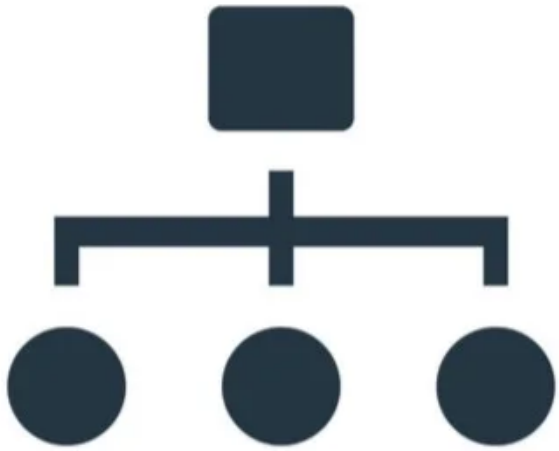
Shreya Malraju Aug 29

Classifying Airplanes,Motorbikes and Schooners - Multiclass Image Classifier



Referred from: https://www.vectorstock.com/royalty-free-vector/classification-icon-simple-element-vector-27025994

INTRODUCTION

The aim of this project is to build an image classifier that categorize images of airplanes,motorbikes and schooners into their respective classes.

Major part of my code is referred from Kaggle-

https://www.kaggle.com/code/maricinnamon/multiclass-classification-caltech101-tensorflow

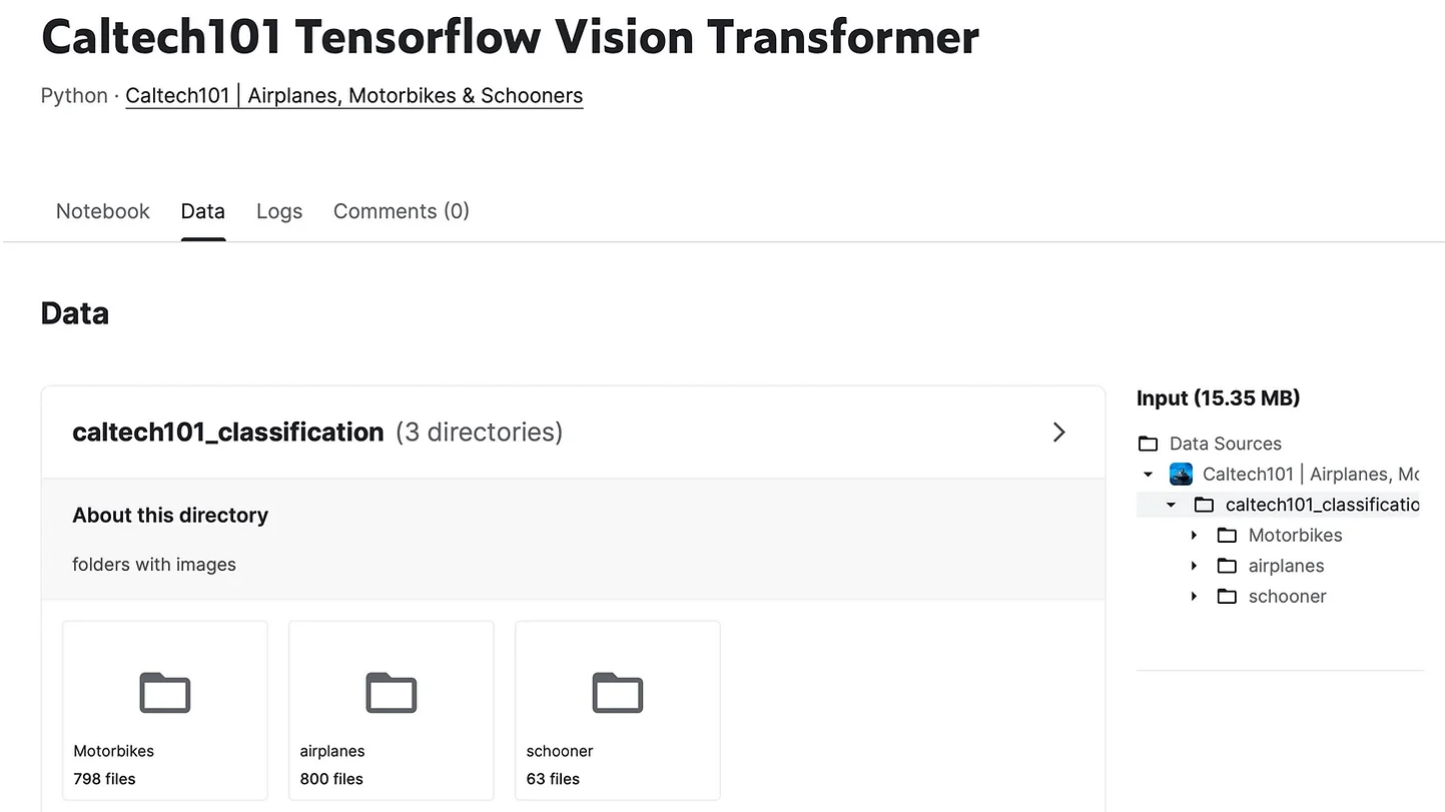
I tried to improve the performance of the above model by experimenting in 2 ways.

