Test
“Clasificación de rostros de acuerdo con sus atributos”

Como parte del proceso para la posición de
“Senior Computer Vision Engineer”

PRESENTA

MSC Gerardo Gudiño García

08 de noviembre de 2021

Agenda

01. Planteamiento de la prueba

02. Lógica de las funciones desarrolladas

03. Características del entorno de ejecución

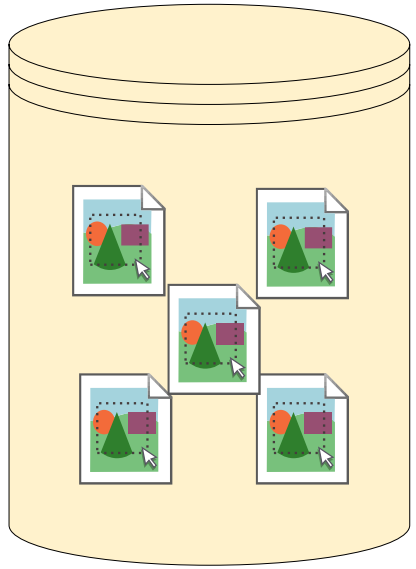
04. Descripción de la prueba ejecutada

05. Resultados

06. Conclusiones

Planteamiento de la prueba

- Utilizar la Base de Datos “CelebA”
- Generar una gráfica donde se expresen los atributos reportados en la base de datos “CelebA”.
- Utilizar una red neuronal convolucional conectada a una ‘Dense’ para obtener los resultados.
- Elegir al menos 4 rasgos distintivos y utilizarlos como clases.
- Observar el porcentaje de acierto de la clasificación en un set de validación propuesto.
- Generar una gráfica sobre el set de validación, utilizando el valor de “accuracy”.



Lógica de las funciones desarrolladas

Función para graficar y seleccionar las clases dependiendo del orden que se defina.

```python
def graph_and_sort_attr(df,order,save_as=None):
    """
        Función para graficar y seleccionar las clases dependiendo del
        orden que se defina
        :param df: DataFrame principal
        :param order: Define el orden para elegir las clases
                0 = mayor número de presencia
                1 = menor número de presencia
                2 = random
        :param save_as: Nombre como se guardará la gráfica si así se desea.
        :return lista con las clases a utilizar
    """
    headers = list(df)[1:]
    dfs = df[headers][df[headers] > 0].sum()
    dfsort = dfs.sort_values()
    dfsort.plot(kind = "bar")
    plt.xlabel("Attributes")
    plt.ylabel("N. of images")
    if save_as:
        plt.savefig(os.path.join("./",save_as))
    plt.show()
    if order == 0:
        return list(dfsort.index[-4:])
    elif order == 1:
        return list(dfsort.index[:4])
    elif order == 2:
        heads = []
        rand_generated = []
        x = 0
        while x < 4:
            r_num = randrange(len(headers))
            if not r_num in rand_generated:
                rand_generated.append(r_num)
                heads.append(headers[r_num])
                x+=1

        return heads
    else:
        return None
```

- Función: graph_and_sort_attr
- Parámetros de entrada:
  - df: DataFrame principal con los datos de las imágenes de la BD “CelebA”
  - order: Define el orden para elegir las clases. Pueden ser los atributos con mayor presencia, menor o de forma random.
  - save_as: Nombre con el cual se guardará la gráfica que se genera.
- Salida:
  - Lista con las clases seleccionadas.

graph_and_sort_attr

- Primero se listan las cabeceras de las columnas que corresponden a los atributos y se excluye la primera, ya que corresponde al nombre de la imagen.
- Con los nombres de las demás columnas, se le aplica un filtro al DataFrame para sumar los valores positivos (1) y así contar el número de presencia que tiene dicho atributo en el DataFrame.

```
headers = list(df)[1:]
dfs = df[headers][df[headers] > 0].sum()
```

graph_and_sort_attr

- Con los valores sumados, se ordenan de menor a mayor. Este nuevo DataFrame se grafica en forma de barras en orden de atributos (attributes) y número de imágenes que tienen ese atributo.
- La gráfica se guarda, si así se desea, o simplemente se muestra.

```
dfsort = dfs.sort_values()
dfsort.plot(kind = "bar")
plt.xlabel("Attributes")
plt.ylabel("N. of images")
if save_as:
    plt.savefig(os.path.join("./",save_as))
plt.show()
```

graph_and_sort_attr

- Del DataFrame ordenado de menor a mayor, se seleccionan los 4 atributos de menor presencia, mayor presencia o de forma random, de acuerdo a lo definido en el parámetro de entrada de ‘order’.

