ST JOSEPH’S COLLEGE (AUTONOMOUS)

BENGALURU-560027

FIRE DETECTION
MULTIVARIATE STATISTICS PROJECT REPORT

Submitted by:
NIKHITHA ELEZEBETH BABY
21BDA14

Under the supervision of
DR SRINIVAS BHOGLE
Department of Advanced Computing
St. Joseph’s College (Autonomous)
36, Lalbagh Road
Bangalore- 575003

ACKNOWLEDGEMENT

The successful completion of any task is incomplete and meaningless without giving any due credit to the people who made it possible without which the project would not have been successful and would have existed in theory.

First and foremost, we are grateful to Dr. Jayati Bhadra, HOD, Department of Ad- vanced Computing, St. Joseph’s College, for giving us this opportunity. We owe a lot of thanks to our supervisor, Dr Srinivas Bhogle, Department of Advanced Computing, St. Joseph’s College, for igniting and constantly motivating us and guiding us in the idea of a creatively and amazingly performed Major Project, in undertaking this endeavor and challenge and also for being there whenever we needed her guidance or assistance.

We would also like to take this moment to show our thanks and gratitude to one and all, who indirectly or directly have given us their hand in this challenging task. We feel happy and joyful and content in expressing our vote of thanks to all those who have helped us and guided us in presenting this project work for our project. Last, but never least, we thank our well-wishers and parents for always being with us, in every sense and constantly supporting us in every possible sense whenever possible.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview and Background

Fire is one of the deadliest risks any living things can encounter in their life. In a concise time frame, fire can wreck an area, say in a forest, hut, house, building, etc. Fire can lead to the loss of expensive property, and in the worst possible scenario, it can also lead to the loss of human life. Detecting fire thus becomes a significant concern.

Our project will carry out fire detection by adopting an RGB (Red, Green, Blue) model based on chromatic and disorder measurement for extracting fire pixels. To prevent this, we will build an early fire detection using image segmentation with the help of the Convolution Neural Network (CNN) model with alarm system VGG16.

Image Localization helps to detect the location of a single object in any given image. In our case, image localization can locate the fire in a given image. We will use the CNN and VGG16 model to train and build predictions over our input images. CNN and is a deep neural network algorithm that is used for solving segmentation problems.

Fire accidents kill 54 people daily in India, yet deaths have declined. Accidental fires caused 6% of all unnatural deaths in India, third-highest after car accidents 53%, and drowning at 9%. During the period, the number of fire accidents reduced by 44 per cent - 16,695 in 2016 to 9,329 in 2020.

Fire-related accidents have, on average, killed 35 people every day in the five years between 2016 and 2020, according to a report by Accidental Deaths and Suicides in India (ADSI), maintained by the National Crime Records Bureau.

Some of the incidences are : There was a fire in Virudhunagar fireworks factory in Tamil Nadu where 25 innocent people were killed. Massive fire engulfs Vizag chemical plant, explosions heard, injuries reported. Several people were injured in an explosion at a chemical factory in Andhra Pradesh's Visakhapatnam. The explosions were followed by a massive fire that was visible across the city. At least 12 fire tenders were deployed to douse the flames.

1.2 Need for the study

We have come across many incidences in and around our locality where fire will cause a huge destruction to the property. After going through many research we do know that there are smoke detectors and various other technologies to get to know whether there is a fire or not. But we realized that there is need for detection of fire before smoke appears where we could detect the source of fire and immediately raise an alert. By doing this we feel that common man should be able to implement our model and he/she can prevent major loss that is being caused by fire. Therefore we have done fire detection at first sight and it will raise an alarm as soon as it detects fire.

1.3 Objectives

Our main aim is to locate the position where the fire is present, which will help the authorities take proper measures to avoid any loss. This project aims to build a deep neural network model that will give the best accuracy in detecting fire.

CHAPTER 2

VGG16 MODEL

2.1 What is VGG16

VGG-16 is a convolutional neural network that is 16 layers deep. It is proved to be a most important milestone in the search of mankind to make computers "see" the world. It is one of the significant innovations that showed the way for several innovations in this field.

As part of our project we have made use of this significant model VGG16. Initially we have cleaned the dataset thoroughly as it was a basic requirement for this model and split the dataset into training and testing set. We loved working with this model as we were able to detect misclassified images as well. Let's look at our model workflow in detail.

2.2 Data Collection

2.2.1 Collecting Fire Images

Our first **data set** contains 600 images of fires. They are mostly urban fire and few wildfires. Initially data was not cleaned. It contains many images without fire such as site before and after the fire, fire-fighters and their equipment only, map of the area of the fire, a cartoon of fire, etc. So we have cleaned the data.

All black and white images are removed. All images without fire or smoke are removed. All the repeated images have been removed. It was a challenge whether or not to keep the picture involving fire damage and lots of ash. Ash might be the indication that there was a fire. But as a fire detection tool there might be little interest in the post damage picture. So we decided to remove those images as well. All the cleaning work was done manually.

2.2.2 Collecting Non-Fire Images

Our second dataset contains 600 non-fire images in various categories: coast, forest, highway, city, mountain, land, street, tall building, etc. and their subcategories. Almost all the images are taken at night. There are **93 such images.** So far we have sufficient non-fire images to match the number of fire images.

Therefore we have 600 images containing fire and 600 images without fire. So in total, we have 1200 images. Throughout this project JEPG image with RGB, the format is chosen as default.

2.2.3 Downloading Images from Website

We used the adobe stock website to collect images. We used the BeautifulSoup package to extract images from the website. We used 43 different keywords to search for fire images and 35 different keywords to search the night images without fire. We cleaned the data removing all the images not containing fire and smoke from the first set.

2.3 Training a Deep Learning Model

2.3.1 Model Architecture

We have 1200 total labeled sample images in total to work with. That means 600 fire images and 600 non-fire images. Among them we have used 900 images for training and 300 for testing equally splitting among both the labels fire and non-fire. Although this data set is of decent size, it is not enough to train the model from scratch using Keras model. And as we see later it is not necessary as well.

In order to be simple we have made use of pre-trained model. We used VGG16 pre-trained model. VGG-16 is a trained Convolutional Neural Network (CNN), from Visual Geometry Group (VGG), Department of Engineering Science, University of Oxford. The number 16 means the number of layers with trainable weights. Pre-trained model are trained in different data, not necessarily similar to the data we are training in this model.

We have still made use of this because of the following reason: As we know that CNN has series of layers. Each layer learns a set of features from the image data. The lower layers learn fundamental patterns like edges, lines, curves etc. The higher layers on the other hand are specific to the images on the model. Hence, the featured learned by the lower level can be general to the large class of images, even the images which model did not see during its training. Because of this reason we only use the base of the pre-trained model removing the top.

We have done this here in two steps. First, we retained all the base and removed only the dense top layer and trained the model. Through this we have received the validation accuracy close to 88%.

In the second step, we unlock the top convolutional model on the base and further train the model. Since we already have a decent accuracy we can imagine that the model is already close to the good model. So we only need to fine tune. We achieve validation accuracy about 89% which is pretty decent result.

We have set up our model with VGG16 base and custom top.

(1) Loss: Since this is classification problem and there are two classes and the binary cross-entropy as the loss function.

