NAALAIYA THIRAN PROJECT – 2022

HX 8001-PROFESSIONAL READINESS FORINNOVATION, EMPLOYABILITY AND ENTREPRENEURSHIP

EMERGING METHODS FOR EARLY DETECTION

OF FOREST FIRES

A PROJECT REPORT

Submitted by

AKASH G	**411519104001**
BALAJI R	**411519104302**
B MANI BHARATHI	**411519104044**
V MANIKANDAN	**411519104045**

(TEAM ID: PNT2022TMID37993)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PERI INSTITUTE OF TECHNOLOGY, MANNIVAKKAM – 600048

Approved by AICTE, New Delhi. Accredited with B+ Grade by NAAC.
(Affiliated to Anna University)

ANNA UNIVERSITY: CHENNAI 600025

NOVEMBER 2022

PERI INSTITUTE OF TECHNOLOGY, MANNIVAKKAM – 600048

Approved by AICTE, New Delhi Accredited with B+ Grade by NAAC.
(Affiliated to Anna University)

NOVEMBER 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this report **"EMERGING METHOD FOR EARLY DETECTION OF FOREST FIRES"** is the bonafide work of **AKASH G (411519104001), B MANIBHARATHI (411519104044) AND V MANIKANDAN (411519104045) , R BALAJI (411519104302)** who carried out **HX 8001 Professional Readiness for Innovation, Employability and Entrepreneurship** offered byIBM and Anna University, Chennai.

SIGNATURE	**SIGNATURE**	**SIGNATURE**
R.S. ABBIRAMI B.TECH., M.E	**A VIJAYANARAYANAN B.E., M.TECH**	**K VARALAKSHMI B. E., M.E**
FACULTY MENTOR	**FACULTY EVALUATOR**	**HEAD OF DEPARTMENT**
Assistant Professor	**Assistant Professor**	**Associate Professor**
Department of Computer Science & Engineering	Department of Computer Science & Engineering	Department of Computer Science & Engineering

TABLE OF CONTENTS

CHAPTER NO	CONTENTS	PAGE NO
	LIST OF FIGURES	i
	LIST OF TABLES	ii
1	INTRODUCTION 1.1 PROJECTOVERVIEW 1.2 PURPOSE	1
2	LITERATURE SURVEY 2.1 EXISTING SOLUTION 2.2 PROBLEM STATEMENTDEFINITION	2
3	IDEATION & PROPOSED SOLUTION 3.1 EMPATHY MAP CANVAS 3.2 IDEATION AND BRAINSTORMING 3.3 PROPOSED SOLUTION 3.4 PROBLEM SOLUTION FIT	3
4	REQUIREMENT ANALYSIS 4.1 FUNCTIONAL REQUIREMENTS 4.2 NON FUCTIONAL REQUIREMENTS	9
5	PROJECT DESIGN 5.1 DATA FLOW DIAGRAMS 5.2 SOLUTION AND TECHNICAL ARCHITECTURE	11
6	PROJECT PLANNING & SCHEDULING 6.1 SPRINT PLANNING ANDESTIMATION 6.2 SPRINT DELIVERYSHEDULE	18
7	CODING & SOLUTIONING 7.1 FEATURE 1 7.2 FEATURE 2	21
8	TESTING AND RESULTS	42
9	PERFORMANCE RESULTS 9.1 PERFORMANCE METRICES	45
10	ADVANTAGES & DISADVANTAGES 10.1 ADVANTAGES 10.2 DISADVANTAGES	46
11	CONCLUSION	47
12	FUTURE SCOPE	48

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
3.1	EMPATHY MAP	3
3.4	PROBLEM SOLUTION FIT	8
5.1	DATA FLOW DIAGRAM FOR EMERGING METHOD FOR EARLYDETECTION OF FOREST FIRE	12
5.2	SOLUTION ARCHITECTURE FOR emerging method for earlydetection of forest fire	13
8.1	SIGN UP PAGE FOR emerging method for early detection of forest fire	43
8.2	LOGIN PAGE for emerging method for early detection of forest fire	44
9.1	PERFORMANCE METRICES	45

