```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
        print(os.path.join(dirname))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,Dropout,Convolution2D,MaxPooling2D,Flatten #action detectionimport tensorflow
import tensorflow as tf
import matplotlib.pyplot as plt
from IPython.display import HTML
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
IMAGE_SIZE = 128

train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True
)
train_generator = train_datagen.flow_from_directory(
        '/content/drive/MyDrive/Cars Dataset/train',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        class_mode="sparse",
)
```

```
Found 3352 images belonging to 7 classes.
```

```
count=0
for image_batch, label_batch in train_generator:
#     print(label_batch)
    print(image_batch[0])
    break
```

```
[[[0.22543551 0.22543551 0.17837669]
  [0.21077232 0.21077232 0.16371349]
  [0.4016731  0.4016731  0.3546143 ]
  ...
  [0.9568628  0.9568628  0.91300875]
  [0.9584651  0.9584651  0.91604483]
  [0.9607844  0.9607844  0.91372555]]

 [[0.3520475  0.3520475  0.30498868]
  [0.2639154  0.2639154  0.21685658]
  [0.3979375  0.3979375  0.3508787 ]
  ...
  [0.9568628  0.9568628  0.91295815]
  [0.95843977 0.95843977 0.9160701 ]
  [0.9607844  0.9607844  0.91372555]]

 [[0.44917092 0.44917092 0.40211207]
  [0.30422154 0.30422154 0.2571627 ]
  [0.39525118 0.39525118 0.34819236]
  ...
  [0.9568628  0.9568628  0.9129075 ]
  [0.9584145  0.9584145  0.91609544]
  [0.9607844  0.9607844  0.91372555]]

 ...

 [[0.09320191 0.08928034 0.11280976]
  [0.09327786 0.08935629 0.11288571]
  [0.0933538  0.08943223 0.11296164]
  ...
  [0.21592927 0.2120077  0.19632143]
  [0.21854743 0.21462587 0.19893959]
  [0.2182389  0.21431734 0.19863106]]

 [[0.09573017 0.0918086  0.11533801]
  [0.09570486 0.09178329 0.1153127 ]
  [0.09567954 0.09175797 0.11528739]
  ...
```

```
   [0.20134674 0.19742517 0.18173888]
   [0.20417634 0.20025477 0.1845685 ]
   [0.20599721 0.20207565 0.18638937]]

  [[0.08026382 0.07634225 0.09987167]
   [0.08011194 0.07619037 0.09971979]
   [0.07996006 0.07603849 0.09956791]
   ...
   [0.22976013 0.22583856 0.21015228]
   [0.23368162 0.22976005 0.21407378]
   [0.23067619 0.22675462 0.21106835]]]
```

```
class_names = list(train_generator.class_indices.keys())
class_names
```

```
['Audi',
 'Hyundai Creta',
 'Mahindra Scorpio',
 'Rolls Royce',
 'Swift',
 'Tata Safari',
 'Toyota Innova']
```

```
test_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True)

test_generator = test_datagen.flow_from_directory(
        '/content/drive/MyDrive/Cars Dataset/test',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        class_mode="sparse"
)
```

```
Found 813 images belonging to 7 classes.
```

```
for image_batch, label_batch in test_generator:
    print(image_batch[0])
    break
```

```
[[[1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  ...
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]]

 [[1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  ...
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]]

 [[1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  ...
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]]

 ...

 [[0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  ...
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]]

 [[0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  ...
  [1.        1.        1.       ]
  [1.        1.        1.       ]
  [1.        1.        1.       ]]

 [[0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  [0.9960785 0.9960785 0.9960785]
  ...
```

```
 [1.        1.        1.        ]
 [1.        1.        1.        ]
 [1.        1.        1.        ]]]
```

```
sz = 128

# Initializing the CNN
model = Sequential()

# First convolution layer and pooling
model.add(Convolution2D(32, (3, 3), input_shape=(sz, sz, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
# Second convolution layer and pooling
model.add(Convolution2D(32, (3, 3), activation='relu'))
# input_shape is going to be the pooled feature maps from the previous convolution layer
model.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers
model.add(Flatten())
# Adding a fully connected layer
model.add(Dense(units=97, activation='relu'))
model.add(Dropout(0.40))
model.add(Dense(units=32, activation='relu'))
model.add(Dense(units=7, activation='softmax')) # softmax for more than 2
```

```
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 126, 126, 32)      896       
                                                                 
 max_pooling2d_8 (MaxPoolin  (None, 63, 63, 32)        0         
 g2D)                                                            
                                                                 
 conv2d_9 (Conv2D)           (None, 61, 61, 32)        9248      
                                                                 
 max_pooling2d_9 (MaxPoolin  (None, 30, 30, 32)        0         
 g2D)                                                            
                                                                 
 flatten_4 (Flatten)         (None, 28800)             0         
                                                                 
 dense_12 (Dense)            (None, 97)                2793697   
                                                                 
 dropout_4 (Dropout)         (None, 97)                0         
                                                                 
 dense_13 (Dense)            (None, 32)                3136      
                                                                 
 dense_14 (Dense)            (None, 7)                 231       
                                                                 
=================================================================
Total params: 2807208 (10.71 MB)
Trainable params: 2807208 (10.71 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(optimizer='adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),metrics=['accuracy'])
```

```
history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=5
)
```

```
Epoch 1/5
105/105 [==============================] - 314s 3s/step - loss: 1.8184 - accuracy: 0.2915 - val_loss: 1.7538 - val_accuracy: 0.3764
Epoch 2/5
105/105 [==============================] - 84s 797ms/step - loss: 1.6880 - accuracy: 0.3729 - val_loss: 1.5904 - val_accuracy: 0.46
Epoch 3/5
105/105 [==============================] - 87s 822ms/step - loss: 1.5617 - accuracy: 0.4335 - val_loss: 1.4325 - val_accuracy: 0.50
Epoch 4/5
105/105 [==============================] - 87s 822ms/step - loss: 1.4318 - accuracy: 0.4717 - val_loss: 1.3584 - val_accuracy: 0.51
Epoch 5/5
105/105 [==============================] - 85s 809ms/step - loss: 1.3464 - accuracy: 0.5075 - val_loss: 1.3073 - val_accuracy: 0.53
```

```
scores = model.evaluate(test_generator)
```

```
26/26 [==============================] - 10s 382ms/step - loss: 1.3057 - accuracy: 0.5301
```

```
scores
```

```
[1.3057115077972412, 0.5301352739334106]
```

```
history.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
type(history.history['loss'])
```

```
list
```

```
len(history.history['loss'])
```

```
5
```

```
history.history['loss'][:5] # show loss for first 5 epochs
```

```
[1.8184212446212769,
 1.688022494316101,
 1.5616528987884521,
 1.4318387508392334,
 1.3464020490646362]
```

