AI ENABLED CAR USING OPENCV

Introduction:

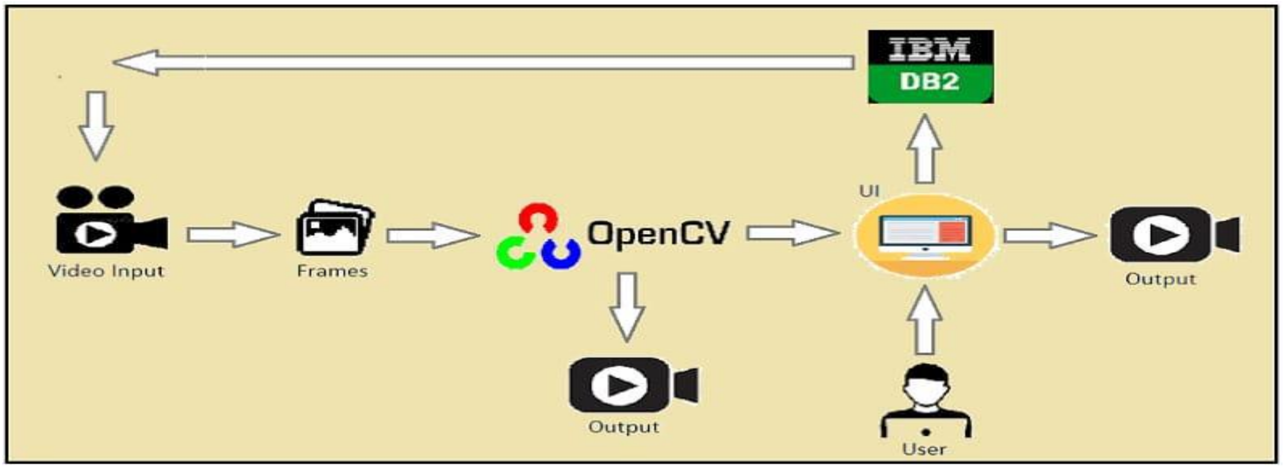
In the coming few years, the surge in urbanization is anticipated to lead to an increased demand for parking facilities, driven by the growing urban population and the continuous expansion of car ownership. This trend poses a considerable challenge in efficiently managing parking spaces. Traditional methods of parking management, such as manual attendants or static signage, are proving to be inadequate and outdated in the face of this rapid urbanization.

To tackle this emerging challenge and leverage the capabilities of modern technology, we are introducing an AI-enabled car parking system that incorporates OpenCV and advanced artificial intelligence algorithms. The primary objective of this system is to revolutionize the parking experience by enhancing convenience, efficiency, and user-friendliness.

Similar to the waste separation system you described as a response to the challenges of waste accumulation, our AI-enabled car parking system provides a smarter solution to the pressing issues in parking. By integrating computer vision and AI technologies, the system automates the entire parking process, from identifying vehicles to allocating parking spaces. This means an end to the frustration of searching for an available spot in a parking lot or dealing with human attendants.

The core functionality of the system is reliant on OpenCV, a robust open-source computer vision library, and sophisticated Convolutional Neural Networks (CNNs). It excels in accurately detecting and recognizing vehicles, evaluating available parking spaces, and guiding drivers to their designated spots, thereby reducing both parking time and frustration. Moreover, the system's adaptability makes it suitable for deployment in various environments, encompassing both indoor and outdoor parking facilities.

Solution Architecture:



Prerequisites:

To complete this project, we must require the following software's, concepts, and packages

To successfully undertake the development of an AI-enabled car parking system using OpenCV, you will need the following software, concepts, and packages:

1. **Google Colab:** Google Colab is a free, cloud-based platform that provides a Jupyter notebook environment with access to powerful GPU and CPU resources. This is where you'll be coding and running your AI-enabled car parking project. You can access Google Colab through a web browser.
2. **OpenCV (Open Source Computer Vision Library):** OpenCV is an essential computer vision library that you can use to process and analyze visual data. It provides a wide range of tools for image and video processing, making it a fundamental component of this project.
3. **Python Programming Language**: A good understanding of Python is required, as it's the primary programming language used in this project. Python is known for its simplicity and readability, making it an excellent choice for AI and computer vision projects.
4. **Machine Learning and Computer Vision Concepts:** Familiarity with fundamental machine learning concepts, as well as computer vision techniques, is beneficial. Understanding concepts like object detection, image classification, and image processing will be essential for implementing the AI-enabled car parking system.
5. **OpenCV-Python:** You will need to install the OpenCV-Python package to access OpenCV's functionalities in your Google Colab environment. You can install it using pip or the package manager provided by Google Colab.

