Practica 3: Multiclasificación de Deportes

10 de julio de 2023

Dataset: Clasificación de imágenes de 100 deportes. Disponible en: https://www.kaggle.com/datasets/gpiosenka/sports-classification

1. Objetivo de la práctica.

Crear un modelo que pueda hacer multiclasificación de imágenes de 100 deportes diferentes y analizar la ventaja del uso de modelos preentrenados.

2. Conceptos.

Ya se habló de las librerías en la Practica 2; TensorFlow, NumPy, y Matplotlib. Así que toca hablar de las demás librerías que se usarán.

Efficientnet es una red neuronal convolucional que está entrenada con más de un millón de imágenes de la base de datos de ImageNet. Como resultado, la red ha aprendido representaciones ricas en características para una amplia gama de imágenes.

Scikit-Learn es una librería enfocada en el aprendizaje automático de la cuál solo usaremos unas métricas para el análisis de los resultados del entrenamiento.

Seaborn es una biblioteca de visualización de Python basada en matplotlib. Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos.

El dataset consiste en 3 carpetas: train(13572), test(500) y valid(500), que a su vez cuentan con 100 subcarpetas con ejemplos de cada deporte, que contienen un total de 14,572 elementos

3. Herramientas a usar.

- Computadora con acceso a internet.
- Los siguientes programas y librerías:
  - Python versión 3.8.8

- TensorFlow versión 2.6.0
- Anaconda versión 4.10.1
- Jupyter-lab 3.014
- NumPy versión 1.21.4
- Pandas versión 1.2.4
- Matplotlib versión 3.3.4
- Zipfile versión 3.2.0
- Efficientnet 1.0.0
- Scikit-learn 1.3.0
- Seaborn 0.11.2

4. Desarrollo.

4.1. Entender el problema.

En lugar de limitarse a la distinción entre dos clases, esta práctica busca desarrollar un modelo capaz de abordar un desafío aún mayor: la clasificación precisa de hasta 100 clases diferentes de deportes. Este enfoque representa un paso importante ya que requerirá una mayor complejidad y sofisticación en la modelización de datos. Propondremos un modelo de multiclasificación basado en nuestros conocimientos y observaremos las limitantes del mismo.

4.2. Criterio de evaluación.

Usaremos la exactitud (accuracy), precisión (precision), sensibilidad (recall), f1-score y matriz de confusión.

4.3. Preparar los datos.

Debemos importar todas las librerías con las que estaremos trabajando durante la práctica.

```
import os
import zipfile
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Estaremos trabajando con Tensorflow, por lo que es necesario comprobar que se esté usando nuestra GPU.

```
tf.config.list_physical_devices('GPU')


[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Aquí definimos la ruta base de nuestros datos. Además, extraeremos las carpetas de los archivos y comprobaremos que se encuentren ahí.

```
Path = "../Practica 3/" #Depende de la ruta en tu maquina

os.makedirs(Path+"deportes", exist_ok=True)
with zipfile.ZipFile('deportes.zip', 'r') as archive:
    # Extrae todo el contenido del achivo ZIP en el directorio
     actual
    archive.extractall("deportes")

os.listdir("deportes")

['EfficientNetB3-sports-0.97.h5',
 'sports.csv',
 'test',
 'train',
 'valid']
```

Gracias a los ejercicios anteriores y que los archivos ya se encuentran separados, podemos comenzar a normalizar las imágenes y generar los conjuntos de entrenamiento, validación y prueba.

```
#Normalizan los valores entre 0 y 1
train = ImageDataGenerator(rescale=1./255)
valid = ImageDataGenerator(rescale=1./255)
test = ImageDataGenerator(rescale=1./255)
```

Aquí comienzan las diferencia; anteriormente el modo de clase (class_mode) con el que definíamos los conjuntos era binario (binary) puesto que eran solamente 2 clases a clasificar. En este caso, para las 100 clases que vamos a manejar, debemos definirlo de manera categórica (categorical).

```
train_dir = "deportes/train"
valid_dir = "deportes/valid"
test_dir = "deportes/test"

train_dataset = train.flow_from_directory(train_dir,
                                  target_size = (224, 224),
                                  color_mode = 'rgb',
                                  batch_size = 50,
                                  class_mode = 'categorical',
                                  shuffle=False)

valid_dataset = train.flow_from_directory(valid_dir,
                                  target_size = (224, 224),
                                  color_mode = 'rgb',
                                  batch_size = 50,
                                  class_mode = 'categorical',
                                  shuffle=False)

test_dataset = test.flow_from_directory(test_dir,
                                target_size = (224, 224),
```

```
                                    color_mode = 'rgb',
                                    batch_size = 50,
                                    class_mode= 'categorical',
                                    shuffle=False)

Found 13572 images belonging to 100 classes.
Found 500 images belonging to 100 classes.
Found 500 images belonging to 100 classes.
```

```
test_dataset.class_indices

{'air hockey': 0,
 'ampute football': 1,
 'archery': 2,
 .
 .
 'wheelchair basketball': 97,
 'wheelchair racing': 98,
 'wingsuit flying': 99}
```

