LOVELY PROFESSIONAL UNIVERSITY

Vehicle Image Multiclass Classification Using Customized CNN and VGG16

Guided by and submited to: Sanjay Kumar Singh

Name: Priyansh Soniya
Reg no: 11711808
Email: priyanshsoniya@gmail.com
Mobile: 9425639042
Github repo: https://github.com/priyansh511/Vehicles-Classification

TensorFlow

Flask

Keras

Abstract

Deep learning is the sub branch of machine learning that involves creationg models using deep neural networks consisting of multiple layers of neurons each with designated weight and activation condition. Each layer has its own working task and contribution to final output.Deep learningis a main contributor behind driverless vehicles, allowing them to perceive a stop sign, or to recognize a walker from live video input. It is used to enable voice control in gadgets like smartphones, tablets, Tvs, etc. Deep learningis getting heaps of consideration of late and in light of current circumstances. It is accomplishing results that were unrealistic previously.

In deep learning, a computation model figures out how to perform arrangement functions from pictures, text, or sound. Deep learning models can accomplish cutting edge accuracy surpassing human-level execution. Models are prepared by utilizing an enormous arrangement of named information and neural organization designs that contain numerous layers.

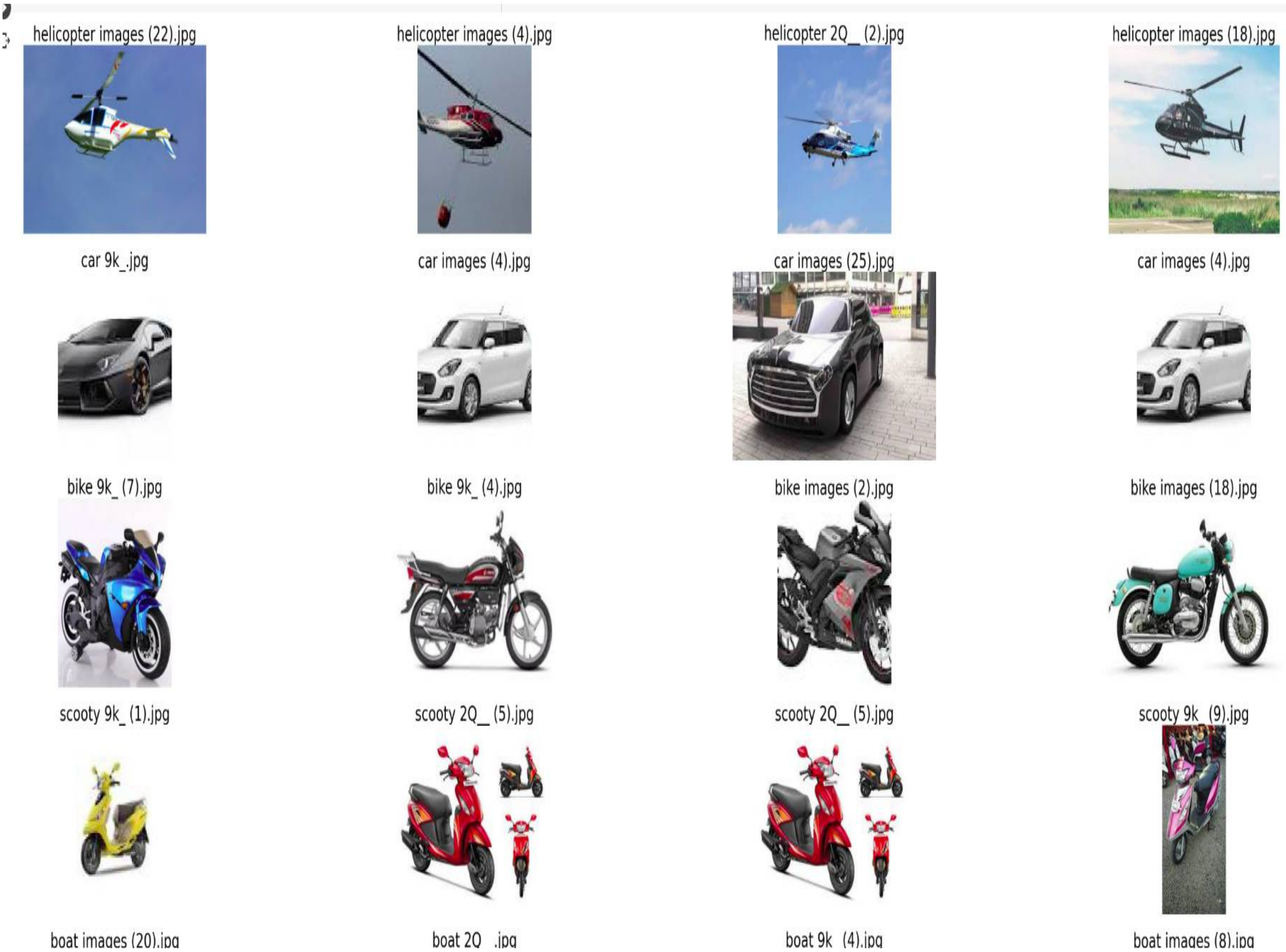
In this model we will create a custom convolutional neural network with the help of KERAS and TENSOFLOW and host it with using FLASK framework. The main aim is to make the different layers of the CNN identify and adapt to different features of vehicle images so that it can classify them into their respective classes.