- Y esos atributos se regresan en forma de lista.

```
if order == 0:
    return list(dfsort.index[-4:])
elif order == 1:
    return list(dfsort.index[:4])
elif order == 2:
    heads = []
    rand_generated = []
    x = 0
    while x < 4:
        r_num = randrange(len(headers))
        if not r_num in rand_generated:
            rand_generated.append(r_num)
            heads.append(headers[r_num])
            x+=1

    return heads
else:
    return None
```

Función que selecciona y agrupa el nombre de la imagen con base en los atributos definidos

```python
def group_data(df_attr,classes_selected,low_performance=False):
    """
        Función que selecciona y agrupa el nombre de la imagen con base en los
        atributos definidos
        :param df_attr: DataFrame de los atributos
        :param classes_selected: nombre de las classes o atributos definidos
        :low_performance: bandera para definir si el dataset debe de ser recortado
                          a un límite de 8000 imágenes por clase.
                          Dependiendo de los recursos que se tengan.
        :return DataFrame: DataFrame con las classes y sus respectivos nombres de las imágenes
                          que pertenecen a las mismas
    """
    df_grouped = df_attr.groupby(classes_selected)["image_id"].unique()
    img_list = []
    for idx, item in df_grouped.items():
        i = 0
        for ix in idx:
            if ix == 1:
                img_list.append([classes_selected[i],item])
                break
            i+=1

    if low_performance:
        print("Low Performance activado!")
        low_img_list=[]
        for c in classes_selected:
            obj = list()
            for il in img_list:
                if c == il[0]:
                    obj += list(il[1])
            low_img_list.append([c,obj])
        img_list = low_img_list[:]
        for i in range(len(img_list)):
            if len(img_list[i][1]) > 8000:
                print("Clase '{}' pasa de {} a 8000 imágenes".format(img_list[i][0],len(img_list[i][1])))
                img_list[i][1] = img_list[i][1][:8000]

    return pd.DataFrame(img_list,columns=["label","img_name"])
```

- Función: group_data
- Parámetros de entrada:
  - df_attr: DataFrame principal de las imágenes con los atributos.
  - classes_selected: Lista con las 4 clases seleccionadas.
  - low_performance: Bandera para construir un dataset modesto para un ambiente limitado de procesamiento o espacio en memoria.
- Salida:
  - Un nuevo dataframe en donde están la etiquetas de la clase y el nombre de las imágenes que pertenecen a dicha clase.

group_data

- Al DataFrame principal se agrupan los valores de acuerdo a su atributo (clase) y, al valor de la columna ‘image_id’, se le aplica el método unique para que no se repitan los nombres de las clases en caso de que pertenezcan a más de una.

```
df_grouped = df_attr.groupby(classes_selected)["image_id"].unique()
```

group_data

- Se recorre el nuevo DataFrame filtrado y se revisa a qué clase pertenece cada una de las imágenes, para así colocar el conjunto de nombre de imágenes con su respectiva etiqueta-clase.

```
img_list = []
for idx, item in df_grouped.items():
    i = 0
    for ix in idx:
        if ix == 1:
            img_list.append([classes_selected[i],item])
            break
        i+=1
```

group_data

- Si la bandera de low_performance está activada, se recortan las listas con los nombres de cada una de las clases hasta un máximo de 8000.

```
if low_performance:
    print("Low Performance activado!")
    low_img_list=[]
    for c in classes_selected:
        obj = list()
        for il in img_list:
            if c == il[0]:
                obj += list(il[1])
        low_img_list.append([c,obj])
    img_list = low_img_list[:]
    for i in range(len(img_list)):
        if len(img_list[i][1]) > 8000:
            print("Clase '{}' pasa de {} a 8000 imágenes".format(img_list[i][0],len(img_list[i][1])))
            img_list[i][1] = img_list[i][1][:8000]
```