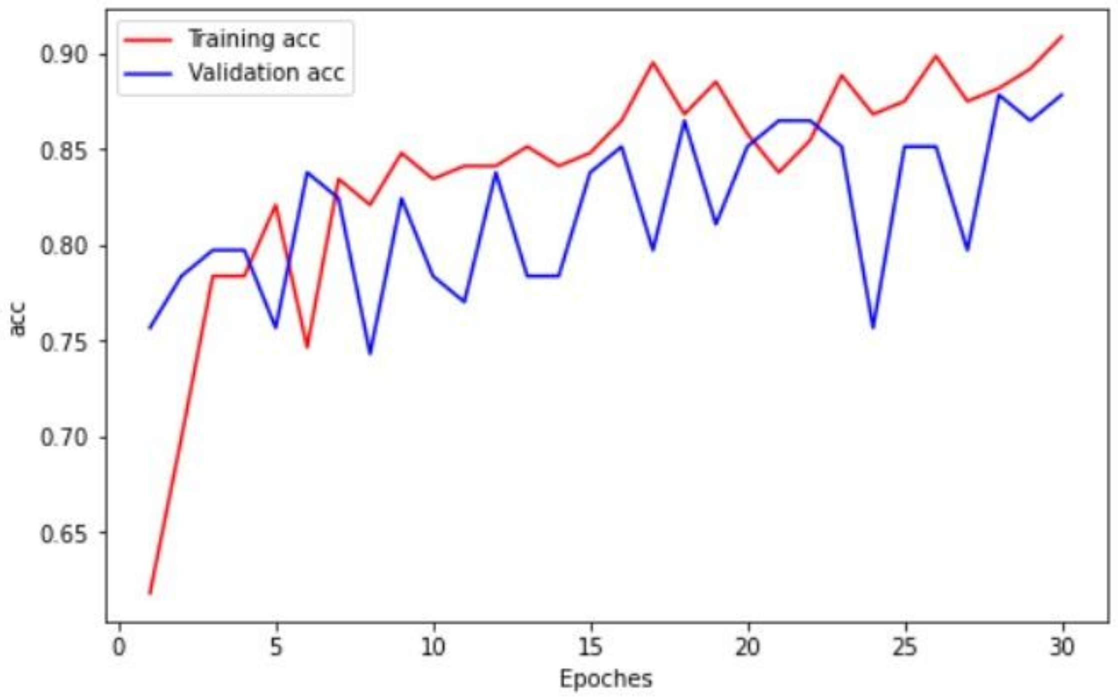
(2) Optimizer: We use RMSprop optimizer with customized learning rate.

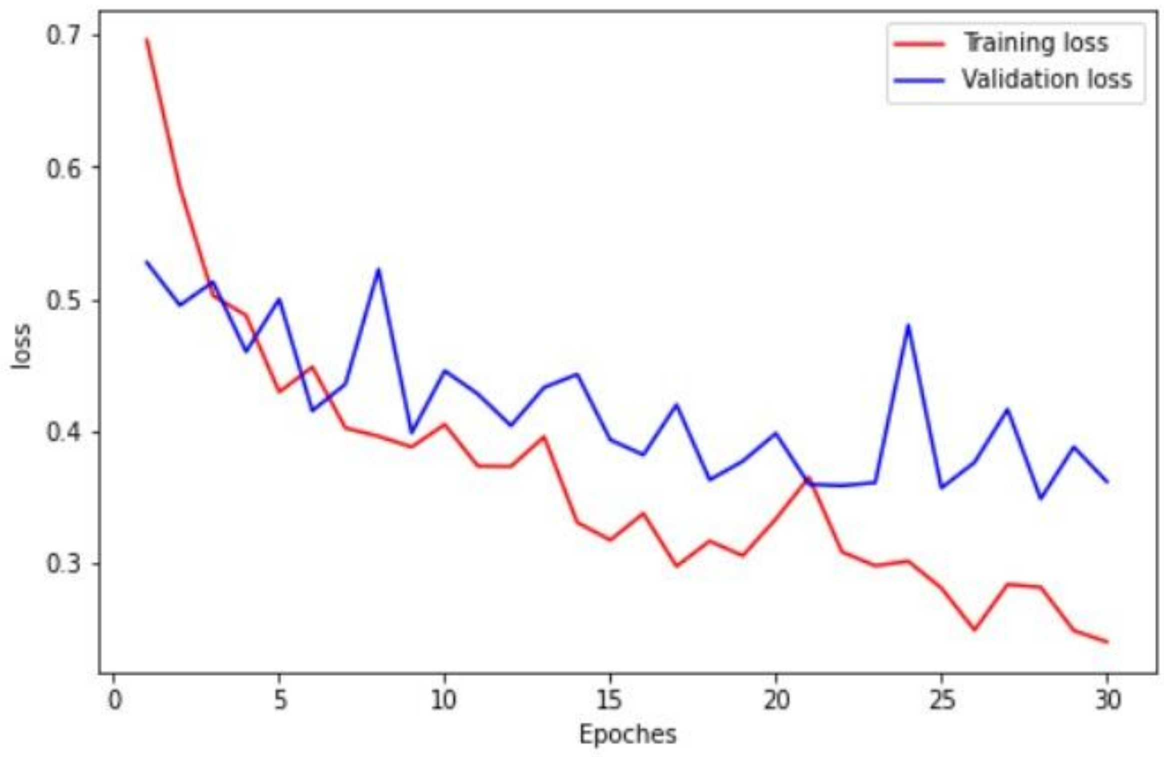
(3) Metrics: In addition to the loss we want to observe the accuracy. We optimize our model based on this metric.

2.3.2 Data Generation & Data Augmentation:

For the large data set it is not convenient to load all the data into memory. So we have used image data generator to load the data from hard disc to memory in small batch. We have done the same for the training and test set. When initiating the image data generator we have also done the data augmentation. We have done the data augmentation in the training set and not on the validation and test set. We have passed the training data from the train generator. We trained for 30 epochs. We passed the validation data from the validation generator and we get validation accuracy about 88%.

Here is the graph of training and validation loss as well as training and validation accuracy.