DATASET

The dataset [1] is obtained from kaggle https://www.kaggle.com/datasets/maricinnamon/caltech101-airplanes-motorbikes-schooners

The dataset consists of 3 folders representing

1. Motorcycles
2. Schooners
3. Airplanes



PROJECT DESIGN

The project is divided into 7 phases namely,

1. Libraries and Variables
2. Data
3. Pre-processing
4. Neural Network Architecture - VGG16
5. Training and Saving the best model
6. Visualizing the obtained results in terms of Loss and Accuracy
7. Testing

Initial Step - Load the data into the runtime. For that, we first need to mount the drive where we are using the google colab. We use opendatasets module of python to directly load the data from the Kaggle by providing the kaggle dataset link and authorizing with our username and key.

```
[1] #First, we need to import dataset from drive

    from google.colab import drive

    drive.mount('/content/drive')

    # Authorization is required for the drive to be mounted

    Mounted at /content/drive
```

```
[2] pip install opendatasets

    Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
    Collecting opendatasets
      Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
    Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from opendatasets) (4.64.1)
    Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from opendatasets) (7.1.2)
    Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (from opendatasets) (1.5.12)
    Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (1.15.0)
    Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2022.9.24)
    Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2.8.2)
    Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (1.24.3)
    Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (6.1.2)
    Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle->opendatasets) (2.23.0)
    Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle->opendatasets) (1.3)
    Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (2.10)
    Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle->opendatasets) (3.0.4)
    Installing collected packages: opendatasets
    Successfully installed opendatasets-0.1.22
```

```
[3] import opendatasets

    opendatasets.download("https://www.kaggle.com/datasets/maricinnamon/caltech101-airplanes-motorbikes-schooners")

    # The above link is to provide dataset from kaggle so that we can import the data into our drive
    # We have to provide our Kaggle credentials and key to provide access and hence data is downloaded

    Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
    Your Kaggle username: shreyamalraju
    Your Kaggle Key: ·····
    Downloading caltech101-airplanes-motorbikes-schooners.zip to ./caltech101-airplanes-motorbikes-schooners
    100%|██████████| 14.5M/14.5M [00:00<00:00, 205MB/s]
```

The data is downloaded into the runtime, hence we can start working on different phases.

Libraries and Variables

First, we shall import some modules that we will be using for processing the images and building the neural network architecture.

```
[4] # Import all the required libraries

    import random
    import datetime
    import cv2
    import imutils
    import numpy as np
    import os
    from sklearn.model_selection import train_test_split
    import matplotlib.pyplot as plt
    from tensorflow.keras.layers import Dense
    from tensorflow.keras.callbacks import ModelCheckpoint
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import Flatten
    from tensorflow.keras.layers import Input
    from tensorflow.keras.optimizers import Adam
    from tensorflow.keras.preprocessing.image import load_img
    from tensorflow.keras.preprocessing.image import img_to_array
    from tensorflow.keras.applications import VGG16
    from tensorflow.keras.models import load_model
    from tensorflow.keras.utils import to_categorical
    from tensorflow.keras.layers import Dropout
    from sklearn.preprocessing import LabelBinarizer
```

Here, we are using modules like os, numpy, tensorflow, matplotlib, keras [3]etc

Now, we define variables for storing the data, labels and classes along with the location of the images in the drive.

```
[5] data = []
    labels = []
    imagePaths = []

    images_path ="/content/drive/MyDrive/caltech101_classification/"

    classes = ["Motorbikes", "airplanes", "schooner"]
```

Data

The class_counter function counts the number of classes to just make sure only 3 classes are created in the furthur process.

```
[6] def class_counter(labels, class_name):
        c = 0
        for label in labels:
            if label == class_name:
                c += 1
        return c
```

```
[7] for cl in classes:
        images_list = []

        path_new = images_path + "/" + cl + "/"

        # get the list of the available images
        for image in os.listdir(path_new):
            # get only images that
            # are located in folder
            if (image.endswith(".jpg")):
                images_list.append(image)

        # sort image_path in ascending order
        images_list = sorted(images_list)

        # loop over the images
        for img in images_list:
            label = cl

            image_path = os.path.sep.join([images_path, cl, img])
            image = cv2.imread(image_path)
            (h, w) = image.shape[:2]

            # load the image
            image = load_img(image_path, target_size=(224, 224))
            image = img_to_array(image)

            data.append(image)
            labels.append(label)
            imagePaths.append(image_path)
```

