AI ENABLED CAR USING OPENCV

# Introduction:

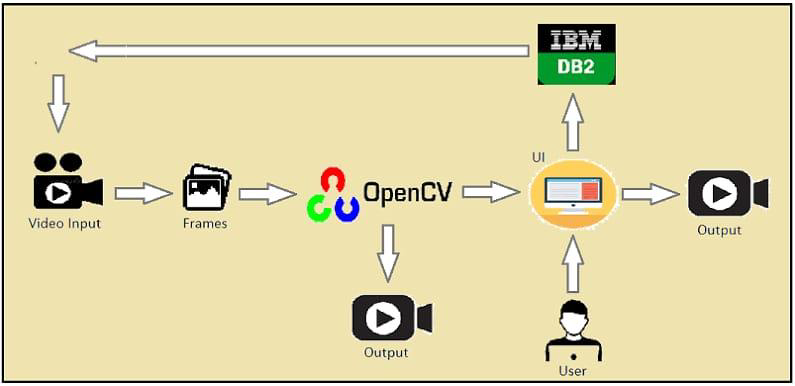
Over the next 25 years, the demand for parking facilities in urban areas is expected to surge as more people migrate to cities and car ownership continues to grow. With this rapid urbanization, the efficient management of parking spaces is becoming a significant challenge. Traditional methods of managing parking, such as manual attendants or static signage, are increasingly proving to be inefficient and outdated.

To address this growing issue and harness the power of modern technology, we are introducing an AI-enabled car parking system that utilizes OpenCV and advanced artificial intelligence algorithms. This system aims to revolutionize the way we park our vehicles by making the process more convenient, efficient, and user-friendly.

Just as the waste separation system you described is a response to the increasing challenges posed by waste accumulation, our AI-enabled car parking system addresses the pressing need for a smarter parking solution. By leveraging computer vision and AI technologies, this system will automate the entire parking process, from vehicle identification to space allocation. Gone will be the days of circling the parking lot in search of an available spot or dealing with human attendants.

This system's core functionality relies on OpenCV, a powerful open-source computer vision library, and sophisticated Convolutional Neural Networks (CNNs). It will accurately detect and recognize vehicles, assess available parking spaces, and guide drivers to their designated spots, reducing parking time and frustration. Furthermore, the system's adaptability will make it suitable for both indoor and outdoor parking facilities.

Technical 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. **Jupyter Notebook:** While you mentioned using Jupyter Notebook in Anaconda, you can also use it in Google Colab. Jupyter Notebook is an interactive environment that allows you to write and execute code in a notebook format, making it ideal for documenting and presenting your project.
6. **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.

# 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](https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add)

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

[**Flask Basics**](https://www.youtube.com/watch?v=lj4I_CvBnt0)

# 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 Hidden 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

# Project Structure:

Create a Project folder which contains files as shown below

* The Dataset folder contains the training and testing images for training our model.

**Milestone 1: Data Collection**

# There was a single image that was given to us . we have reframed it into occupies and empty from the single image that was given to us .

**Download the Dataset-** [**https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr**](https://drive.google.com/drive/folders/13vNq4XZLV15uxxXrVkj33Jr7VflrhKWr)

# Milestone 2: Image Preprocessing

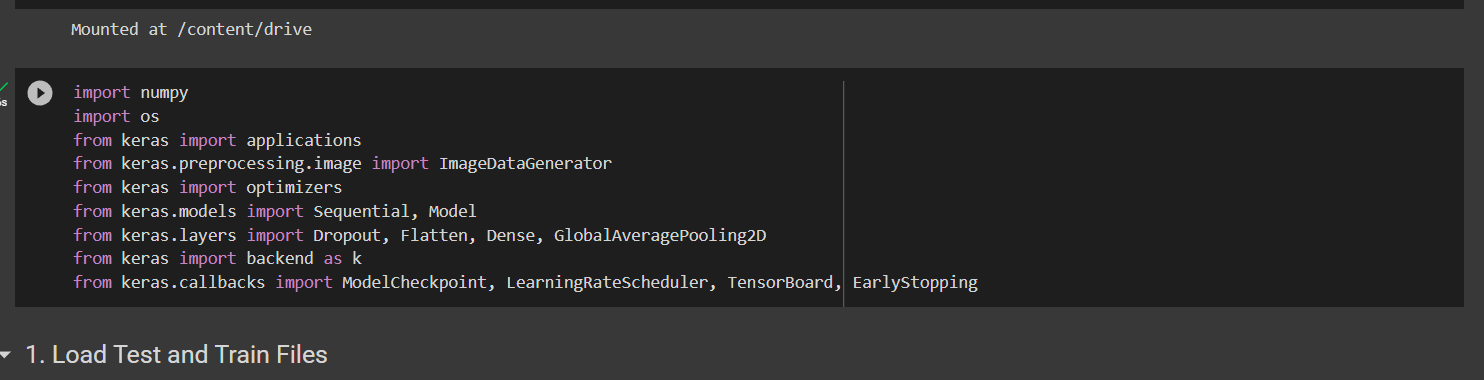
In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

