DEEP FOREST FIRE DETECTION USING DEEP LEARING

Submitted in partial fulfilment of the requirements for the degree of

MASTER OF TECHNOLOGY

in

Computer Science and Engineering

by

MUNUGAPATI PRAVEEN: 14K91D5815

Under the guidance of
Dr. A SURESH RAO SIR (M. TECH, PHD)

HOD C.S.E

(DEAN OF COLLEGE & VICE PRINCIPLE)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
TKR COLLEGE OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
ACCREDITED BY NBA AND NAAC WITH 'A' GRADE)
Medbowli, Meerpet, Saroornagar, Hyderabad-500097

DECLARATION BY THE CANDIDATE

I, Mr. **MUNUGAPATI PRAVEEN** bearing Hall Ticket Number: **14K91D5815** hereby declare that the major project report titled **DEEP FOREST FIRE DETECTION USING DEEP LEARING** under the guidance of **Dr. A SURESH RAO SIR HOD** in Department of Computer Science and Engineering is submitted in partial fulfilment of the requirements for the award of the degree of Master of Technology in Computer Science and Engineering.

Place: Meerpet **MUNUGAPATI PRAVEEN:14K91D5815**

Date:

CERTIFICATE

This is to certify that the main project report entitled **DEEP FOREST FIRE DETECTION USING DEEP LEARING,** being submitted by Mr. **MUNUGAPATI PRAVEEN** bearing Hall Ticket Number **14K91D5815** in partial fulfilment of requirements for the award of degree of Master of Technology in Computer Science and Engineering, to the TKR College of Engineering and Technology is a record of bonafide work carried out by him/her under my guidance and supervision.

Signature of the Guide
Dr. A SURESH RAO SIR

Signature of the HOD
Dr. A SURESH RAO SIR

Place : Meerpet

Date :

TABLE OF CONTENTS

ABSTRACT

Early forest fire detection is of great importance to avoid the huge damage of forests caused by fires. Early fire detection focuses on smoke detection. The forest area is gradually decreased because of increasing forest fire and human activities. The satellite sensor is used to collect the forest thermal image in different places and analyse the data in these images to detect the fire region if they occur. Image processing technique can effectively predict the fire in the forest. The input image is pre-processed to enhance the image quality, because the input image has the noise, so the pre-processing technique is used to eliminate the noise in this system and enhance the image quality.

The pre-processed image is taking to the segmentation process; it processes the image to adjacent the forest sub-area. In this system, the affected area is separately detected, and it gives the accurate forest fire in this system because the output image intensity is better to stabilize the average value of the image. In our proposed system we propose a deep learning method that uses a Convolutional Neural Network (CNN) to predict the forest fire detection.

The convolutional layer is the main building block of the convolutional neural network. Usually, the layers of the network are fully connected in which a neuron in the next layer is connected to all the neurons in the previous layer. We are going to detect the fire in the forest result based on the accuracy which we get in train and test of the dataset based CNN algorithm using that we show the graph result.

ACKNOWLEDGEMENTS

The joy and exhilaration that come with completing a design successfully would be absent if we failed to admit the people whose support and guidance made it possible and who have culminated our sweats with success.

I am grateful to my internal guide **Dr. A. Suresh Rao,** HOD in the Department of Computer Science and engineering at TKR College of engineering and technology for his or her assistance and direction during the completion of my thesis or dissertation.

For his assistance and direction during the writing of our thesis and dissertation, department head, **Dr. A. Suresh Rao,** Professor of Computer Science and engineering at the TKR College of engineering and technology is also to be tanked.

I would like to express my sincere thanks to **Dr. D. V. Ravi Shankar,** the principle of the TKR College of engineering and technology for allowing me to work on this thesis or dissertation.

Eventually, I would want to express my gratefulness to everyone who helped me complete this thesis or discussion. For their stimulant and support, my family and musketeers earn a special word of thanks.

Place: Meerpet **MUNUGAPATI PRAVEEN:14K91D5815**

Date:

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Motivation

Wildfires are devastating events causing immense damage to life, property, and ecosystems. Early detection is crucial for minimizing wildfire impacts. This is where Deep Learning (DL) comes in as a powerful tool for wildfire detection. Here's why DL is particularly motivating for this purpose:

Accuracy: DL models, especially Convolutional Neural Networks (CNNs), excel at identifying patterns in imagery. They can learn to distinguish fire and smoke from other visual elements in real-time camera footage or satellite data, leading to highly accurate detection.

Early Detection: DL models can analyse subtle variations in colour, temperature, and texture that might be indicative of an incipient wildfire. This allows for earlier detection compared to traditional methods, providing a crucial head start for response efforts.

24/7 Monitoring: DL models can be integrated into automated systems for continuous monitoring of forests and other fire-prone regions. This tireless vigilance surpasses human capabilities and reduces the dependence on manual intervention.

Data Integration: DL can handle various data sources like satellite imagery, drone footage, and ground sensor readings. By incorporating this multi-dimensional data, DL models can create a more comprehensive picture of fire risk and activity.

Adaptability: DL models can be continuously trained on new data, allowing them to adapt to changing environmental conditions and improve their wildfire detection accuracy over time.

In conclusion, DL offers a promising approach to wildfire detection due to its potential for high accuracy, early identification, real-time monitoring, and continuous improvement. This technology has the potential to revolutionize wildfire management and significantly reduce their devastating consequences.

1.2 Problem definition

Forest fire is a global environmental problem causing extensive damage every year. According to the International Union for Conservation of Nature (IUCN) Report "Global Review of Forest Fire 2000," wildfire is a natural phenomenon. However, over 90% of all wild land fires are due to human action causing significant forest loss, that is 6–14 million hectares of forest per annum, and 30% of the CO2 in the atmosphere produced from forest fires. This leads to enormous economic losses, damage to environmental, recreational and amenity assets, global warming and loss of life. There is a strong recognition that action is needed to catalyse a strategic international response to forest fires.

Recently, a record-breaking fire was seen ripping through the Amazon and Australia, which caused great damage to the ecosystem and contributed to the increase in air pollution and cause a death of almost 480 million animals. Even though it could have been easily prevented by the authorities, due to lack of proper surveillance and human negligence it worsened.

1.3 Limitations of Existing System

Satellite-Based Systems

Earth-orbiting satellites and even air-floating instruments were used for forest fire observation and detection. Satellite images collected by two main satellites launched for forest fire detection purposes, the advanced high-resolution radiometer 2, and the spectroradiometer of moderate resolution. Sadly, these satellites can provide pictures of the Earth's regions every two days, which is a long time for fire scanning; weather conditions can affect the quality of satellite images as well.

Optical Sensor and Digital Camera

There are actually two different types of sensor networks for fire detection, camera tracking and wireless sensor network. The advent of sensors, digital camera, image processing, and industrial computers led to the development of an electronic, automatic early detection and forest fire warning system.

Different types of detection sensors can be used in terrestrial systems :

- Video camera, sensitive to visible smoke spectrum recognizable during the day and fire identifiable at night
- .infrared thermal imaging cameras focused on fire heat flow detection
- IR spectrometers to recognize smoke spectrum characteristics
- light detection and range systems — measuring laser rays emitted from smoke particles.

Disadvantages:

- The performance of the model is very less, it achieved just 79% accuracy.
- The model has to use a greater number of feature engineering techniques and ensemble methods in order to increase the performance.

1.4 Proposed System

In this project, an ensemble prediction system is proposed for detecting user's emotions from their tweets. The ensemble system consists of voting classifier with Logistic Regression, Stochastic gradient descent and Linear SVM but where as in existing system the authors used voting classifier ensemble with LR and SGD. Here feature extraction techniques TF, TF-IDF are used in order to convert the pre-processed tweets into real valued vectors. During preprocessing, tokenization, lemmatization of data will be carried out. To train a supervised ML classification model with texts, the data needs to be converted into a meaningful numerical representation which is known as vectorization.

Chapter 2

LITERATURE REVIEW

2.1 Review of Literature

Deep learning has emerged as a powerful tool for wildfire detection in recent years. Convolutional Neural Networks (CNNs), a type of deep learning architecture, have shown significant promise in automating the process of identifying wildfires in imagery collected from satellites, drones, and ground-based cameras.

VGG16, a pre-trained deep learning model developed by Simonyan and Zisserman (2014) [1], is one of the CNN architectures explored for wildfire detection. Its effectiveness stems from its ability to learn features directly from image data, eliminating the need for manual feature engineering.

Here's a breakdown of key findings from the literature:

- **Transfer Learning:** VGG16 is often employed using transfer learning. This technique leverages the weights pre-trained on a large image dataset (like ImageNet) and fine-tunes them for the specific task of wildfire detection. This approach proves advantageous when dealing with limited wildfire image datasets, as it reduces training time and improves accuracy compared to training from scratch .

- **Performance:** Studies report varying degrees of success with VGG16 for wildfire detection. Accuracy can range from 79% to 98% depending on the dataset, pre-processing techniques, and hyperparameter tuning While VGG16 performs well, it may be surpassed by other CNN architectures like Xception or ResNet in certain scenarios .

- **Challenges:** Some limitations are associated with using VGG16 for wildfire detection. The model's complexity can lead to high computational costs, and its deep architecture might be susceptible to overfitting, especially with small datasets .

Overall, VGG16 offers a viable approach for wildfire detection with transfer learning. However, it's crucial to consider its limitations and explore potentially superior performing architectures depending on the specific application and dataset characteristics.

Implementation & Methodology:

1. Data Acquisition and Pre-processing

- **Data Collection:**
    - Source images from various sources like satellites, drones, or ground-based cameras. High-resolution imagery is preferred for capturing subtle details of wildfires.
    - Public datasets specifically curated for wildfire detection can be a good starting point (e.g., NASA Fire Information for Resource Management System [FIRMS]).
- **Data Pre-processing:**
    - Annotate the collected images to identify regions containing wildfires (positive samples) and those without wildfires (negative samples). This annotation process is crucial for supervised learning, where the model learns from labeled data.
    - Preprocess the images for consistency. This might involve resizing images to a standard format, adjusting color channels, or applying normalization techniques to ensure the model focuses on relevant features and reduces training time.
    - Consider data augmentation techniques to artificially increase the dataset size and improve model generalization. Techniques like random cropping, flipping, or adding noise can be employed without altering the underlying information about wildfires.