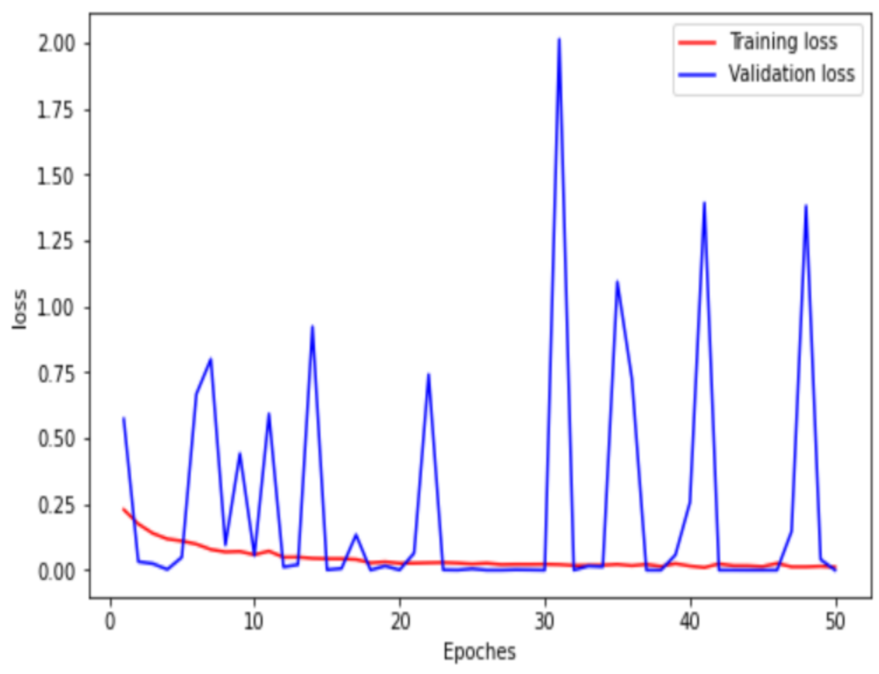


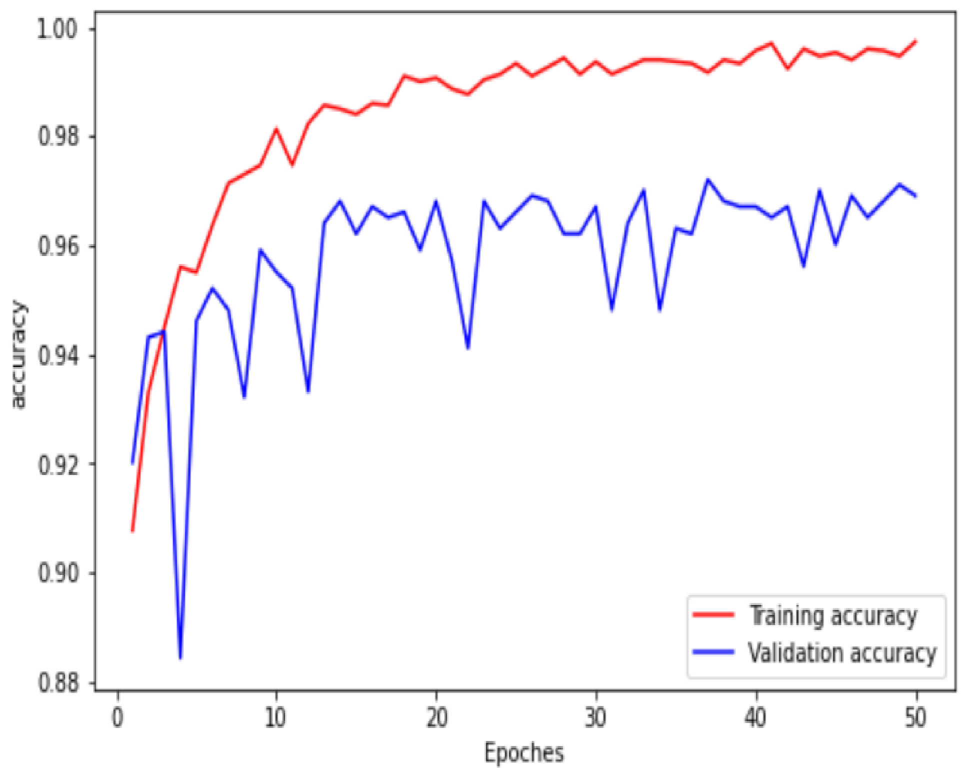


2.4 Fine Tuning the Model:

Previously, we trained with only top layer removed from VGG16. Here we unlock top base layer from VGG16 and fine tune the model. Doing so we reduce the learning rate by an order from 10−4 to 10−5 .

For fine tuning of our model we have trained for the 50 epochs. The model surpass the validation accuracy of 88% shortly after 30 epochs.

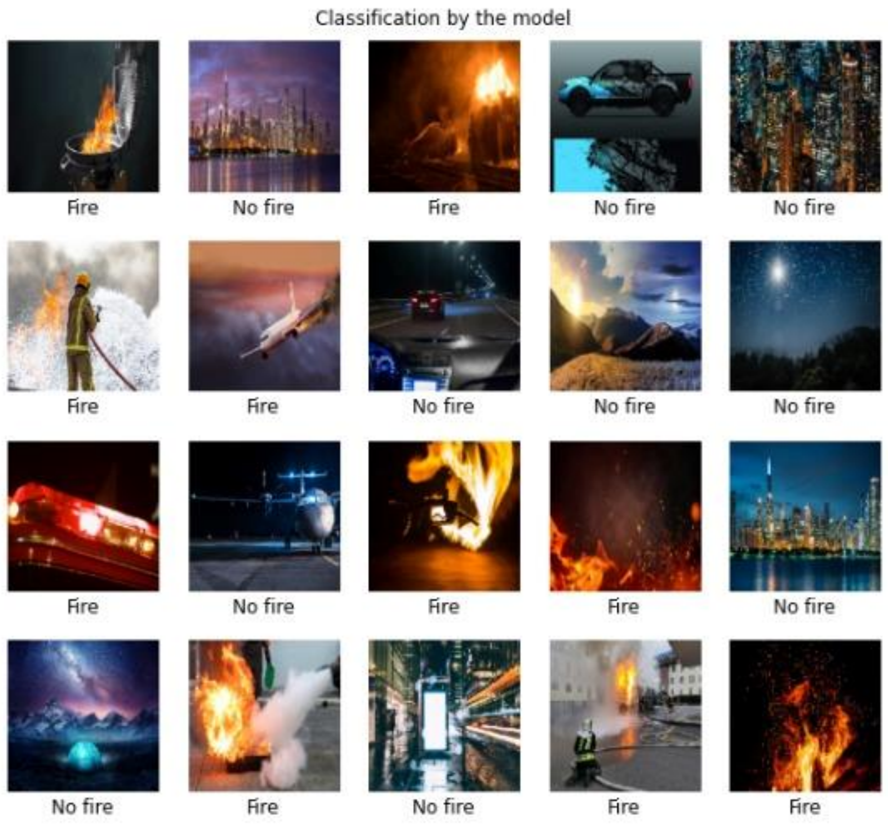

Training loss
Validation loss
loss
Epoches


Training accuracy
Validation accuracy
accuracy
Epoches

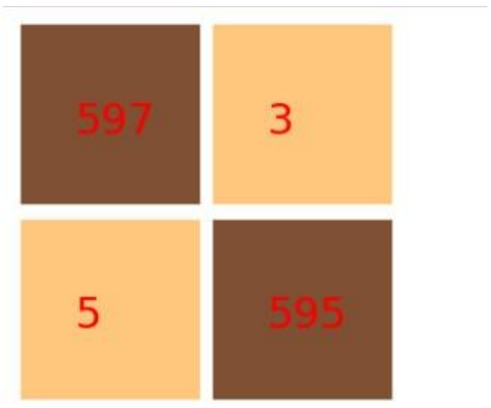
2.5 Error Analysis:

In this section we analyze the error of the model, i.e. misclassified images.

Firstly we can see few examples of the correctly classified images**.**



We can clearly see the classification done by our model. It has detected fire images as fire and non-fire images as non-fire. Then we visualize the confusion matrix.



And finally, we see separately fire images classified as non-fire in the figure given below.

Some of the misclassified images have fire but that is too small. Even human observer can get easily confused. Some of the big explicit fire images are miss classified too. May be that is painting of fire but not the picture. Misclassified fire images are mostly bonfire, stove fire, fire torch, kitchen fire etc. This is not big surprise because there were not enough fire sample in training set in that categories. Looking at this misclassified set some of the picture actually seems to have fire. So we can assume that the problem is about the mis- labeling. Others don’t have fire but have artificial red light or are picture with hue of dawn and dusk almost appearing as fire. Overall the model has done a good job in separating those images with solid 89% accuracy.

Therefore we can see that detection of fire has been done in the given image. VGG16 has been able to identify the fire images as well as nonfire images. This will be of great help for future studies when we implement an alarm to this model. The best part of VGG16 is that we are able to identify the misclassified images.