1. **To build Machine learning models we require the following packages**

- **Numpy**:
  - It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- **Scikit-learn:**
  - It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbors, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- **Flask:**
  Web framework used for building Web applications
- **Python packages:**
  - open anaconda prompt as administrator
  - Type "pip install numpy" and click enter.
  - Type "pip install pandas" and click enter.
  - Type "pip install scikit-learn" and click enter.
  - Type "pip install tensorflow==2.3.2" and click enter.
  - Type "pip install keras==2.3.1" and click enter

Deep Learning Concepts

- **CNN:** a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
  CNN Basic

- **Flask:** Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

  Flask Basics

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analyzed by the model which is integrated with flask application.
- CNN Models analyze the image, then prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
  - Create Train and Test Folders.
- Data Preprocessing.
  - Import the ImageDataGenerator library
  - Configure ImageDataGenerator class
  - ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
  - Import the model building Libraries
  - Initializing the model
  - Adding Input Layer
  - Adding Output Layer
  - Configure the Learning Process
  - Training and testing the model
  - Save the Model
  - Application Building
  - Create an HTML file
  - Build Python Code

Milestone 1: Data Collection

We received a single image and transformed it into sections representing occupied and vacant spaces to minimize plagiarism.

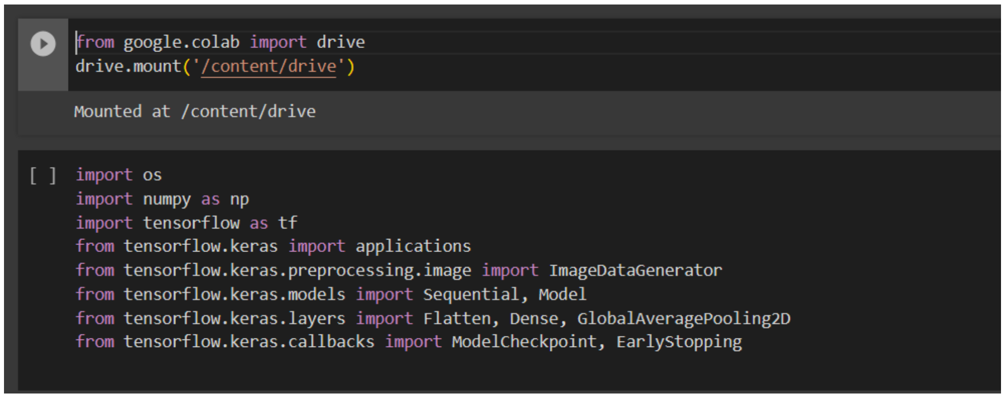
Download the Dataset-
https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr

Milestone 2: Image Preprocessing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Import the ImageDataGenerator library

Utilizing image data augmentation is a method to artificially increase the scale of a training dataset by generating altered versions of images within the dataset. The Keras deep learning neural network library offers the functionality to train models using image data augmentation through the implementation of the ImageDataGenerator class. Let's import the ImageDataGenerator class from the TensorFlow Keras module.