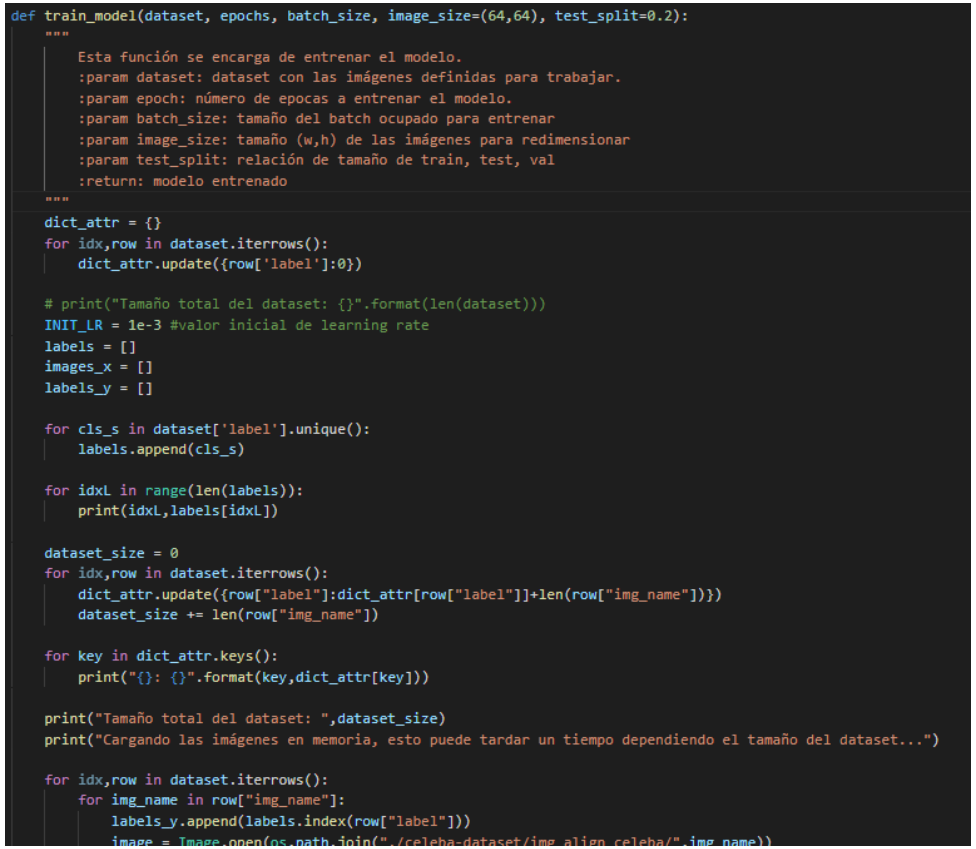
group_data

- A la lista final de etiqueta-clase con los nombres de las imágenes que pertenecen a ella, se convierte en un nuevo DataFrame con las columnas “label” & “img_name”.

```
return pd.DataFrame(img_list,columns=["label","img_name"])
```

Función para preparar y entrenar el modelo.