# Activity 1:Import the ImageDataGenerator library

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

Let us import the ImageDataGenerator class from tensorflow Keras



# Activity 2: Configure ImageDataGenerator class

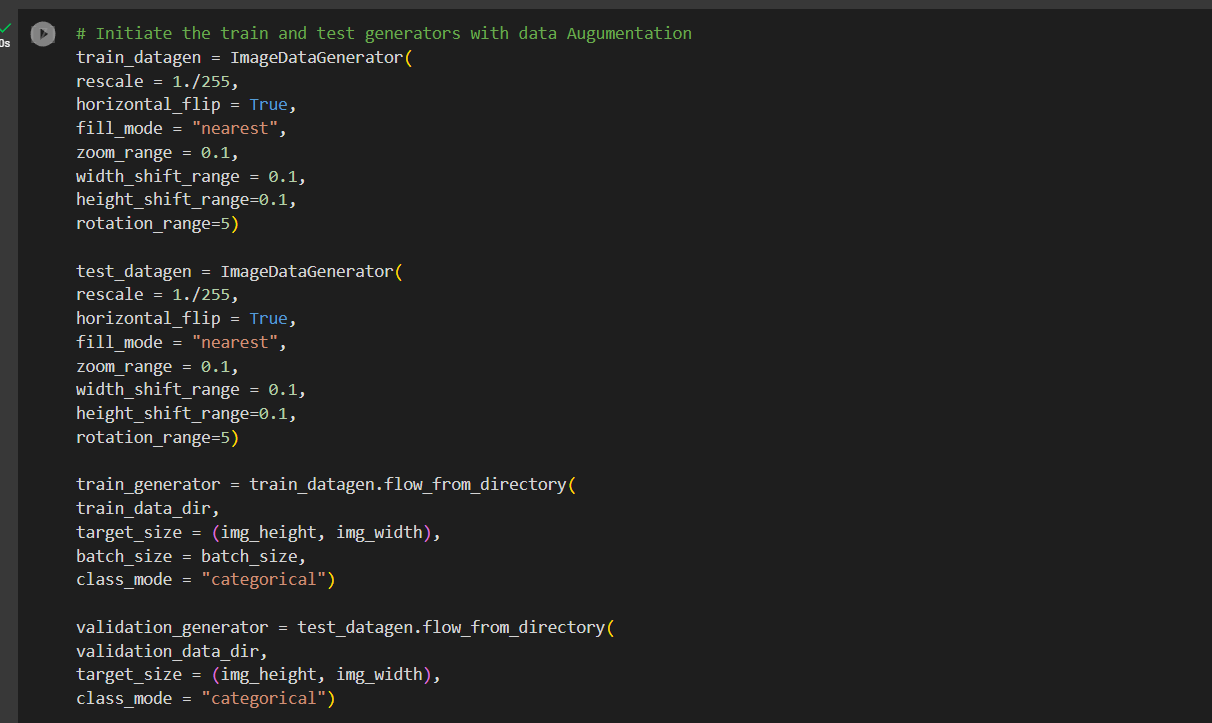
ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

There are five main types of data augmentation techniques for image data; specifically:

* Image shifts via the width\_shift\_range and height\_shift\_range arguments.
* The image flips via the horizontal\_flip and vertical\_flip arguments.
* Image rotations via the rotation\_range argument
* Image brightness via the brightness\_range argument.
* Image zoom via the zoom\_range argument.

An instance of the ImageDataGenerator class can be constructed for train and test.