La página del ejercicio de Kaggle especifica que todas las imágenes ya se ecnuentran con la medida de 224p x 224p x 3 canales de color. Podemos sacar una imagen de uno de las carpetas de las categorías para poder comprobarlo

```
sample = plt.imread(Path+"deportes/train/hydroplane racing/039.jpg"
    )
samp = sample.shape
plt.imshow(sample)
print('Sample_Image_size(H*W):', samp)
```

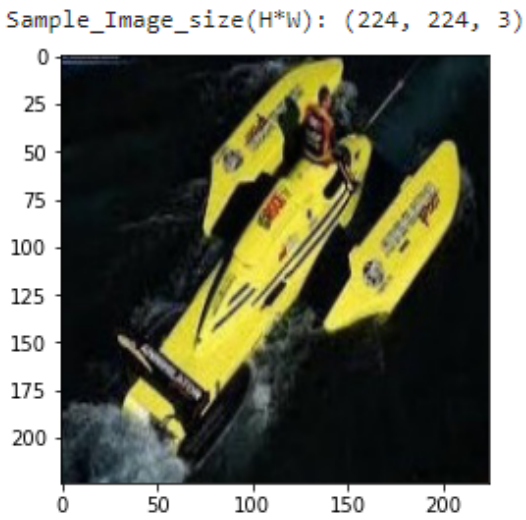

Figura 1: Imagen ejemplo del conjunto de prueba. Una imagen con medidas de 224p x 224p

4.4. Construir el Modelo.

Ahora como parte del preprocesamiento de imágenes se les aplicará una técnica llamada Padding; consiste en agregar un "borde" a la matriz de las imágenes antes de la convolución para resaltar las características de los bordes y ayudar a la red a considerarlas para el entrenamiento.

Esta técnica ayudará bastante ya que la cantidad de clases con las que estamos trabajando es considerablemente mayor a la de las prácticas anteriores y debemos ayudar al modelo a que las categorice de mejor manera.

Al igual que en la práctica anterior, usaremos una estructura de red neuronal convolucional sencilla y alteraremos la última capa densa para que haga la clasificación de 100 clases diferentes.

Además de eso, la función de activación se vera cambiada a la función Softmax lo que convierte la salida de la última capa de su red neuronal en lo que es esencialmente una distribución de probabilidad. Esto quiere decir que en vez de ser un resultado de 0 o 1, será de 0 a 1 reflejando lo "probable"que sea que una imagen pertenezca a una categoría.

```
model1 = keras.models.Sequential([
    keras.layers.Conv2D(32, (3,3), padding='same', activation='relu
    ', input_shape = (224, 224, 3)),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(64, (3,3), padding='same', activation='relu
    '),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(128, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(256, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(512, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Flatten(),

    keras.layers.Dense(512, activation='relu'),

    keras.layers.Dense(100, activation = 'softmax')
])

model1.summary()
```

Ya con la estructura de la red creada solo nos queda mandar a llamar el entrenamiento. Este lo realizaremos haciendo uso de la función de model.fit donde se le dirá a la red con qué imágenes se le alimentará, cuántas veces tendrá que hacer el entrenamiento y qué imágenes usará para corroborar si su clasificación es correcta.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 256)       295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 256)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 512)       1180160
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dense_1 (Dense)              (None, 100)               51300
=================================================================
Total params: 14,465,444
Trainable params: 14,465,444
Non-trainable params: 0
_________________________________________________________________
```

Figura 2: Esta es la estructura de la red neuronal con la que se van a entrenar con las imágenes y cuya capa de salida está alterada para clasificar categóricamente las imágenes.

```
# setting hyperparameters
model1.compile(optimizer='adam', # set the optimizer
               loss='categorical_crossentropy', # loss function
               metrics=['accuracy']) #accuracy metric
```

```
history = model1.fit(
              train_dataset,
              epochs=10,
              validation_data=valid_dataset,
              verbose=2
              )
```