INTRODUCTION

The aim of the model is to classify the images of vehicles into their respective classes. We are using convolutional neural network which contains dedicated layers to identify the different features of the images by applying certain weighted filters and thein compressing and highlighting more defined features from the image array. Next we will compare our model to a pretrained model VGG16 and pretrained imagenet weights and check the degree of accuracy the both achieve. The technology used in the model is Tensorflow version 2.3 and KERAS api. These technology allow us to build powerful and precise deep learning models easily and fast.

DATASET

The dataset is used is vehicles dataset from kaggle which is a raw dataset of multiple images of vehicles in 10 categories such as trains, cars, bikes, buses,etc. It is downloaded from kaggle and is colected from a google image downloading chrome extension. The dataset is devided into two parts Train and Test which consist of 574 and 72 images respectively.



Images before augmentation

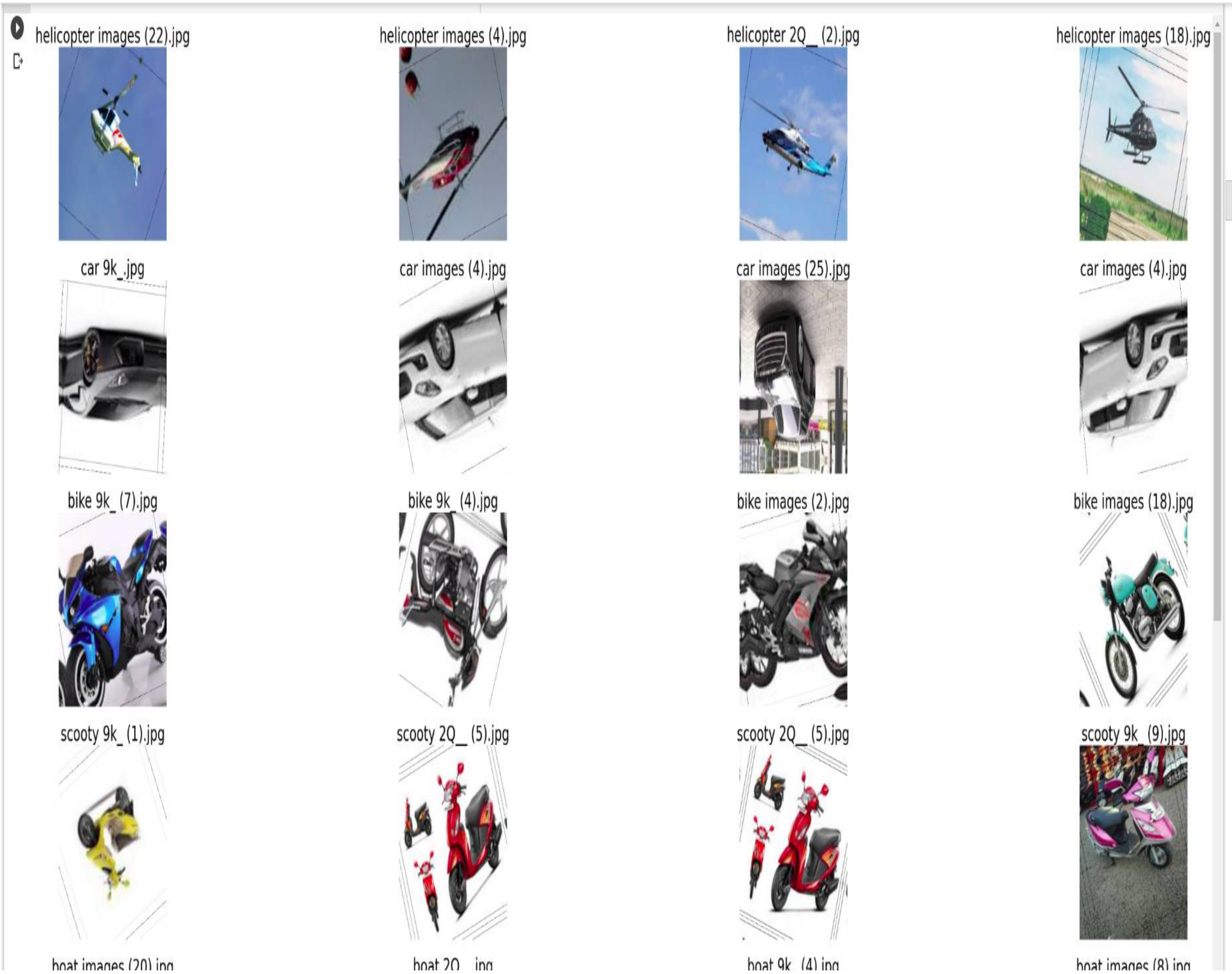
Preprocesing

The images of the dataset are of diferent sizes and shapes. So they need to be resized and rescaled. Also to make the model familiar with multiple aspects of images and reduce overfitting we introduce augmentation in images. Augmentaion is adding noise or enhancements in the image so that we get new images to train the model. The augmentation methods used are:

1. **Rescaling**: Deviding the image with the heighest pixel value (255) so as to have a uniform range of 0 to 1 in the image array.

2. **Resizing:** Resizing every image to a fixed size. (300 x 300)

3. **Random_rotation:** Rotating the image to the given factor.
4. **Random_crop:** Cropping image to a certain size.
5. **Random_Filp:** Randomly flip the image 180 degree vertically or horizontally or both.



