Machine Learning Engineer Nanodegree

Capstone project

Project Report

Vijaybhasker Pagidoju

vijay@linkedIn

Definition

Project Overview

This project is intended to resolve a problem of identifying invasive species. The Solution is drawn by using Machine learning Algorithms, Deep Neural Networks and Transfer Learning.

The Intention of the project is to help researchers and environment by identifying invasive species, harmful to the habitat and eco system.

The Solution will save researchers valuable time, find the invasive species within very less time.

Problem Statement

Tangles of kudzu overwhelm trees in Georgia while cane toads threaten habitats in over a dozen countries worldwide. These are just two invasive species of many which can have damaging effects on the environment, the economy, and even human health. Despite widespread impact, efforts to track the location and spread of invasive species are so costly that they're difficult to undertake at scale.

Currently, ecosystem and plant distribution monitoring depends on expert knowledge. Trained scientists visit designated areas and take note of the species inhabiting them. Using such a highly qualified workforce is expensive, time inefficient, and insufficient since humans cannot cover large areas when sampling.

Because scientists cannot sample a large quantity of areas, some machine learning algorithms are used in order to predict the presence or absence of invasive species in areas that have not been sampled. The accuracy of this approach is far from optimal, but still contributes to approaches to solving ecological problems.

This Problem is part of Kaggle's competition.

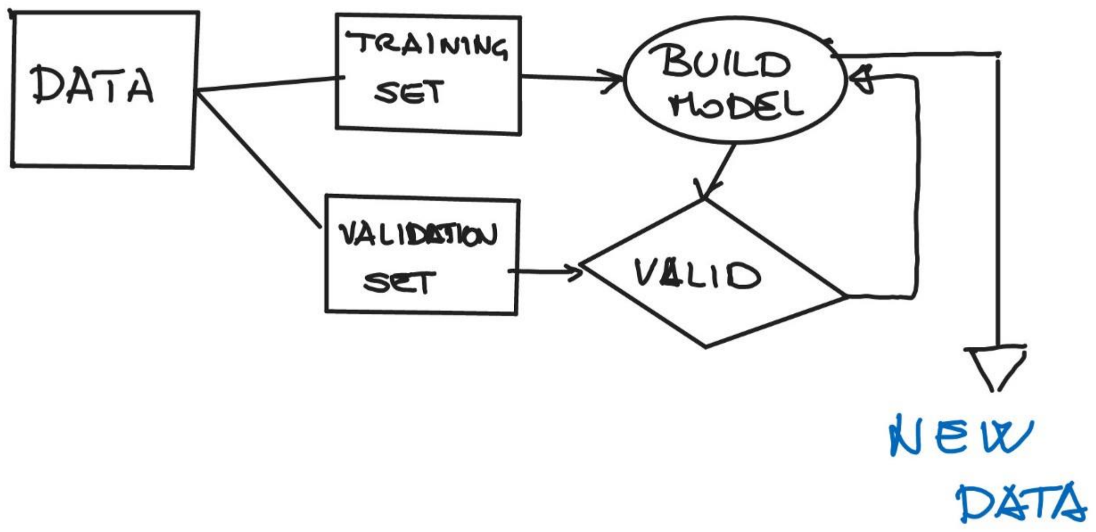
Our solution is to identify invasive species in the images like the above.

Analysis:

From the Problem statement, it is clear that the problem is a image classification based one and there is a training and testing sets of data provided with the label for the training set.

Being a labels training set and the solution's motive is to attain most accurate labels for the testing data set.

This is a Supervised Learning Problem



So, the intended steps are as below

Importing Datasets

Preprocessing the Data

Creating a Cross Validation Set

Building a Model

Testing the Model built

Optimizing the Model

Predicting the Labels of the Test Set

Datasets

The Data is provided on kaggle as part of the competition

The Datasets provided are

Train Dataset: A training dataset consisting of 2295 images of plants and the environment.