```
Epoch 1/10
272/272 - 81s - loss: 4.6584 - accuracy: 0.0024 - val_loss: 4.6061 - val_accuracy: 0.0100
Epoch 2/10
272/272 - 66s - loss: 4.6072 - accuracy: 0.0041 - val_loss: 4.6062 - val_accuracy: 0.0100
Epoch 3/10
272/272 - 67s - loss: 4.6066 - accuracy: 0.0056 - val_loss: 4.6062 - val_accuracy: 0.0100
Epoch 4/10
272/272 - 67s - loss: 4.6061 - accuracy: 0.0103 - val_loss: 4.6063 - val_accuracy: 0.0100
Epoch 5/10
272/272 - 66s - loss: 4.6056 - accuracy: 0.0080 - val_loss: 4.6064 - val_accuracy: 0.0100
Epoch 6/10
272/272 - 67s - loss: 4.6049 - accuracy: 0.0141 - val_loss: 4.6065 - val_accuracy: 0.0100
Epoch 7/10
272/272 - 67s - loss: 4.6045 - accuracy: 0.0141 - val_loss: 4.6066 - val_accuracy: 0.0100
Epoch 8/10
272/272 - 66s - loss: 4.6041 - accuracy: 0.0141 - val_loss: 4.6067 - val_accuracy: 0.0100
Epoch 9/10
272/272 - 66s - loss: 4.6037 - accuracy: 0.0141 - val_loss: 4.6069 - val_accuracy: 0.0100
Epoch 10/10
272/272 - 67s - loss: 4.6033 - accuracy: 0.0141 - val_loss: 4.6070 - val_accuracy: 0.0100
```

Figura 3: Aquí se ve el progreso del entrenamiento durante las últimas 10 épocas. Podemos ver cómo en la parte de validación la exactitud comienza a sobreajustarse

Viendo los resultados del entrenamiento, podemos ver que tenemos problemas con el entrenamiento de la red neuronal; la pérdida en ambos conjuntos está estancada, lo que significa que el modelo no está aprendiendo. La exactitud sufre de de lo mismo, indicando que la clasificación se está realizando de una manera incorrecta. Y como tercer factor, el tiempo de entrenamiento es bastante por época, lo que puede significar que no haya logrado captar ninguna característica significativa de los datos.

Harmos una graficación del entrenamiento para poder sacar las últimas conclusiones de estos probelmas.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 8))
plt.subplot(2, 1, 1)
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
```

```
#plt.ylim([min(plt.ylim()),1])
plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.ylabel('Cross Entropy')
#plt.ylim([0,1.0])
plt.title('Training and Validation Loss')
plt.xlabel('epoch')
plt.show()
```

Figura 4: Las gráficas que muestran la exactitud y pérdida del entrenamiento del modelo en el conjunto de entrenamiento y validación.

Ya con la gráfica se confirman dos posibles casos; que la cantidad de imágenes que estamos usando para el entrenamiento sean pocas o la estructura de nuestro modelo es demasiado sencilla.

4.4.1. Aumento de Datos.

Vamos a resolver el primer caso y aplicaremos la técnica de aumento de datos (data augmentation) que consiste en aplicar transformaciones a nuestro conjunto de entrenamiento para así aumentar la cantidad muestras que usará la red. Estas transformaciones pueden ser rotación, desplazamiento a lo largo y ancho, deformación, acercamiento, un reflejo horizontal o vertical de la imagen. Todas estas transformaciones se hacen en un rango aleatorio o con un límite definido por el usuario.

```
train_dataset_augmented = ImageDataGenerator(
                     rescale = 1./255.,
                     rotation_range = 40,
                     width_shift_range = 0.2,
                     height_shift_range = 0.2,
                     shear_range = 0.2,
                     zoom_range = 0.2,
                     horizontal_flip = True,
                     vertical_flip =True
                     )
```

```
train_dataset_augmented = train.flow_from_directory(
                              train_dir,
                              target_size = (224, 224),
                              color_mode = 'rgb',
                              batch_size = 50,
                              class_mode = 'categorical'
                              )
```

Este nuevo conjunto de entrenamiento lo cargaremos a un "nuevo"modelo de red neuronal con la misma estructura que hemos usado. Este nuevo modelo usará las mismas imágenes de validación, puesto que las transformaciones del conjunto de entrenamiento son solo para ayudar a la red a entrenar mejor.

```
model2 = keras.models.Sequential([
    keras.layers.Conv2D(32, (3,3), padding='same', activation='relu
    ', input_shape = (224, 224, 3)),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(64, (3,3), padding='same', activation='relu
    '),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(128, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(256, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Conv2D(512, (3,3), padding='same', activation='
    relu'),
    keras.layers.MaxPooling2D(),

    keras.layers.Flatten(),
```

```
    keras.layers.Dense(512, activation='relu'),

    keras.layers.Dense(100, activation = 'softmax')
])
```

```
# setting hyperparameters
model1.compile(optimizer='adam', # set the optimizer
               loss='categorical_crossentropy', # loss function
               metrics=['accuracy']) #accuracy metric
```