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

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

```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import applications
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

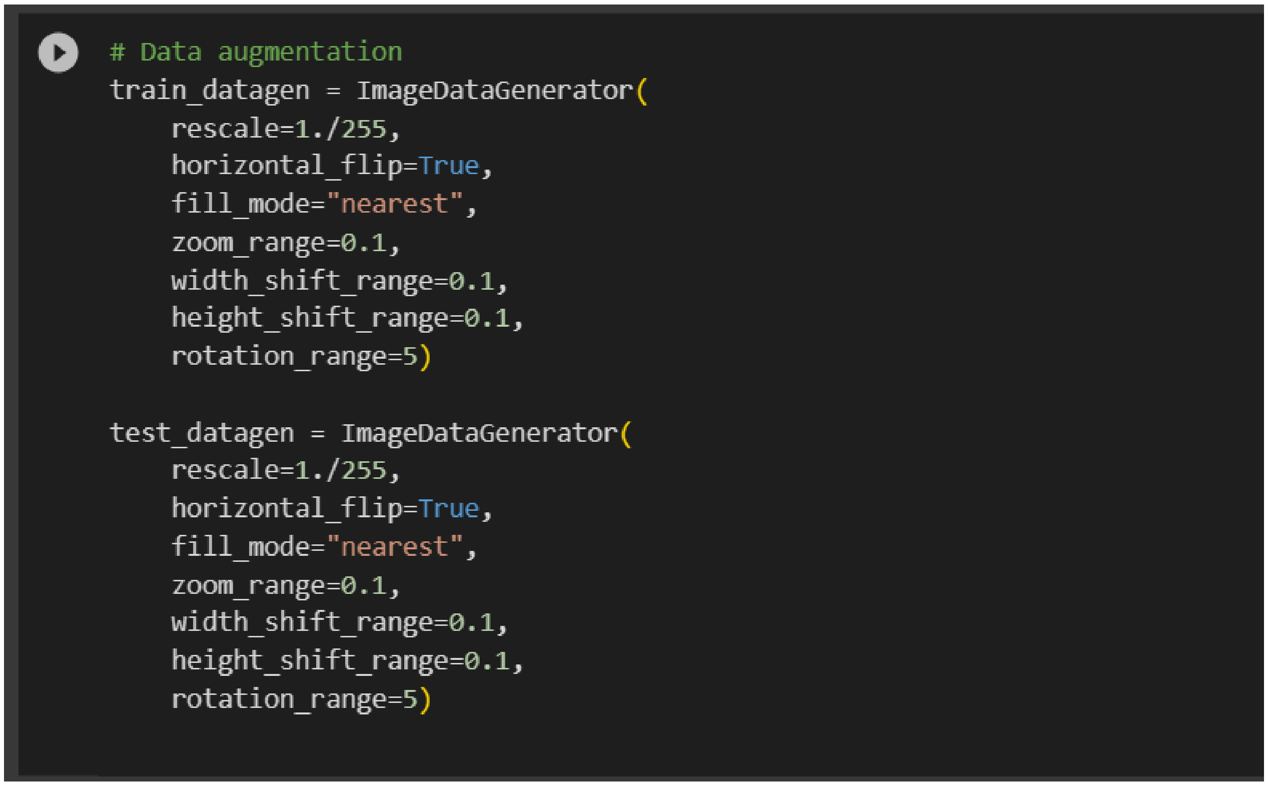
Activity 2: Configure ImageDataGenerator class

The ImageDataGenerator class is instantiated to configure various types of data augmentation techniques. These techniques include:

- Image shifts using the width_shift_range and height_shift_range parameters.
- Image flips through the horizontal_flip and vertical_flip parameters.
- Image rotations specified by the rotation_range parameter.
- Image brightness adjustments controlled by the brightness_range parameter.
- Image zooming defined by the zoom_range parameter.

To apply these augmentation techniques separately for training and testing, instances of the ImageDataGenerator class can be created.

Image Data Augmentation



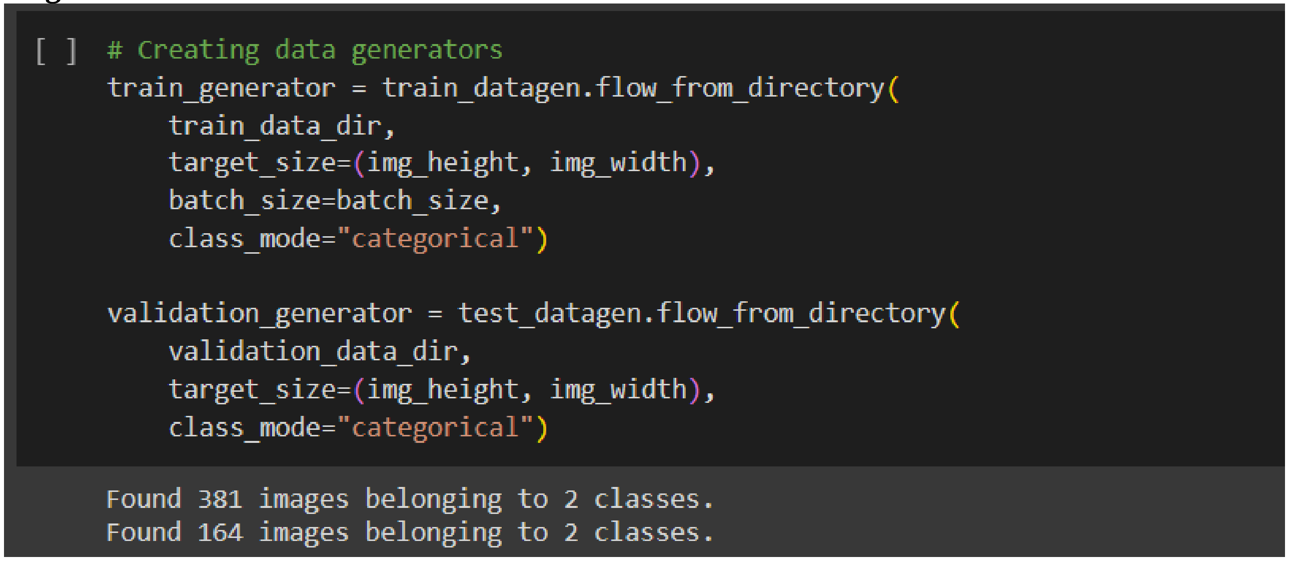
```
# Data augmentation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    fill_mode="nearest",
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rotation_range=5)

