas adicionales para aumentar su complejidad. La primera capa que agregamos es una capa convolucional 2D, que se desplaza sobre la imagen con una ventana de 3x3 y espera una entrada de imagen de 250 x 250 píxeles.

Usamos la función de activación ‘ReLU’ (Unidades Lineales Rectificadas) para introducir no linealidad en el modelo. Esta función es simple y ofrece tiempos de computación rápidos.

```
#Creacion del dataset
multi_data = pd.concat(
[
image_paths[image_paths.name == "George_W_Bush"].sample(75),
```

```
image_paths[image_paths.name == "Colin_Powell"].sample(75),
image_paths[image_paths.name == "Tony_Blair"].sample(75),
image_paths[image_paths.name == "Donald_Rumsfeld"].sample(75),
image_paths[image_paths.name == "Gerhard_Schroeder"].sample(75),
image_paths[image_paths.name == "Ariel_Sharon"].sample(75),
]
)
# retener datos finales de prueba
multi_train, multi_test = train_test_split(multi_data, test_size=0.2)
# dividir en datos de validación
multi_train, multi_val = train_test_split(multi_train, test_size=0.2)

# Configuración para el modelo CNN
multi_classifier = Sequential()
multi_classifier.add(Conv2D(32, (3, 3), input_shape=(250, 250, 3),
activation="relu"))
multi_classifier.add(MaxPooling2D(pool_size=(2, 2)))
multi_classifier.add(Flatten())
multi_classifier.add(Dense(units=128, activation="relu"))

# Como estamos entrenando en varias clases, necesitamos varias unidades
de clasificación (una para cada clase). También usamos una
# función de activación softmax
multi_classifier.add(Dense(units=6, activation="softmax"))
# Cambiamos la función de pérdida a categorical_crossentropy
multi_classifier.compile(
optimizer="adam", loss="categorical_crossentropy", metrics=["accuracy"]
)

# mover imágenes a directorios separados
directory_mover(multi_train, "train_multi/")
directory_mover(multi_val, "val_multi/")
directory_mover(multi_test, "test_multi/")

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
rescale=1.0 / 255, shear_range=0.2, zoom_range=0.2,
horizontal_flip=True
)
test_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0 / 255)
multi_training_set = train_datagen.flow_from_directory(
top_base + "/train_multi/",
target_size=(250, 250),
```

```
batch_size=32,
class_mode="categorical",
)
multi_val_set = test_datagen.flow_from_directory(
top_base + "/val_multi/",
target_size=(250, 250),
batch_size=32,
class_mode="categorical",
)
multi_test_set = test_datagen.flow_from_directory(
top_base + "/test_multi/",
target_size=(250, 250),
batch_size=32,
class_mode="categorical",
)


multi_history = multi_classifier.fit_generator(
multi_training_set,
# establecer pasos por época igual al número de imágenes de
entrenamiento
steps_per_epoch=len(multi_training_set),
epochs=14,
validation_data=multi_val_set,
validation_steps=len(multi_val_set),
)
```