We then loop over the images in the folder to append them into data, labels, imagepaths.

```
[57] counter_mtb = class_counter(labels, "Motorbikes")
     counter_arp = class_counter(labels, "airplanes")
     counter_sch = class_counter(labels, "schooner")

     print(counter_sch, counter_arp, counter_mtb)

     max_number = max(counter_mtb, counter_arp, counter_sch)
     max_number

     63 800 798
     800
```

In the above code, we calculate the number of images in each class and notice that Schooner class has 63 images and Airplanes have 800 wheras Motorbikes have 798.

We see that 63 schooners are not enough to perform the training and obtain best results, hence we need to artificially expand the dataset by augmenting the existing data. Hence we should perform scaling and rotation to increase images to 800. We also augment motorbikes so that all the 3 classes have equal number of images.

```
[9]  def do_scale(img):
         # scale range
         scale_val = random.uniform(0.8, 1.2)
         iScaled = cv2.resize(img.copy(),
                              None,
                              fx=scale_val,
                              fy=scale_val)

         return iScaled
```

```
[10] def do_rotate(img):
         (h, w) = img.shape[:2]

         # degrees range
         rotate_val = random.uniform(-5, 5)

         # image center
         center = (w / 2, h / 2)

         # Rotation Matrix
         M = cv2.getRotationMatrix2D(center,
                                     rotate_val,
                                     scale=1)

         iRotated = cv2.warpAffine(img.copy(),
                                   M,
                                   (w, h))
         return iRotated
```

This piece of code performs scaling and rotating of the images.

Scaling - used to resize the images, so we can create multiple copies of the same images with different size.

Rotation - We rotate the images to save multiple copies of the same image in different angles and hence we can expand the dataset.

The below code performs augmentation. Augmentation[6] is done on the existing images. For loop is applied on every image and we scale and rotate every image until the count is equal to the maximum images of a class.

```
def augment_data(counter, max_number, class_name):

    # while we don't have a lot of images
    while counter < max_number:
        # loop through each image in list

        for img in data:
            # check the number of images again

            if counter < max_number:
                # make scaling
                imgAug = img.copy()
                imgAug = do_scale(imgAug)

                # temporary save the new image
                cv2.imwrite("imgAug.jpg", imgAug)

                # load the new image
                imgAug = load_img("imgAug.jpg", target_size=(224, 224))
                imgAug = img_to_array(imgAug)

                # delete it from memory
                os.remove("imgAug.jpg")

                # add new image, it's label and path
                data.append(imgAug)
                labels.append(class_name)
                imagePaths.append(image_path)

                # recalculate a counter
                counter = class_counter(labels, class_name)
            else:
                break
```

```
        # make rotating
        if counter < max_number:
            imgAug = img.copy()
            imgAug = do_rotate(imgAug)

            # temporary save the new image
            cv2.imwrite("imgAug.jpg", imgAug)

            # load the new image
            imgAug = load_img("imgAug.jpg", target_size=(224, 224))
            imgAug = img_to_array(imgAug)

            # delete it from memory
            os.remove("imgAug.jpg")

            # add new image and it's label and path
            data.append(imgAug)
            labels.append(class_name)
            imagePaths.append(image_path)

            # recalculate a counter
            counter = class_counter(labels, class_name)
        else:
            break
```

We augment motorbikes and schooner classes to make the images equal to the number of images in airplanes class.

```
[12] augment_data(counter_mtb, max_number, "Motorbikes")
     augment_data(counter_sch, max_number, "schooner")
```

After the classes are augmented, we see that there are 800 images in each class.