- Función: train_model
- Parámetros de entrada:
  - dataset: DataFrame que se generó previamente en donde están la etiqueta de la clase y la lista de los nombres de las imágenes que pertenecen a la misma.
  - epochs: Número de épocas que se entrenará el modelo.
  - batch_size: Tamaño del batch que se someterá al entrenamiento.
  - image_size: Tamaño en w,h con que se redimensionarán las imágenes.
  - test_Split: relación del tamaño que se definirán para los datasets de train, val and test.
- Salida:
  - Modelo entrenado



```
def train_model(dataset, epochs, batch_size, image_size=(64,64), test_split=0.2):
    """
        Esta función se encarga de entrenar el modelo.
        :param dataset: dataset con las imágenes definidas para trabajar.
        :param epoch: número de epocas a entrenar el modelo.
        :param batch_size: tamaño del batch ocupado para entrenar
        :param image_size: tamaño (w,h) de las imágenes para redimensionar
        :param test_split: relación de tamaño de train, test, val
        :return: modelo entrenado
    """
    dict_attr = {}
    for idx,row in dataset.iterrows():
        dict_attr.update({row['label']:0})

    # print("Tamaño total del dataset: {}".format(len(dataset)))
    INIT_LR = 1e-3 #valor inicial de learning rate
    labels = []
    images_x = []
    labels_y = []

    for cls_s in dataset['label'].unique():
        labels.append(cls_s)

    for idxL in range(len(labels)):
        print(idxL,labels[idxL])

    dataset_size = 0
    for idx,row in dataset.iterrows():
        dict_attr.update({row["label"]:dict_attr[row["label"]]+len(row["img_name"])})
        dataset_size += len(row["img_name"])

    for key in dict_attr.keys():
        print("{}: {}".format(key,dict_attr[key]))

    print("Tamaño total del dataset: ",dataset_size)
    print("Cargando las imágenes en memoria, esto puede tardar un tiempo dependiendo el tamaño del dataset...")

    for idx,row in dataset.iterrows():
        for img_name in row["img_name"]:
            labels_y.append(labels.index(row["label"]))
            image = Image.open(os.path.join("./celeba-dataset/img_align_celeba/",img_name))
```

train_model

- Se definen las variables de las listas para las etiquetas (labels), valores en ‘x’ (matrices de las imágenes) y valores en ‘y’ (índice de las etiquetas), así como el valor inicial del ‘learning rate’, para este valor se utiliza uno por defecto.

```
INIT_LR = 1e-3 #valor inicial de learning rate
labels = []
images_x = []
labels_y = []
```

train_model

- Se recorre el dataset y se van cargando los valores de los índices para las imágenes (labels_y) y la matriz de cada imagen (images_x) en memoria.

- Al final estas listas se pasan a un tipo de numpy array.

```
print("Cargando las imágenes en memoria, esto puede tardar un tiempo dependiendo el tamaño del dataset...")

for idx,row in dataset.iterrows():
    for img_name in row["img_name"]:
        labels_y.append(labels.index(row["label"]))
        image = Image.open(os.path.join("./celeba-dataset/img_align_celeba/",img_name))
        new_image = image.resize(image_size)
        images_x.append(np.array(new_image))

labels_y = np.array(labels_y)
images_x = np.array(images_x, dtype=np.uint8)
```

train_model

- Se divide el dataset en train, test y val.
- De la misma forma se regularizan los pixeles de las imágenes para comprender un valor de 0 – 1 y así asegurar un mayor éxito al momento de entrenar.
- Las etiquetas, de la misma forma, se regularizan a un formato ‘One-Hot encoding’, para un mejor funcionamiento de la red neuronal.

```
#dividir el dataset en train, val and test. Relación 80%-20%
train_x,test_x,train_y,test_y = train_test_split(images_x,labels_y,test_size=test_split)
train_x = train_x.astype('float32')
test_x = test_x.astype('float32')
train_x = train_x / 255.
test_x = test_x / 255.

train_y_one_hot = keras.utils.to_categorical(train_y)
test_y_one_hot = keras.utils.to_categorical(test_y)

train_x,val_x,train_label,val_label = train_test_split(train_x, train_y_one_hot, test_size=test_split, random_state=13)
```

train_model

- Se definen las capas para el entrenamiento.
- Primero se genera una capa convolucional con valores básicos para realizar una prueba rápida pero efectiva.
  - Se aplican 32 filtros de 3x3 y se define el tamaño de las imágenes de entrada, en este caso 64x64 y 3 canales al ser RGB.
  - Se agregan las respectivas capas de ReLu, MaxPooling y Dropout para optimizar de mejor manera la red neuronal y evitar el **overfitting**.

```
# Definiendo las capas para el entrenamiento
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(32, kernel_size=(3, 3),activation='linear',padding='same',input_shape=(image_size[0],image_size[1],3)))
model.add(keras.layers.LeakyReLU(alpha=0.1))
model.add(keras.layers.MaxPooling2D((2, 2),padding='same'))
model.add(keras.layers.Dropout(0.5))
```

train_model

- Ahora se crea una capa densa ‘dense’, la cual es una capa tradicional, de igual forma con valores estándar.
  - Será con 32 filtros y una activación lineal.
  - Igual con la función Dropout para evitar el **overfitting**.
- Por último la capa de salida la cual tendrá el mismo número de filtros que el número de nuestras clases y con activación softmax.