CHAPTER 3

CNN MODEL

3.1Introduction

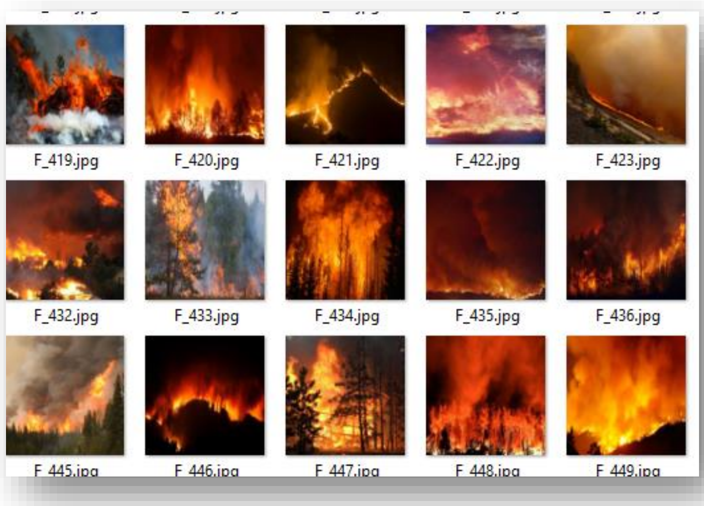
CNN is a type of neural network model which allows us to extract higher representations for the image content. Unlike the classical image recognition where you define the image features yourself, CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification.

3.2 Data Collection

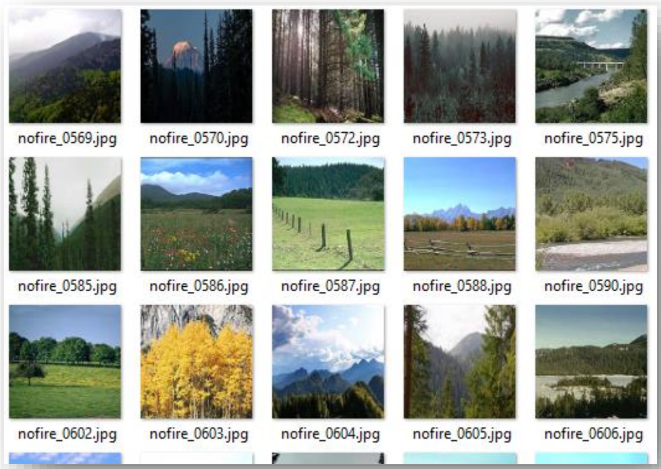
The secondary source fire image dataset used in this project are primarily from the various Kaggle source and Google images. We collected images with fire and non-fire dataset because our model detects both fire and non-fire images. However, the dataset was not cleansed. We removed some non-fire images from our fire image dataset manually.

Our dataset was split into training and testing sets. There were two types in the training data: fire and non-fire. For both fire and non-fire images, about 7086 images were taken into consideration. 565 images were taken into the account for testing data.

Example of fire image dataset :



Example for non-fire image dataset :



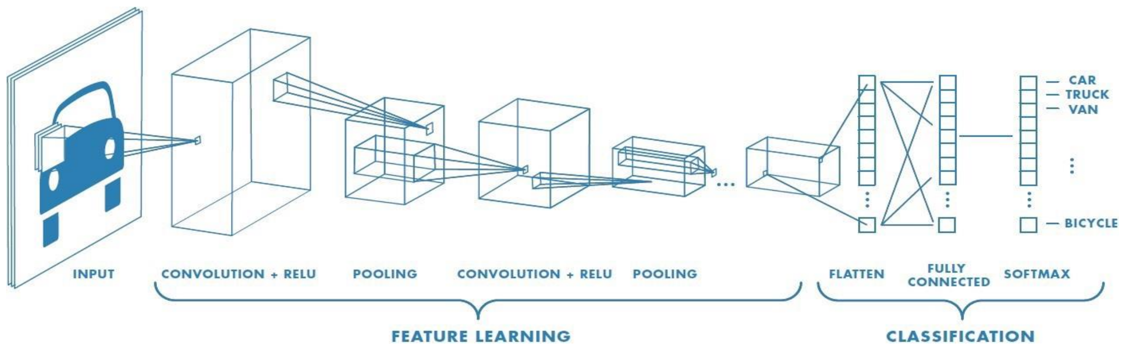
3.3 Methodology

We used CNN model, a variant of deep learning for the project. So how CNN works!

CNN also called Conv-Net is a class of deep neural networks which is based on shared-weights architecture and translation invariance characteristics. In CNN, the network employs a mathematical operation called Convolution. Convolution is a specialized kind of linear operation. A CNN consists of an input and an output layer with the multiple hidden layers between input and output layer.

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing. The term ‘Convolution” in CNN denotes the mathematical function of convolution which is a special kind of linear operation wherein two functions are multiplied to produce a third function which expresses how the shape of one function is modified by the other. In simple terms, two images which can be represented as matrices are multiplied to give an output that is used to extract features from the image.

There are three types of layers that make up the CNN which are the convolutional layers, pooling layers, and fully-connected (FC) layers. When these layers are stacked, a CNN architecture will be formed. In addition to these three layers, there is one more important parameter which is the activation function which is defined below:



1. **Convolutional Layer**

   This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM). The output is termed as the Feature map which gives us information about the image such as the corners and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

2. **Pooling Layer**

   In most cases, a Convolutional Layer is followed by a Pooling Layer. The primary aim of this layer is to decrease the size of the convolved feature map to reduce the computational costs. This is performed by decreasing the connections between layers and independently operates on each feature map. Depending upon method used, there are several types of Pooling operation.

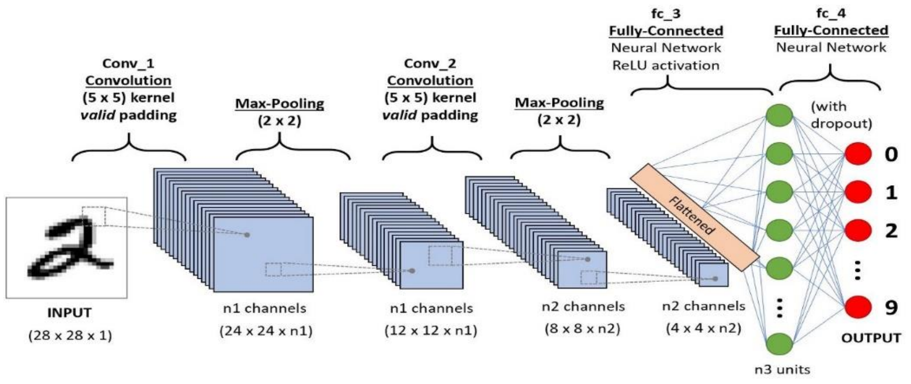
   In Max Pooling, the largest element is taken from feature map. Average Pooling calculates the average of the elements in a predefined sized Image section. The total sum of the elements in the predefined section is computed in Sum Pooling. The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

3. **Fully Connected Layer**

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture. In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