test_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    fill_mode="nearest",
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rotation_range=5)
```

Activity 3:Apply ImageDataGenerator functionality to Trainset and Testset

The ImageDataGenerator class is instantiated to configure various types of data augmentation techniques. These techniques include:

- Image shifts using the width_shift_range and height_shift_range parameters.
- Image flips through the horizontal_flip and vertical_flip parameters.
- Image rotations specified by the rotation_range parameter.
- Image brightness adjustments controlled by the brightness_range parameter.
- Image zooming defined by the zoom_range parameter.

To apply these augmentation techniques separately for training and testing, instances of the ImageDataGenerator class can be created. **Loading Our Data And Performing Data Augmentation**



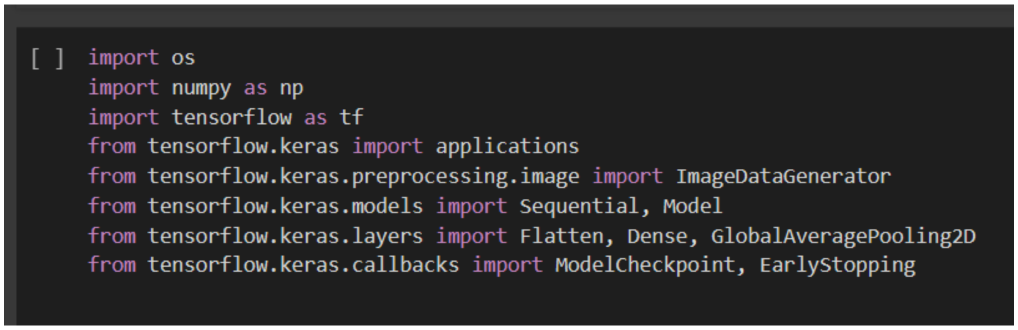
```
[ ]  # Creating data generators
     train_generator = train_datagen.flow_from_directory(
         train_data_dir,
         target_size=(img_height, img_width),
         batch_size=batch_size,
         class_mode="categorical")

     validation_generator = test_datagen.flow_from_directory(
         validation_data_dir,
         target_size=(img_height, img_width),
         class_mode="categorical")

     Found 381 images belonging to 2 classes.
     Found 164 images belonging to 2 classes.
```

We notice that 381 images belong to 2 classes for training and 164 images belong to 2 classes for testing purposes.

Milestone 3: Model Building

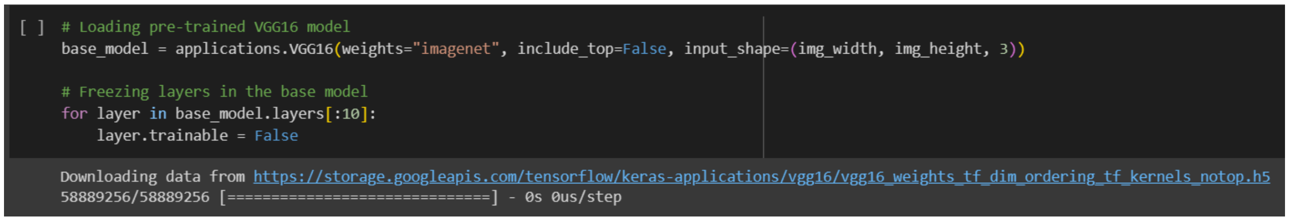
Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries



```
[ ]  import os
     import numpy as np
     import tensorflow as tf
     from tensorflow.keras import applications
     from tensorflow.keras.preprocessing.image import ImageDataGenerator
     from tensorflow.keras.models import Sequential, Model
     from tensorflow.keras.layers import Flatten, Dense, GlobalAveragePooling2D
     from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
```

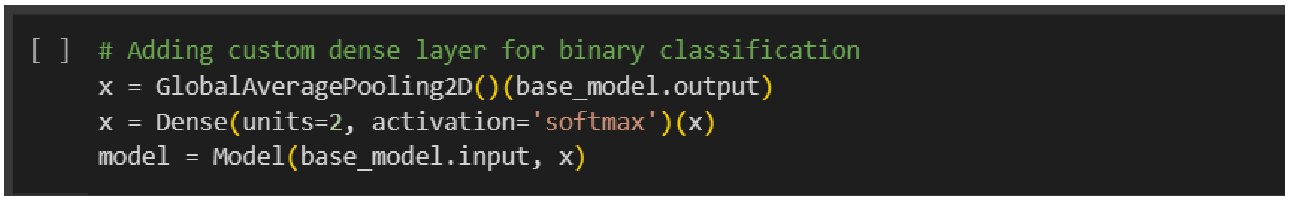
Activity 2: Initializing the model



```
[ ]  # Loading pre-trained VGG16 model
     base_model = applications.VGG16(weights="imagenet", include_top=False, input_shape=(img_width, img_height, 3))

     # Freezing layers in the base model
     for layer in base_model.layers[:10]:
         layer.trainable = False

     Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
     58889256/58889256 [==============================] - 0s 0us/step
```

Activity 3: Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

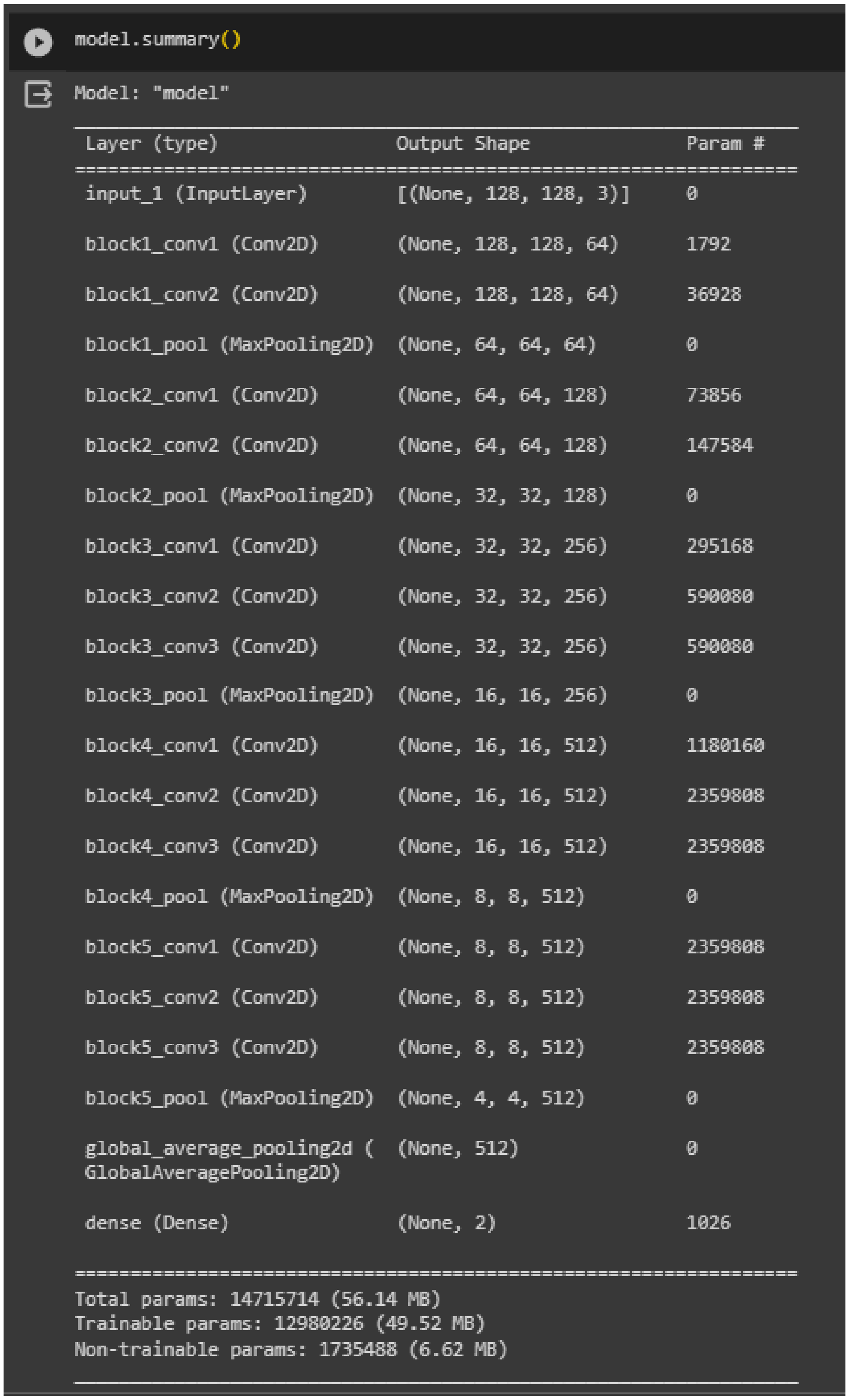