2. Model Selection and Training

- Deep Learning Model Selection:
    - Convolutional Neural Networks (CNNs) are the preferred choice for image classification tasks like wildfire detection. Popular CNN architectures for this application include:

        - VGG16/VGG19: Well-established CNN architectures demonstrating good performance in various image classification tasks.
        - ResNet: A deeper and more complex CNN architecture known for its ability to handle large datasets and avoid overfitting.
        - You Only Look Once (YOLO): A real-time object detection model suitable for applications requiring fast processing and localization of wildfires.
    - Pre-trained models can be leveraged as a starting point for transfer learning. These models, trained on vast image datasets, can be fine-tuned for the specific task of wildfire detection, significantly reducing training time and computational resources.

- **Model Training:**
    - Split the pre-processed dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps monitor performance during training and prevent overfitting, and the testing set evaluates the final model's generalizability on unseen data.
    - Train the chosen model on the training set using an appropriate optimizer (e.g., Adam) and loss function (e.g., binary cross-entropy for binary classification of fire/no-fire).
    - Monitor the model's performance on the validation set during training. Early stopping techniques can be used to prevent overfitting if the validation accuracy plateaus or starts to decline.
    - Fine-tune hyperparameters (learning rate, batch size, etc.) to optimize the model's performance.

3. Model Evaluation and Testing

- Performance Metrics:

- Evaluate the trained model's performance on the unseen testing set using metrics like accuracy, precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly classify wildfire and non-wildfire images.

- Analyse confusion matrices to identify potential biases or weaknesses in the model's performance. A confusion matrix visualizes how often the model correctly or incorrectly classified wildfire and non-wildfire images.

- **Addressing Biases and Overfitting:**

  - If the model exhibits biases or struggles to generalize well, consider techniques like data augmentation, class balancing (if dealing with imbalanced datasets), or regularization techniques (e.g., dropout layers) to improve model robustness.

4. Deployment and Integration

- Deployment Considerations:

  - Choose a suitable platform for deploying the trained model. Options include deploying it on a cloud server for real-time processing of image streams or integrating it with existing infrastructure like fire monitoring systems.

  - Consider the computational resources required by the chosen model and optimize for efficiency if deployment on resource-constrained devices is necessary.

- Integration:

  - Integrate the deployed model with an image acquisition system (cameras, drones) to receive a continuous stream of images for processing.

  - Develop a user interface for visualizing the results and potentially interacting with the system (e.g., acknowledging false positives).

  - Consider integrating the system with alert systems to notify relevant authorities upon wildfire detection.

This approach provides a foundational framework for implementing a wildfire detection system using deep learning. The specific details of each step (data selection, model architecture, deployment) can be tailored and optimized based on project requirements and resource constraints.

Chapter 3

REQUIREMENT ANALYSIS

3.1 Functional Requirements

Requirements engineering or requirements analysis is the process of identifying the requirements and expectations for a new product. In order to define expectations, handle difficulties, and record all of the key needs for the product, it necessitates routine communication with the stakeholders and end-users.

The link between the system's input and output is described in the functional requirements, which also specify which output file should be created from a certain input file. A thorough explanation of all data inputs, their sources, and the range of acceptable inputs must be included in every functional requirement

3.2 Non-Functional Requirements

Describe any system components that are visible to users but are not directly connected to how the system functions. Quantitative requirements like as precision (i.e., how precisely the system's numerical responses are) and reaction time (i.e., how quickly the system responds to user requests) are examples of non-functional needs.

- Portability
- Reliability
- Usability
- Time Constraints
- Responsive design should be implemented
- Space Constraints
- Performance
- Standards
- Ethics

3.3 Software Requirement Specifications

3.3.1 Software Requirements

Operating system : Windows 10.

Coding Language : Python.

Front-End : Python.

Back-End : Flask

Designing : Html, CSS, JavaScript.

3.3.2 Hardware Requirements

Processor - Pentium IV

RAM - 4 GB (min)

Hard Disk - 20 GB

Key Board - Standard Windows Keyboard

Mouse - Two or Three Button Mouse

Monitor - SVGA

3.4 Software Development Life Cycle

The Systems Development Life Cycle (SDLC) is a concept used in the fields of systems engineering, information systems, and software engineering to describe the process of creating new systems or upgrading existing ones, as well as the concepts and techniques that go into their design.

3.4.1 Requirements Analysis and Design

Analyses are used to compile the system's requirements. This stage comprises a careful analysis of the organizational business needs. The business procedure could change. The focus of design is on high-level design, such as what programmers are required and how they will interact, low-level design (how

individual programs will run), interface design (how will interfaces appear), and data design (what data will be required). During these times, the overall architecture of the software is established. Design and analysis play a significant role throughout the whole development cycle. Any design error might be extremely expensive to repair later on in the software development process. Extreme care is used at this phase.

Extreme care is used at this phase.

3.4.2 Implementation

This stage involves turning the designs into code. Computer programs can be written in a conventional programming language or with the aid of an application generator. Compilers, interpreters, and debuggers are some of the programming tools used to create the code. Different high-level programming languages are used for coding, including PYTHON 3.6 and Anaconda Cloud. The type of application determines the best programming language to use. An organized system is created.

3.4.3 Testing

This stage involves testing the system. Typically, software are written as a series of independent modules, each of which is carefully tested individually. After that, the system is tested as a whole. The separate parts are assembled and evaluated together. The system is tested to make sure the interfaces between modules function (integration testing), that it operates on the designated platform and with the anticipated volume of data (volume testing), and that it achieves what the user wants it to do (acceptance/beta testing).

3.4.4 Maintenance

The system will eventually need maintenance. Software will very definitely change after being sent to the user. Several causes have contributed to the transformation. The system may experience change as a result of unexpected input values. Furthermore, changes to the system could have an immediate effect on how the program functions. The program should be able to adapt to any changes that may come along after implementation.

3.4.5 SDLC Methodologies

This document, which specifies every need for the system, is essential to the creation of the system's life cycle (SDLC). It will serve as the basis for testing and be used by programmers. Future standard modifications will need to have explicit change approval. In his 1988 essay "A spiral Model of Software Development and Enhancement." The SPIRAL MODEL was explained by Barry Boehm. Although this model wasn't the first to examine iterative development, it was the first to provide an explanation for why such models are so well-liked.

The intended duration of the iterations was six months to two years. Each phase starts with a design objective and ends with a customer evaluation of the project's status up to that point. Engineering and analysis work are used at every level of the project with an eye towards its ultimate goal.

The following are the processes for the Spiral Model: The new system requirements are stated as completely as is practical.

1. To do this, it is often necessary to interview many customers who represent all of the various user categories.
2. Whether internal or external, as well as other aspects of the existing system.
3. A preliminary design is created for the new system.
4. To make a first prototype of the new system, the preliminary design is applied.
5. Typically, this is a system that has been reduced in size and comes close to the original.
6. Specifications of the finished product.
    - Evaluating the advantages, disadvantages, and dangers associated with the original prototype. Outlining the demands for the second prototype.
    - Organizing the design of the second prototype.
    - Constructing and testing a second prototype.

The customer might choose to abandon the project entirely if the risk is thought to be too great. Risk factors include everything that might, in the view of the client, lead to a less-than satisfactory final product, including overruns in development expenses, mistakes in running costs, or other elements.

- The current prototype is evaluated similarly to how the last prototype was examined, and if required, a new prototype is made utilizing the previously mentioned four-step process.
- The aforementioned steps are continued until the buyer is satisfied that the redesigned prototype faithfully represents the desired final product.
- The finished system is constructed based on the modified prototype.
- A thorough testing and evaluation of the finished system. Routine maintenance is performed often to prevent widespread failures and save downtime. 3.5 Modules

3.5.1 NumPy

The following operations can be carried out by a developer using NumPy:

- actions on arrays that combine logic and mathematics.
- Shape change using algorithms and Fourier transformations.
- Operations in linear algebra are algebraic operations. Linear algebra and random number generation functions are both included in NumPy.
- A Python-based replacement for MATLAB is NumPy.
- Along with SciPy (Scientific Python) and Matplotlib (a charting toolkit), NumPy is commonly used. This combination usually takes the place of MATLAB, a well-known technical computing environment.

3.5.2 Pandas

Python's Pandas library is free and open source. High-performance data structures and data analysis tools are offered ready for use. Pandas is a widely used data science and analytics package that operates on top of NumPy.

Multi-dimensional arrays and a large variety of mathematical array operations are supported by the low-level data structure known as NumPy. The interface for Pandas is more advanced.

Additionally, it offers robust time series capability and simplified tabular data alignment. Pandas primary data structure is the DataFrame. As a 2-D data structure, it enables the storage and manipulation of tabular data.

On the DataFrame, Pandas offers a robust feature set. As an illustration, consider data alignment, data statistics, data slicing, grouping, merging, concatenating, etc.

Beginning the Pandas Installation Process

Pandas module installation requires Python 2.7 or higher. Use the below command to install it if you are using conda. onda install pandas

Use the command below to install the pandas module if you are using PIP. pip3.7 install pandas

The following line of code should be added to your Python script to import Pandas and NumPy: import pandas as pd. import numpy as np

We must import this dependency as Pandas depends on the NumPy library.

3.5.3 Pickle

The serialisation and deserialization of a Python object structure is accomplished using Python Pickle. Python allows for the pickling of any object to enable disc storage. The object is first serialized by Python Pickle before being transformed into a character stream, which is then filled with all the information needed to recreate the object in another Python script. It should be noted that the documentation claims that the pickle module is not protected from data that has been intentionally or mistakenly created. As a result, never unpickle data that you have obtained from an unauthorized or shady source.

