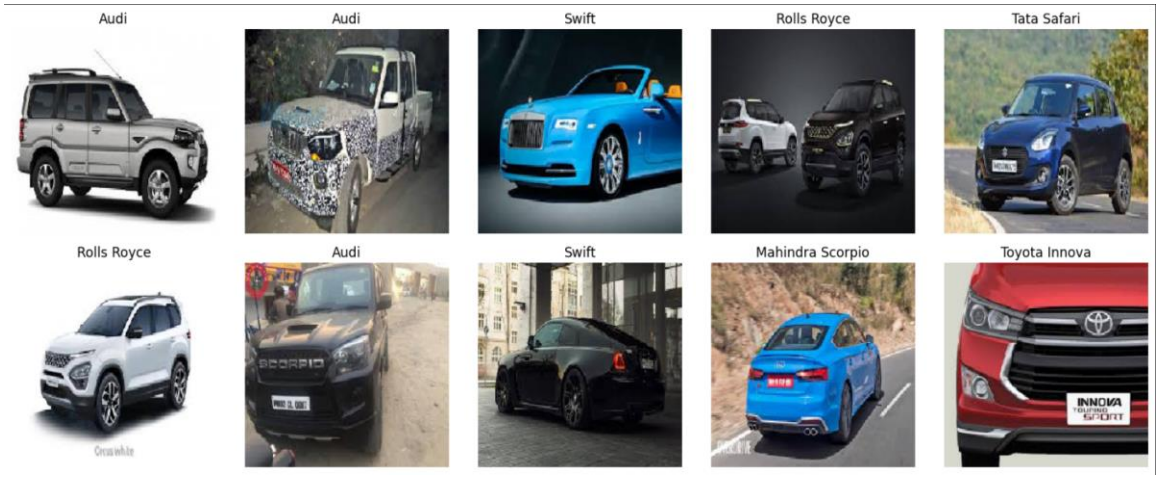
Modulo 2. Inteligencia Artificial

Keyuan Zhao A01366831

Contexto del dataset

El dataset fue obtenido en la plataforma de Kaggle, en donde tiene almacenado un total de 4,165 de imágenes en formato JPG, las cuales está almacenado en dos carpetas principales, train y test, cada una de las carpetas contiene subcarpetas en donde está divididos por marcas de carros: Audi, Hyundai Creta, Mahindra Scorpio, Rolls Royce, Swift, Tata Safari y Toyota Innova. La carpeta de train contienen 3,352 imágenes y la carpeta de test contiene 813 imágenes.

Para la aplicación de framework se utilizó la librería de Tensorflow con Keras y algunos modelos de aplicación para procesamiento de capas.



Modelo 1: CNN

El primer modelo se utilizó CNN como arquitectura para el entrenamiento del modelo. La decisión de usar CNN como modelo base fue por su gran ventaja de poder reducir el número de parámetros sin perder la calidad de los modelos. Basando en su arquitectura, se generó 9 capas para la construcción del modelo.

```
model = Sequential()

model.add(layers.Convolution2D(32, (3, 3), input_shape=(128, 128, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

model.add(layers.Convolution2D(32, (3, 3), activation='relu'))

model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())

model.add(layers.Dense(units=96, activation='relu'))
model.add(layers.Dropout(0.40))
model.add(layers.Dense(units=32, activation='relu'))
model.add(layers.Dense(units=7, activation='softmax'))
```

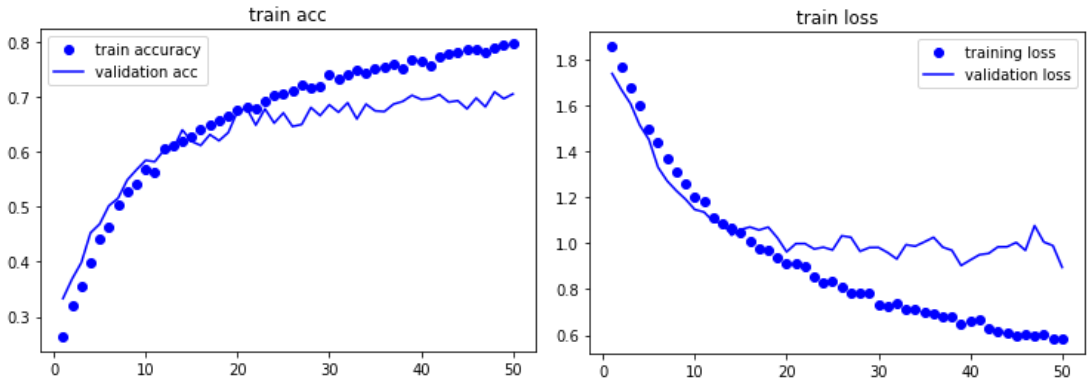
Después se utilizó Adam como optimizador y “sparse categorical crossentropy” para nuestra clasificación de etiquetas, ya que en nuestro dataset cuenta con más de dos clases de vehículos y es efectivo usando dicha métrica para calcular el error del modelo (TensorFlow, 2023).