Image Data Augmentation



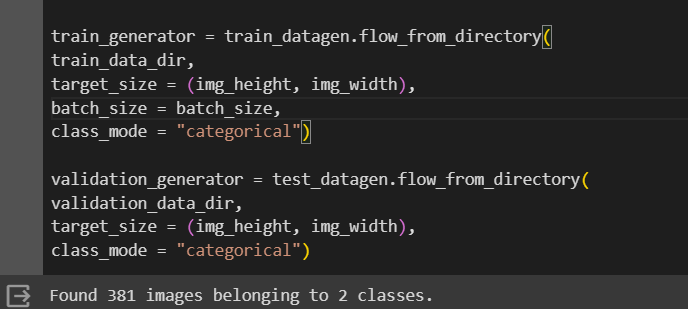
# Activity 3:Apply ImageDataGenerator functionality to Trainset and Testset

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code.For Training set using flow\_from\_directory function.

This function will return batches of images from the subdirectories occupied and empty from 0 to 1{occupied: 0, empty : 1 }

**Arguments:**

* directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
* batch\_size: Size of the batches of data which is 64.
* target\_size: Size to resize images after they are read from disk.
* class\_mode:
  + ‘int': means that the labels are encoded as integers (e.g. for sparse\_categorical\_crossentropy loss).
  + 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical\_crossentropy loss).
  + 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary\_crossentropy).
  + None (no labels).

**Loading Our Data And Performing Data Augmentation**

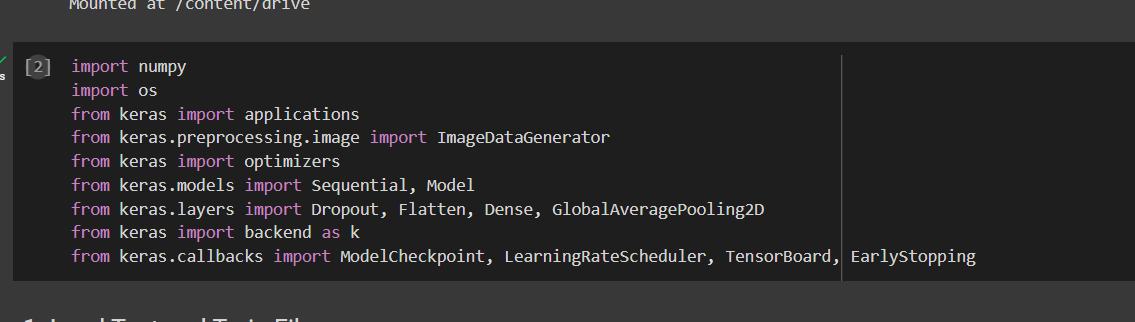
We notice that 381 images belong to 2 classes for training and 164 images belong to 2 classes for testing purposes.

# Milestone 3: Model Building

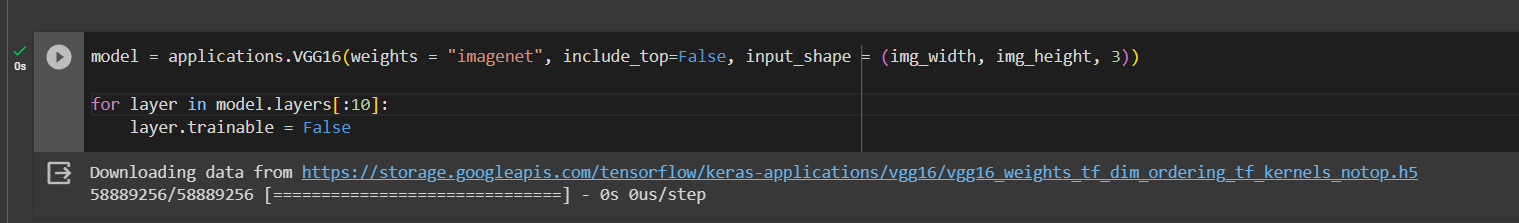
Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