Pre-Processing

First, we have to normalize the data i.e., change the range from [0,255] to [0,1]

```
[ ] data = np.array(data, dtype="float32") / 255.0

[ ] labels = np.array(labels)
    imagePaths = np.array(imagePaths)

[ ] lb = LabelBinarizer()
    labels = lb.fit_transform(labels)

[ ] split = train_test_split(data,
                             labels,
                             imagePaths,
                             test_size=0.05,
                             random_state=42)

[ ] (trainImages, testImages) = split[:2]
    (trainLabels, testLabels) = split[2:4]
    (trainPaths, testPaths) = split[4:]

[ ] f = open("testing_multiclass.txt", "w")
    f.write("\n".join(testPaths))
    f.close()
```

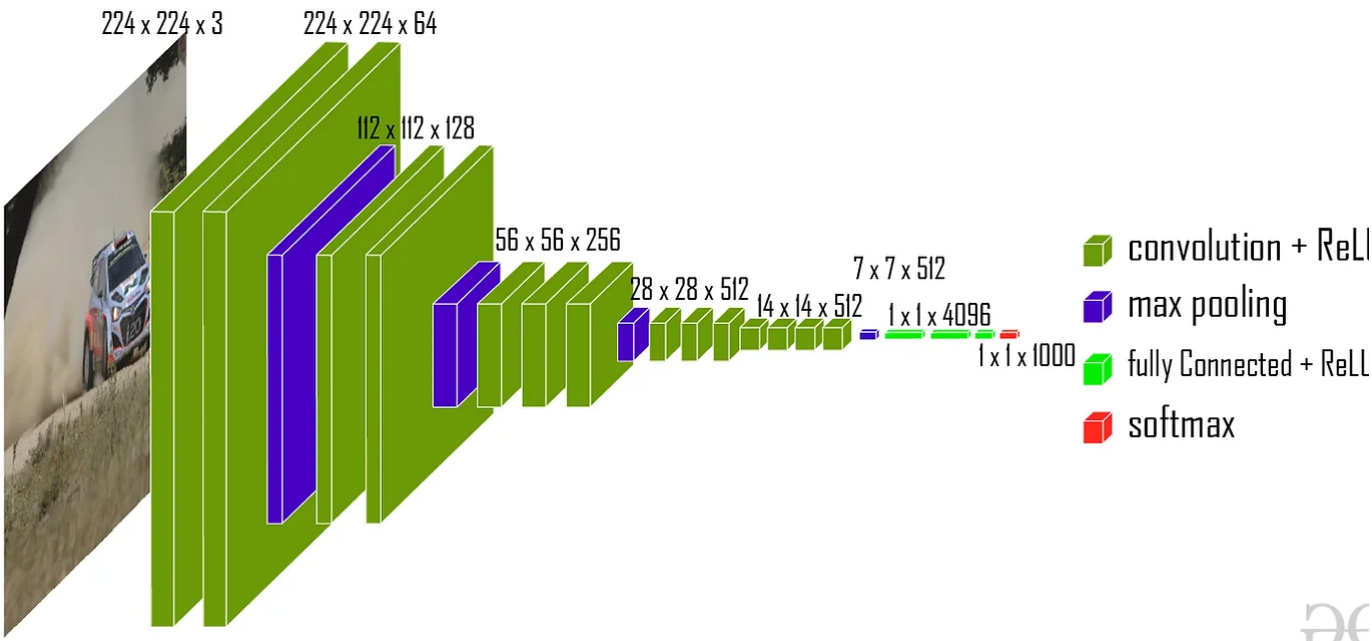
We then convert labels and imagepaths into numpy arrays and later, we convert class labels into encoding.

We then split the data into train and test data. The whole dataset is divided into 95% training and 5% testing. And then unpack the split variable into other variables like train and test images,labels and paths.

We save the names of the test images in a text file to perform testing on the NN later.

Neural Network Architecture

In this project, VGG16[2] neural network is used. Basically, VGG16 is a 16 layer deep neural network.



Referred from : https://www.geeksforgeeks.org/vgg-16-cnn-model/

VGG16[5] algorithm is very much efficient even upto classifying the images in 1000 classes.

```
[ ]  vgg = VGG16(weights="imagenet",
                 include_top=False,
                 input_tensor=Input(shape=(224, 224, 3)))

     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

[ ]  vgg.trainable = False

[ ]  flatten = vgg.output
     flatten = Flatten()(flatten)

[ ]  softmaxHead = Dense(512, activation="relu")(flatten)
     softmaxHead = Dropout(0.5)(softmaxHead)
     softmaxHead = Dense(512, activation="relu")(softmaxHead)
     softmaxHead = Dropout(0.5)(softmaxHead)

     softmaxHead = Dense(len(lb.classes_),
                         activation="softmax",
                         name="class_label")(softmaxHead)

[ ]  model = Model(
         inputs=vgg.input,
         outputs=(softmaxHead))
```

We freeze all the layers of VGG to prevent from training and we flatten the max-pooling layer which is the output of the vgg. We use softmax activation function to classify the images.

We then have to add this output to the model.