The methods are really ‘straightforward for recovering pickled data. Pickle.load() is the method you must use to do that. The file object you obtain when opening a file in readbinary (rb) mode is the main parameter for the pickle load method. Simple! Surely not. Using the pickle dump code, let's create the code toretrieve the data we pickled.

CHAPTER 4

SYSTEM DESIGN

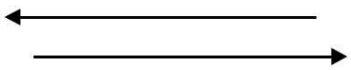
4.1 DFDs and UML diagrams

A data flow diagram (DFD) is a visual tool used to illustrate the movement of data within a system. On the other hand, UML is a modelling language designed for object-oriented software development. It consists of a range of diagrams that can be utilized to represent the structure and behaviour of a software system.

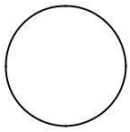
4.1.1 Data Flow Diagram

Data flow diagrams provide a clear illustration of how data is transmitted within a system. They use symbols and arrows to visually indicate the direction of information flow

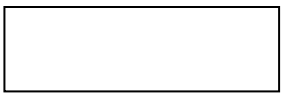
Data flow:



Process:

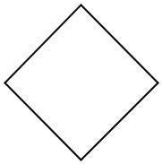


Source:



Data Store:

Rhombus: decision



4.1.2 UML

The UML stands for Unified modeling language, is a standardized general-reason visible modeling language with inside the area of Software Engineering. It is used for specifying, visualizing, constructing, and documenting the primary artifacts of the software program application device. It allows in designing and characterizing, especially those software program application systems that consist of the concept of Object orientation. It describes the walking of the software program application and hardware systems.

Goals of UML

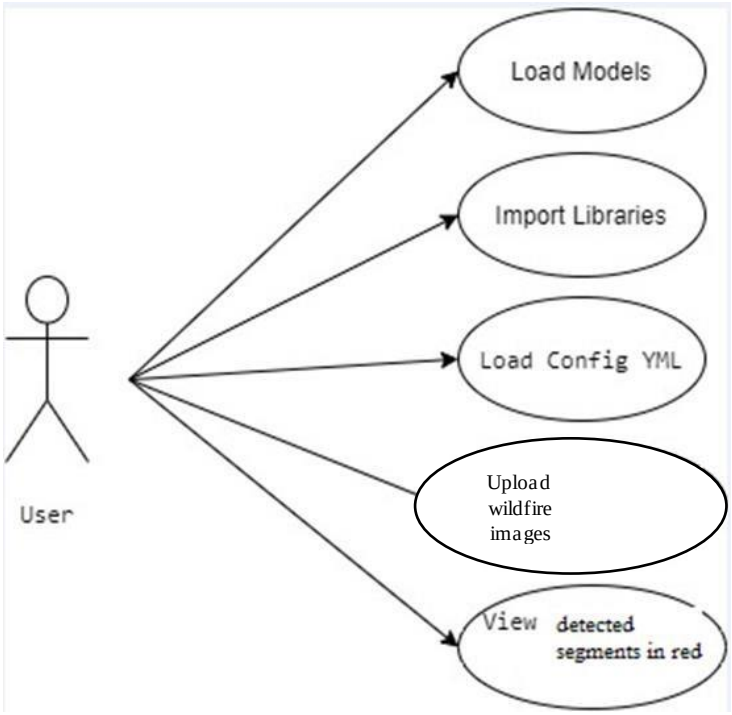
- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts. • Be independent of particular programming languages and development process.

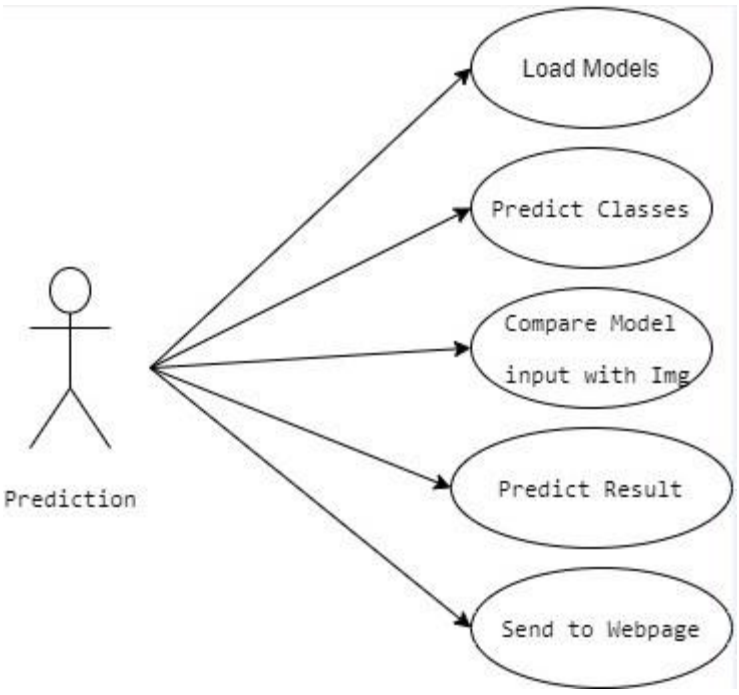
4.1.3 Use Case Diagram

To identify actors in a system, it's important to consider who uses the system regularly, who is responsible for system maintenance, and what external hardware the system interacts with. Use cases represent how the system is utilized from the user's perspective, with the flow of events defining the ideal scenario of user interactions with the system.

Use-case diagrams are used to illustrate the system's behavior and show the relationships between actors and use cases. These diagrams use various communication links, such as associations, generalizations, extensions, and inclusions, to depict the interactions between actors and the system.

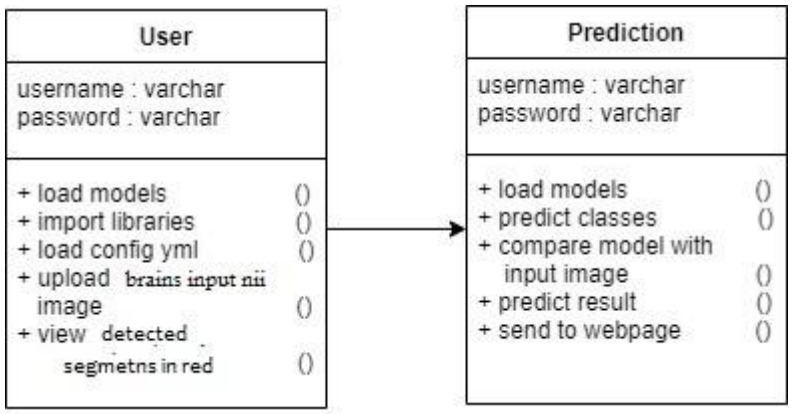
Generalization connections show relationships between actors who rely on each other's functions to accomplish their roles within the system, while extension links represent optional functions that may connect actors to use cases or a group of similar use cases that are not functionally related.


Load Models
Import Libraries
Load Config YML
Upload wildfire images
View detected segments in red
User


Load Models
Predict Classes
Compare Model input with Img
Predict Result
Send to Webpage
Prediction

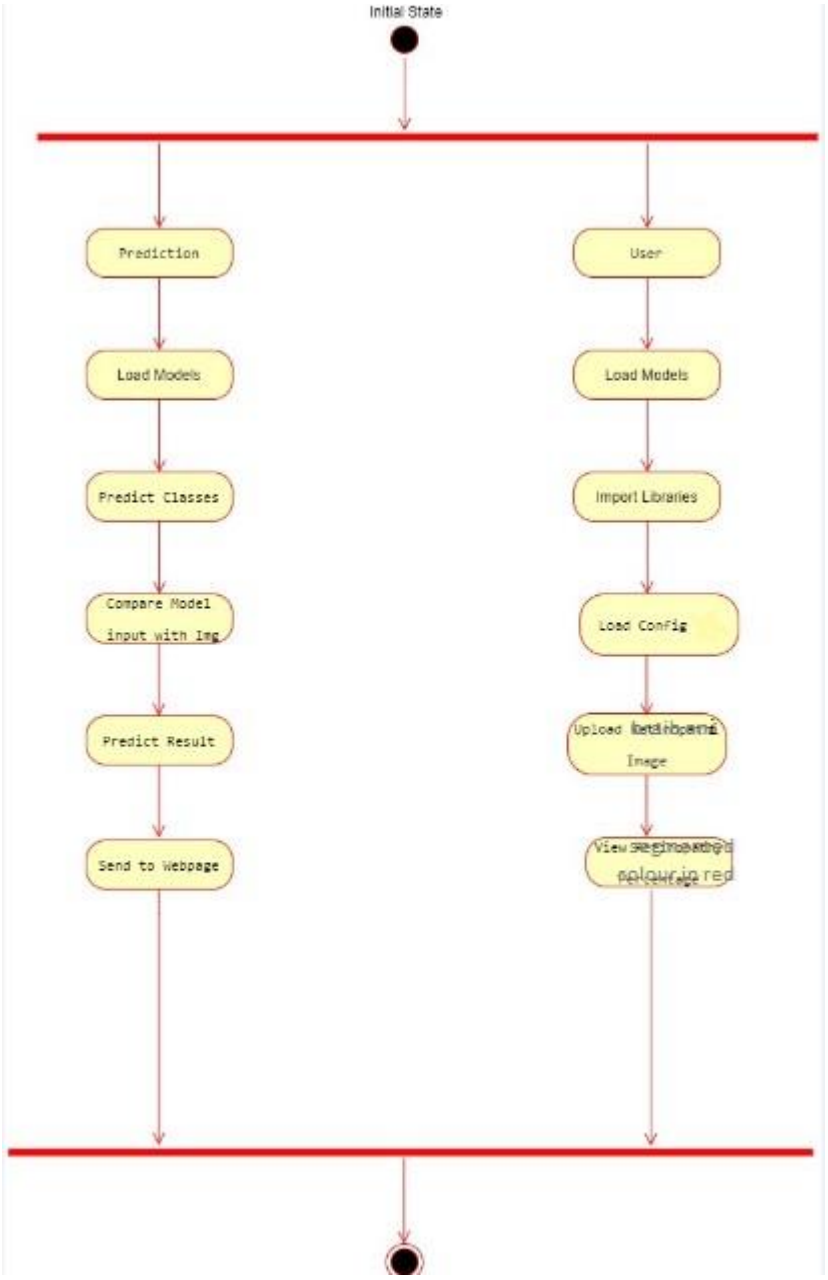
4.1.4 Class Diagram

Class diagrams are a type of UML diagram that provides a static representation of a system. These diagrams illustrate the system's structure by depicting static classes, attributes, and operations. They can be used to describe the rules of a system, and can be mapped to object-oriented programming. Class diagrams depict relationships between classes, such as one-to-one, one-to-many, many-to many, and many-to-one, and can also show inheritance relationships between classes, with any class capable of being extended from another class.