Nótese que el cambio principal es aquí, usando el nuevo conjunto de datos de entrenamiento aumentado.

```
history2 = model2.fit(
            train_dataset_augmented,
            epochs=10,
            validation_data=valid_dataset,
            verbose=2
            )
```

```
Epoch 1/10
272/272 - 163s - loss: 4.5805 - accuracy: 0.0133 - val_loss: 4.5337 - val_accuracy: 0.0240
Epoch 2/10
272/272 - 167s - loss: 4.2739 - accuracy: 0.0432 - val_loss: 3.9957 - val_accuracy: 0.0820
Epoch 3/10
272/272 - 169s - loss: 3.8144 - accuracy: 0.0960 - val_loss: 3.4438 - val_accuracy: 0.1260
Epoch 4/10
272/272 - 168s - loss: 3.4759 - accuracy: 0.1485 - val_loss: 3.1622 - val_accuracy: 0.1980
Epoch 5/10
272/272 - 178s - loss: 3.2020 - accuracy: 0.1982 - val_loss: 3.1162 - val_accuracy: 0.2260
Epoch 6/10
272/272 - 182s - loss: 3.0064 - accuracy: 0.2398 - val_loss: 2.8211 - val_accuracy: 0.2600
Epoch 7/10
272/272 - 173s - loss: 2.8535 - accuracy: 0.2720 - val_loss: 2.4651 - val_accuracy: 0.3480
Epoch 8/10
272/272 - 169s - loss: 2.7168 - accuracy: 0.2986 - val_loss: 2.6835 - val_accuracy: 0.3060
Epoch 9/10
272/272 - 186s - loss: 2.5973 - accuracy: 0.3216 - val_loss: 2.3811 - val_accuracy: 0.3540
Epoch 10/10
272/272 - 168s - loss: 2.5000 - accuracy: 0.3411 - val_loss: 2.2661 - val_accuracy: 0.3960
```

Figura 5: Aquí se ve el progreso del entrenamiento durante las últimas 10 épocas. Gracias a la modificación, vemos una mejoría en el entrenamiento.

Gracias a los resultados, podemos ver que el aumento de datos ayudó significativamente al entrenamiento de la red, pasando de menos de un 2 % de exactitud de clasificación correcta a un 34 % en el conjunto de entrenamiento y un 39 % en el de validación.

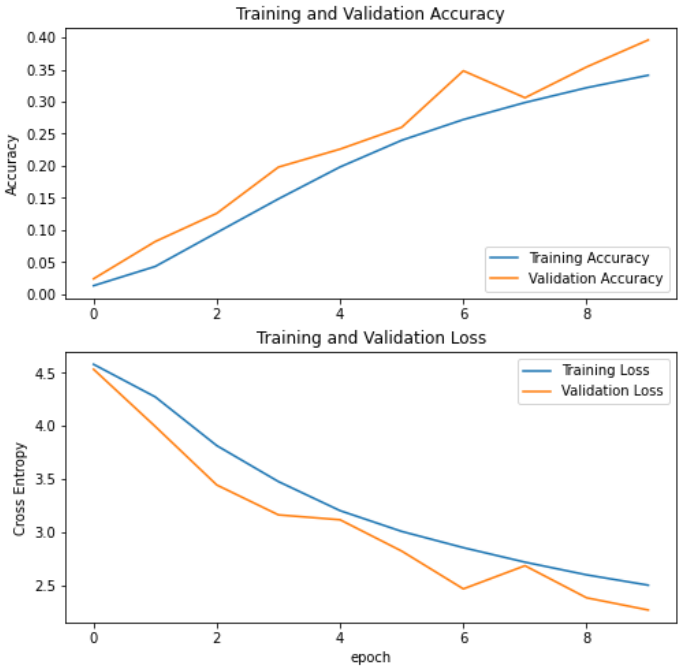

Figura 6: Las gráficas que muestran la exactitud y pérdida del entrenamiento del modelo en el conjunto de entrenamiento de datos aumentados y validación.

La mejoría de entrenamiento de los datos es buena, pero ahora tocaría modificar el modelo usado para poder conseguir un resultado más satisfactorio, o también tenemos una opción más segura y directa; usar una red preentrenada.

4.4.2. Redes Preentrenadas.

Las Redes Neuronales Preentrenadas son, como dice su nombre, modelos que han sido entrenados con conjuntos de datos grandes y generalizados. Un ejemplo de estas Redes Preentrenadas es EfficientNet para clasificación de imágenes.

Estos modelos preentrenados ayudan a reducir el tiempo invertido en encontrar un modelo para estos casos pues nos "ahorra" el hecho de estar haciendo una red neuronal desde cero; tanto en la complejidad del modelo como el poder computacional necesario para poder hacer que de un buen resultado. Esto gracias a que las Redes Preentrenadas pueden ser modificadas en las últimas capas para ajustarse al problema que necesitemos resolver.