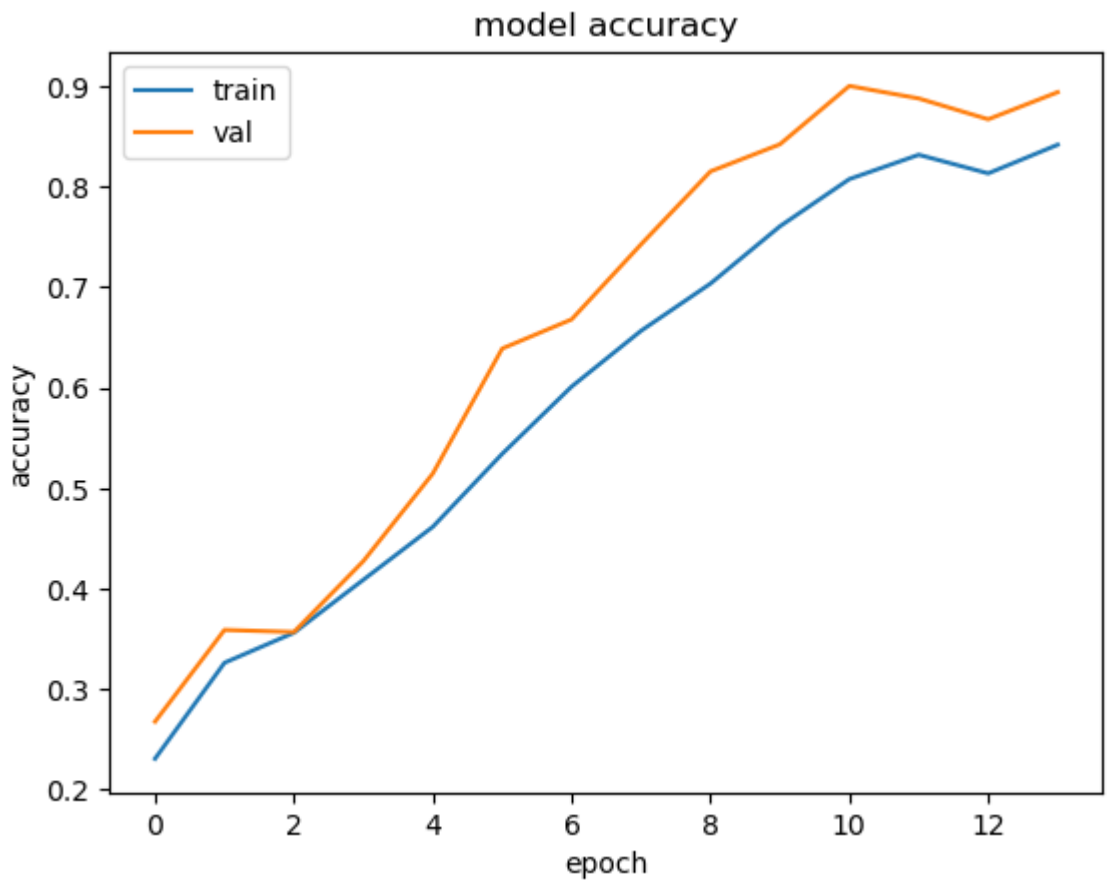
Salida:
Se movieron 288 imagenes a la carpeta train_multi/.
Se movieron 72 imagenes a la carpeta val_multi/.
Se movieron 90 imagenes a la carpeta test_multi/.
Found 702 images belonging to 6 classes.
Found 482 images belonging to 6 classes.
Found 509 images belonging to 6 classes.
…
Epoch 11/14
22/22 [==============================] - 16s 733ms/step - loss: 0.5994 - accuracy: 0.8077 - val_loss: 0.4206 - val_accuracy: 0.9004
Epoch 12/14
22/22 [==============================] - 16s 726ms/step - loss: 0.5397 - accuracy: 0.8319 - val_loss: 0.4045 - val_accuracy: 0.8880
Epoch 13/14

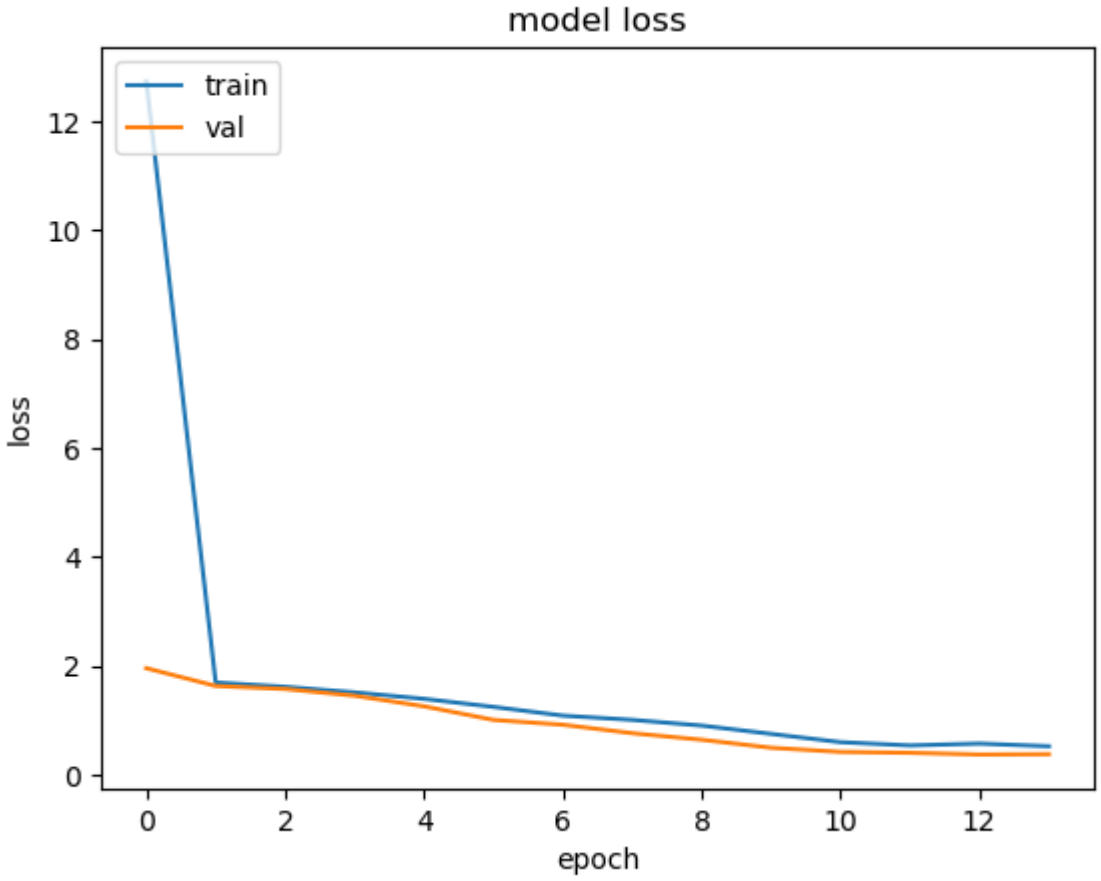
22/22 [==============================] - 17s 754ms/step - loss: 0.5737 - accuracy: 0.8134 - val_loss: 0.3728 - val_accuracy: 0.8672
Epoch 14/14
22/22 [==============================] - 17s 745ms/step - loss: 0.5245 - accuracy: 0.8419 - val_loss: 0.3788 - val_accuracy: 0.8942