Images after augmentation

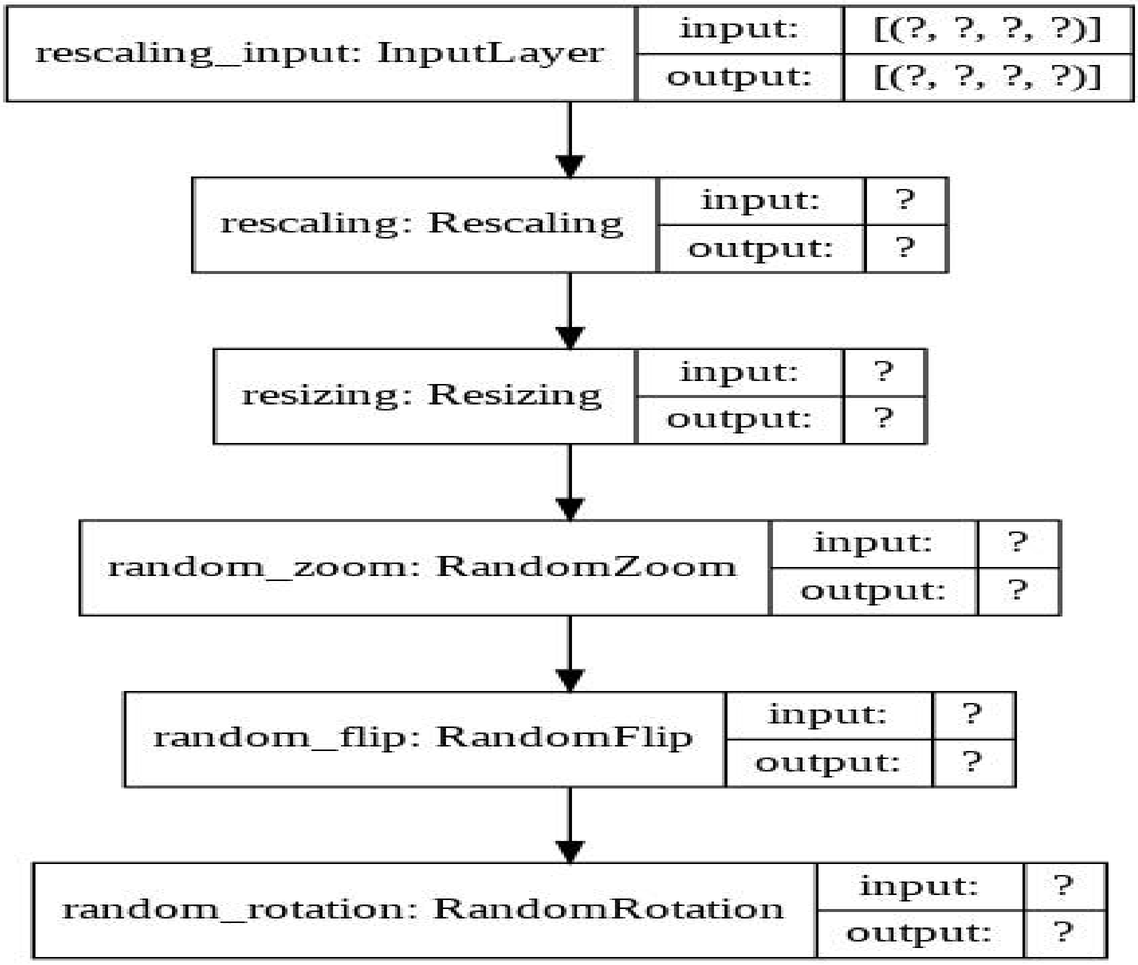
Custom CNN Model:

Instead of processing every pixel in the image array like deep neural networks, Convolutional Neural Networks works on finding features of the images like edges, shape, color variations etc.
Our custom CNN is built by two types of layes:
1. Conv2D: It process the image array by applying filters of provided size on the image and changing the image to theoutput value. Hence it detects and highlights the distinguishable features from image.

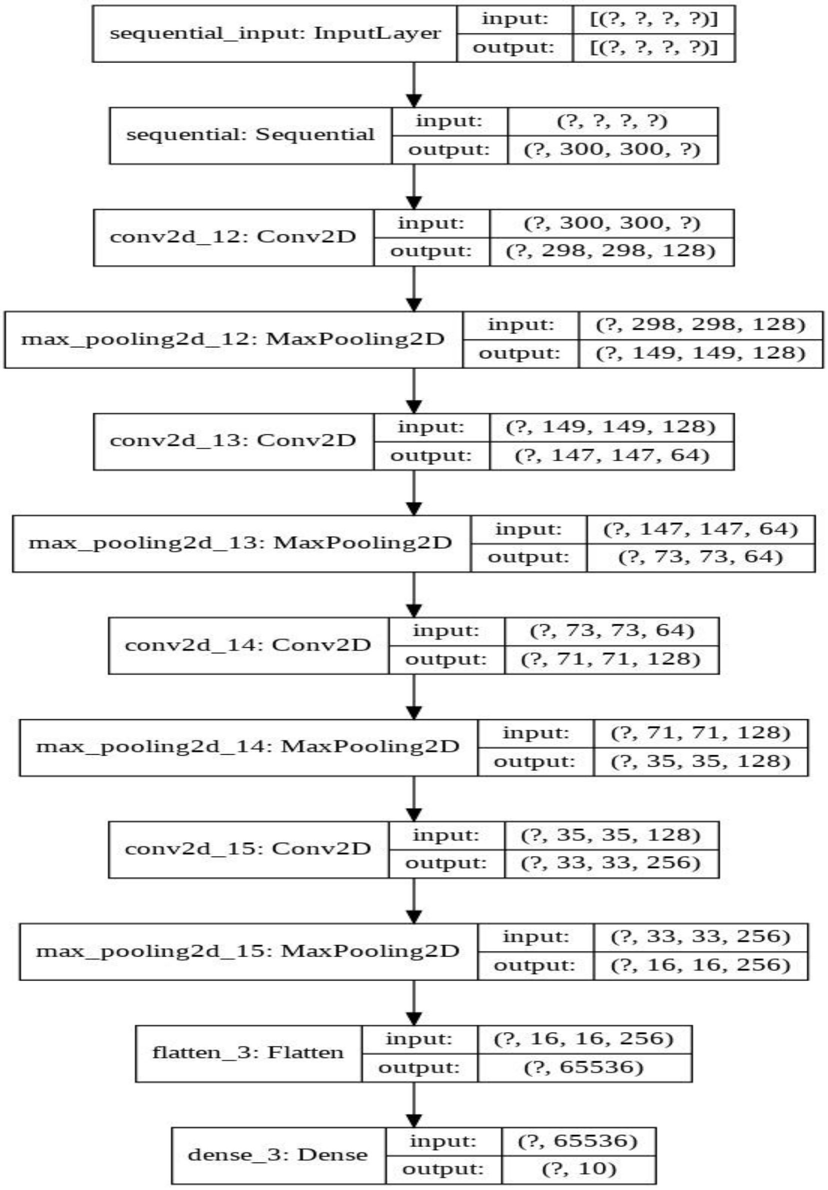
2. MaxPooling2D: It find the higest pixel from a given window size so as to enhance the image features and compress important aspects.

```
model_custom=keras.Sequential([

    augmenting,
    layers.Conv2D(128,(3,3),activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(64,(3,3),activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(128,(3,3),activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Conv2D(256,(3,3),activation='relu'),
    layers.MaxPooling2D(2,2),
    layers.Flatten(),
    layers.Dense(len(classes),activation='softmax')
])
```