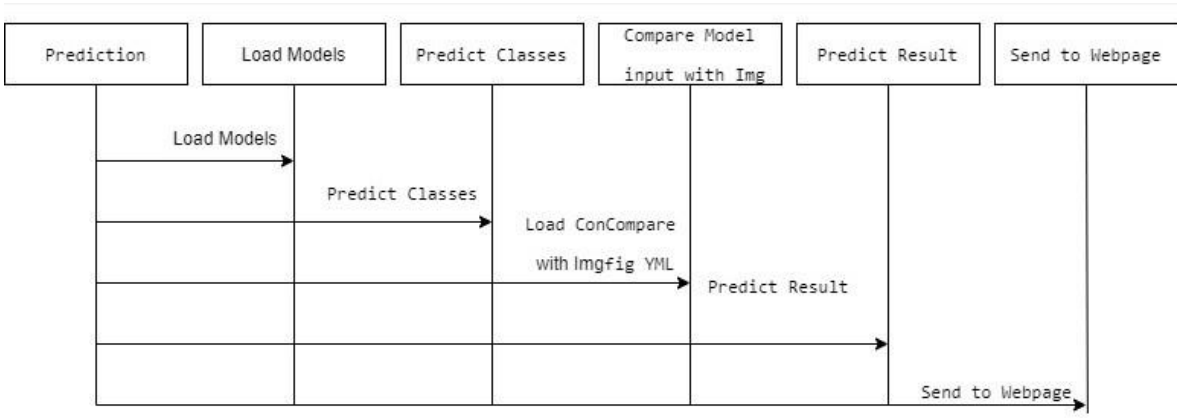
4.1.5 Activity Diagram

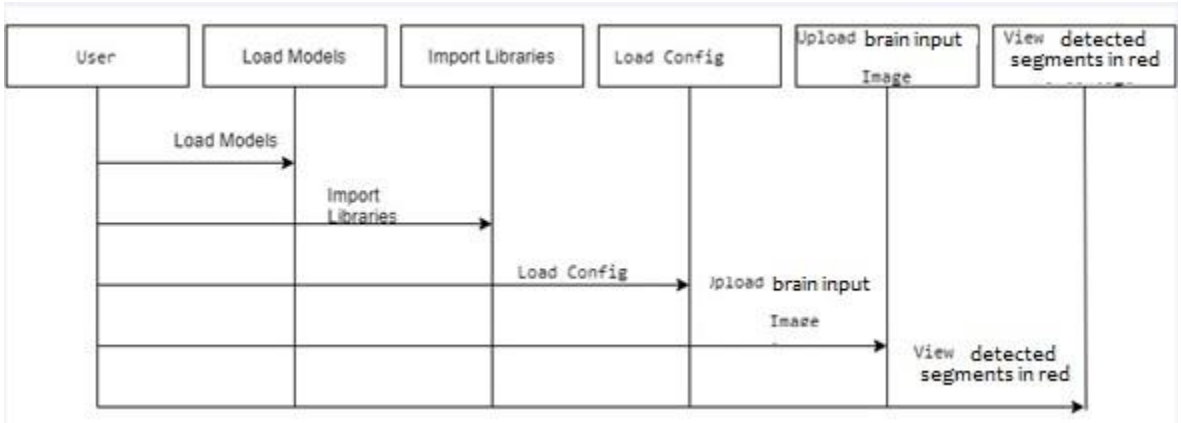
Activity diagrams are graphical representations that illustrate the sequence of actions or control flow within a system, similar to flowcharts or data flow diagrams. They are commonly used in business process modeling and can also be utilized in use case diagrams to depict steps. These diagrams can depict both sequential and concurrent activities and have a starting state and a final state to represent the beginning and end of the process being depicted.



4.1.6 Sequence Diagram

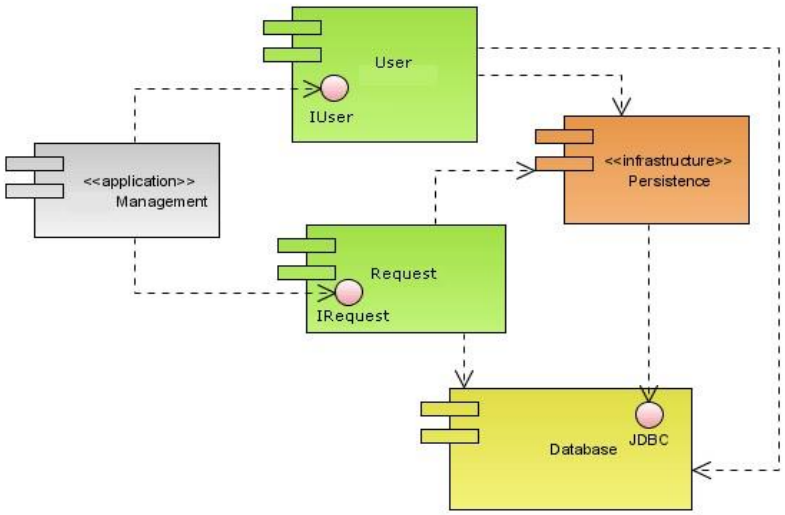
Sequence diagrams are graphical representations that illustrate how processes are executed, showing the collaboration between objects in a system. These diagrams are focused on the timing of interactions, with the vertical axis representing time and displaying the sequence of interactions visually.

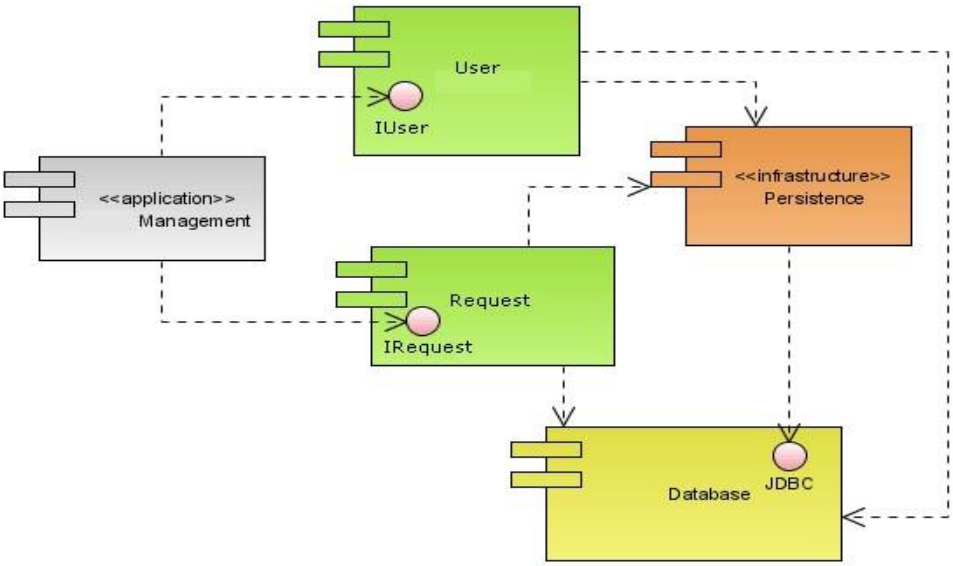




4.1.7 Component Diagram

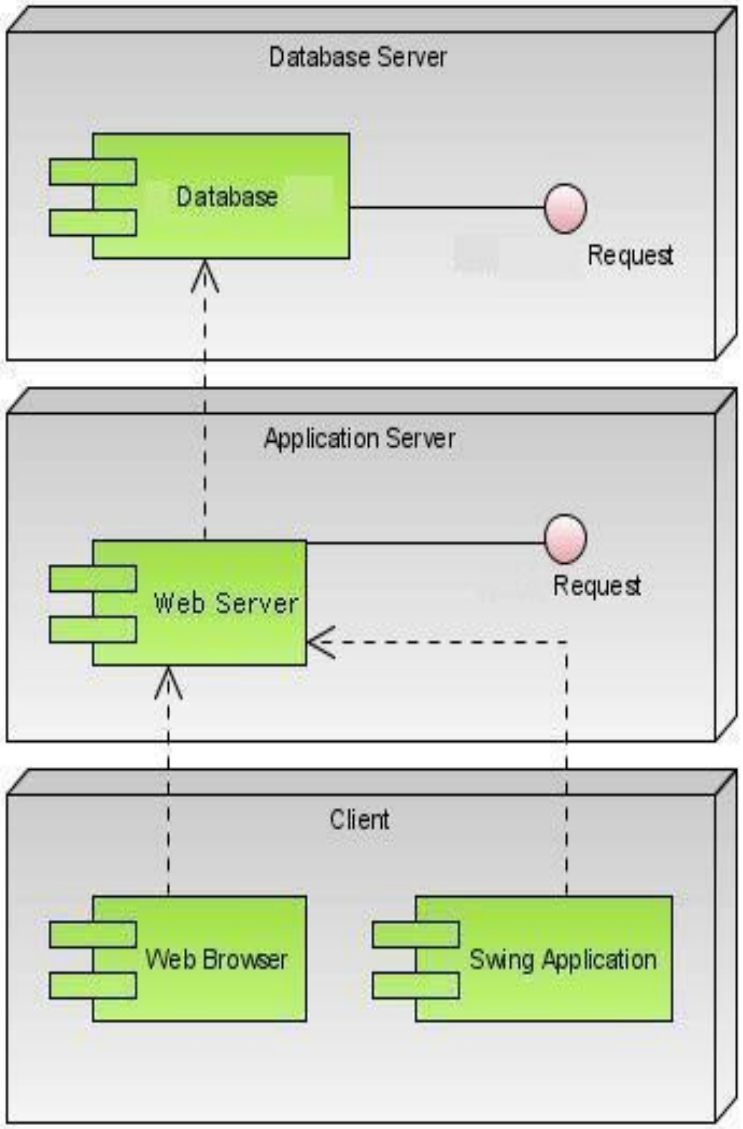
A component diagram is a type of static structure diagram in the Unified Modeling Language that illustrates the organization and interrelationships between software components, including their interfaces, dependencies, and associations. It provides a visual representation of the components that constitute a system and their relationships, facilitating the identification of independent parts that can be developed separately and the management of system complexity. Component diagrams are frequently used in the initial stages of system development to depict system architecture. Components are typically depicted as rectangles with the component's name written inside, and the connections between them are indicated by lines connecting the rectangles, which may have optional arrows indicating the direction of the connection.