4. **Activation Functions**

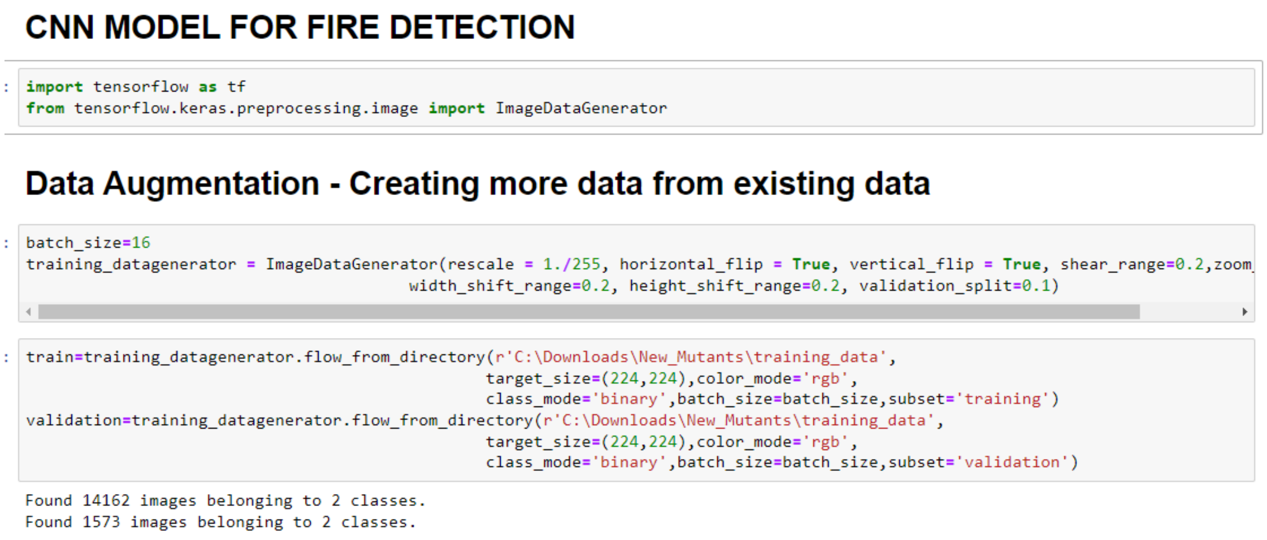
Finally, one of the most important parameters of the CNN model is the activation function. They are used to learn and approximate any kind of continuous and complex relationship between variables of the network. In simple words, it decides which information of the model should fire in the forward direction and which ones should not at the end of the network. It adds non-linearity to the network. There are several commonly used activation functions such as the ReLU, Softmax, tanH and the Sigmoid functions. Each of these functions have a specific usage. For a binary classification CNN model, sigmoid and softmax functions are preferred an for a multi-class classification, generally softmax is used.



In Python Programming, the model type that is most commonly used is the Sequential type. It is the easiest way to build a CNN model in keras. It permits us to build a model layer by layer. The ‘add()’ function is used to add layers to the model. As explained above, the proposed system architecture consists of three Convolution and Pooling pairs followed by a Flatten layer which is usually used as a connection between Convolution and the Dense layers.

3.4 Model Demonstration

3.4.1Training CNN model



CNN MODEL FOR FIRE DETECTION

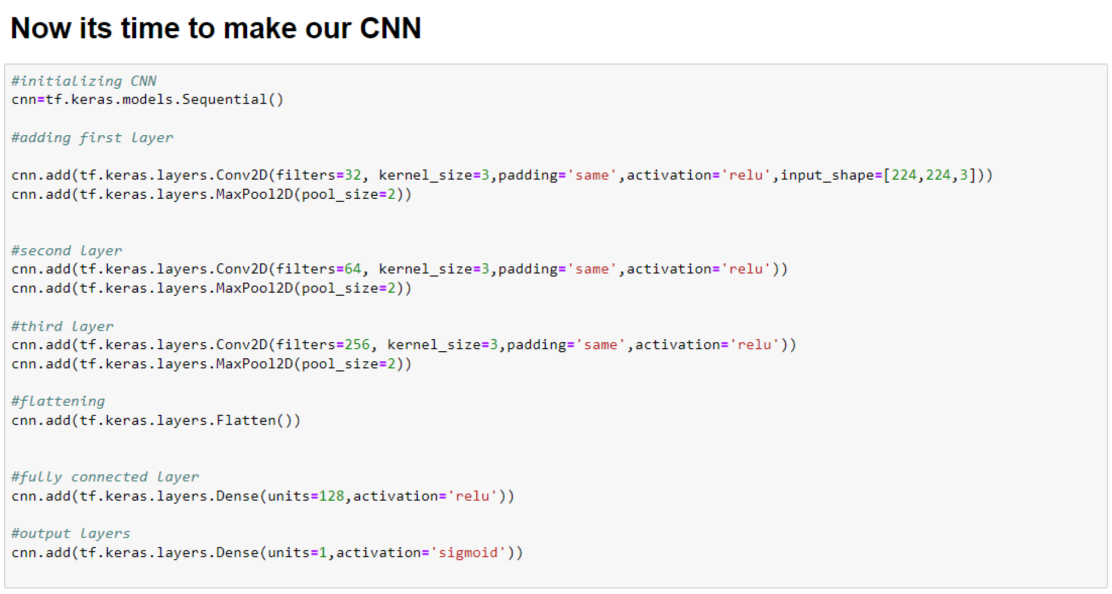
```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Data Augmentation - Creating more data from existing data

```
batch_size=16
training_datagenerator = ImageDataGenerator(rescale = 1./255, horizontal_flip = True, vertical_flip = True, shear_range=0.2,zoom_
                                            width_shift_range=0.2, height_shift_range=0.2, validation_split=0.1)
```

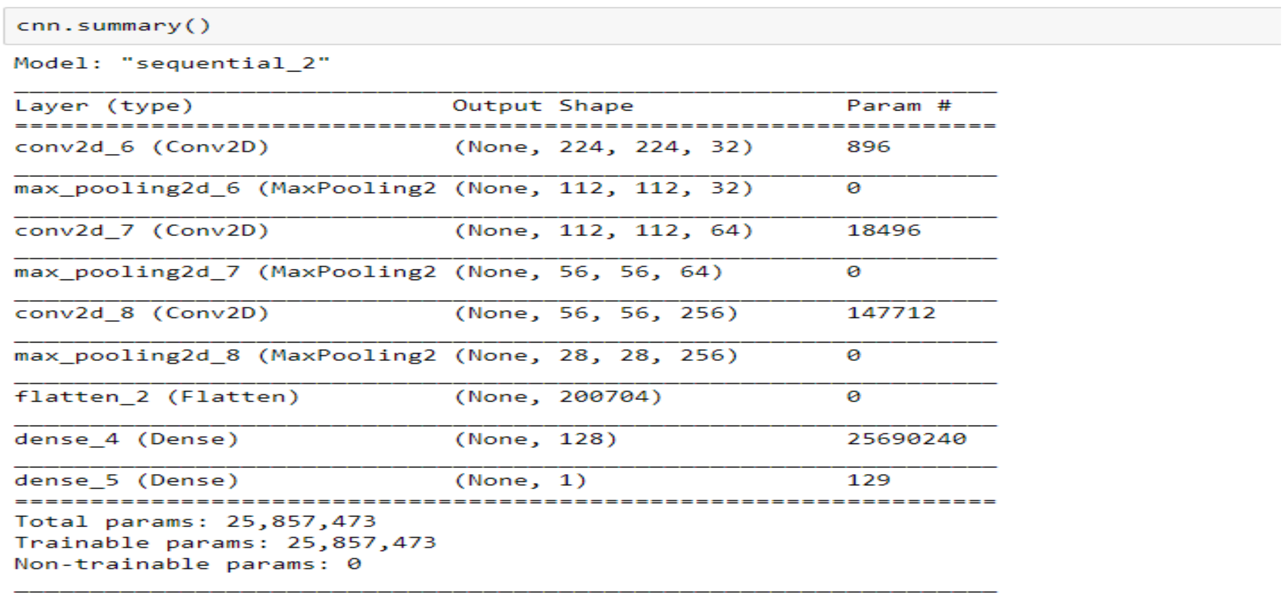
```
train=training_datagenerator.flow_from_directory(r'C:\Downloads\New_Mutants\training_data',
                                                  target_size=(224,224),color_mode='rgb',
                                                  class_mode='binary',batch_size=batch_size,subset='training')
validation=training_datagenerator.flow_from_directory(r'C:\Downloads\New_Mutants\training_data',
                                                  target_size=(224,224),color_mode='rgb',
                                                  class_mode='binary',batch_size=batch_size,subset='validation')
```