Test Dataset: A testing dataset consisting of 1531 images of the plants and the environment.

Training Labels: Labels for the training dataset of 2295 images are available in train_labels.csv file.

Sample submission file: A sample submission file in csv format is provided with 1531 rows with headers, as a sample format for the final submission labels csv file.

Metrics

The submission labels will be evaluated based on area under ROC curve between the predicted probability and observed target.

The Project will evaluate the train and cross validation sets in the metrics of accuracy.

The project will make use of various loss functions such as rmsprop to identify the loss and evaluate to mitigate.

Accuracy and Loss will be evaluated and observed further on graph visualizations to understand the progress.

Solution

The Project intends to use Neural networks for the building the model with various layers of nodes.The model will make use of various hyper parameters to tune the model.

A continuous evaluation of validation loss and accuracy during the training.

There will be further evaluation of the training and validation accuracies and then based on them, the project will import the pretrained networks based on ImageNet dataset(Resnet50,vgg16,vgg19,inceptionv3,etc).

The project intends to create a hybrid model using the pretrained network and fully connected densed layers after the pretrained network.

Finally, the the labels for the testing dataset will be identified.

Benchmark

The project being a part of the kaggle's competition, the benchmark model at this time of this project submission is at a score 0.9970, which is highly unlikely to be achieved.The Project intends to understand the problem and route to the solution with clear steps and leading to a optimal validation accuracy as possible and make 1st ever kaggle competition submission.

Methodology

Importing Libraries

Initially there are a stack of imports that are required for the project to set go..

The project imports following

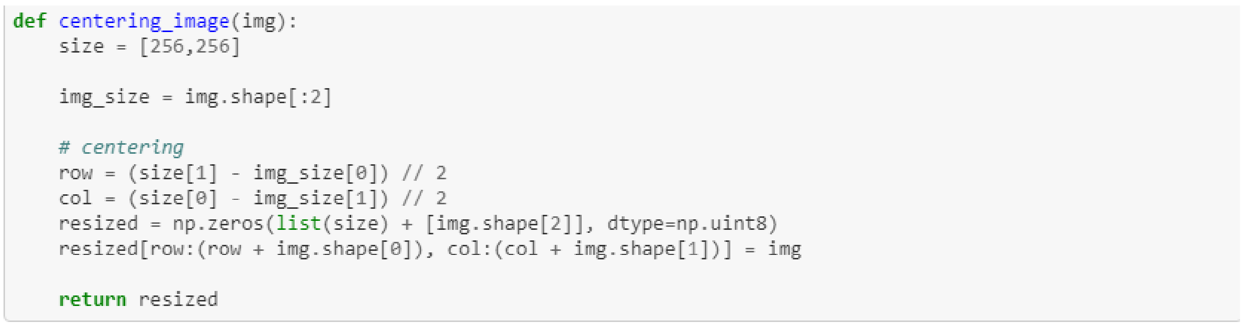
Pandas, NumPy, Matplotlib, OpenCV, Keras, Sklearn, tqdm, Pretrained networks from keras,PIL, ImageDatagenerator, Keras Datasets,Conv2D.