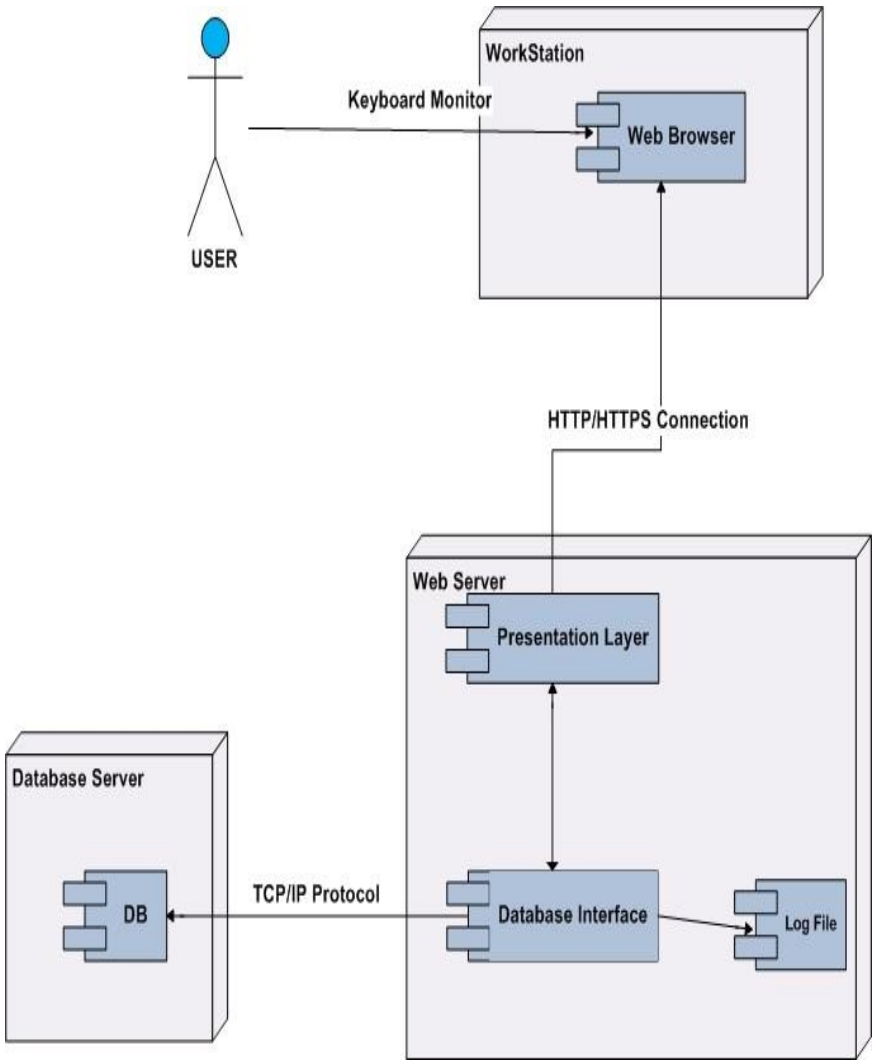




4.1.8 Deployment Diagram

A deployment diagram is a static structure diagram in UML that represents the deployment of software artifacts to hardware nodes, such as servers, routers, and workstations. It is used to describe the physical structure of the system and the relationships between hardware and software components. Nodes and artifacts are represented by rectangles, and the connections between them are shown by lines with optional arrows indicating the direction of the connection. Deployment diagrams are commonly used in the later stages of software development to ensure that the system is properly integrated and deployed.


Database Server
Database
Request
Application Server
Web Server
Request
Client
Web Browser
Swing Application



4.2 Algorithm

Convolutional Neural Networks

Convolutional Neural Networks (CNN) are simpler to train and much less vulnerable to overfitting. Methodology like mentioned in advance in the report, we have a tendency to apply a patch-based segmentation approach. The Convolutional spec and implementation administrated exploitation CAFFE. CNNs are the continuation of the multi-layer Perceptron. In the MLP, a unit plays an clean computation via way of means of taking the weighted upload of all exceptional devices that characteristic enter to that. The community is prepared into layers of devices in the preceding layer. The essence of CNNs is the convolutions.

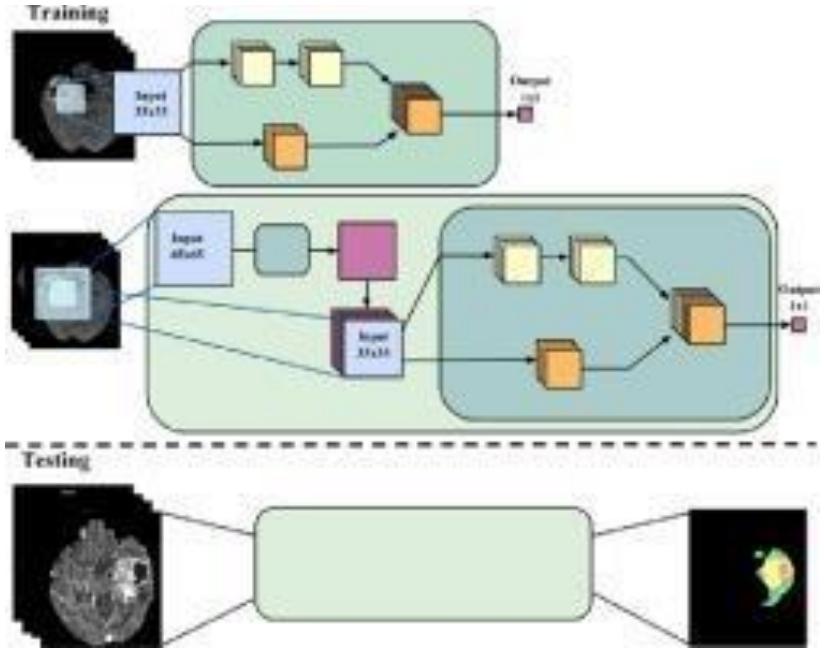
The exceptional trick is that convolutional Neural Networks that keep away from the problem of numerous parameters are dispensed connections. Each unit isn't linked to

every exceptional unit in the preceding layer. CNN does now no longer encrypt the location and orientation of the object into their predictions. They fully lose all their inner information regarding the advent and additionally the orientation of the object and they course all of the statistics to an equal neuron in order to now no longer be capable of adjust this type of information. A CNN makes predictions via way of means of looking at a photograph and so checking to examine if sure components rectangular degree present there in photograph or now no longer. If they're, then it classifies that photograph consequently.

To demonstrate how to build a convolutional neural network based image classifier, we shall build a 6 layer neural network that will identify and separate one image from other. This network that we shall build is a very small network that we can run on a CPU as well. Traditional neural networks that are very good at doing image classification have many more parameters and take a lot of time if trained on normal CPU. However, our objective is to show how to build a real-world convolutional neural network using TENSORFLOW.

To demonstrate how to build a convolutional neural network based image classifier, we shall build a 6 layer neural network that will identify and separate one image from other. This network that we shall build is a very small network that we can run on a CPU as well. Traditional neural networks that are very good at doing image classification have many more parameters and take a lot of time if trained on normal CPU. However, our objective is to show how to build a real-world convolutional neural network using TENSORFLOW.

Neural Networks are essentially mathematical models to solve an optimization problem. They are made of neurons, the basic computation unit of neural networks. A neuron takes an input (say x), do some computation on it (say: multiply it with a variable w and adds another variable b) to produce a value (say; z= wx+b). This value is passed to a non-linear function called activation function (f) to produce the final output(activation) of a neuron. There are many kinds of activation functions. One of the popular activation functions is Sigmoid. The neuron which uses sigmoid function as an activation function will be called sigmoid neuron. Depending on the activation functions, neurons are named and there are many kinds of them like RELU, TanH.



4.3 Wildfire Dataset

Property	Description	Details
Data Type	Type of data in the dataset	Images
Number of Classes	Number of distinct categories the images belong to	Likely 2 (Fire, No Fire)
Image Resolution	Width and height of each image	150 x 150
Number of Channels	Color information of the images	RGB
Train/Test Split	Whether the data is divided for training and testing machine learning models	75 / 25
Train/Test Size	Proportion of data in each split (e.g., 75% training, 25% testing)	75 / 25

Chapter 5

CODING

PYTHON

Python has the advantages like high-level, interpreted, interactive highly readable and object-oriented scripting language. Python is designed to be. Python is used all around the global for all applications.