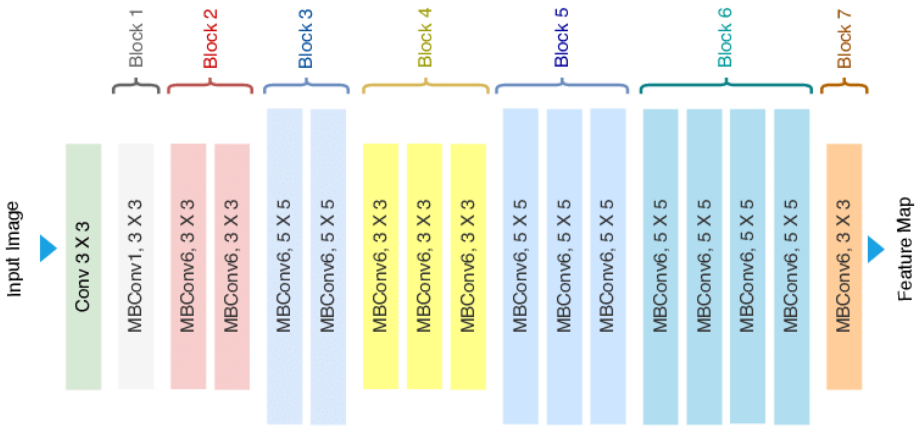

Figura 7: Esta es la arquitectura de EfficentNetB0

Para este problema, usaremos EfficentNetB0, la versión más pequeña de los modelos de EfficientNet ya que en términos de eficiencia y rendimiento de recursos es la que más se adapta y nos ayuda con nuestras limitantes. Si cuenta con un equipo más poderoso, siéntase en la libertad de trabajar con un modelo más grande.

En resumen, la arquitectura de EfficientNetB0 es la siguiente:

1. Entrada: Imagen de 224 x 224 pixeles.
2. Convolución inicial: Convolución de 3x3 con 32 filtros.
3. Bloques de MBConv(Mobile Inverted Bottleneck Convolution): Son unos bloques de convolución característicos de EfficienNet. Estos bloques son los que ayudan a reducir el costo computacional de la red. Estos bloques lo que hacen es:
   - Aplicar convoluciones a cada filtro de la imagen de manera separada.
   - Aplicar una convolución 1x1 para combinar las características extraídas.
   - Aplicar un aumento de filtros, y aplicar los 2 pasos anteriores una vez más
4. Global Average Pooling: Reduce la dimensionalidad de la red para mandarla a la capa final.
5. Capa completamente conectada: Es la capa de salida con una función de clasificación softmax.

Ahora, cargaremos EfficienNetB0 a nuestra libreta para poder trabajar con nuestros datos.

```
import efficientnet.tfkeras as efn

base_model =  efn.EfficientNetB0(input_shape = (224, 224, 3), # Shape of our images
                                 include_top = False, # Leave out the last fully connected layer
                                 weights = 'imagenet')

for layer in base_model.layers:
    layer.trainable = False
```

Importamos EfficentNet desde la librería Keras, lo asignamos a base_model y excluimos la última capa conectada: esto para poder configurar nosotros la capa de salida para nuestro caso, que es clasificar 100 deportes diferentes.

Algo muy importante es la parte de "layer.trainable = False" ya que esto "congela " las capas del modelo, lo que hace que a la hora de usarlo para clasificar, no actualizará los pesos que tiene productos del preentrenamiento. Solo se modificarán los pesos de las capas agregadas al final.

Es hora de armar el modelo y agregarle esas capas finales y comenzar el entrenamiento.

```
model3 = keras.models.Sequential([
    base_model,
    keras.layers.GlobalAveragePooling2D(),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.1),
    keras.layers.Dense(256, activation='relu'),
    keras.layers.Dropout(0.1),
    keras.layers.Dense(100, activation = 'softmax')
])
```

Agregamos una capa GlobalAveragePooling2d después del modelo preentrenado para tener un vector con las características extraídas. Después se usan las capas Densas y de Dropout para capturar capturar las relaciones no lineales de las características y evitan el sobreajutse. Y finalmente la capa Densa de salida para calificar la imagen en una de las 100 categorías.