Define hyperparameters like learning rate, number of epochs, and batch size.

```
[ ]  INIT_LR = 1e-4
     NUM_EPOCHS = 9
     BATCH_SIZE = 32

[ ]  losses = {
         "class_label": "categorical_crossentropy",
     }

[ ]  trainTargets = {
         "class_label": trainLabels,
     }

[ ]  testTargets = {
         "class_label": testLabels,
     }
```

After that, to set the loss method, we must define the dictionary and also for target training and testing output.

Training and Saving the best model

```
[ ]  model_path = "model.h5"

     model_checkpoint_callback = ModelCheckpoint(
         filepath=model_path,
         monitor='val_accuracy',
         mode='max',
         save_best_only=True)
```

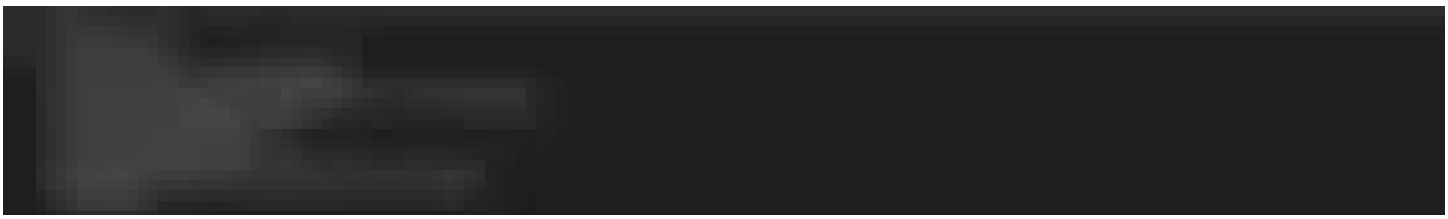
```
[ ]  opt = Adam(INIT_LR)

     model.compile(loss=losses,
                   optimizer=opt,
                   metrics=["accuracy"])

     print(model.summary())
```

Here, we must save the best model from all the epochs and also we compile the model and obtain the model summary.

We then perform training on the model.



The following is the result obtained -

We see that the best accuracy is obtained from 8th epoch and it is 93.33 percent.

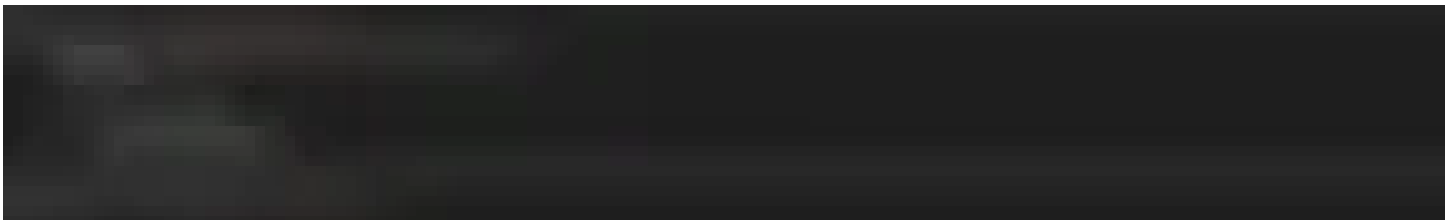
Visualizing the obtained results in terms of Loss and Accuracy

We plot the graph between training loss and validation loss and obtain the following