```
Found 14162 images belonging to 2 classes.
Found 1573 images belonging to 2 classes.
```



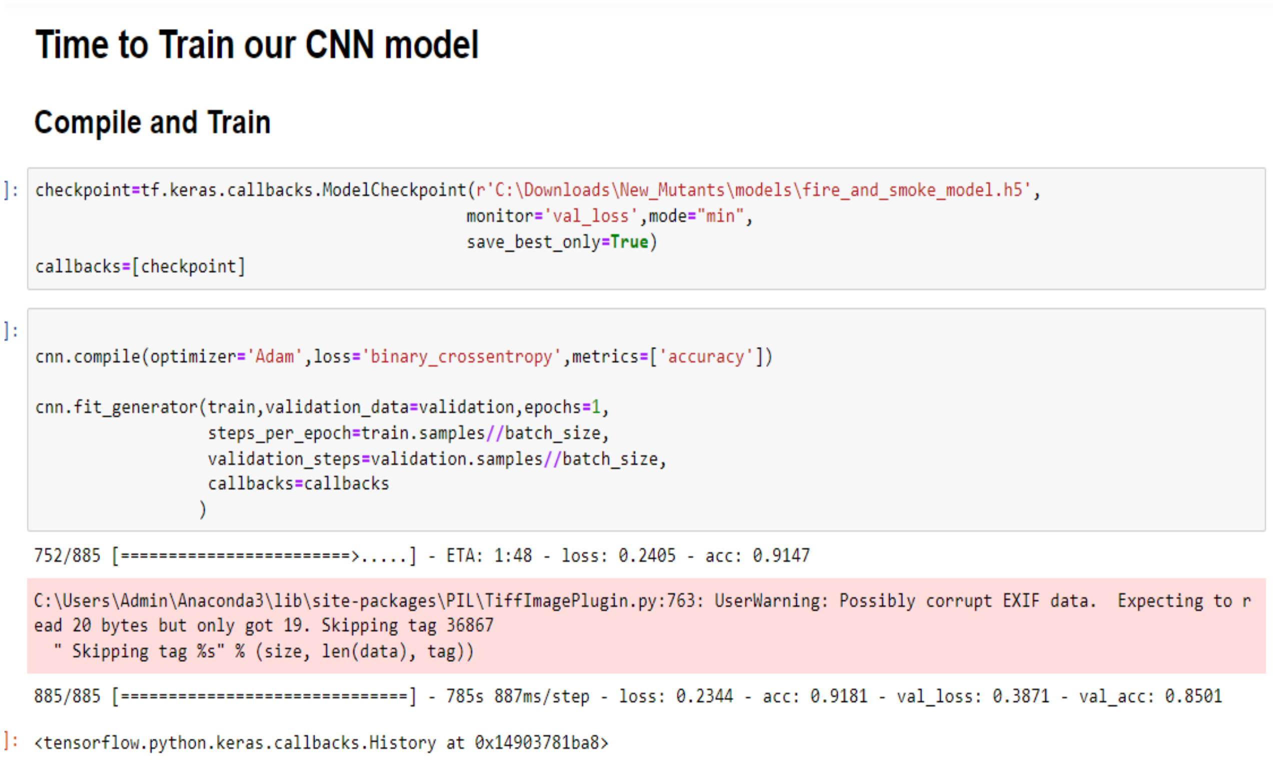
Now its time to make our CNN

```
#initializing CNN
cnn=tf.keras.models.Sequential()

#adding first layer

cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3,padding='same',activation='relu',input_shape=[224,224,3]))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))


#second layer
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))

#third layer
cnn.add(tf.keras.layers.Conv2D(filters=256, kernel_size=3,padding='same',activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2))

#flattening
cnn.add(tf.keras.layers.Flatten())


#fully connected layer
cnn.add(tf.keras.layers.Dense(units=128,activation='relu'))

#output layers
cnn.add(tf.keras.layers.Dense(units=1,activation='sigmoid'))
```



```
cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_6 (Conv2D)            (None, 224, 224, 32)      896       
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 112, 112, 32)      0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 112, 112, 64)      18496     
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 56, 56, 64)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 56, 56, 256)       147712    
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 28, 28, 256)       0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 200704)            0         
_________________________________________________________________
dense_4 (Dense)              (None, 128)               25690240  
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129       
=================================================================
Total params: 25,857,473
Trainable params: 25,857,473
Non-trainable params: 0
_________________________________________________________________
```



Time to Train our CNN model

Compile and Train

```
]: checkpoint=tf.keras.callbacks.ModelCheckpoint(r'C:\Downloads\New_Mutants\models\fire_and_smoke_model.h5',
                                               monitor='val_loss',mode="min",
                                               save_best_only=True)
   callbacks=[checkpoint]
```

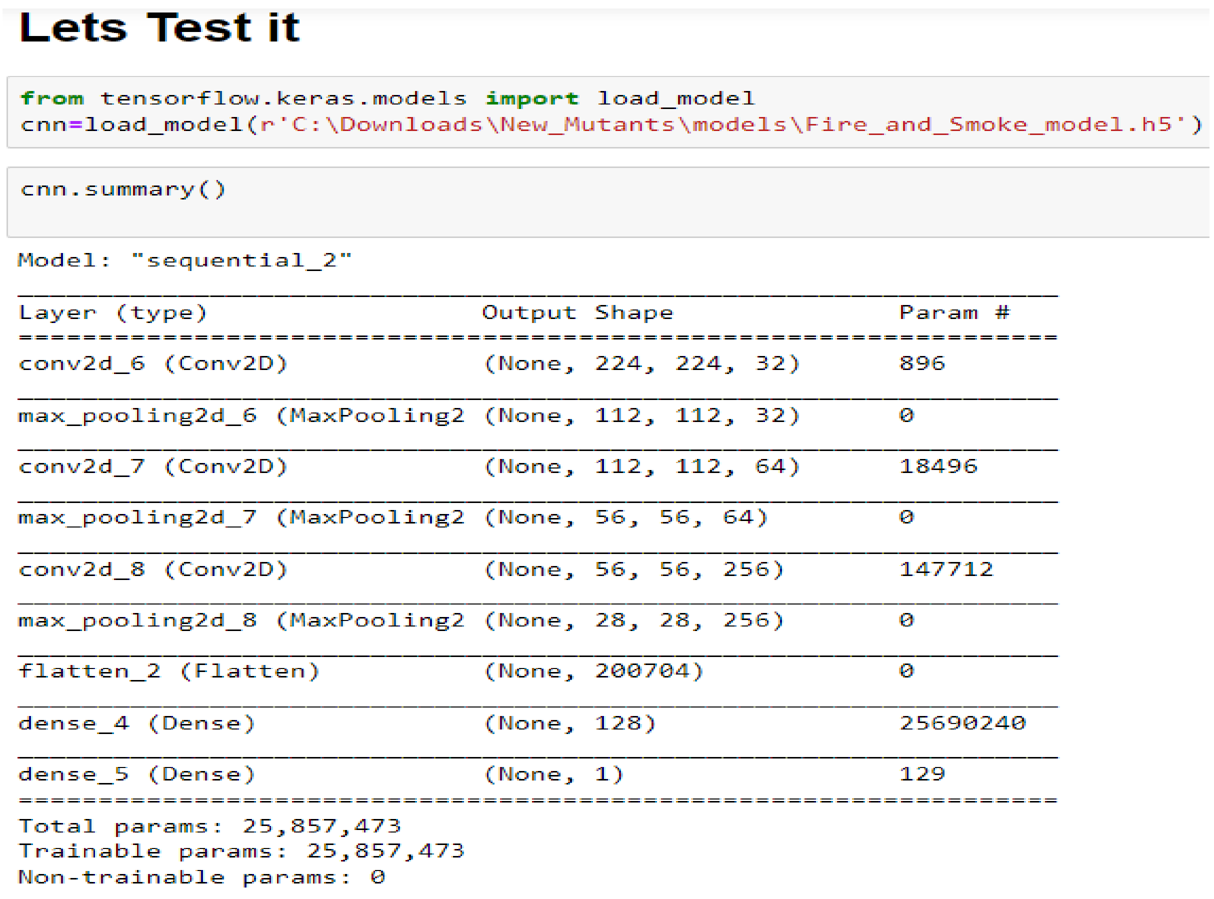
```
]: 
   cnn.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])

   cnn.fit_generator(train,validation_data=validation,epochs=1,
                     steps_per_epoch=train.samples//batch_size,
                     validation_steps=validation.samples//batch_size,
                     callbacks=callbacks
                    )
```