Ahora compilamos el modelo y lo ponemos a entrenar.

```
# setting hyperparameters
model3.compile(optimizer='adam', # set the optimizer
               loss='categorical_crossentropy', # loss function
               metrics=['accuracy']) #accuracy metric

history3 = model3.fit(
            train_dataset_augmented,
            epochs=10,
            validation_data=valid_dataset,
            verbose=2
            )
```

```
Epoch 1/10
272/272 - 168s - loss: 1.3204 - accuracy: 0.6298 - val_loss: 0.6068 - val_accuracy: 0.8220
Epoch 2/10
272/272 - 165s - loss: 1.1718 - accuracy: 0.6642 - val_loss: 0.5244 - val_accuracy: 0.8340
Epoch 3/10
272/272 - 168s - loss: 1.0824 - accuracy: 0.6890 - val_loss: 0.5610 - val_accuracy: 0.8420
Epoch 4/10
272/272 - 169s - loss: 0.9994 - accuracy: 0.7108 - val_loss: 0.5115 - val_accuracy: 0.8340
Epoch 5/10
272/272 - 163s - loss: 0.9424 - accuracy: 0.7300 - val_loss: 0.4131 - val_accuracy: 0.8760
Epoch 6/10
272/272 - 168s - loss: 0.8950 - accuracy: 0.7367 - val_loss: 0.4332 - val_accuracy: 0.8580
Epoch 7/10
272/272 - 168s - loss: 0.8645 - accuracy: 0.7459 - val_loss: 0.4448 - val_accuracy: 0.8680
Epoch 8/10
272/272 - 166s - loss: 0.8210 - accuracy: 0.7580 - val_loss: 0.4274 - val_accuracy: 0.8800
Epoch 9/10
272/272 - 166s - loss: 0.7941 - accuracy: 0.7653 - val_loss: 0.4054 - val_accuracy: 0.8880
Epoch 10/10
272/272 - 165s - loss: 0.7615 - accuracy: 0.7753 - val_loss: 0.3917 - val_accuracy: 0.8800
```

Figura 8: Aquí se ve el progreso del entrenamiento durante las últimas 10 épocas. Gracias al uso de la red preentrenada, el resultado es bastante bueno.

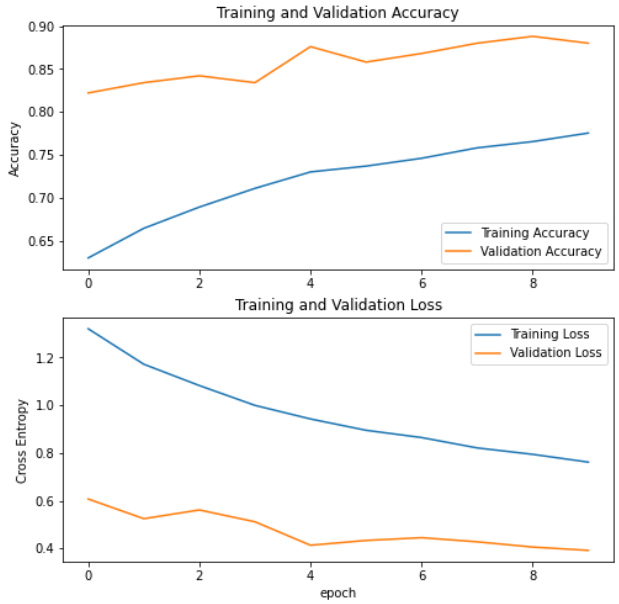

Figura 9: Las gráficas que muestran la exactitud y pérdida del entrenamiento del modelo preentrenado en el conjunto de entrenamiento de datos aumentados y validación.

Con el uso de los datos aumentados y EfficienNetB0 el entrenamiento de la red nos arrojó una exactitud del 77 % en el conjunto de entrenamiento y de 88 % en el conjunto de validación.

4.5. Análisis de errores.

Ahora que ya terminamos de entrenar, vamos a revisar diferentes métricas para ver el rendimiento de nuestra red:

- Precisión (Precision): Indica la proporción de clases que fueron clasificadas como verdaderos positivos (TP) entre todos los ejemplos que el modelo clasificó como positivos.
$$\frac{TP}{TP + FP}$$
- Sensibilidad (Recall): Indica la proporción de clases que fueron clasificadas como verdaderos positivos (TP) entre todos los ejemplos que realmente son positivos.
$$\frac{TP}{TP + FN}$$
Donde FN son falsos negativos
- F1-score: Es la media armónica de la precisión y la exhaustividad. Se utiliza la media armónica porque castiga más fuertemente los valores extremos que la media aritmética, proporcionando un balance más justo entre las dos métricas.
$$F1 = 2 \cdot \frac{\text{Precisión} \cdot \text{Recall}}{\text{Precisión} + \text{Recall}}$$
- Matriz de confusión: Proporciona una representación visual de los resultados de la predicción de un modelo comparándolos con los valores reales.