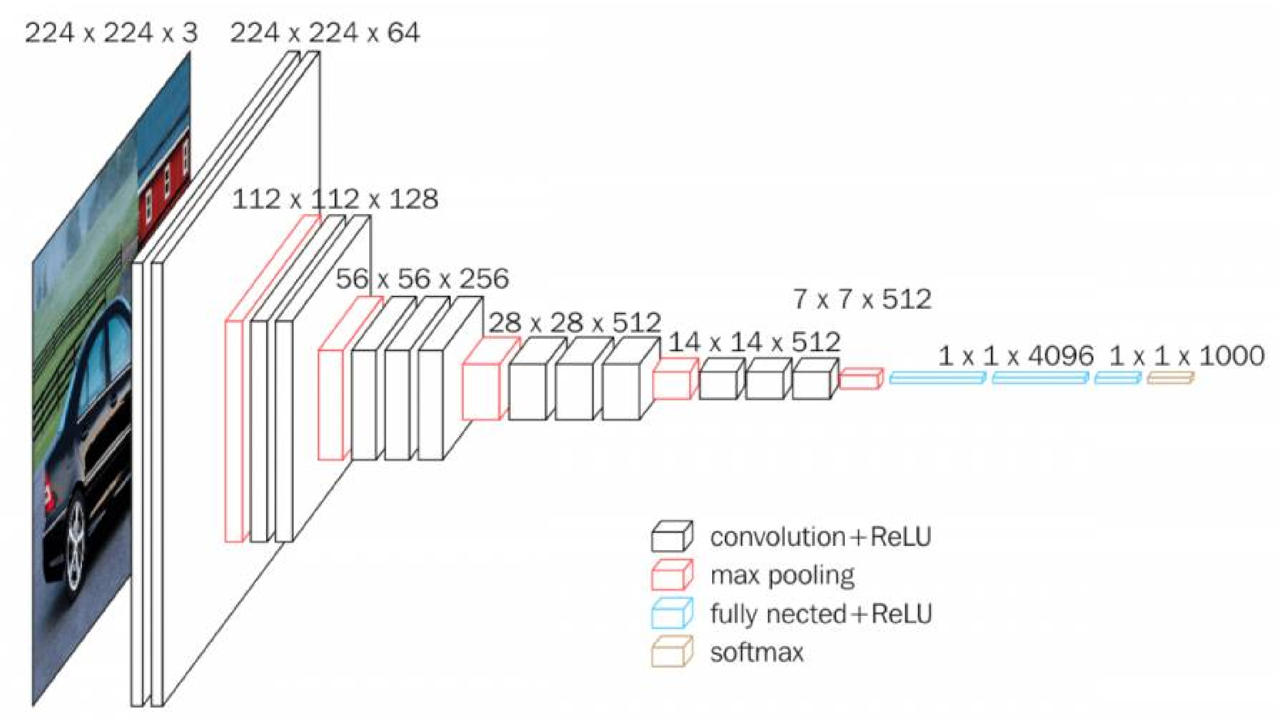
layers in augmentation model



layers in CNN model

VGG16

VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition".The model achieves 92.7% top-5 test accuracy in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes. The model is made of multiple Conv2D and MaxPooling2d layers and outputs in Dense layer.
We use it to check and compare the accuracy of our model.




```
model_vgg=keras.Sequential([
    augmenting,
    vgg,
    layers.Flatten(),
    layers.Dense(512,activation='relu'),
    layers.Dense(len(classes),
        activation='softma  x')
])
```

Deployment

The model is deployed using FLASK web framework which is a python based web development framework. It is v ery lightwieght and very efficient in making and deploying light web applications. It works on Jninja and werkzeug WSGI web development library.