# Activity 1: Importing the Model Building Libraries

Importing the necessary libraries



# Activity 2: Initializing the model

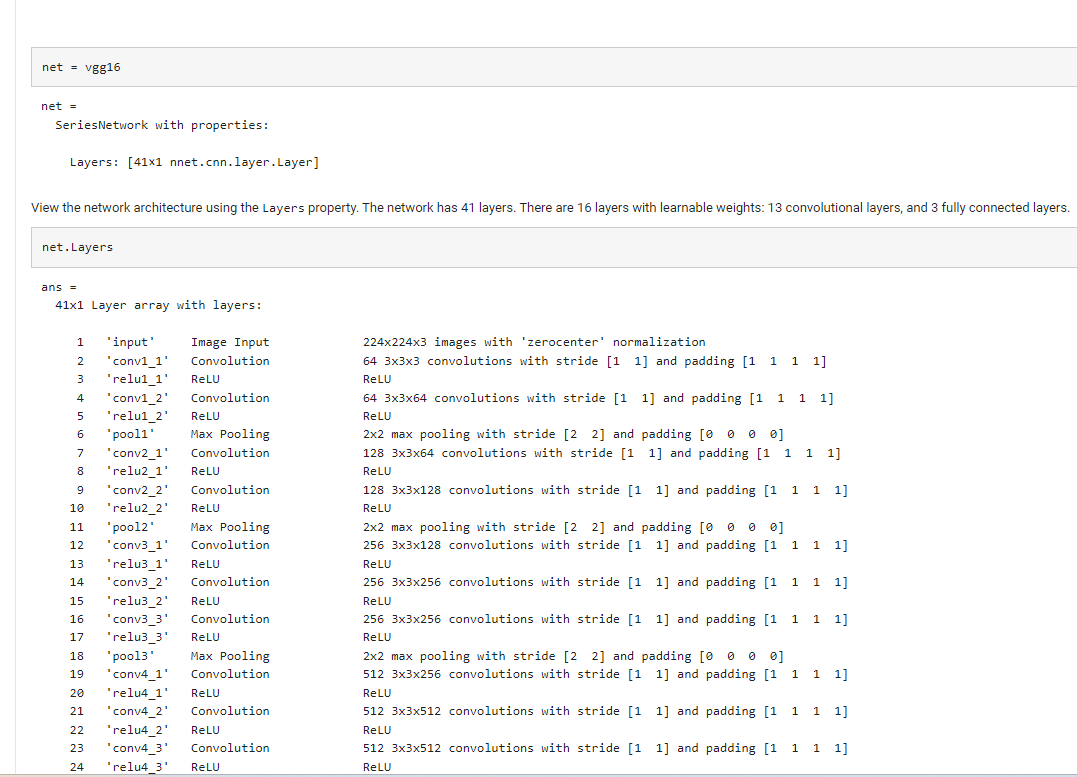


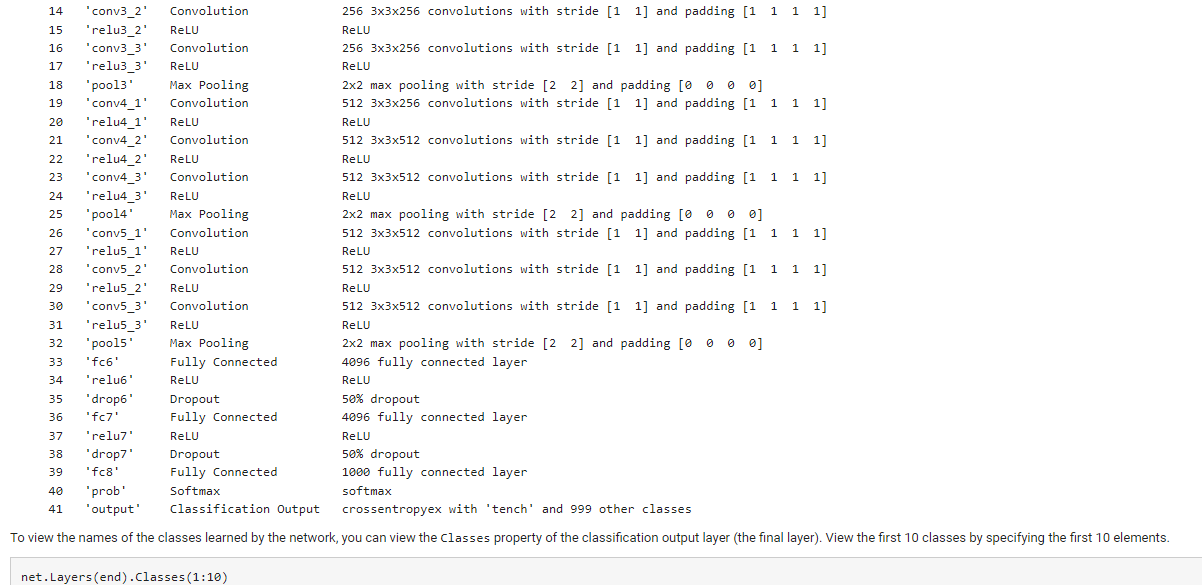
# Activity 3: Adding CNN Layers

* For information regarding CNN Layers refer to the link Link: <https://victorzhou.com/blog/intro-to-cnns-part-1/>
* As the input image contains three channels, we are specifying the input shape as (128,128,3).
* We are adding a convolution layer with activation function as “relu” and

with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.