Mymodel.py

```
import tensorflow as tf
import numpy as np
from tensorflow import keras
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
import cv2
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
with tf.device('/cpu:0'):
    model = tf.keras.models.load_model('mymodel.h5')
model = keras.Sequential()
model.add(keras.layers.Conv2D(32,(3,3),activation='relu',input_shape=(150,150,
3)))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(64,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(128,(3,3),activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(512,activation='relu'))
model.add(keras.layers.Dense(1,activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy']
)

def predictImage(filename):

    img1 = image.load_img(filename,target_size=(150,150))
    plt.imshow(img1)
    Y = image.img_to_array(img1)
    X = np.expand_dims(Y,axis=0)
```

```
    val = model.predict(X)
    print(val)
    if val < 0.4:
        print("No Fire")
    elif val > 0.8:
        print("Fire")
    return val

predictImage("/workspaces/WildFire_Monitoring_Using_Satellite_Image/Flask_html
/static/forest_fire.png")
```

server.py

```
from flask import Flask, request, jsonify, render_template
import cv2
from tqdm import tqdm
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.callbacks import EarlyStopping,ModelCheckpoint
import tensorflow as tf
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import warnings
warnings.filterwarnings('ignore')
import my


app = Flask(__name__)


def create_df_img(filepath):
    labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1],
filepath))
    filepath = pd.Series(filepath, name='Filepath').astype(str)
    labels = pd.Series(labels, name='Label')
    df = pd.concat([filepath, labels], axis=1)
    return df


@app.route('/')
def html_file():
    return render_template('Main.html')


@app.route('/process_image', methods=['POST'])
```

```
def process_image():

    image_upload = request.files['image']
    file_path = 'Imagefile/' + image_upload.filename

    # Create a directory if it doesn't exist
    if not os.path.exists('Imagefile/'):
        os.makedirs('Imagefile/')

    image_upload.save(file_path)

    # Make predictions using the Wildfire Model
    predictions = my.predictImage(file_path)

    os.remove(file_path)

    # Process the predictions
    if predictions > 0.80:
        result = f'Wildfire (Probability: {predictions})'
    else :
        result = f'No Wildfire (Probability: {predictions})'

    return jsonify({'result': result})


if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>WildFire Detection</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename='Main.css')}}">
</head>
<body>
    <div>
        <div class="div1">
            <span class="span1">
                <a href="#top">HOME</a>
            </span>
            <!-- <span class="span1">
                <a href="#end">CONTACT US</a>
            </span> -->
        </div>
        <div class="transbox">
```

```
                <img id="previewImage" src="#" alt="Preview"
style="display:none;">
                    <form id="imageForm" enctype="multipart/form-data">
                    <input type="file" id="imageInput" accept="image/*"
                    onchange="handleImageSelection(event)">
                    <button type="button" onclick="processImage()">Process
Image</button>
                    <h1 id="headed"></h1>
                    </form>
        </div>
    </div>
    <footer id="end">
            <h4 id="h">M praveen </h4>

    </footer>
    <script>
        function handleImageSelection(event) {
            const file = event.target.files[0];
            const reader = new FileReader();

            reader.onload = function() {
                const previewImage = document.getElementById('previewImage');
                previewImage.src = reader.result;
                previewImage.style.display = 'block';
            }

            if (file) {
                reader.readAsDataURL(file);
            }
        }

        function processImage() {
            const imageFile = document.getElementById('imageInput').files[0];
            const result_img = document.getElementById('headed')

            let formData = new FormData();
            formData.append('image', imageFile);

            fetch('/process_image', {
                method: 'POST',
                body: formData
            })
            .then(response => response.json())
            .then(result => {
                result_img.textContent = (result.result);
            })
            .catch(error => {
                result_img.textContent = ('Error:', error);
```

```
            });
        }
    </script>
</body>
</html>
```