Before deployment on the hosting server FLASK offers a pseudo hosting server using FLASK_ngrok library that very helpful in development checking and building app.

```
!pip install flask-ngrok

from flask import Flask, render_template, request
from flask_ngrok import run_with_ngrok
app =
Flask(__name__,template_folder='/content/templates')
run_with_ngrok(app)

@app.route('/', methods=['POST','GET'])
def index():
if request.method=='POST':
x=request.files['image']
print(x)
x=numpy.asarray(x)
img__=tf.expand_dims(x,0)
res1=predict_custom(img__)
res2=preidict_vgg(img__)
data-{img:x,res1:res1,res2:res2}
return render_template("predict.html",)
else:
return render_template('index.html')

@app.route('/about',methods=['POST','GET'])
def hello():
retuen render_template("about.html")


app.run()
```

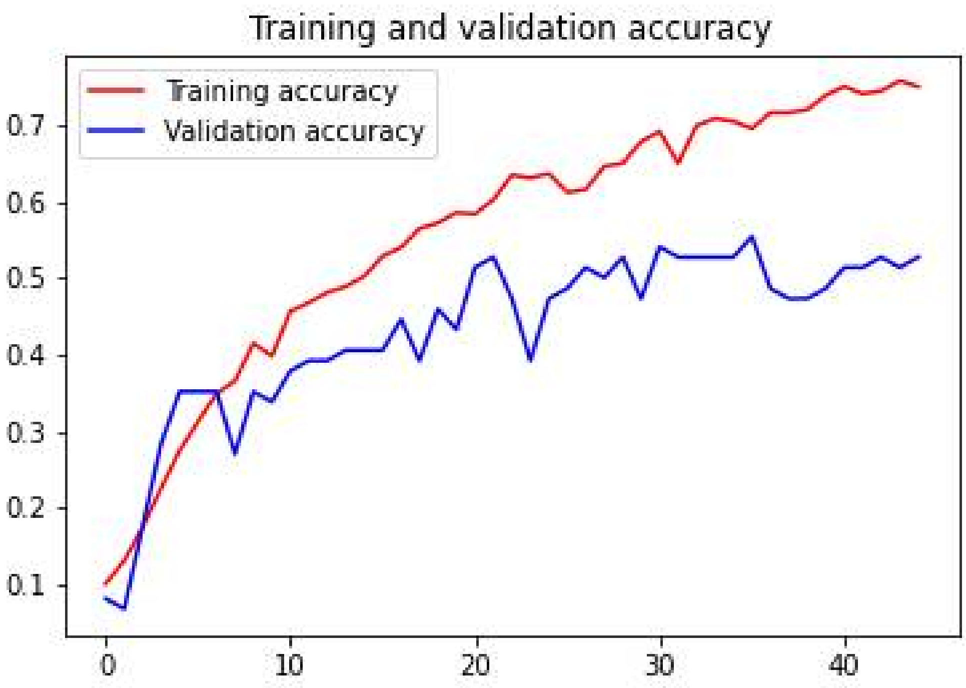
Results

Custom model:

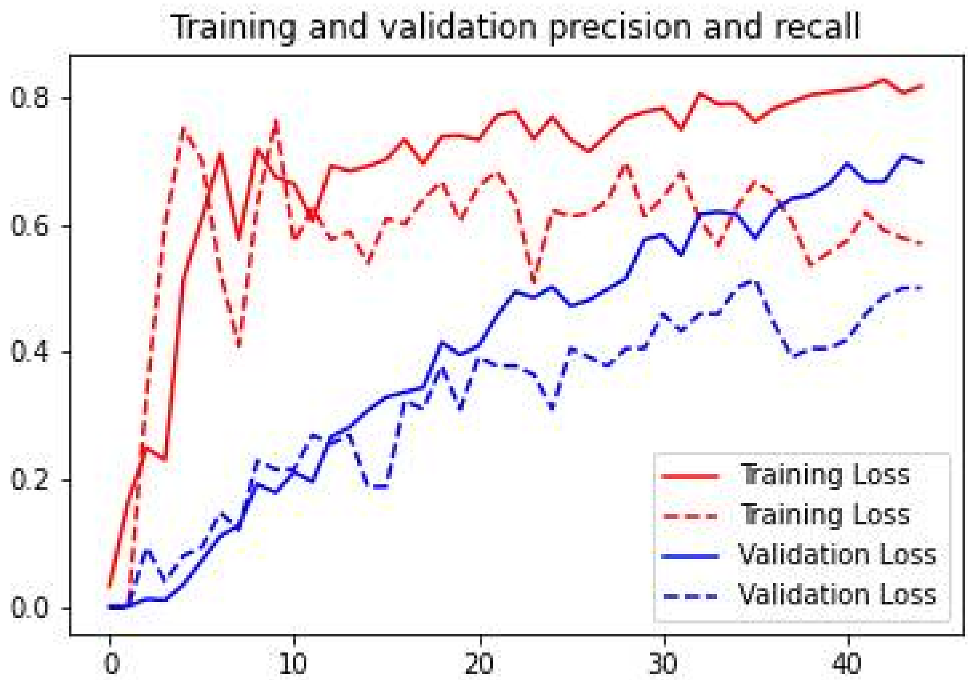
```
print('accuracy',acc[-1])
print('val_accuracy',val_acc[-1])
print('precision',precision[-1])
print('val_precision',val_precision[-1])
print('recall',recall[-1])
print('val_recall',val_recall[-1])
model_custom.evaluate_generator(val_generator)
```

```
accuracy 0.75
val_accuracy 0.5270270109176636
precision 0.8159645199775696
val_precision 0.5692307949066162
recall 0.6969696879386902
val_recall 0.5
[2.5282795429229736, 0.5270270109176636, 0.5692307949066162, 0.5]
```

train accuracy (best) 82%
val acuracy (best) 61%
precision 81%
recall 69%



train accuracy vs val accuracy



train precision,train recall vs val precision, val recall

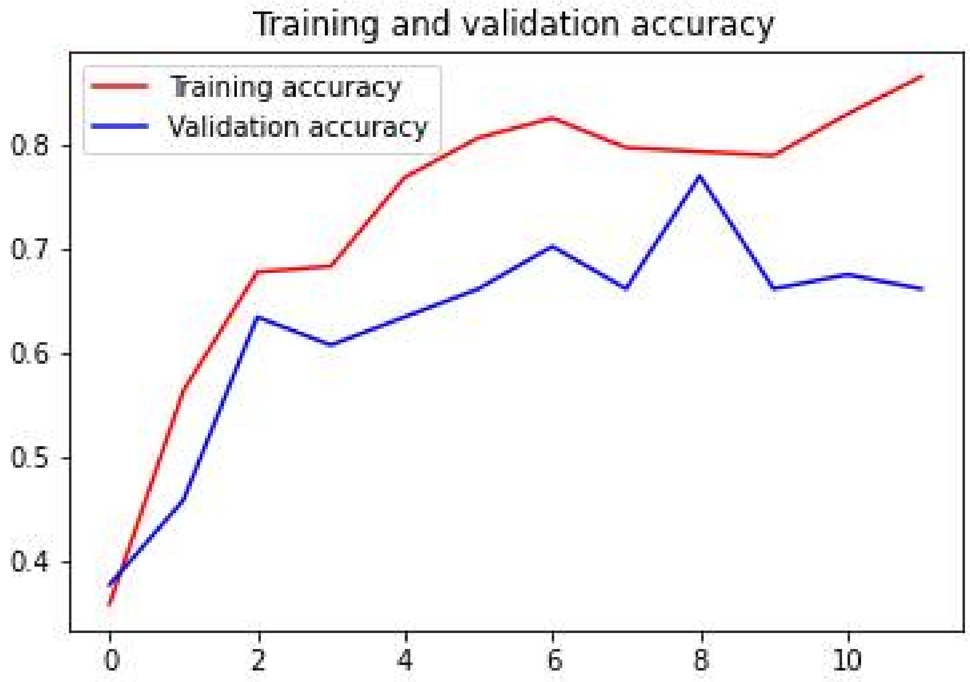
Even after 45 eppochs the model only gives around 85% training accuracy and 60% validation accuracy. After around 40 epochs it is seen that the validation accuracy deos not increase but validation loss started increasing that suggests that the model started overfitting.