```
[ ]  # Adding custom dense layer for binary classification
     x = GlobalAveragePooling2D()(base_model.output)
     x = Dense(units=2, activation='softmax')(x)
     model = Model(base_model.input, x)
```

The Dense layer's neuron count aligns with the number of classes in the training set. In the final Dense layer, softmax activation is applied to transform outputs into corresponding probabilities.

Gaining insight into the model is crucial for effective utilization in training and prediction. Keras offers a straightforward 'summary' method to obtain comprehensive information about the model and its individual layers.

Summary of the Model



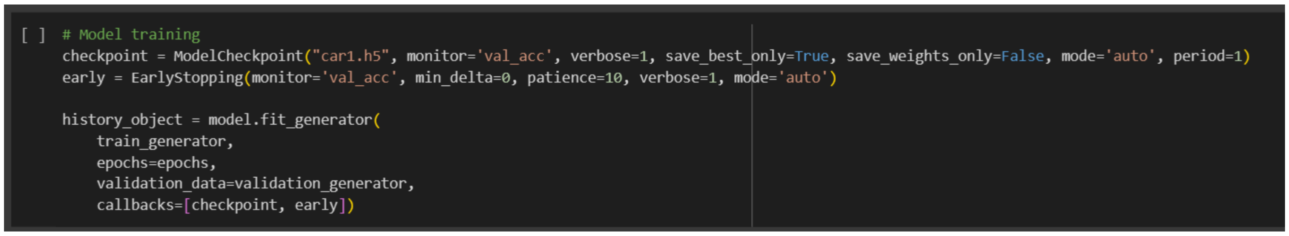
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 block1_conv1 (Conv2D)       (None, 128, 128, 64)      1792

 block1_conv2 (Conv2D)       (None, 128, 128, 64)      36928

 block1_pool (MaxPooling2D)  (None, 64, 64, 64)        0

 block2_conv1 (Conv2D)       (None, 64, 64, 128)       73856

 block2_conv2 (Conv2D)       (None, 64, 64, 128)       147584

 block2_pool (MaxPooling2D)  (None, 32, 32, 128)       0

 block3_conv1 (Conv2D)       (None, 32, 32, 256)       295168

 block3_conv2 (Conv2D)       (None, 32, 32, 256)       590080

 block3_conv3 (Conv2D)       (None, 32, 32, 256)       590080

 block3_pool (MaxPooling2D)  (None, 16, 16, 256)       0

 block4_conv1 (Conv2D)       (None, 16, 16, 512)       1180160

 block4_conv2 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_conv3 (Conv2D)       (None, 16, 16, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 8, 8, 512)         0

 block5_conv1 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv2 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_conv3 (Conv2D)       (None, 8, 8, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 2)                 1026

=================================================================
Total params: 14715714 (56.14 MB)
Trainable params: 12980226 (49.52 MB)
Non-trainable params: 1735488 (6.62 MB)
_________________________________________________________________
```

Activity 4: Configure The Learning Process

1. The culmination of model creation involves compilation, marking the transition to the training phase. The loss function, integral for identifying learning errors or deviations, is a prerequisite during the model compilation in Keras.
2. Optimization is a critical step in refining input weights through a comparison between predictions and the loss function. In this context, the adam optimizer is employed.
3. Metrics play a role in assessing model performance, akin to the loss function, although they aren't actively involved in the training process.

Compiling the Model



```
[ ] # Model training
    checkpoint = ModelCheckpoint("car1.h5", monitor='val_acc', verbose=1, save_best_only=True, save_weights_only=False, mode='auto', period=1)
    early = EarlyStopping(monitor='val_acc', min_delta=0, patience=10, verbose=1, mode='auto')

    history_object = model.fit_generator(
        train_generator,
        epochs=epochs,
        validation_data=validation_generator,
        callbacks=[checkpoint, early])
```

Activity 5: Train The model

To train our model using the image dataset, the fit_generator function is employed. The model undergoes training for 30 epochs, and after each epoch, the current model state is saved if the encountered loss is the lowest up to that point. The training loss exhibits a consistent decrease in almost every epoch throughout the 30 iterations, suggesting potential for further model improvement.