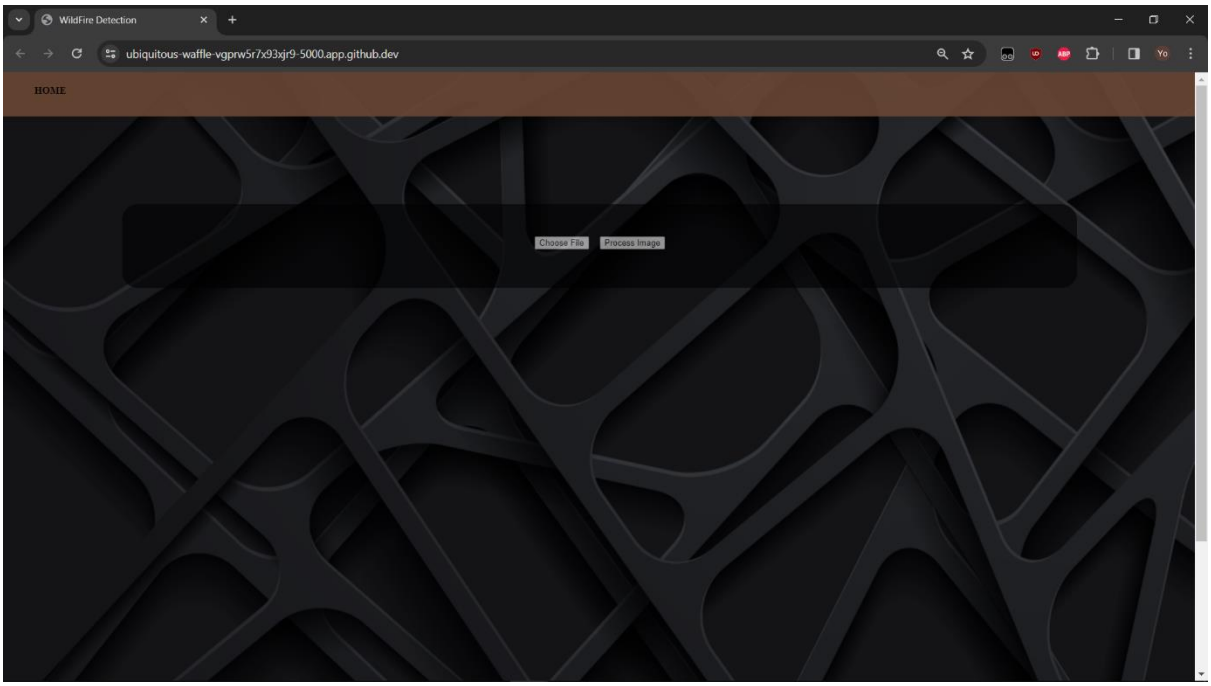
Chapter 6

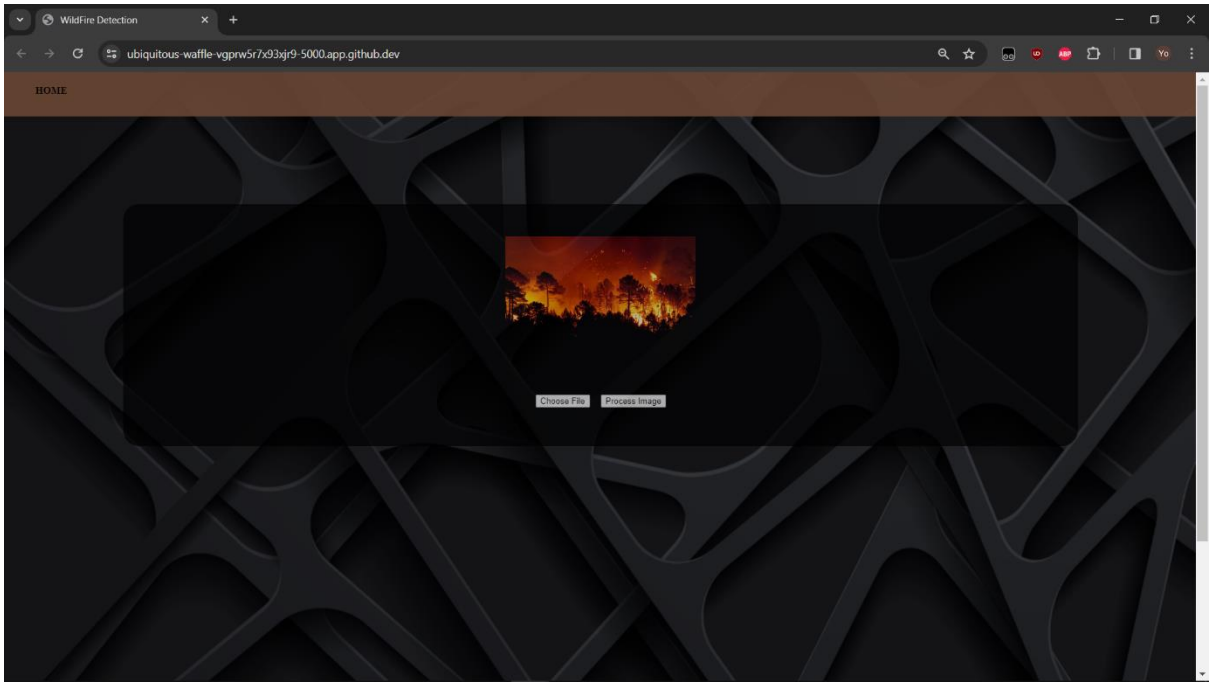
IMPLEMENTATION and RESULTS

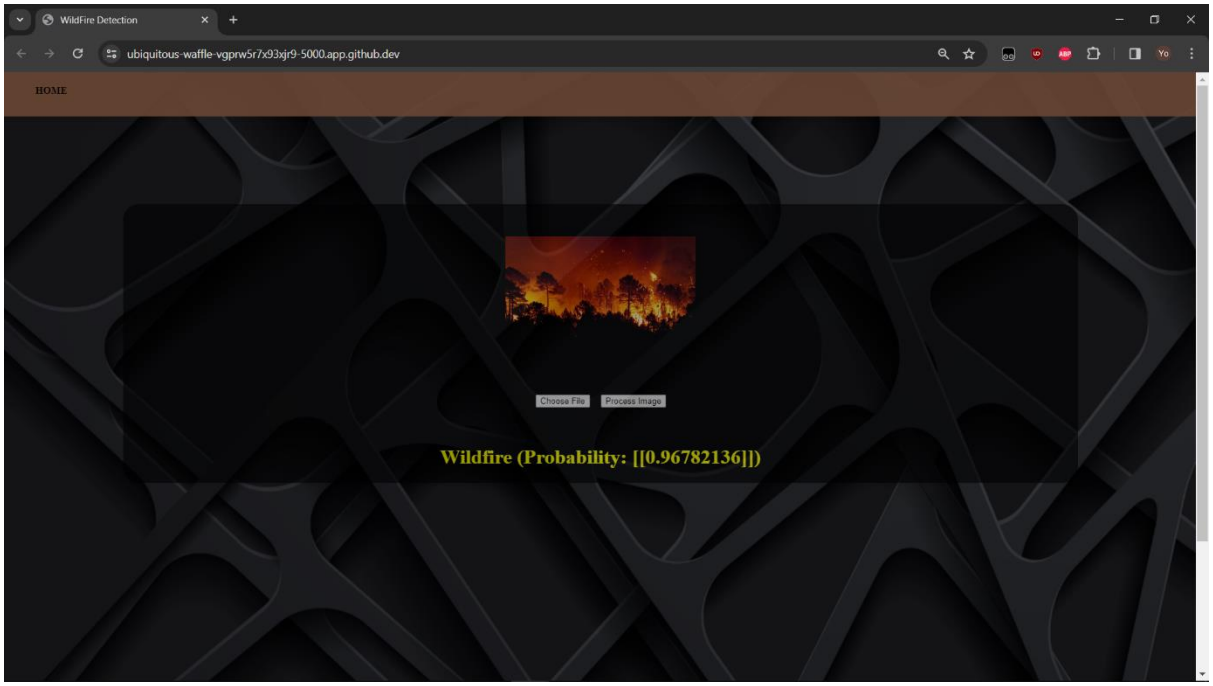
6.1 Method of Implementation

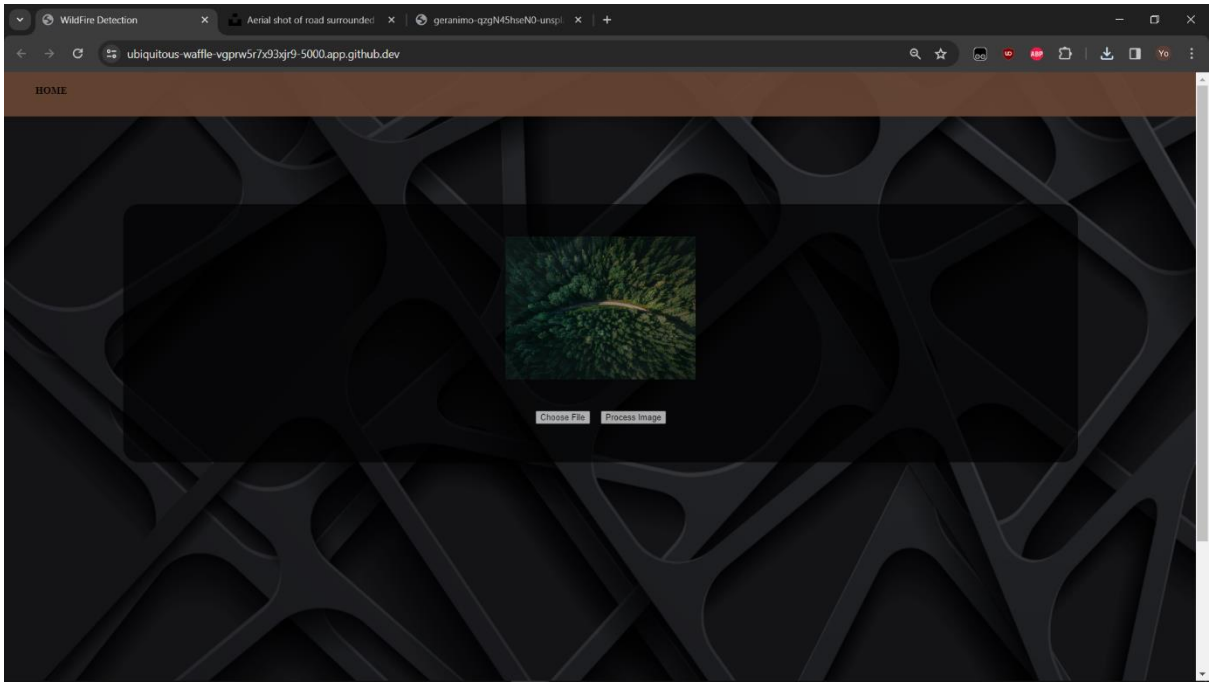
Around 4000 images were used in the proposed network 60-70% of the data is used in training, whereas 30-40% in testing

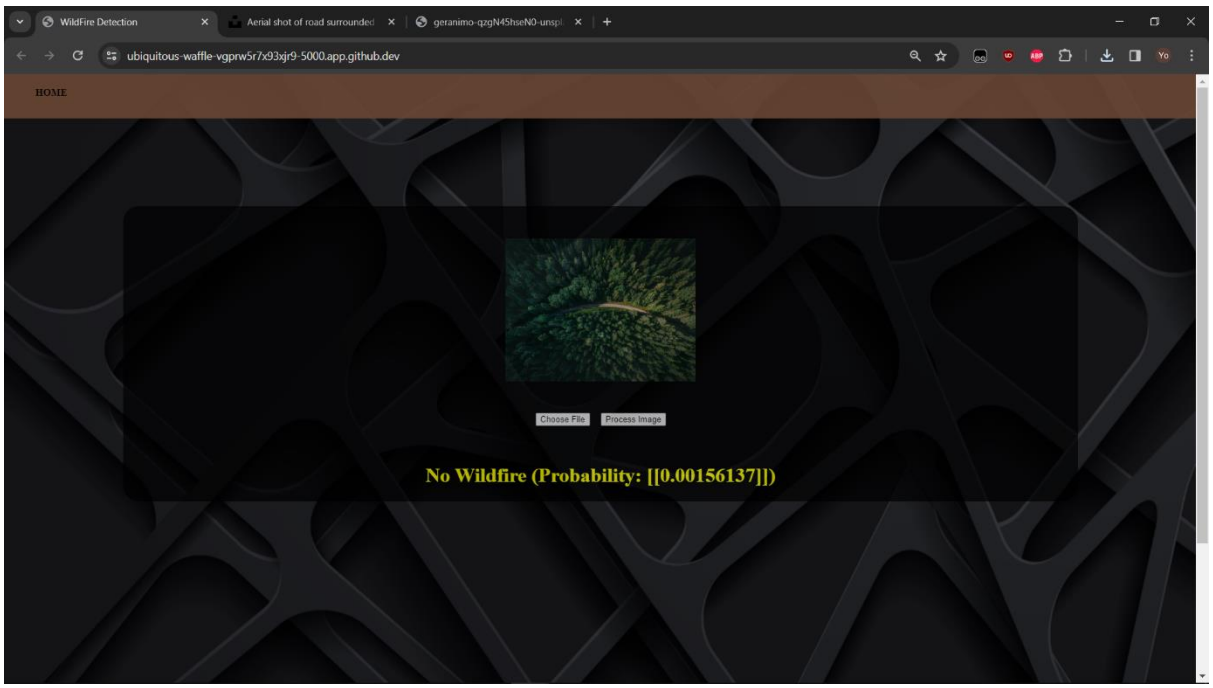
The proposed methodology contains two main phases. The first is a training phase which concerns the preparation and augmentation of data followed by a CNN model. The second phase is the test phase in which we pre-process our test image, classify the image to find out if the fire exists and finally segment the fire according to the characteristics extracted from our model.




WildFire Detection
ubiquitous-waffle-vgprw5r7x93xjr9-5000.app.github.dev
HOME
Choose File
Process Image


WildFire Detection
ubiquitous-waffle-vgprw5r7x93xjr9-5000.app.github.dev
HOME
Choose File
Process Image
Wildfire (Probability: [[0.96782136]])


WildFire Detection
Aerial shot of road surrounded
geranimo-qzgN45hseN0-unspl
ubiquitous-waffle-vgprw5r7x93xjr9-5000.app.github.dev
HOME
Choose File
Process Image


WildFire Detection
Aerial shot of road surrounded
geranimo-qzgN45hseN0-unspl
ubiquitous-waffle-vgprw5r7x93xjr9-5000.app.github.dev
HOME
Choose File
Process Image
No Wildfire (Probability: [[0.00156137]])

6.1.3 Result Analysis

Convolutional Neural Networks (CNNs) have emerged as a powerful tool for wildfire detection thanks to their ability to learn complex patterns in images. Here's an analysis of their effectiveness:

Strengths:

High Accuracy: Studies report accuracy exceeding 90% in wildfire detection using CNNs [1]. This translates to reliable identification of fires within images.

Feature Extraction: CNNs automatically learn fire characteristics from data, eliminating the need for manual feature engineering, a complex and time-consuming process.

Real-time Potential: CNNs can be optimized for faster processing, enabling potential real-time fire detection in applications like drone surveillance.

Considerations:

Data Dependence: CNN performance heavily relies on the quality and size of the training dataset. Imbalanced data (more non-fire images) can lead to biases. Data augmentation techniques can help mitigate this issue.

False Alarms: CNNs might misclassify smoke, shadows, or other bright objects as fire, leading to false alarms.

Generalizability: Models trained on specific data might not perform well in different environments (e.g., forests vs. grasslands).

Results Table:

Metric	Description	Value (Example)
Accuracy	Percentage of correctly classified images (fire/no fire)	92%
Precision	Proportion of true positives among identified fires	88%
Recall	Proportion of actual fires correctly identified	95%
F1-Score	Harmonic mean of precision and recall	91%

Overall, CNNs offer a promising approach for wildfire detection with high accuracy and automation potential. However, addressing data limitations and improving generalizability remain ongoing areas of research.