```
752/885 [========================>.....] - ETA: 1:48 - loss: 0.2405 - acc: 0.9147
C:\Users\Admin\Anaconda3\lib\site-packages\PIL\TiffImagePlugin.py:763: UserWarning: Possibly corrupt EXIF data.  Expecting to r
ead 20 bytes but only got 19. Skipping tag 36867
  " Skipping tag %s" % (size, len(data), tag))
885/885 [==============================] - 785s 887ms/step - loss: 0.2344 - acc: 0.9181 - val_loss: 0.3871 - val_acc: 0.8501
```

```
]: <tensorflow.python.keras.callbacks.History at 0x14903781ba8>
```

We have obtained over 91% of accuracy.

3.4.2 Testing CNN model

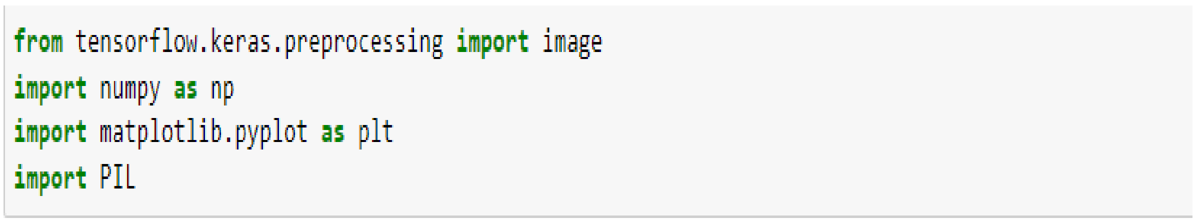


Lets Test it

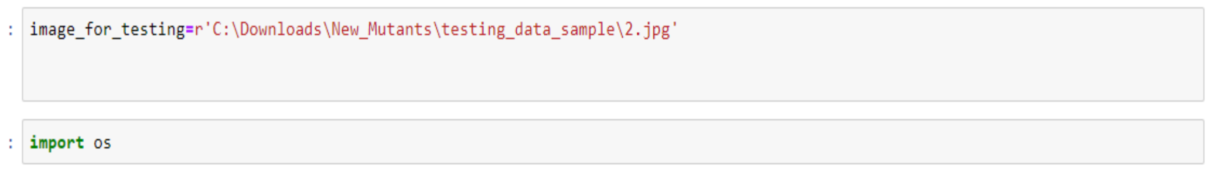
```
from tensorflow.keras.models import load_model
cnn=load_model(r'C:\Downloads\New_Mutants\models\Fire_and_Smoke_model.h5')
```

```
cnn.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 224, 224, 32)      896
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 56, 56, 256)       147712
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 28, 28, 256)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               25690240
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 129
=================================================================
Total params: 25,857,473
Trainable params: 25,857,473
Non-trainable params: 0
```

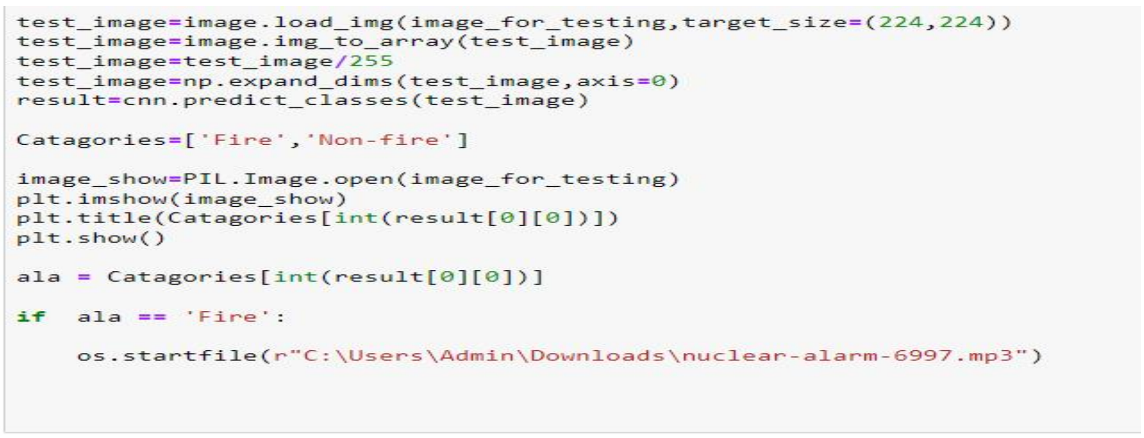


```
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt
import PIL
```

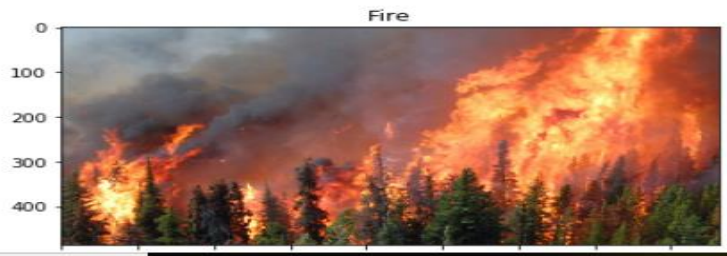


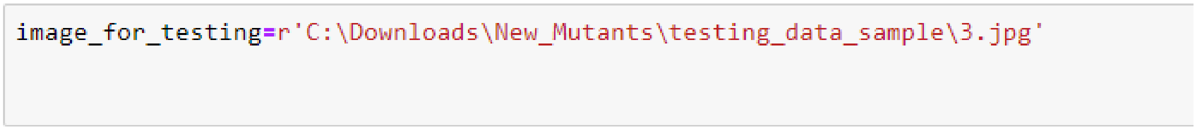
```
image_for_testing=r'C:\Downloads\New_Mutants\testing_data_sample\2.jpg'
```