Data Preprocessing

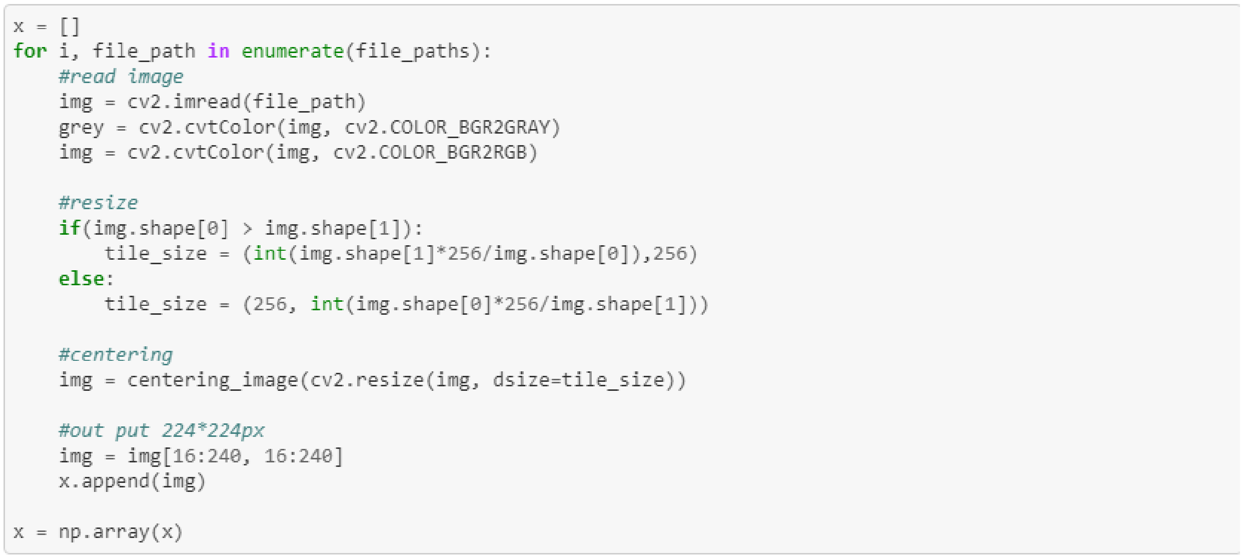
The training labels should first be evaluated to find any discrepency and should delete if any NA values are available and the same with sample submisssion csv file.The training and testing datasets should be evaluated to find the number of images provided in them.

After the initial evaluation, the images are set as a numpy array and then further to this the images are centered and bought to a common pixels so that there is a consistency across all the images to have a consistent training of the datasets.

Then, all the images in the datasets should be normalized by dividing by 255 to get rid of any inconsistency and effects.



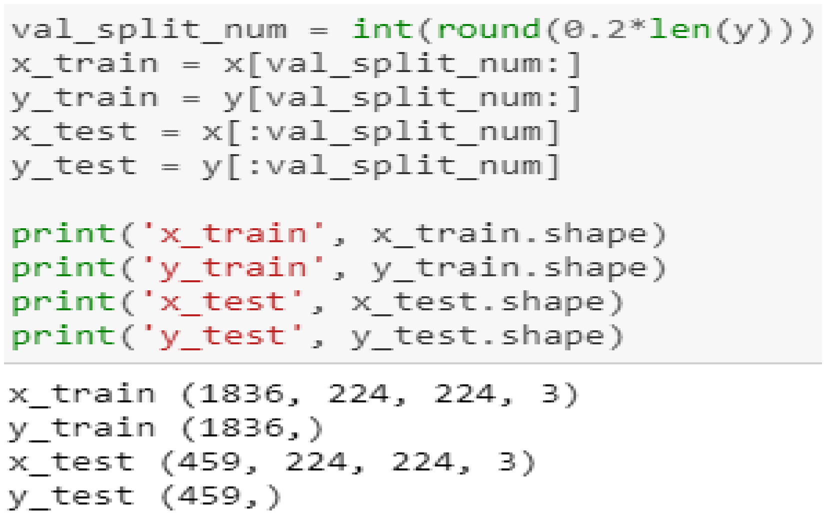
```python
def centering_image(img):
    size = [256,256]

    img_size = img.shape[:2]

    # centering
    row = (size[1] - img_size[0]) // 2
    col = (size[0] - img_size[1]) // 2
    resized = np.zeros(list(size) + [img.shape[2]], dtype=np.uint8)
    resized[row:(row + img.shape[0]), col:(col + img.shape[1])] = img

    return resized
```



```python
x = []
for i, file_path in enumerate(file_paths):
    #read image
    img = cv2.imread(file_path)
    grey = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    #resize
    if(img.shape[0] > img.shape[1]):
        tile_size = (int(img.shape[1]*256/img.shape[0]),256)
    else:
        tile_size = (256, int(img.shape[0]*256/img.shape[1]))

    #centering
    img = centering_image(cv2.resize(img, dsize=tile_size))

    #out put 224*224px
    img = img[16:240, 16:240]
    x.append(img)

x = np.array(x)
```

There should be cross validation set that should be formed, to test the trained model, which should not be a part of Training and testing set.