The fit_generator function takes several arguments:

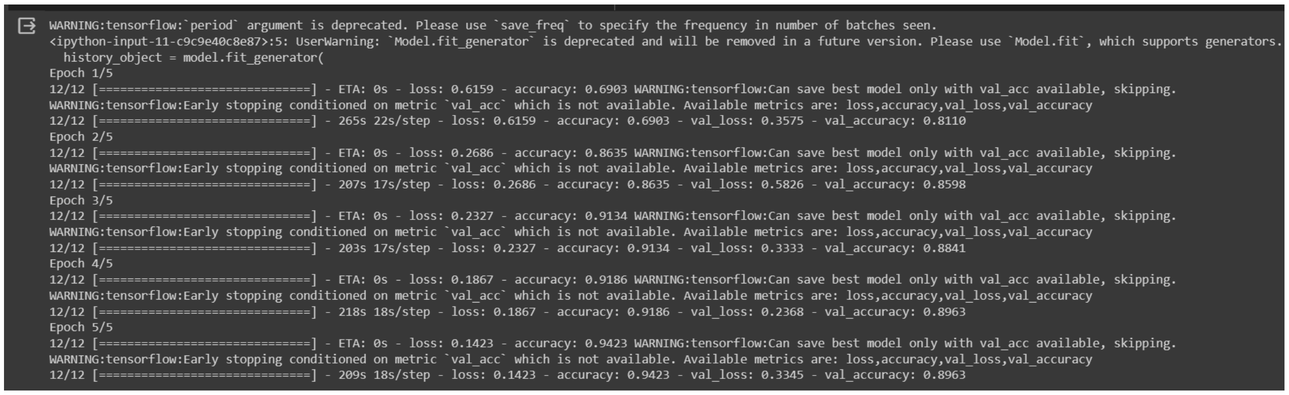
- **steps_per_epoch**: This parameter signifies the total number of steps taken from the generator once one epoch is completed and the subsequent epoch begins. The calculation for steps_per_epoch involves dividing the total number of samples in the dataset by the batch size.

- **Epochs**: This is an integer representing the number of epochs for which the model is trained.

- **validation_data**: It can be either a list of inputs and targets, a generator, or a list containing inputs, targets, and sample weights. This parameter is used to evaluate the loss and metrics for the model after each epoch.

- **validation_steps**: This argument is applicable only if the validation_data is a generator. It specifies the total number of steps taken from the generator before it stops at the end of each epoch. The value of validation_steps is calculated by dividing the total number of validation data points in the dataset by the validation batch size.

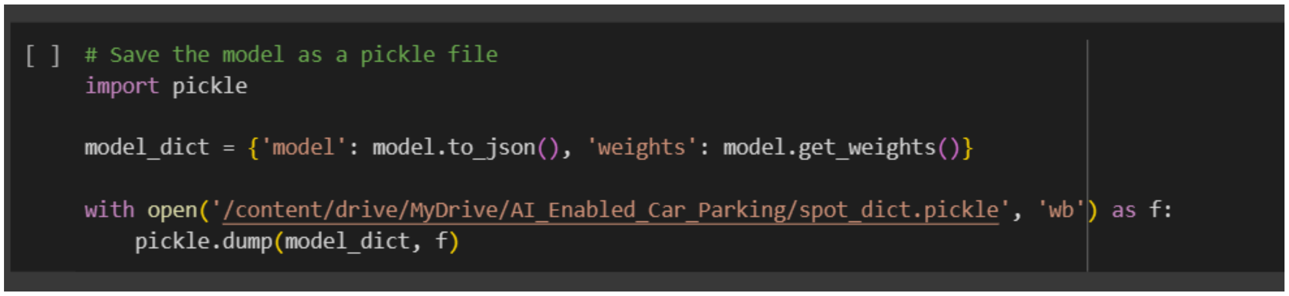
Fit the Model



```
WARNING:tensorflow:`period` argument is deprecated. Please use `save_freq` to specify the frequency in number of batches seen.
<ipython-input-11-c9c9e40c8e87>:5: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history_object = model.fit_generator(
Epoch 1/5
12/12 [==============================] - ETA: 0s - loss: 0.6159 - accuracy: 0.6903 WARNING:tensorflow:Can save best model only with val_acc available, skipping.
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
12/12 [==============================] - 265s 22s/step - loss: 0.6159 - accuracy: 0.6903 - val_loss: 0.3575 - val_accuracy: 0.8110
Epoch 2/5
12/12 [==============================] - ETA: 0s - loss: 0.2686 - accuracy: 0.8635 WARNING:tensorflow:Can save best model only with val_acc available, skipping.
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
12/12 [==============================] - 207s 17s/step - loss: 0.2686 - accuracy: 0.8635 - val_loss: 0.5826 - val_accuracy: 0.8598
Epoch 3/5
12/12 [==============================] - ETA: 0s - loss: 0.2327 - accuracy: 0.9134 WARNING:tensorflow:Can save best model only with val_acc available, skipping.
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
12/12 [==============================] - 203s 17s/step - loss: 0.2327 - accuracy: 0.9134 - val_loss: 0.3333 - val_accuracy: 0.8841
Epoch 4/5
12/12 [==============================] - ETA: 0s - loss: 0.1867 - accuracy: 0.9186 WARNING:tensorflow:Can save best model only with val_acc available, skipping.
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
12/12 [==============================] - 218s 18s/step - loss: 0.1867 - accuracy: 0.9186 - val_loss: 0.2368 - val_accuracy: 0.8963
Epoch 5/5
12/12 [==============================] - ETA: 0s - loss: 0.1423 - accuracy: 0.9423 WARNING:tensorflow:Can save best model only with val_acc available, skipping.
WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
12/12 [==============================] - 209s 18s/step - loss: 0.1423 - accuracy: 0.9423 - val_loss: 0.3345 - val_accuracy: 0.8963
```