```
import os
```

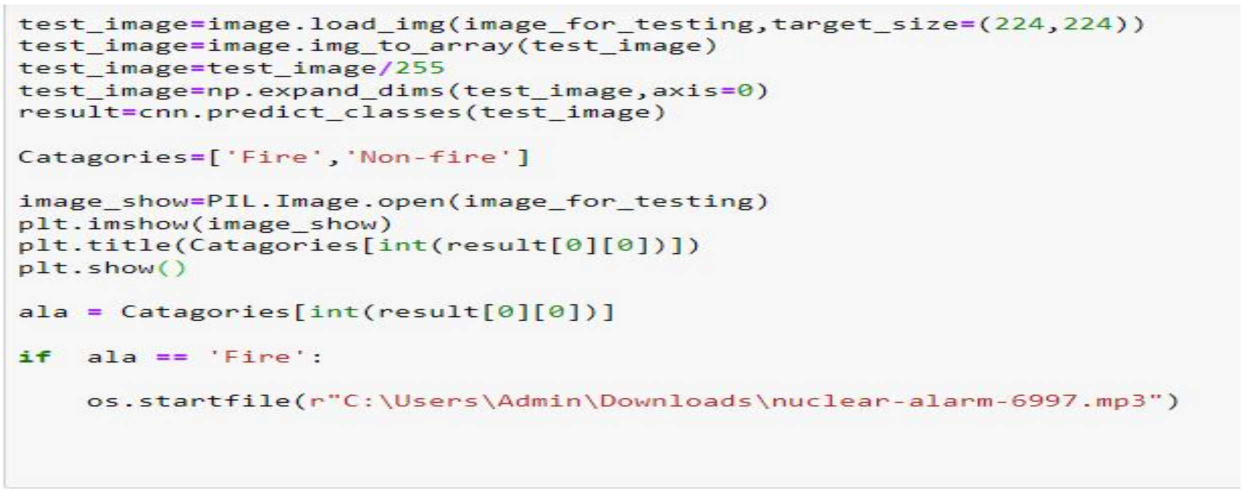


```
test_image=image.load_img(image_for_testing,target_size=(224,224))
test_image=image.img_to_array(test_image)
test_image=test_image/255
test_image=np.expand_dims(test_image,axis=0)
result=cnn.predict_classes(test_image)

Catagories=['Fire','Non-fire']

image_show=PIL.Image.open(image_for_testing)
plt.imshow(image_show)
plt.title(Catagories[int(result[0][0])])
plt.show()

ala = Catagories[int(result[0][0])]

if  ala == 'Fire':

    os.startfile(r"C:\Users\Admin\Downloads\nuclear-alarm-6997.mp3")
```

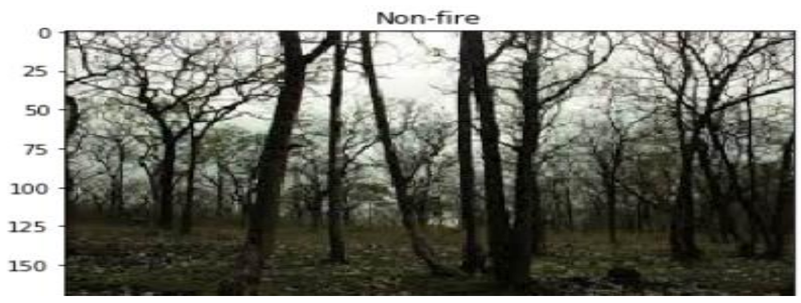




```
image_for_testing=r'C:\Downloads\New_Mutants\testing_data_sample\3.jpg'
```



```
test_image=image.load_img(image_for_testing,target_size=(224,224))
test_image=image.img_to_array(test_image)
test_image=test_image/255
test_image=np.expand_dims(test_image,axis=0)
result=cnn.predict_classes(test_image)

Catagories=['Fire','Non-fire']

image_show=PIL.Image.open(image_for_testing)
plt.imshow(image_show)
plt.title(Catagories[int(result[0][0])])
plt.show()

ala = Catagories[int(result[0][0])]

if  ala == 'Fire':

    os.startfile(r"C:\Users\Admin\Downloads\nuclear-alarm-6997.mp3")
```



3.5 Alarm System:

The main advantage and function of a fire alarm system is to ensure ultimate safety. So, we have implemented this system in our model. Our team's main goal in developing this project is to alert people when a fire starts in their home, forest, or building. So we created a small alarm system in which when a fire image is detected, the alarm set for fire emergency rings. Most people can be alerted if a fire breaks out in their neighbourhood by using this method.

3.6 Challenges:

Because the CNN model requires a large amount of data, obtaining a large amount of data proved difficult. Attempting to implement it in our laptop was another challenge, but the college lab system came to our rescue. There were misclassifications in the images because there was only a small amount of fire detected in the image.

CHAPTER 4
SUMMARY

4.1 Ups and Downs

The best part of our project was that we were able to work with two models completely i.e. CNN and VGG16 and to some extent we have some idea on Yolo.

The workflow was very smooth with ups and downs. There were moments where we were on cloud nine for obtaining the output but there were few moments we were like fish out of water. At the end as a team we were clear about our goal and could march towards it successfully.

In CNN model the biggest success was implementing alarm when it detects fire. As our project aim was to raise an alarm at the moment when there is a fire.

Similarly we were able to detect the misclassified images in VGG16.

Some of the fish out of water moments were as follows: Even though there was a spark of fire out model failed to detect the fire. So we have to remove those images.

As we know that Yolo algorithm could take video of the fire as input and we were on top of our toes to implement it but due to some input issues and some of the errors did not allow us to proceed further.

In place of alarm we wanted to send an Email notification. Even though this was successful we felt the need for immediate action when fire was detected so we set up alarm.

With all these as an individual team member learned a lot.

4.2 Eureka

Bravo!! Alarm is working.

Oh No!! We are getting better accuracy.

4.3 Learning

It was a knowledge gaining experience. Both models have widened our knowledge in the field of machine learning. Today we are happy for learning the basics of two models. We have learned to fix the errors that occurred during the project with the help of our fellow mates.

We have learned the working of CNN as well as VGG16. Each of it has its own features, uses and drawbacks.

Apart from the project we have learned to work in the team by lifting up each others and passing on our knowledge.

As we have worked with two models and if someone asks which was better we will definitely suggest CNN model as we were able to fix alarm when model detects fire.

4.4 Use Cases:

Other than fire detection VGG16 can also be used in disease diagnosis using medical imaging like x-ray or MRI. It can also be used in recognizing street signs from a moving vehicle.

In view of the CNN model’s reasonable accuracy for fire detection, its size, and the rate of false alarms, the system can be helpful to disaster management teams in controlling fire disasters in a short time. Through this we can avoid huge losses.

CHAPTER 5

SCOPE FOR FUTURE STUDY

5.1 Suggesting YOLO

You only look once (YOLO) is a state-of-the-art, real-time object detection algorithm. The fire detection task aims to identify fire or flame in a video and put a bounding box around it. It gives a demo on how to build a fire detection detector using YOLOv5.Unlike most of the previous detection algorithms which apply a model to an image at multiple locations andhigh-scoring regions are considered as detection, YOLO uses a completely different approach.

YOLO uses a single network for the evaluation of the entire image, this makes it faster than most of the other algorithms like RCNN and Fast R-CNN. We had already begun to work but due to some errors we could not complete it.

As we were implementing this we found that the fire detection results were fairly good even though the model was trained only for a few epochs.However, we observed that the trained model tends to predict the red emergency light on top of a police car as fire. It might be due to the small number of negative samples in the training dataset.

So there is a room for improvement for detection of fire using Yolo.

5.2 Future direction for CNN and VGG16 model

The classification in our model is binary classification. Therefore we can add smoke images too so that we can still improve the model for the better results. When it comes for VGG16 the training time is more for the VGG16 we can find some ways for the better training procedures. Adding alarm or an notification to the VGG16 model when it detects fire is also of a great help.

Two major drawbacks with VGG16 we faced are: Firstly It is painfully slow to train. Secondly the network architecture weights themselves are quite large. Therefore we can focus on overcoming these drawbacks as we too have tried by reducing the number of images but we weren’t successful.

CNN model mainly focuses on the detection of fire scenes under observation. Future studies may focus on deploying the model into raspberry pi and using necessary support packages to detect the real time fire by making challenging and specific scene understanding datasets for fire detection methods and detailed experiments.

CHAPTER 6

CONCLUSION

We feel grateful for whatever we have learned and we have found some room for improvement in order to do well. We feel that doing some experiment with CNN Model would be our next step as it can help small vendors to identify the fire easily and help to prevent major loss.

We thank Dr Srinivas Bhogle for guiding us and give us the opportunity to do work with this topic. We have we have met the requirements.