Una vez entrenado el modelo empezamos a realizar las comparativas y graficar los resultados del modelo

```
# Gráficos para comparar resultados del modelo
plt.plot(multi_history.history["accuracy"])
plt.plot(multi_history.history["val_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train", "val"], loc="upper left")
plt.show()

plt.plot(multi_history.history["loss"])
plt.plot(multi_history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "val"], loc="upper left")
plt.show()
```


model accuracy
train
val
accuracy
epoch
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
2
4
6
8
10
12


model loss
train
val
loss
epoch
0
2
4
6
8
10
12
0
2
4
6
8
10
12

```
multi_test_names = []
for i in range(len(multi_test_set.filenames)):
multi_test_names.append(multi_test_set.filenames[i])
for i in range(len(multi_test_names)):
multi_test_names[i] = multi_test_names[i].split("/")[0]
multi_test_name_order = list(OrderedDict.fromkeys(multi_test_names))


# Función para realizar predicciones en el modelo CNN
def predictions(dir_name, classifier, binary):
predictions = []
for im in os.listdir(dir_name):
test_image = image.load_img(dir_name + str(im), target_size=(250, 250))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
if binary is True:
result = float(str(classifier.predict(test_image))[2])
else:
result = np.argmax(classifier.predict(test_image))
predictions.append(result)
return predictions

# función para encontrar la precisión y el recall de las predicciones,
def prec_acc(predictions_frame):
precision = []
accuracy = []
recall = []
for i in range(len(set(predictions_frame.Predictions))):
# Verdaderos positivos: Casos donde la predicción y el valor actual son
i
tp = predictions_frame[
np.logical_and(
predictions_frame["Actual"] == i, predictions_frame["Predictions"] == i
)
].shape[0]
# Verdaderos negativos: Casos donde ni la predicción ni el valor actual
son i
tn = predictions_frame[
np.logical_and(
predictions_frame["Actual"] != i, predictions_frame["Predictions"] != i
)
].shape[0]
```

```
# Falsos positivos: Casos donde la predicción es i pero el valor actual
no
fp = predictions_frame[
np.logical_and(
predictions_frame["Actual"] != i, predictions_frame["Predictions"] == i
)
].shape[0]
# Falsos negativos: Casos donde el valor actual es i pero la predicción
no
fn = predictions_frame[
np.logical_and(
predictions_frame["Actual"] == i, predictions_frame["Predictions"] != i
)
].shape[0]
# Total de predicciones
total_preds = predictions_frame.shape[0]

# Verificación para evitar la división por cero en la precisión
if (tp + fp) > 0:
precision.append(tp / (tp + fp))
else:
precision.append(
0
) # o np.nan o cualquier otro valor que indique una precisión
indefinida

# Calcular la precisión
accuracy.append((tp + tn) / total_preds)

# Verificación para evitar la división por cero en el recall
if (tp + fn) > 0:
recall.append(tp / (tp + fn))
else:
recall.append(
0
) # o np.nan o cualquier otro valor que indique un recall indefinido

return (accuracy, precision, recall)
```

```
multi_predictions_0 = predictions(
top_base + "/test_multi/" + multi_test_name_order[0] + "/",
multi_classifier,
```

```
binary=False,
)
multi_predictions_1 = predictions(
top_base + "/test_multi/" + multi_test_name_order[1] + "/",
multi_classifier,
binary=False,
)
multi_predictions_2 = predictions(
top_base + "/test_multi/" + multi_test_name_order[2] + "/",
multi_classifier,
binary=False,
)
multi_predictions_3 = predictions(
top_base + "/test_multi/" + multi_test_name_order[3] + "/",
multi_classifier,
binary=False,
)
multi_predictions_4 = predictions(
top_base + "/test_multi/" + multi_test_name_order[4] + "/",
multi_classifier,
binary=False,
)
multi_predictions_5 = predictions(
top_base + "/test_multi/" + multi_test_name_order[5] + "/",
multi_classifier,
binary=False,
)
```