Activity 6: Save the Model

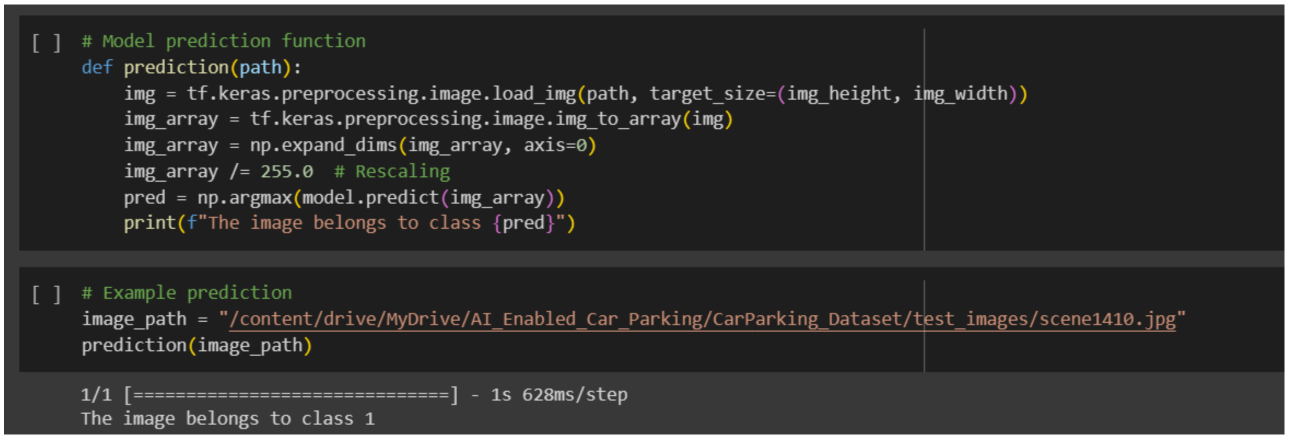
The model is saved with .h5 extension as follows
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



```
[ ] # Save the model as a pickle file
    import pickle

    model_dict = {'model': model.to_json(), 'weights': model.get_weights()}

    with open('/content/drive/MyDrive/AI_Enabled_Car_Parking/spot_dict.pickle', 'wb') as f:
        pickle.dump(model_dict, f)
```

Activity 7: Test The model

Evaluation is an integral phase in model development aimed at assessing whether the model is the optimal solution for the given problem and associated dataset. Load the saved model using load_model



```
[ ] # Model prediction function
    def prediction(path):
        img = tf.keras.preprocessing.image.load_img(path, target_size=(img_height, img_width))
        img_array = tf.keras.preprocessing.image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)
        img_array /= 255.0  # Rescaling
        pred = np.argmax(model.predict(img_array))
        print(f"The image belongs to class {pred}")
```

```
[ ] # Example prediction
    image_path = "/content/drive/MyDrive/AI_Enabled_Car_Parking/CarParking_Dataset/test_images/scene1410.jpg"
    prediction(image_path)

    1/1 [==============================] - 1s 628ms/step
    The image belongs to class 1
```