VGG16:

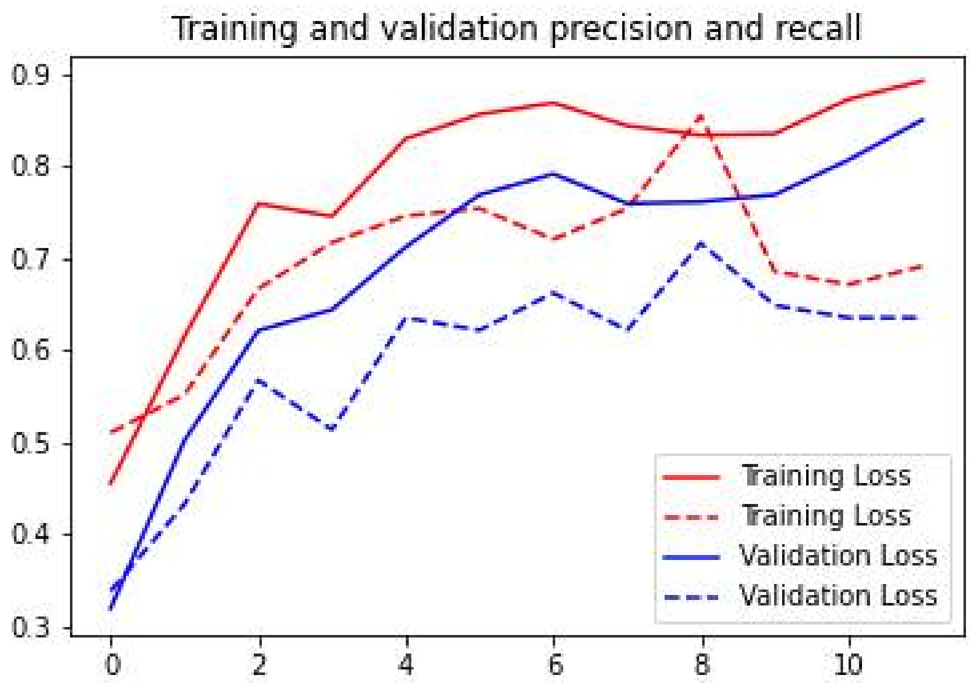
```
print('accuracy',acc[-1])
print('val_accuracy',val_acc[-1])
print('precision',precision[-1])
print('val_precision',val_precision[-1])
print('recall',recall[-1])
print('val_recall',val_recall[-1])
model_vgg.evaluate_generator(val_generator)
```

```
accuracy 0.8655303120613098
val_accuracy 0.662162184715271
precision 0.8926441073417664
val_precision 0.6911764740943909
recall 0.8503788113594055
val_recall 0.6351351141929626
[1.2841216325759888, 0.662162184715271, 0.6911764740943909, 0.6351351141929626]
```

train accuracy (best) 88%
val acuracy (best) 76%
precision 90%
recall 85%



VGG16 train vs val accuracy



The VGG16 model gives around 90% train accuracy and around 75% validation accuracy in around 10 epochs. Then it starts overfitting.