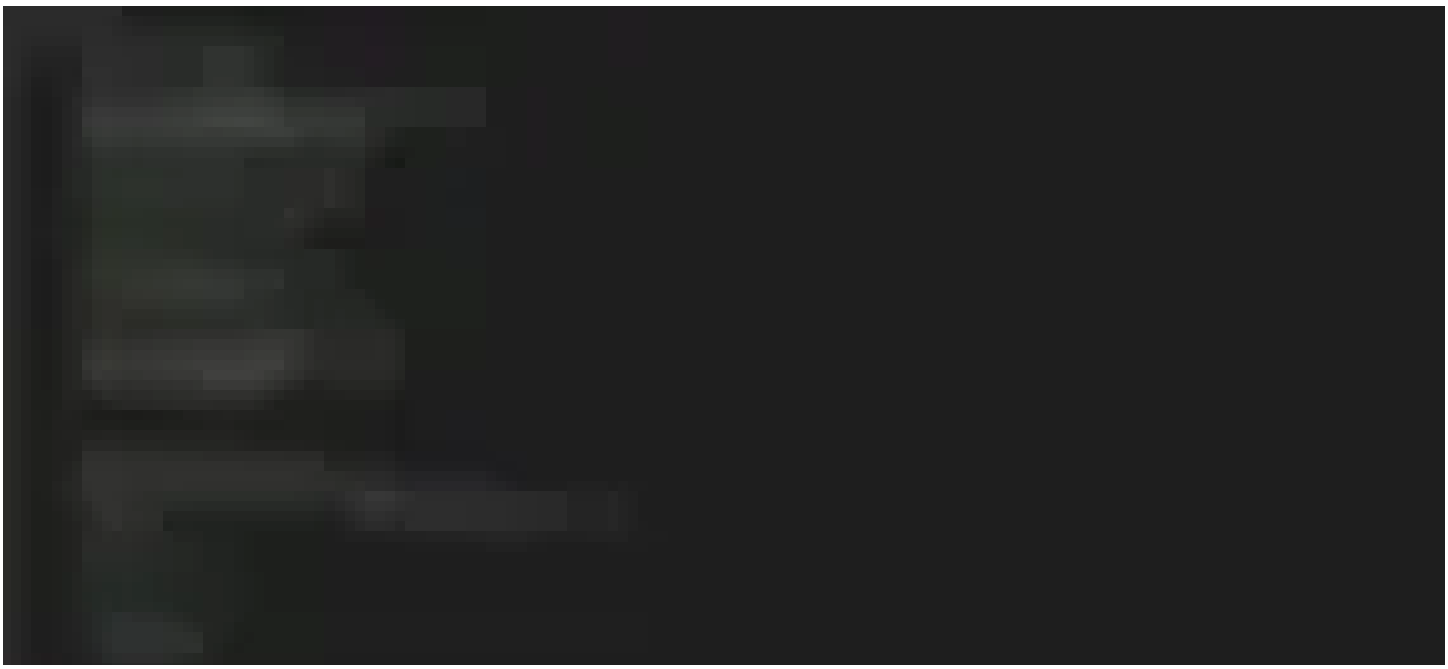
The graph between Accuracy and validation accuracy is as follows

Testing

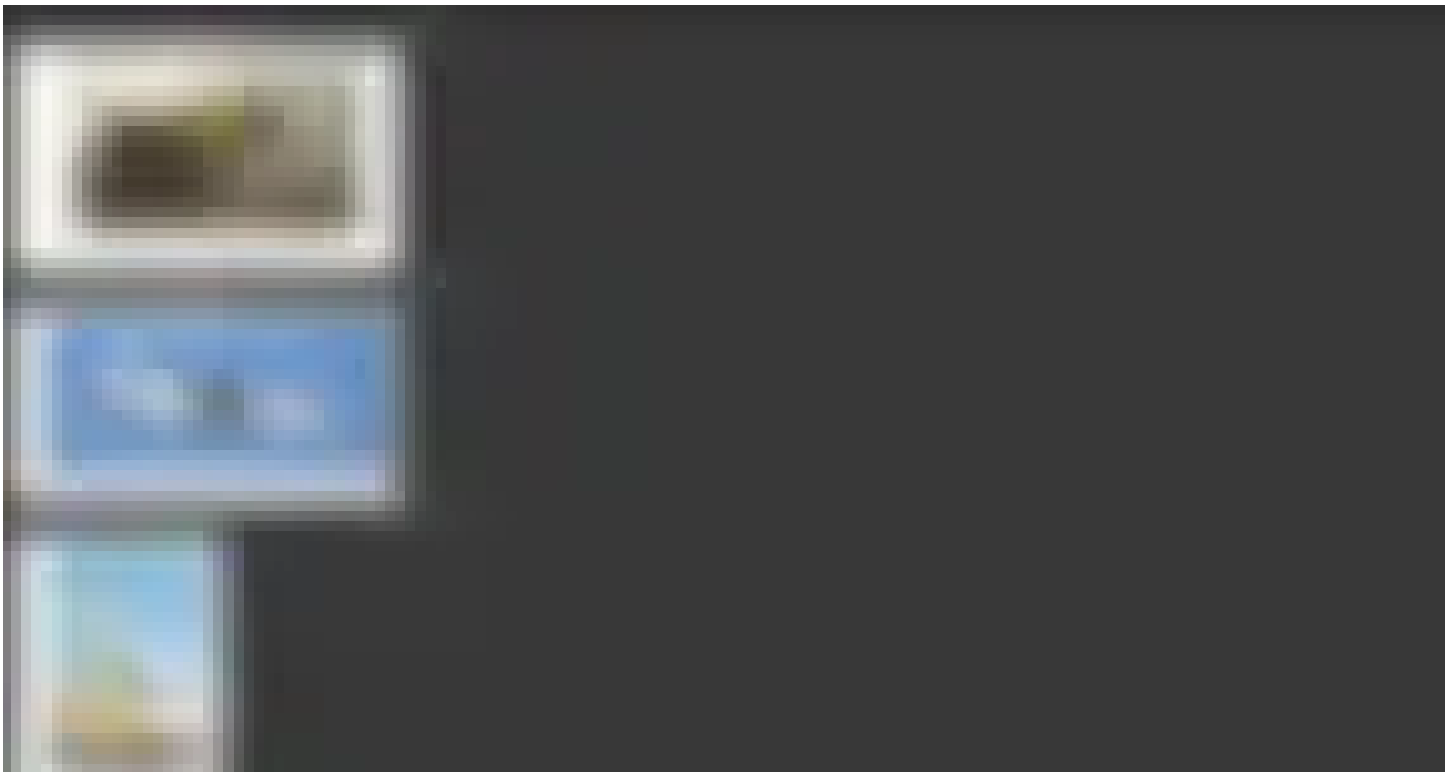
We load the best model obtained for testing and then predict the labels of test images.