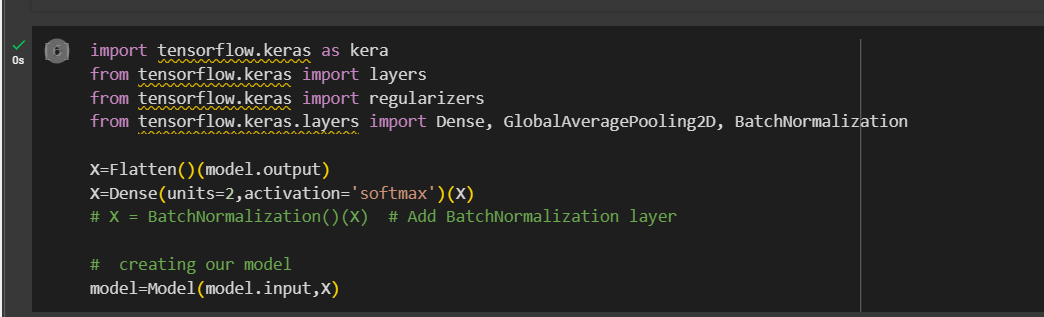
* Max pool layer is used to downsample the input.( Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter)
* Flatten layer flattens the input. Does not affect the batch size.





# Activity 5: Adding Dense Layers

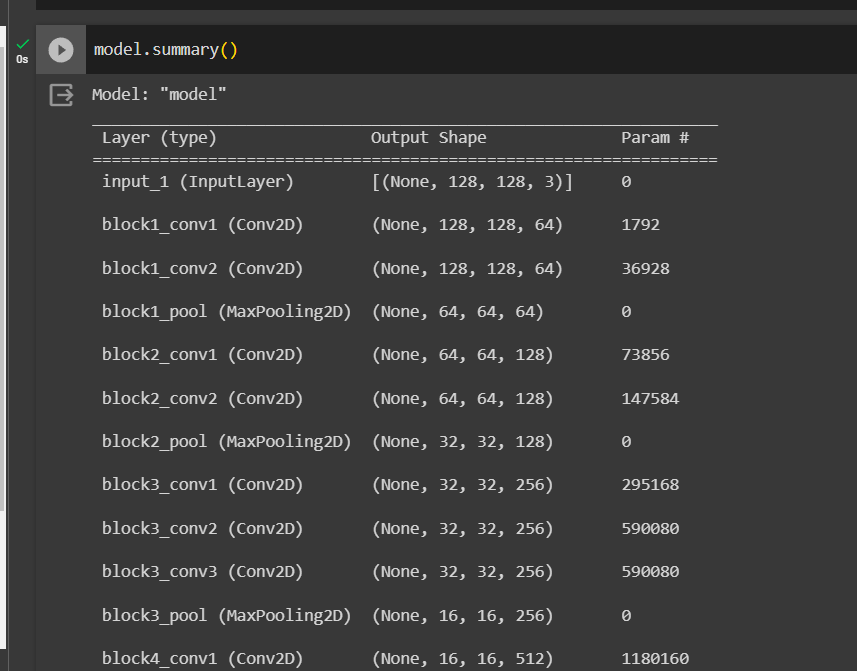
A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

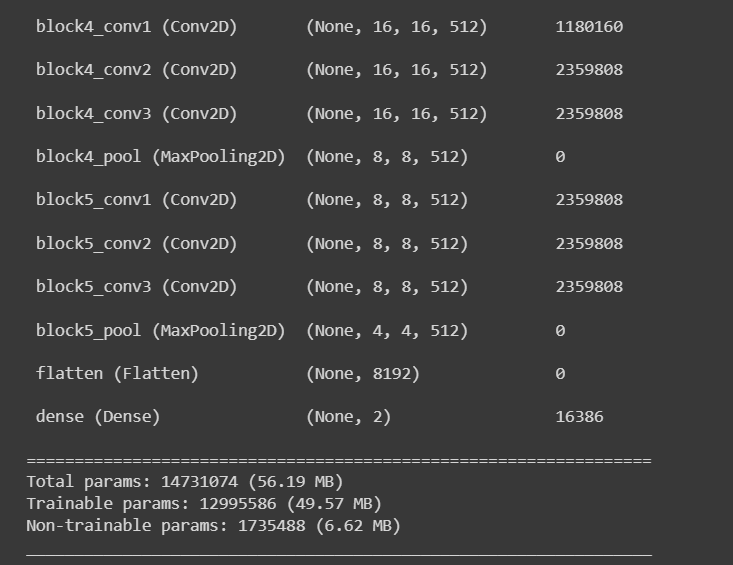


The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, summary to get the full information about the model and its layers.