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(32, activation='linear'))
model.add(keras.layers.LeakyReLU(alpha=0.1))
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(len(labels), activation='softmax'))

model.summary()
```

train_model

- Se compila el modelo con valores estándar y se pone a entrenar con los datos previos que se obtuvieron o se fueron creando como es el caso para los dataset de train y val.

```
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=keras.optimizers.Adagrad(lr=INIT_LR, decay=INIT_LR / 100),metrics=['accuracy'])
train_model_out = model.fit(train_x, train_label, batch_size=batch_size,epochs=epochs,verbose=1,validation_data=(val_x, val_label))
```

- El mismo modelo se evalua con el dataset de test y se obtienen los valores de pérdida (loss) y de precisión (accuracy).
- Al final se regresa el modelo entrenado.

```
test_eval = model.evaluate(test_x, test_y_one_hot, verbose=1)

print('Test loss:', test_eval[0])
print('Test accuracy:', test_eval[1])

return train_model_out
```

Método para graficar los resultados del modelo entrenado.

```python
def graph_model(model,value='accuracy',save_as=None):
    """
        Método para graficar los resultados del modelo entrenado
        ['accuracy','loss','val_loss','val_accuracy']
        :param model: modelo entrenado a graficar
        :param value: valor a graficar ['accuracy','loss']
        :param save_as: Nombre como se guardará la gráfica si así se desea.
    """
    plt.plot(model.history[value])
    plt.plot(model.history['val_{}'.format(value)])
    plt.title('Model {}'.format(value))
    plt.ylabel('{}'.format(value))
    plt.xlabel('Epoch')
    plt.legend(['train', 'val'], loc='upper left')
    if save_as:
        plt.savefig(os.path.join("./",save_as))
    plt.show()
```

- Método: graph_model
- Parámetros de entrada:
  - model: Modelo entrenado
  - value: Valor a graficar [‘accuracy’, ‘loss’]
  - save_as: Nombre como se guardará la gráfica si así se desea.

graph_model

- Se obtienen los valores del modelo en ‘history’, para crear la gráfica.
- Si se desea se guarda la gráfica o, sino, simplemente se muestra.