The Training set is shuffled to make the dataset random, to make the input more generic and less prone to Overfitting.Then, the Training set is divided to Training set and Cross Validation set. The Data is usually split as per the standard 80:20 dataset split.



```
val_split_num = int(round(0.2*len(y)))
x_train = x[val_split_num:]
y_train = y[val_split_num:]
x_test = x[:val_split_num]
y_test = y[:val_split_num]

print('x_train', x_train.shape)
print('y_train', y_train.shape)
print('x_test', x_test.shape)
print('y_test', y_test.shape)
```

```
x_train (1836, 224, 224, 3)
y_train (1836,)
x_test (459, 224, 224, 3)
y_test (459,)
```

Now, the images from the dataset should randomly be witnessed to see if all the images are available.

Building a Model

The Model architecture is built from scratch using Convolutional neural network layers.

The Model is built using

One Input layer

Three Hidden layers

One output layer

The Batch normalization input layer is produced with the input as required.

The three hidden layers use varied number of filters, even padding, same kernel size.

After every hidden layer, there is a Maxpooling with pool_size set to 2, to reduce the high feature dimensionality from the filter sizes, which will help the model to mitigate overfitting.

One more measure taken is to add dropout layer at a random permutation, so that there is a random drop of input to make the training more generic and this also helps the model to be away from problem of overfitting.

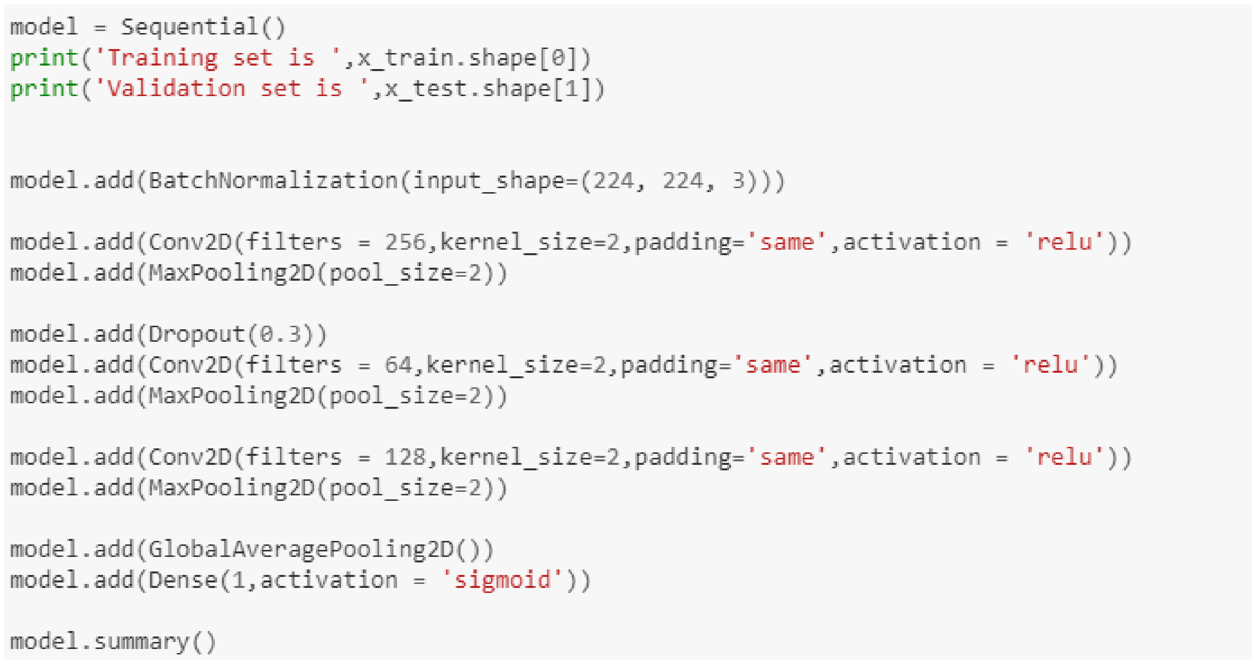
Before adding the final Dense layer, A globalAveragepooling layer is added to make the feature dimensionality reduced to 1, to input it to final layer.