4.5.1. Precisión, Sensibilidad y F1-Score

```
y_true = test_dataset.classes
y_pred = np.argmax(model3.predict(test_dataset), axis = 1)
f1 = f1_score(y_true, y_pred, average='macro')
print("F1 Score:", f1)
print(classification_report(y_true, y_pred, target_names=
    test_dataset.class_indices.keys()))
```

```
10/10 [==============================] - 3s 235ms/step
F1 Score: 0.8766594516594516
                     precision    recall  f1-score   support

           air hockey       1.00      0.60      0.75         5
      ampute football       1.00      1.00      1.00         5
              archery       1.00      1.00      1.00         5
         arm wrestling       1.00      1.00      1.00         5
          axe throwing       1.00      1.00      1.00         5
        balance beam       1.00      1.00      1.00         5
        barell racing       1.00      0.80      0.89         5
             baseball       0.75      0.60      0.67         5
           basketball       0.83      1.00      0.91         5
       baton twirling       1.00      0.80      0.89         5
            bike polo       0.56      1.00      0.71         5
            billiards       0.71      1.00      0.83         5
                  bmx       0.00      0.00      0.00         5
              bobsled       1.00      1.00      1.00         5
              bowling       0.80      0.80      0.80         5
               boxing       1.00      1.00      1.00         5
         bull riding       0.83      1.00      0.91         5
      bungee jumping       0.83      1.00      0.91         5
        canoe slamon       1.00      1.00      1.00         5
       cheerleading       1.00      0.40      0.57         5
  chuckwagon racing       1.00      0.60      0.75         5
             cricket       0.83      1.00      0.91         5
             croquet       1.00      1.00      1.00         5
             curling       1.00      1.00      1.00         5
           disc golf       1.00      0.80      0.89         5
             fencing       1.00      0.80      0.89         5
        field hockey       0.71      1.00      0.83         5
      sailboat racing       0.71      1.00      0.83         5
            shot put       0.71      1.00      0.83         5
         shuffleboard       1.00      0.80      0.89         5
       sidecar racing       0.80      0.80      0.80         5
          ski jumping       1.00      1.00      1.00         5
          sky surfing       0.75      0.60      0.67         5
           skydiving       0.83      1.00      0.91         5
       snow boarding       1.00      0.60      0.75         5
  snowmobile racing       1.00      0.80      0.89         5
       speed skating       0.83      1.00      0.91         5
      steer wrestling       1.00      1.00      1.00         5
       sumo wrestling       1.00      0.60      0.75         5
              surfing       1.00      0.80      0.89         5
             swimming       1.00      0.80      0.89         5
         table tennis       0.80      0.80      0.80         5
               tennis       1.00      1.00      1.00         5
        track bicycle       1.00      1.00      1.00         5
              trapeze       1.00      0.60      0.75         5
          tug of war       0.80      0.80      0.80         5
            ultimate       1.00      0.60      0.75         5
         uneven bars       1.00      1.00      1.00         5
          volleyball       0.83      1.00      0.91         5
       water cycling       0.83      1.00      0.91         5
          water polo       0.71      1.00      0.83         5
      weightlifting       1.00      1.00      1.00         5
wheelchair basketball       1.00      1.00      1.00         5
   wheelchair racing       0.80      0.80      0.80         5
      wingsuit flying       1.00      1.00      1.00         5

            accuracy                           0.88       500
           macro avg       0.90      0.88      0.88       500
        weighted avg       0.90      0.88      0.88       500
```

Figura 10: Tabla para mostrar la Precisión, Sensibilidad y F1-Score del modelo.

Aquí podemos ver las clases y las métricas; la precisión es para ver los clasificados como positivos, las sensibilidad es para ver cuántos de esos positivos SI son positivos y el f1-score es para ver qué tanto se desvían las otras dos métricas. En el caso del air hockey podemos ver que la sensibilidad detectó que al menos un 40 % de las imágenes que clasificó el modelo no son de esa clase. En el caso del billar, la precisión estuvo menor que la sensibilidad, por lo que en f1-score subió. Y tenemos el caso del BMX donde nada fue clasificado de manera correcta. En promedio, el modelo clasifica con 88 % de precisión las imágenes de deportes.

Podemos ver una tabla para comparar el Air Hockey y el Ampute Football y comprobar qué clases clasificó erróneamente.

```
classes = dict(zip(test_dataset.class_indices.values(),
    test_dataset.class_indices.keys()))
Predictions = pd.DataFrame({"Image Index" : list(range(len(
    test_dataset.labels))),
                            "Test Labels" : test_dataset.labels,
                            "Test Classes" : [classes[i] for i in
    test_dataset.labels],
                            "Prediction Labels" : y_pred,
                            "Prediction Classes" : [classes[i] for
    i in y_pred],
                            "Path": test_dataset.filenames,
                            "Prediction Probability" : [x for x in
    np.asarray(tf.reduce_max(model3.predict(test_dataset), axis =
    1))]
                           })
Predictions.head(8)
```

	Image Index	Test Labels	Test Classes	Prediction Labels	Prediction Classes	Path	Prediction Probability
0	0	0	air hockey	0	air hockey	air hockey\1.jpg	0.964227
1	1	0	air hockey	86	table tennis	air hockey\2.jpg	0.854314
2	2	0	air hockey	0	air hockey	air hockey\3.jpg	0.510868
3	3	0	air hockey	11	billiards	air hockey\4.jpg	0.983025
4	4	0	air hockey	0	air hockey	air hockey\5.jpg	0.729177
5	5	1	ampute football	1	ampute football	ampute football\1.jpg	0.996712
6	6	1	ampute football	1	ampute football	ampute football\2.jpg	0.995570
7	7	1	ampute football	1	ampute football	ampute football\3.jpg	0.527842

Figura 11: Tabla de predicciones del modelo.