```
plt.plot(model.history[value])
plt.plot(model.history['val_{}'.format(value)])
plt.title('Model {}'.format(value))
plt.ylabel('{}'.format(value))
plt.xlabel('Epoch')
plt.legend(['train', 'val'], loc='upper left')
if save_as:
    plt.savefig(os.path.join("./",save_as))
plt.show()
```

Características del entorno de ejecución

El entorno en el cual se desarrollo y se generaron las pruebas, cuenta con lo siguiente:

- SO Ubuntu 20.04
- Procesador Intel i7 8va a 2.20GHz
- RAM 16GB
- Gráfica Nvidia Quadro P600 4GB
- CUDA 11.4
- Python 3.7.7
- Tensorflow 2.1.0

Descripción de la prueba ejecutada

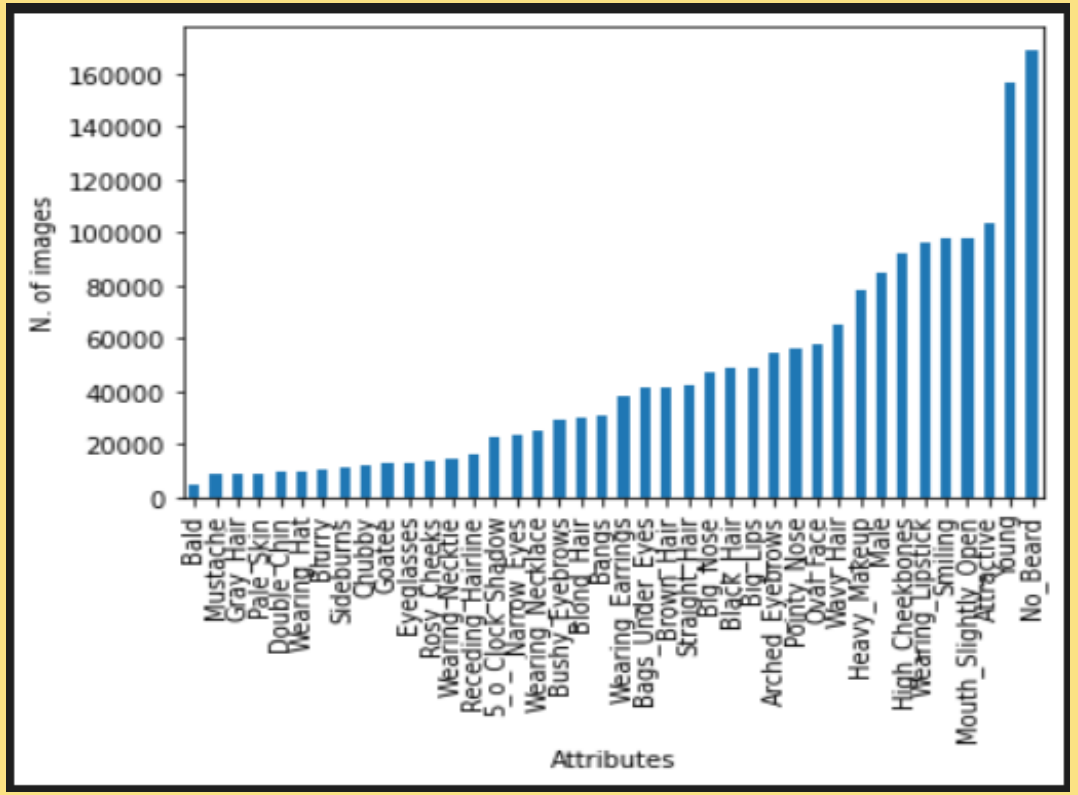
- La prueba se realizó de acuerdo a lo descrito en el planteamiento de la misma, con un total de 202599 imágenes.
- Se escogió utilizar las 4 clases con la menor presencia, para realizarla de una manera rápida y por las limitaciones del entorno de ejecución.
- Se realizó con un total de 250 épocas (epochs) y redimensionando las imágenes a 64x64.
- Igual, por haberse escogido las clases con menor presencia y que el dataset resultaba no tan pesado, se deshabilitó la bandera de '*low_performance*'. Por lo cual no se recortó en nada al dataset para realizar el entrenamiento.