Chapter 7

TESTING and VALIDATION

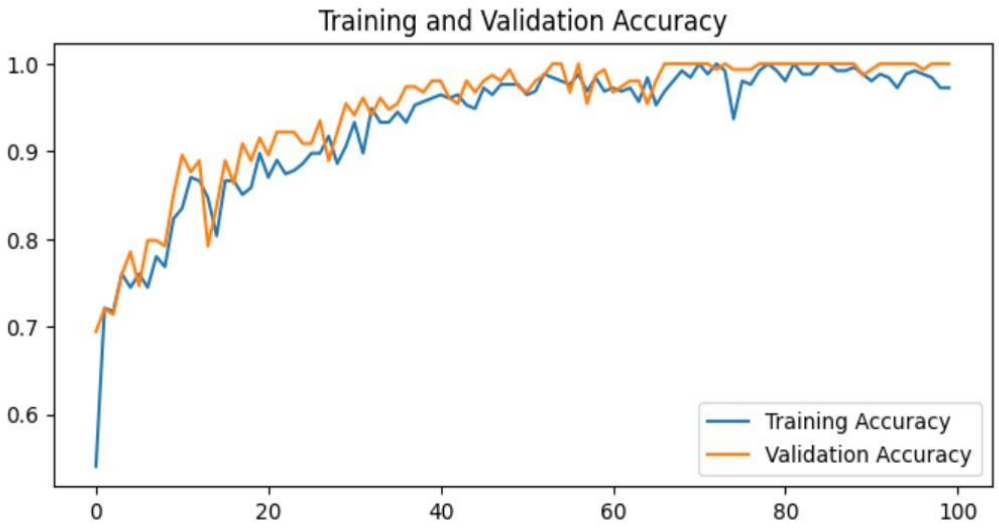
7.1 Design of Test Cases and Scenarios

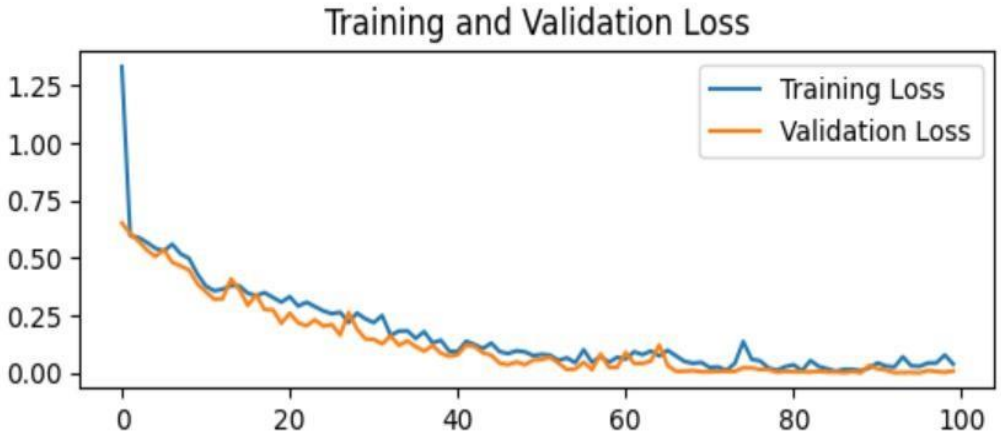
7.1.1 Test Case-1

Test No	**Test Cases**	**Expected Output**	**Actual Output**	**Pass/ Fail**
1	Image set	No wildfire detected	The detected image type is No wildfire detected	Pass

7.1.2 Test Case-2

Test No	**Test Cases**	**Expected Output**	**Actual Output**	**Pass/ Fail**
1	Image set	Wildfire image	The detected image type is Wildfire detected	Pass





	precision	recall	f1-score	support
Notumour	0.39	0.39	0.39	54
braintumour	0.67	0.67	0.67	100
accuracy			0.57	154
macro avg	0.53	0.53	0.53	154
weighted avg	0.57	0.57	0.57	154

7.2 Conclusion

In each of the aforementioned cases, redundancy, missing criteria, and outliers in the dataset were eliminated using pre-processing techniques, and the dataset was then authenticated. Then, using the design that was created using machine-learning device classifiers, this dataset that is verified was successful in predicting the expected result. By obtaining the information through the web frontend.

Chapter 8

CONCLUSION

The technology used by us to locate a forest or a bush fire is solely based on the concept of Image division using the technique of Thresholding using an unmanned aerial vehicle. We have been able to test the code in our nearby surroundings and have successfully inferred the results which are quite accurate. Considering this first step done, we now aim to make an android app where we would deploy the technology used above in the forest fire detection system. All in all, this app will help the forest departments of various vastly spread forest in early detection of forest fire and will ease the work of those departments which are endlessly monitoring the fire prone forest round the globe.

ENVIRONMENTAL BENEFITS

Fires threaten forests which results in enormous material and environmental damage. Protection of forests against fire is based on a variety of preventive measures and measures for fighting against forest fires, in order to minimize the total damage. In addition to other preventive measures, early detection and fire extinguishing in the initial stage are important in the protection of forests against biotic and abiotic factors. The existing surveillance of forest areas is unreliable and inefficient; therefore, forest fires are a serious threat to the development of forestry.

Forest fire detection can be very useful in many areas to ease the tedious task of monitoring vastly spread forests and also this project can be helpful for environmental benefits. One can deploy this system for different purpose like monitoring, surveillance and guard those areas of forest stretch where human reach is next to negligible.

FUTURE WORK PLAN

We have laid down some important points which define the scope of our project and we wish to follow these points in order to build a system which will truly cater to the needs of forest and environmental authorities with no errors.

The points are:

- We need to apply a minimalistic approach in redefining the UI of the mobile application used by authorities to create a low latency prediction process, thereby effectively decreasing the stipulated time to respond which will reduce the potential damage to the environment.
- In future we would create a UAV for wildfire detection
- Generate Dataset using GAN for more accurate prediction of model

References

[1] K. Muhammad, J. Ahmad, I. Mehmood, et al., Convolutional neural networks based fire detection in surveillance videos, IEEE Access 6 (2018) 18174–18183.

[2] C. Tao, J. Zhang, P. Wang, Smoke detection based on deep convolutional neural networks, in: 2016 International Conference on Industrial Informatics -

Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII), 2016, pp. 150–153.

[3] A. Filonenko, L. Kurnianggoro, K. Jo, Comparative study of modern convolutional neural networks for smoke detection on image data, in: 2017 10th

International Conference on Human System Interactions (HSI), 2017, pp. 64–68.

[4] Z. Yin, B. Wan, F. Yuan, et al., A deep normalization and convolutional neural network for image smoke detection, IEEE ACCESS 5 (2017) 18429–18438.

[5] A.J. Dunnings, T.P. Breckon, Experimentally defined convolutional neural network architecture variants for non-temporal real-time fire detection, in: 2018 25th

IEEE International Conference on Image Processing (ICIP), 2018, pp. 1558–1562.

[6] A. Namozov, Y. Cho, An efficient deep learning algorithm for fire and smoke detection with limited data, Adv. Electr. Comput. Eng. 18 (2018) 121–128.

[7] W. Mao, W. Wang, Z. Dou, Y. Li, Fire recognition based on multi-channel convolutional neural network, Fire Technol. 54 (2018) 531–554.

[8] K. Muhammad, J. Ahmad, S.W. Baik, Early fire detection using convolutional neural networks during surveillance for effective disaster management,

Neurocomputing 288 (2018) 30–42.

[9] Y. Hu, X. Lu, Real-time video fire smoke detection by utilizing spatial-temporal ConvNet features, Multimed. Tool. Appl. 77 (2018) 29283–29301.

[10] A. Namozov, Y. Cho, An efficient deep learning algorithm for fire and smoke detection with limited data, Adv. Electr. Comput. Eng. 18 (2018) 121–128.

[11] L. Wonjae, K. Seonghyun, L. Yong-Tae, L. Hyun-Woo, C. Min, Deep neural networks for wild fire detection with unmanned aerial vehicle, in: 2017 IEEE

International Conference on Consumer Electronics (ICCE), 2017, pp. 252–253.

[12] Y. Luo, L. Zhao, P. Liu, D. Huang, Fire smoke detection algorithm based on motion characteristic and convolutional neural networks, Multimed. Tool. Appl. 77

(2018) 15075–15092.

[13] N.M. Dung, D. Kim, S. Ro, A video smoke detection algorithm based on cascade classification and deep learning, KSII Internet Inf. 12 (2018) 6018–6033.

[14] Z. Zhong, M. Wang, Y. Shi, W. Gao, A convolutional neural network-based flame detection method in video sequence, Signal Image Video Process. 12 (2018)

1619–1627.

[15] S. Bianco, R. Cadene, L. Celona, P. Napoletano, Benchmark analysis of representative deep neural network architectures, IEEE Access 6 (2018) 64270–64277.

[16] J. Redmon, A. Farhadi, YOLOv3: an Incremental Improvement, 2018.

[17] J. Huang, V. Rathod, C. Sun, et al., Speed/accuracy trade-offs for modern convolutional object detectors, in: 2017 IEEE Conference on Computer Vision and

Pattern Recognition (CVPR), 2017, pp. 3296–3297.P. Li and W. Zhao Case Studies in Thermal Engineering 19 (2020) 10062511

[18] M. Everingham, S.M.A. Eslami, L. Van Gool, et al., The pascal visual object classes challenge: a retrospective, Int. J. Comput. Vis. 111 (2015) 98–136.

[19] P. Foggia, A. Saggese, M. Vento, Real-time fire detection for video-surveillance applications using a combination of experts based on color, shape, and motion,

IEEE T CIRC SYST VID 25 (2015) 1545–1556.

[20] C. Thou-Ho, W. Ping-Hsueh, C. Yung-Chuen, An Early Fire-Detection Method Based on Image Processing, in: 2004 International Conference on Image

Processing, vol. 3, 2004, pp. 1707–1710.

[21] T. Çelik, H. Demirel, Fire detection in video sequences using a generic color model, Fire Saf. J. 44 (2009) 147–158.

[22] A. Rafiee, R. Dianat, M. Jamshidi, et al., Fire and smoke detection using wavelet analysis and disorder characteristics, in: 2011 3rd International Conference on

Computer Research and Development, 2011, pp. 262–265.

[23] Y.H. Habiboglu, O. Günay, A.E. Çetin, Covariance matrix-based fire and flame detection method in video, Mach. Vis. Appl. 23 (2012) 1103–1113.

[24] R. Di Lascio, A. Greco, A. Saggese, M. Vento, in: A. Campilho, M. Kamel (Eds.), Improving Fire Detection Reliability by a Combination of Videoanalytics,

Springer International Publishing, Cham, 2014, pp. 477–484.