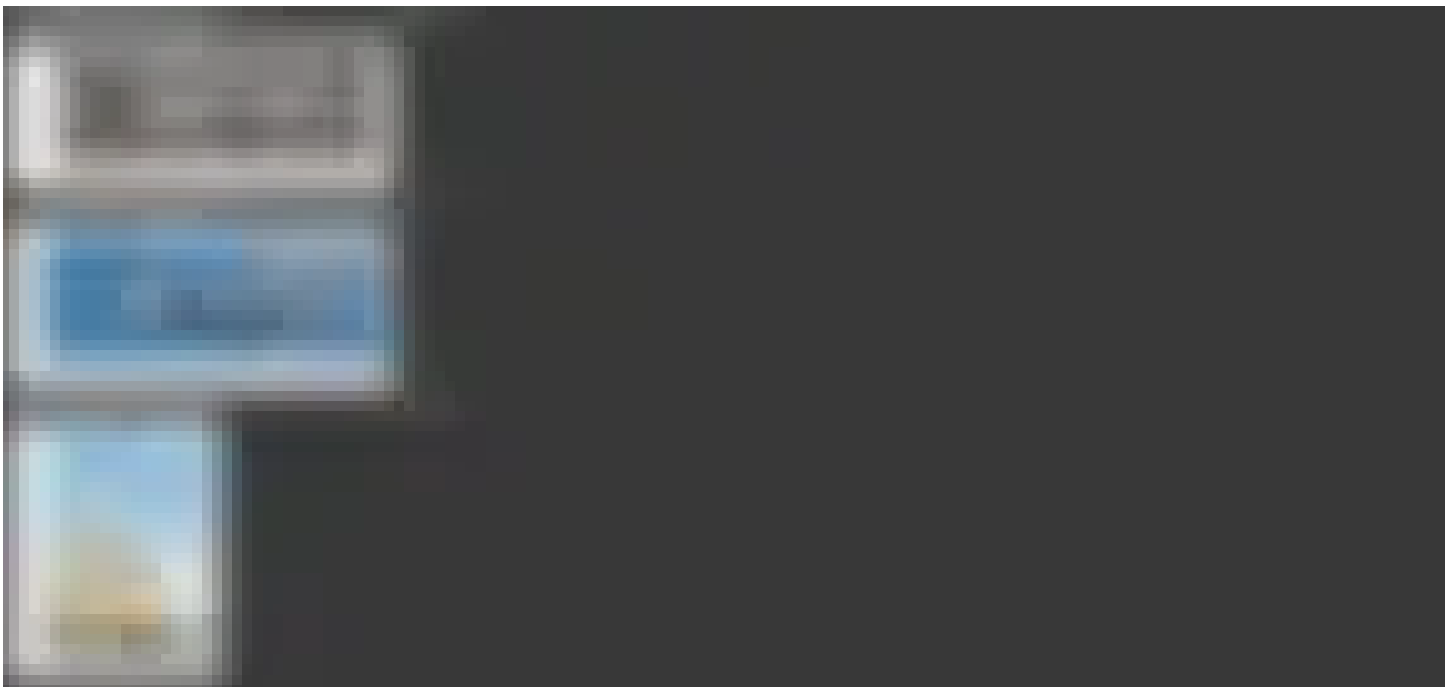


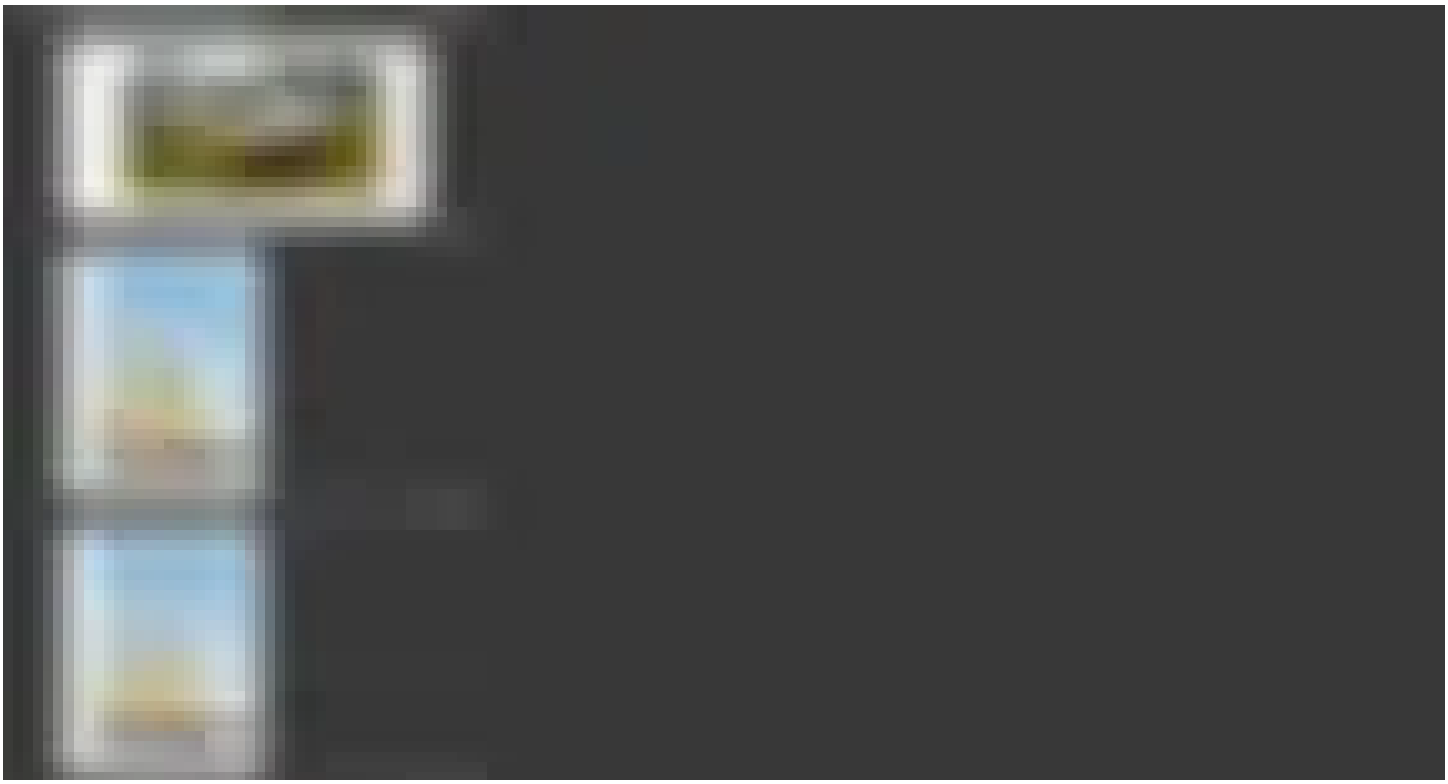
We predict the class label of the 10 testing images and observe that all the values are predicted correctly and hence the model performs pretty decent.



Output







CONTRIBUTION

The above model performs pretty well but in the process of increasing the accuracy, I tried experimenting with different hyperparameters and different number of layers.

Experiment 1 : Increasing the number of epochs -

Previously, the number of epochs were 9 I tried increasing them to 40 and hence got better accuracy.

We see that previously, the accuracy was 93.33 percent, but after increasing the epochs, the accuracy got increased to 98.61 and hence the model is performing even better.

We also analysed the loss and accuracy by plotting the performance.

Experiment 2 : Number of Layers

Previously, there were only 2 layers and I tried increasing it to 3 and hence the accuracy is also increased from 93.33 to 97.22

From the above two experiments, the highest accuracy is obtained from increasing the number of epochs and hence the performance of the model is increased.

REFERENCES

[1] https://www.kaggle.com/code/maricinnamon/multiclass-classification-caltech101-tensorflow

[2]https://www.mathworks.com/help/deeplearning/ref/vgg16.html;jsessionid=c9b7423621bb9e3cf9c8d63d2cec

[3]https://keras.io/guides/sequential_model/

[4]https://www.tensorflow.org/learn#build-models

[5]https://www.mathworks.com/help/deeplearning/ref/vgg16.html

[6]https://towardsdatascience.com/non-linear-augmentations-for-deep-leaning-4ba99baaaaca