The Model is using relu as an activation functins for all the hidden layers and sigmoid as an activation function for the final layer.

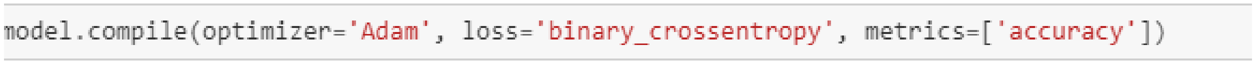
This entire model built results in a total parameters to be trained as 101,965.

The Built Model is further compiled using Adam as an optimizer, with loss as binary cross entropy and accuracy as metrics.

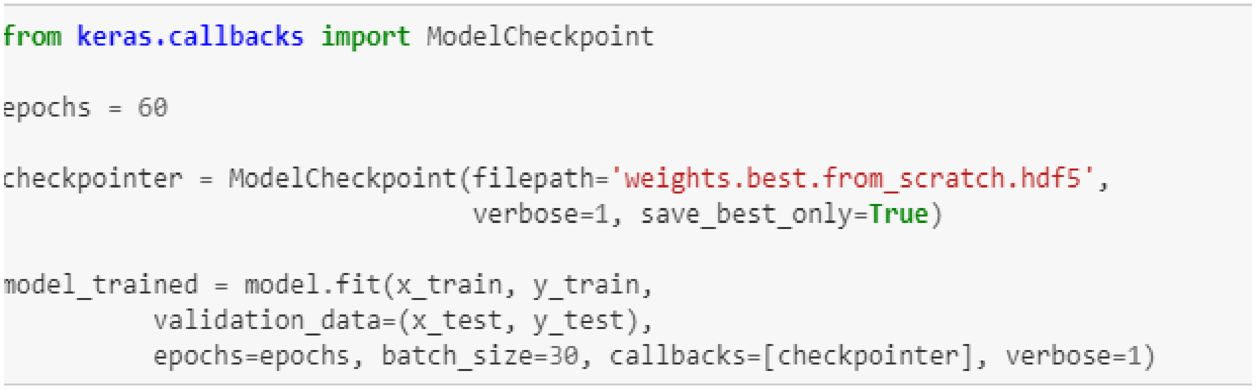
The Model is trained for 60 epochs with a batch size of 30.The weights that result in incremental accuracy and decrementing the loss are stored in a file to further load them to the model after the training to improve accuracy of the model.



```
model = Sequential()
print('Training set is ',x_train.shape[0])
print('Validation set is ',x_test.shape[1])


model.add(BatchNormalization(input_shape=(224, 224, 3)))

model.add(Conv2D(filters = 256,kernel_size=2,padding='same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Dropout(0.3))
model.add(Conv2D(filters = 64,kernel_size=2,padding='same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 128,kernel_size=2,padding='same',activation = 'relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(GlobalAveragePooling2D())
model.add(Dense(1,activation = 'sigmoid'))

model.summary()
```