...
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 70ms/step
1/1 [==============================] - 0s 24ms/step

```
multi_predictions_frame = pd.DataFrame(
list(
zip(
multi_predictions_0
+ multi_predictions_1
+ multi_predictions_2
+ multi_predictions_3
+ multi_predictions_4
+ multi_predictions_5,
```

```
[0] * len(multi_predictions_0)
+ [1] * len(multi_predictions_1)
+ [2] * len(multi_predictions_2)
+ [3] * len(multi_predictions_3)
+ [4] * len(multi_predictions_4)
+ [5] * len(multi_predictions_5),
)
),
columns=["Predictions", "Actual"],
)
```

```
# Creación de un dataframe para guardar las predicciones
multi_predictions_frame = pd.DataFrame(
list(
zip(
multi_predictions_0
+ multi_predictions_1
+ multi_predictions_2
+ multi_predictions_3
+ multi_predictions_4
+ multi_predictions_5,
[0] * len(multi_predictions_0)
+ [1] * len(multi_predictions_1)
+ [2] * len(multi_predictions_2)
+ [3] * len(multi_predictions_3)
+ [4] * len(multi_predictions_4)
+ [5] * len(multi_predictions_5),
)
),
columns=["Predictions", "Actual"],
)

multi_accuracy = prec_acc(multi_predictions_frame)
print("Precision:" + str(multi_accuracy[1]))
print("Recall:" + str(multi_accuracy[2]))
print(multi_test_name_order)
```

Salida:
Precision:[0.868421052631579, 1.0, 0.5673758865248227, 0.9692307692307692, 0.9384615384615385, 0.7857142857142857]

Recall:[0.9295774647887324, 0.6666666666666666, 0.975609756097561, 0.63, 0.8026315789473685, 0.9166666666666666]
['Ariel_Sharon', 'Colin_Powell', 'Donald_Rumsfeld', 'George_W_Bush', 'Gerhard_Schroeder', 'Tony_Blair']

Los valores de precisión proporcionados para las seis categorías son bastante altos en general, manteniéndose la gran mayoría en porcentajes arriba del 90%, esto quiere decir que el modelo en los datos que se usaron la mayoría de veces pudo identificarlo correctamente, siendo una excepción 'Tony Blair' con un 72.58%, lo que sugiere que cuando el modelo predice que una imagen es de 'Tony Blair', hay una mayor proporción de veces que se equivoca comparado con las otras categorías. Así como en el recall mantiene porcentajes relativamente elevados, por lo que sugiere que el modelo mantiene un rendimiento sólido, pero que puede ser mejorado si se le dedica mayor tiempo a este modelo.

Podemos concluir que las CNN con los ajustes adecuados pueden ser grandes herramientas para identificar y clasificar varias características de los objetos con el uso de imágenes, esto puede ser muy útil en el campo de la visión computacional, pero es importante denotar que para que un modelo con CNN pueda funcionar de manera adecuada hay que estar conciente de los factores que rodean estas herramientas, ya que la precisión de estas se puede ver muy afectado dependiendo de los parámetros que le demos al modelo, así como reconocer que hay mucho margen de mejora en este modelo en específico con el uso de mejores datos, que sean más diversos, así como aumentar la complejidad de las capas para el entrenamiento del modelo.