En esta tabla tenemos, de izquierda a derecha, el índice de la imagen, la clase a la que corresponde, el nombre de la clase a la que corresponde, la clase a la que la red la clasificó, el nombre le la clase a la que la red la clasificó, la ruta en el equipo de la imagen clasificada, y la probabilidad con la que clasificó la imagen.

Podemos ver ya el 60 % de sensibilidad que marcaba la tabla anterior, puesto que se clasificó erróneamente el tenis de mesa y el billar. Estas clasificaciones erróneas tiene sentido puesto que las características de estos 3 deportes son muy similares; incluyen una mesa, un objeto a golpear (un puck, una pelota o una bola de billar) y un objeto para golpear (un brazo sosteniendo un empujador, una paleta o el taco de billar). Esto gracias a que las características no fueron resaltadas de manera apropiada por el entrenamiento y eso haya causado estos resultados.

4.5.2. Matriz de Confusión

```
preds = model3.predict_generator(test_dataset)
y_pred = np.argmax(preds, axis=1)
g_dict = test_dataset.class_indices
classes = list(g_dict.keys())

cf_matrix = confusion_matrix(test_dataset.classes, y_pred,
    normalize='true')
plt.figure(figsize = (20,15))
sns.heatmap(cf_matrix, annot=False, xticklabels = sorted(set(
    test_dataset.classes)), yticklabels = sorted(set(test_dataset.
    classes)))
plt.title('Normalized Confusion Matrix')
plt.show()
```

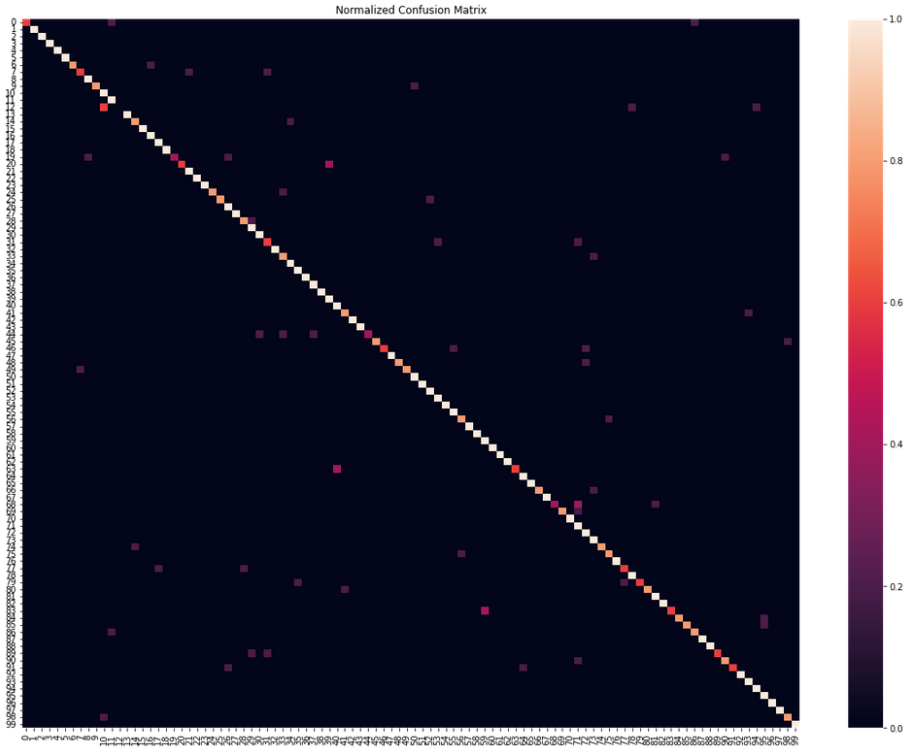

Figura 12: La matriz de confusión es una representación visual de las clasificaciones dadas por el modelo después del entrenamiento comparadas con las clasificaciones verdaderas.

Las matriz de confusión ayuda a ver de manera más clara qué clases tuvieron más errores de clasificación, puesto que se derivan de la diagonal donde se supone que se encuentran todas las clasificaciones correctas. Podemos observar que unas

clases fueron clasificadas erróneamente a clases cercadas, esto posiblemente a que sean deportes muy parecidos. O está el caso donde ninguna imagen fue clasificada de manera correcta, cercano al inicio de la diagonal.

Estos resultados ameritarían la modificación de unos pesos de la red o aumentar más el conjunto de datos de entrenamiento, pero los resultados son favorables en muchos sentidos

4.6. Implementación.

Consulta el Notebook de Jupyter en la carpeta de **Practica 3** llamado ***TF-Deportes.ipynb***