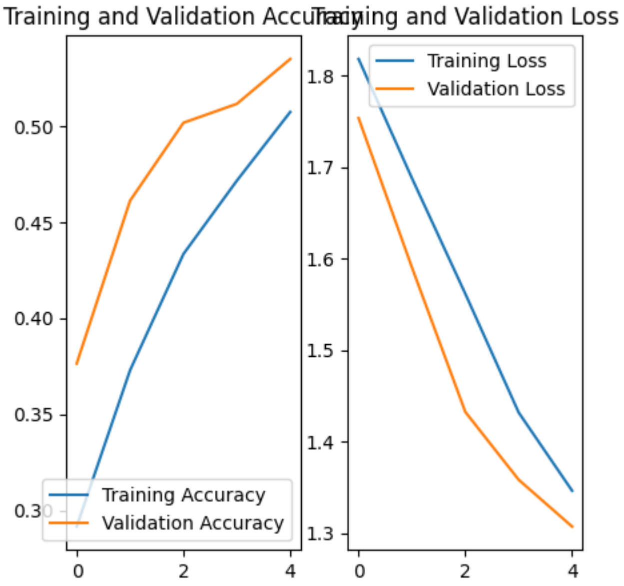
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

```
import matplotlib.pyplot as plt
EPOCHS = 5

plt.figure(figsize=(5, 5))
plt.subplot(1, 2, 1)
plt.plot(range(EPOCHS), acc, label='Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(range(EPOCHS), loss, label='Training Loss')
plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```




```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
```

```
        confidence = round(100 * (np.max(predictions[0])), 2)
        return predicted_class, confidence


    plt.figure(figsize=(15, 15))
    for images, labels in test_generator:
        for i in range(6):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i])

            predicted_class, confidence = predict(model, images[i])
            actual_class = class_names[int(labels[i])]

            plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

            plt.axis("off")
        break
```

```
1/1 [==============================] - 0s 116ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 28ms/step
```