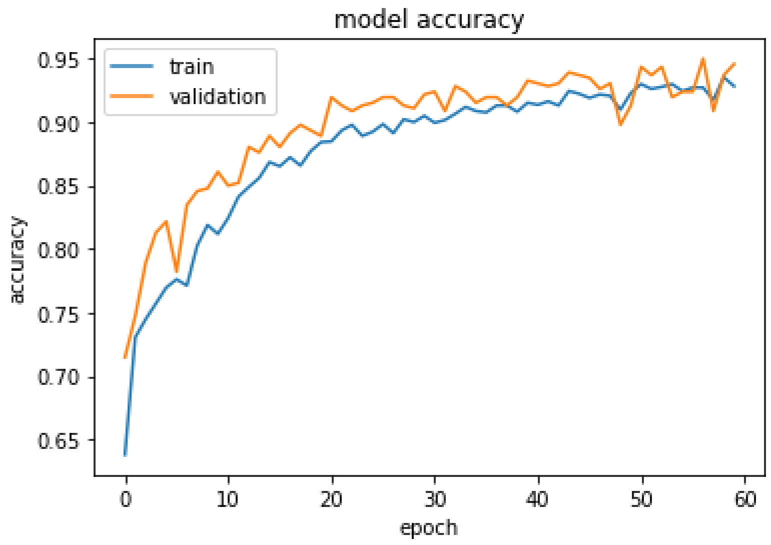


```
model.compile(optimizer='Adam', loss='binary_crossentropy', metrics=['accuracy'])
```

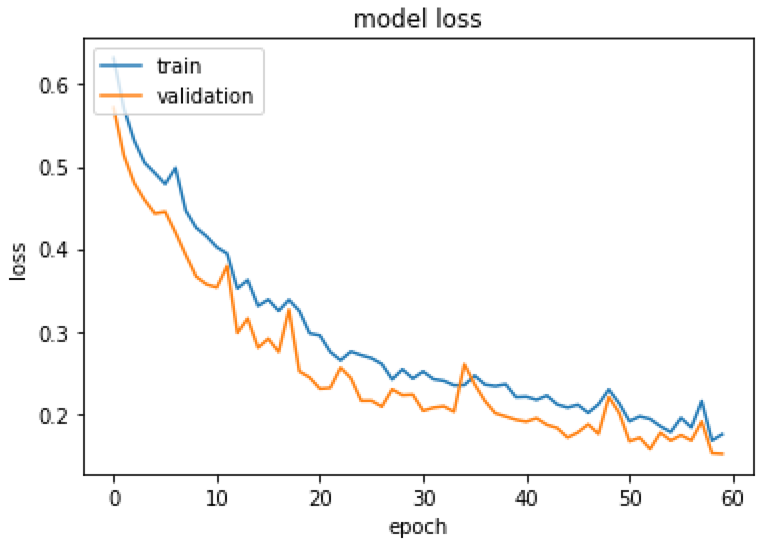


```
from keras.callbacks import ModelCheckpoint

epochs = 60

checkpointer = ModelCheckpoint(filepath='weights.best.from_scratch.hdf5',
                               verbose=1, save_best_only=True)

model_trained = model.fit(x_train, y_train,
          validation_data=(x_test, y_test),
          epochs=epochs, batch_size=30, callbacks=[checkpointer], verbose=1)
```

Training and Validation Accuracy Graph



Training and Testing Loss Graph



Optimizing the Model building using Transfer Learning

After observing the built model validation accuracy and validation loss , in order to optimize the models accuracy, A hybrid model using transfer learning is built.

A Pretrained Neural Network on ImageNet Dataset is loaded and then the pretrained network is loaded to the new model and then a fully connected layers are added to make the model specific to images in the dataset.

In the present project, VGG16 neural network is used from the keras applications.

Once the vgg16 pretrained network is loaded with the same input shape of 224,224. There are three hidden dense layers and one output layers added to the model.

The three hidden layers have varied filter sizes and use same activation function relu.The final layer is reduced to 1 filter size, using activation function sigmoid.

As the hybrid model is built, its compiled using same loss function and metrics as accuracy. The Models uses the same optimizer with different tailered parameter to decreased the learning rate.