Para al final, entrenarlo con 50 épocas dándole nuestras imágenes del test para ver qué tanto de efectividad está aprendiendo nuestro modelo.

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

```
history = model.fit(train_generator,
                    validation_data=test_generator,
                    epochs = 50)
```

Una vez entrenado el modelo, graficamos la comparación del acurracy y loss de train y test. En la gráfica de “trian acc” podemos notar que el modelo fue aprendiendo correctamente y validando con las imágenes de test, lo cual para test obtuvo un 70% de exactitud en clasificar una imagen. Por otro lado, la gráfica de “train loss” notamos que el error fue disminuyendo conforme pasan las épocas; para test no favorable a partir de la época 20, ya que el error no bajo más del 1.



Para el evaluar nuestro primer modelo utilizamos con 4 de steps, lo cual nos dio como resultado 56.25% de exactitud y 1.54 de loss clasificando las imágenes de nuestro test. Debido por su baja exactitud, este modelo lo tomamos como base para comparar con otros modelos.

```
[43]
    test_loss, test_acc = model.evaluate(test_generator, steps = 4)
    print('\ntest acc :\n', test_acc)
    print('\ntest loss :\n', test_loss)
```

```
4/4 [==============================] - 0s 32ms/step - loss: 1.5444 - acc: 0.5625

test acc :
 0.5625

test loss :
 1.5443692207336426
```

Modelo 2: Transfer Learning con VGG16

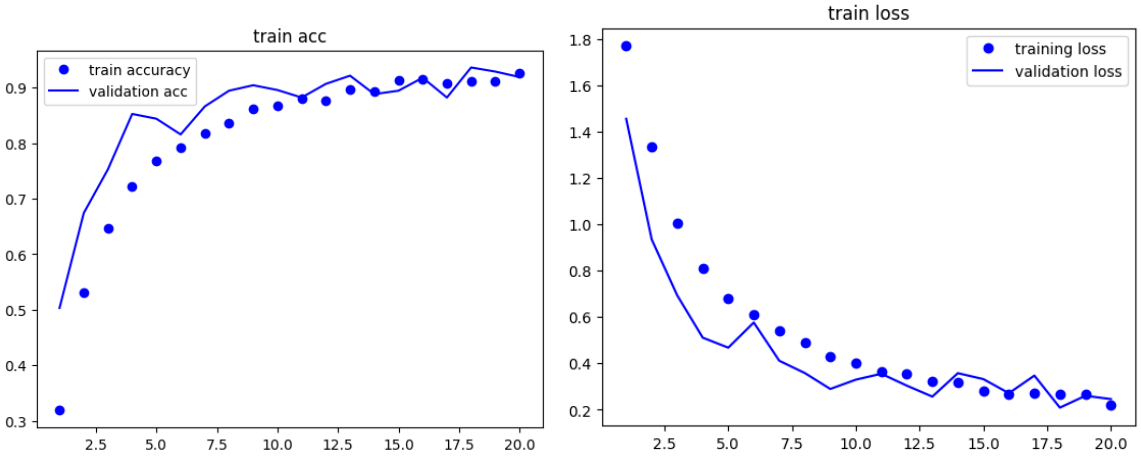
Para el modelo 2 ocupamos Transfer Learning cargando un modelo pre-entrenado de VGG16, ingresando capas de CNN que ocupamos en el modelo 1.

```
from tensorflow.keras.applications.vgg16 import VGG16
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False ## Not trainable weights

model = Sequential()

model.add(base_model)
model.add(layers.Convolution2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))

# model.add(layers.Convolution2D(32, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())

model.add(layers.Dense(units=96, activation='relu'))
model.add(layers.Dropout(0.40))
model.add(layers.Dense(units=32, activation='relu'))
model.add(layers.Dense(units=7, activation='softmax'))
```

Para este modelo ocupamos los mismos parámetros que el modelo 1 para entrenarlo, la única diferencia fue que modificamos la cantidad de épocas a 20 para reducir el tiempo de entrenamiento. Esto es debido a que VGG16 es un modelo pre-entrenado y no es necesario agregar más épocas para el entrenamiento.

```
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])


history = model.fit(train_generator,
                    validation_data=test_generator,
                    epochs = 20)
```

Después del entrenamiento del modelo se generó las siguientes gráficas. Podemos observar que en la gráfica de “train acc”, la efectividad de train y test fue mejorando conforme pasan las épocas, ambas llegando a cerca de 100% de efectividad. Por otro lado, la gráfica de “train loss” también tiene un comportamiento adecuado, es notorio ver como baja el loss conforme pasan las épocas.