**Summary of the Model**

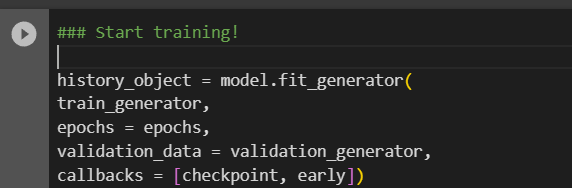




# Activity 6: Configure The Learning Process

* The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
* Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
* Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Compiling the Model



# Activity 7: Train The model

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

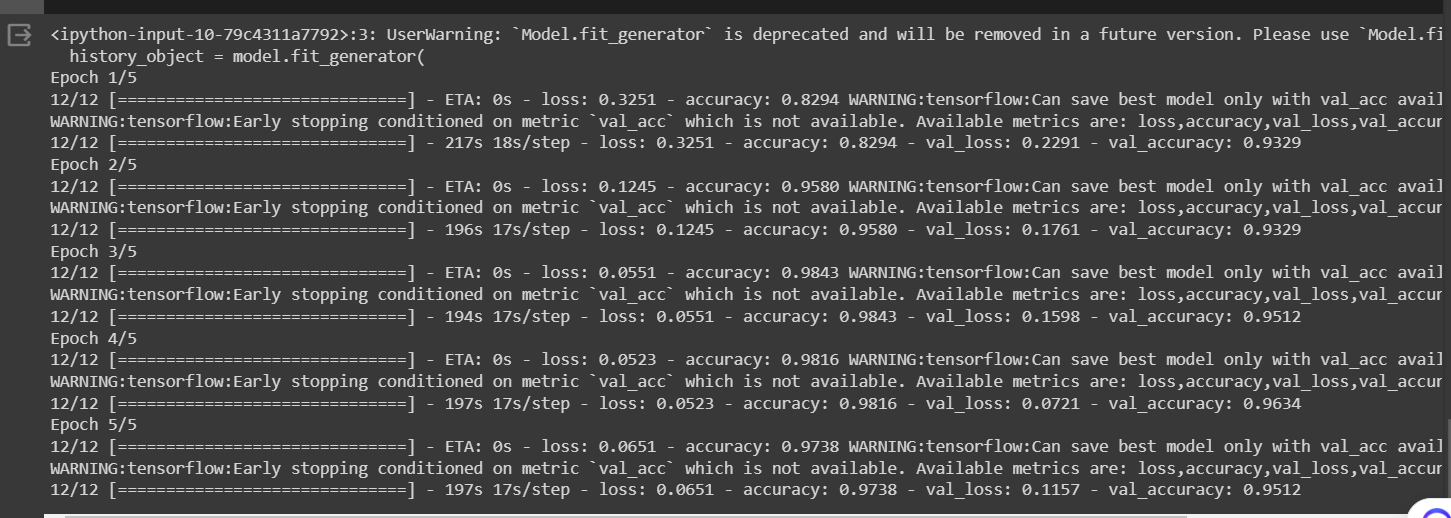
**fit\_generator** functions used to train a deep learning neural network

# Arguments:

* steps\_per\_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps\_per\_epoch as the total number of samples in your dataset divided by the batch size.
* Epochs: an integer and number of epochs we want to train our model for.
* validation\_data can be either:
  + an inputs and targets list
  + a generator
  + an inputs, targets, and sample\_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
* validation\_steps: only if the validation\_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is

stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.

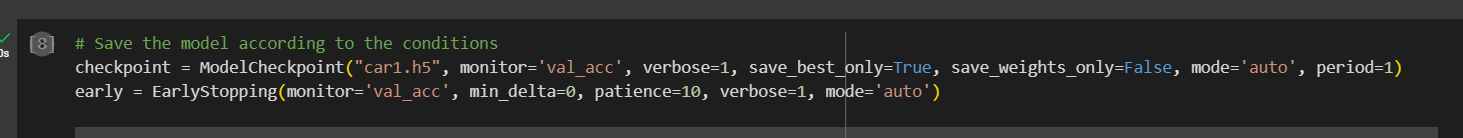
**Fit the Model**



# Activity 8: 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.



# Activity 9: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.

Load the saved model using load\_model