This results in total trainable parameters to 21,178,689.

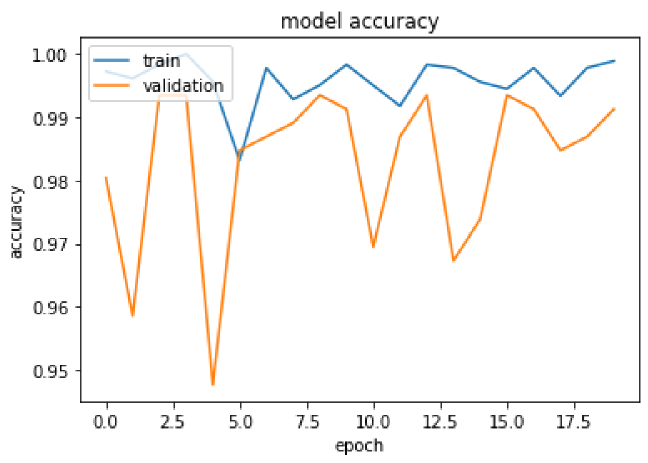
The Model uses imagedatagenerator to rotate the images at different angles to further get a good training accuracy for the images.

The Model is then trained for 20 epochs with a batch size of 32 and the best weigts are stores as earlier to load the model for better working.

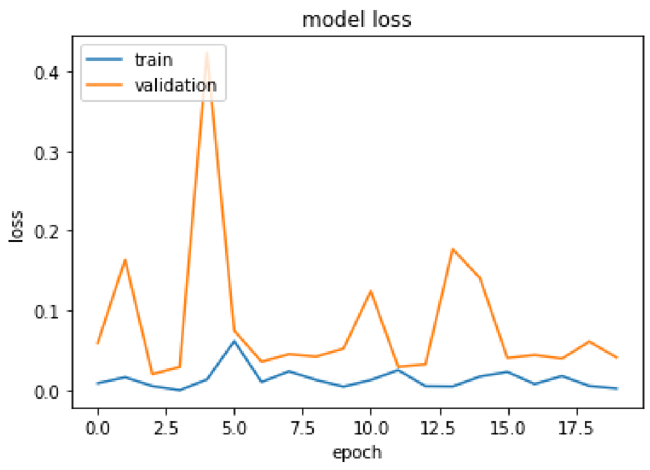
```
add_model = Sequential()
add_model.add(Flatten(input_shape=base_model.output_shape[1:]))

add_model.add(Dense(256, activation='relu'))

add_model.add(Dense(128, activation='relu'))

add_model.add(Dense(64, activation='relu'))

add_model.add(Dense(1, activation='sigmoid'))

vgg16_model = Model(inputs=base_model.input, outputs=add_model(base_model.output))
vgg16_model.compile(loss='binary_crossentropy', optimizer=Adam(lr=1E-4, beta_1=0.9, beta_2=0.999, epsilon=
1e-08, decay=1E-4),
              metrics=['accuracy'])
vgg16_model.summary()
```

```
batch_size = 32
epochs = 20

vgg16_train_datagen = ImageDataGenerator(
        rotation_range=31,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True)
vgg16_train_datagen.fit(x_train)

vgg16_checkpointer = ModelCheckpoint(filepath='weights.best.vgg16.hdf5',
                               verbose=1, save_best_only=True)


vgg16_history = vgg16_model.fit_generator(
    vgg16_train_datagen.flow(x_train, y_train, batch_size=batch_size),
    steps_per_epoch=x_train.shape[0] // batch_size,
    epochs=epochs,callbacks=[vgg16_checkpointer],
    validation_data=(x_test, y_test),
)
```

Training and Validation Accuracy Graph



Training and Validation Loss Graph



Test Prediction

As, the Problem is part of the Kaggle competition,

Using our trained hybrid model, Prediction is made for all the images in the test dataset and the labels are stored in the submit.csv file and uploaded to the kaggle submissions.