Realizando las evaluaciones para este modelo con las imágenes de test con 4 de steps, notamos que la efectividad fue de 92.57%, lo cual es bastante alta a comparación del modelo 1. Tomamos este modelo como base para buscar si todavía se puede aumentar la efectividad con nuestro último modelo.

```
test_loss, test_acc = model.evaluate(test_generator, steps = 4)
print('\ntest acc :\n', test_acc)
print('\ntest loss :\n', test_loss)
```

```
4/4 [==============================] - 7s 1s/step - loss: 0.2080 - acc: 0.9258

test acc :
 0.92578125

test loss :
 0.2079908549785614
```

Modelo 3: CNN

Para este modelo se utilizó la arquitectura de CNN al igual que el modelo 1, lo único que modificó fue cambiarle el tamaño de las capas para que se pueda adaptar mejor con las capas.

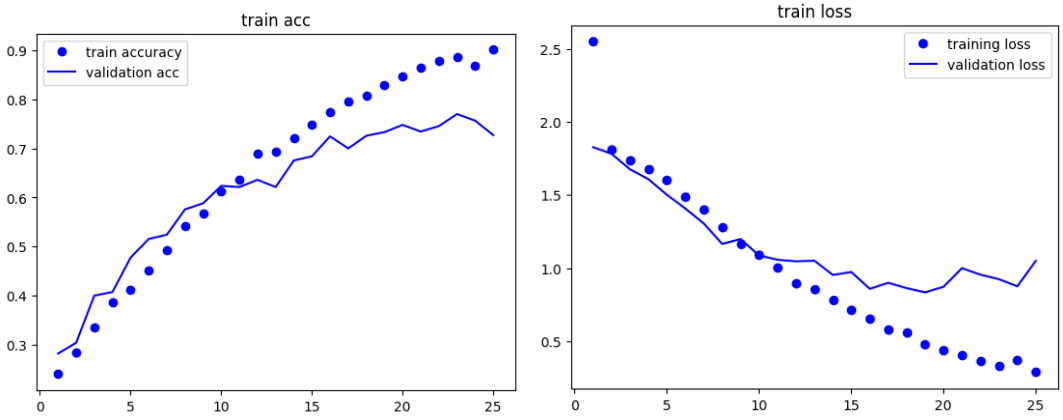
```
model = Sequential([
        layers.Conv2D(128, (3,3), activation='relu', input_shape=(256, 256, 3)),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(256, (3,3), activation='relu'),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(512, (3,3), activation='relu'),
        layers.MaxPooling2D(2,2),
        layers.Conv2D(512, (3,3), activation='relu'),
        layers.MaxPooling2D(2,2),
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dense(7, activation='softmax')
])
```

Al momento de compilar el modelo se utilizó “categorial crossentropy”, lo cual es muy similar a “sparse categorial crossentropy” por el manejo de las etiquetas, la diferencia clave es la forma que representa las etiquetas del destino.

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

```
history = model.fit(
      train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size,
      epochs=25,
      validation_data=test_generator,
      validation_steps=test_generator.samples/test_generator.batch_size,
      verbose=1)
```

Al generar las gráficas para este modelo, notamos que el entrenamiento del modelo no fue tan efectivo como el modelo 2, ya que si observamos la gráfica de “train acc”, para la efectividad del test no fue suficiente como para el train. Esto pasa igual que en la gráfica de “train loss”, el comportamiento del error para el test no fue lo suficiente y presenta con detalles.



Al igual que los modelos anteriores, evaluamos el modelo con 4 steps y obtuvimos 75.78% de efectividad en clasificar una imagen, lo cual no fue tan suficiente como el modelo 2.

```
test_loss, test_acc = model.evaluate(test_generator, steps = 4)
print('\ntest acc :\n', test_acc)
print('\ntest loss :\n', test_loss)
```

```
4/4 [==============================] - 4s 1s/step - loss: 1.0038 - acc: 0.7578

test acc :
 0.7578125

test loss :
 1.0038481950759888
```

Conclusiones

El modelo que se destaca para la realización de este proyecto es el modelo 2, ya que por su alta efectividad en la evaluación nos permite clasificar de manera óptima las una de las 7 clases de vehículos mencionados anteriormente. Además, notamos la importancia de Transfer Learning como herramienta para mejorar el rendimiento del modelo gracias por su gran capacidad de adopción de capas externas.

Considero que en un trabajo futuro de este proyecto será necesario agregar más clases para ampliar el dataset, ya que no es suficiente con solo 7 clases de marcas de automóviles. Al aumentar más tipos de automóviles, tal vez es factible utilizar los mismos pasos que hicimos para el modelo 2 usando Transfer Learning.

Referencias

TensorFlow. (2023). “tf.keras.losses.CategoricalCrossentropy”. https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy

Dataset: https://www.kaggle.com/datasets/kshitij192/cars-image-dataset