```
#carga el dataframe de los atributos
path_attr = "celeba-dataset/list_attr_celeba.csv"
df_attr = pd.read_csv(path_attr)
# Selecciona las clases a entrenar
classes_selected = graph_and_sort_attr(df_attr,1,save_as="attr_graph.jpg")
# selecciona y agrupa las imágenes dependiendo de las clases a entrenar en un nuevo DataFrame
dataset = group_data(df_attr,classes_selected)
# proceso para entrenar el modelo
trained_model = train_model(dataset,250,64)
# método para graficar los resultados del modelo
graph_model(trained_model,save_as="val_graph.jpg")
```

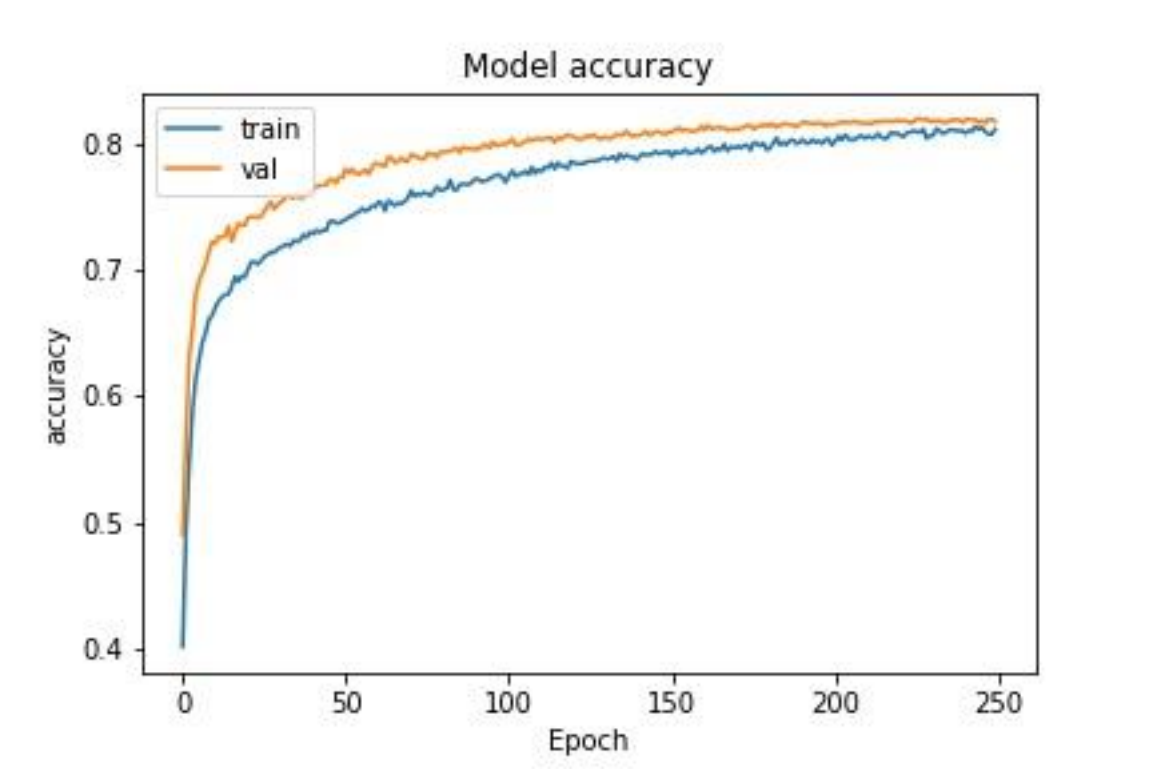
Resultados

Después de realizar el entrenamiento se obtuvieron los siguientes resultados:

- Las 4 clases con menor presencia fueron:
  - [‘Bald’, ‘Mustache’, ‘Gray_Hair’, ‘Pale_Skin’]
  - Resultando en un dataset de 27472 imágenes
  - Dividido en las clases:
    - Bald, 4547
    - Mustache, 7751
    - Gray_Hair, 6871
    - Pale_Skin, 8303
  - Recordar que las imágenes se eligen únicamente y se asignan a la primer clase que coincidan.



Resultados



El dataset se entrenó por 250 épocas (epochs) con una redimensión a 64x64, en un tiempo máximo de 30 minutos.

Los resultados después de entrenar fueron:

- loss: 0.4935 ~ 49%
- accuracy: 0.8110 ~ 81%
- val_los: 0.4636 ~ 46%
- val_accuracy: 0.8171 ~ 81%
- test_loss: 0.4633 ~ 46%
- test_accuracy: 0.8242 ~ 82%

Conclusiones

Como se puede observar en los resultados, el modelo, después de las 200 épocas, tiene una precisión de +80%, lo cual lo hace, en este punto, un modelo estable y un tanto confiable para el tipo de modelo que es, al tener únicamente una capa convolucional y una capa oculta ‘dense’, me parece un buen porcentaje de precisión.

A partir de la época 200 el valor de precisión tiende a crecer de forma muy lenta y se empieza a posicionar de forma horizontal, indicando que está a punto de llegar a su límite de aprendizaje o de llegar a un estado de overfitting.

Al llegar a este punto, conviene analizar si las imágenes de entrada pueden seleccionarse o aplicarse un filtro o preprocesamiento para mejorar sus características, o en dado caso, aumentar el número de capas y, del mismo modo, aumentar el número de épocas a las que se someterá el entrenamiento. Aunque, en mi experiencia, este tipo de modelos de clasificación, no son tan confiables ya que dependen muchísimo de que las imágenes que se necesiten analizar sean muy parecidas a las imágenes que se utilizaron para entrenar, ya que al mover un poco el objeto o atributo característico con el que se definió a que clase pertenece, da unos resultados muy alejados a lo que realmente es o, de plano, erróneos.