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
3.2	IDEATION AND BRAINSTORMING	4
3.3	PROPOSED SOLUTION	7
4.1	FUNCTIONAL REQUIREMENTS FOR Emerging Method for Early Detection of Forest Fire	9
4.2	NON-FUNCTIONAL REQUIREMENTS OF Emerging Method for Early Detection of Forest Fire	10
6.1	SPRINT PLANNING AND ESTIMATION FOR Emerging Method for Early Detection of Forest Fire	18
6.2	SPRINT PLANNING DONE FOR Emerging Method for Early Detection of Forest Fire	19

CHAPTER 1
INTRODUCTION

Forest fires are a major environmental issue, creating economic and ecological damage while endangering human lives. There are typically about 100,000 wildfires in the United States every year. Over 9 million acres of land have been destroyed due to treacherous wildfires. It is difficult to predict and detect Forest Fire in a sparsely populated forest are and it is more difficult if the prediction is done using ground-based methods like Camera or Video-Based approach. Satellites can be an important source of data prior to and also during the Fire due to its reliability and efficiency. The various real-time forest fire detection and prediction approaches, with the goal of informing the local fire authorities.

1.1 PROJECT OVERVIEW

Several forest fire detection and prevention system have been made and successfully applied but the issue of forest fire risk has lot more things to research and improve the system so it works more effectively and efficiently. Forest fire management requires high reliability system, so in any incident the system will inform them before any fire accident. Due to the lack of emergency plan and the need of efficient system is much higher.

The installation of early detection system in a better way by identifying the sites is also a major challenge. With the complete information and warning of incident the smart system can help to reduce the risk and prevent the forest fire which can save the biodiversity and lives of millions. we are interfacing Arduino Uno R3 with different IoT sensors such as smoke sensor, PIR Sensor, temperature sensor, which is capable of detecting the fire conditions and sends the information to the concerned authority on detecting fire conditions.

1.2 PURPOSE

The forest fires destroys the wildlife habitat, damages the environment, affects the climate, spoils the biological properties of the soil, etc. So the forest fire detection is a major issue in the present decade. At the same time the forest fire have to be detectedas fast as possible.

CHAPTER 2

LITERATURE SURVEY

In residential areas ION detectors are advantageous for flaming fire detection, while photo detectors are beneficial for non flaming fire detection. However, to achieve more reliable and fault-tolerant results and higher detection rates more than one sensor should be used. This assures that flaming and non flaming fires can be discriminated.

- Although temperature sensors are probably the simplest and the most obvious sensors for fire detection, studying various sources in this field reveals that all researchers agree on the fact that it alone cannot accurately indicate fire and gas (e.g., CO, CO2) concentrations are main features for fire detection.

- Fire Weather Index (FWI) and other indices resulted from several decades of forestry research can be used as strong indications for forest fire detection.

- The WSN community needs to use the general knowledge about fire patterns, best combination of sensors and appropriate detection techniques from the fire-related disciplines. It is apparent that selection of sensors was often carried out randomly or assumption-basely.

2.1 EXISTING SOLUTION

Detection of forest fire and smoke in wildland areas is done through remote sensing-based methods such as satellites, high-resolution static cameras fixed on the ground, and unmanned aerial vehicles (UAVs).

2.2 PROBLEM STATEMENT DEFINITION

The user interacts with a web camera to read the video. Once the input image from the video frame is sent to the model, if the fire is detected it is showcased on the console, and alerting sound will be generated and an alert message will be sent to the Authorities.

- Data Collection.
- Collect the dataset or create the dataset.
- Image Preprocessing.
- Import Image Data Generator Library.
- Define the parameters /arguments for Image Data Generator class
- Applying ImageDataGenerator on trainset and test set.
- Model Building
- Import the model building Libraries

CHAPTER 3

IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a collaborative visualization used to express clearly what one knows about a particular type of user. It externalizes knowledge about users in order to create a shared understanding of user needs, and aid in decision making.

Empathy maps are split into 4 quadrants (Says, Thinks, Does, and Feels), with the user in the middle.

The empathy map for Inventory management system for retailers is shown in Fig 3.1

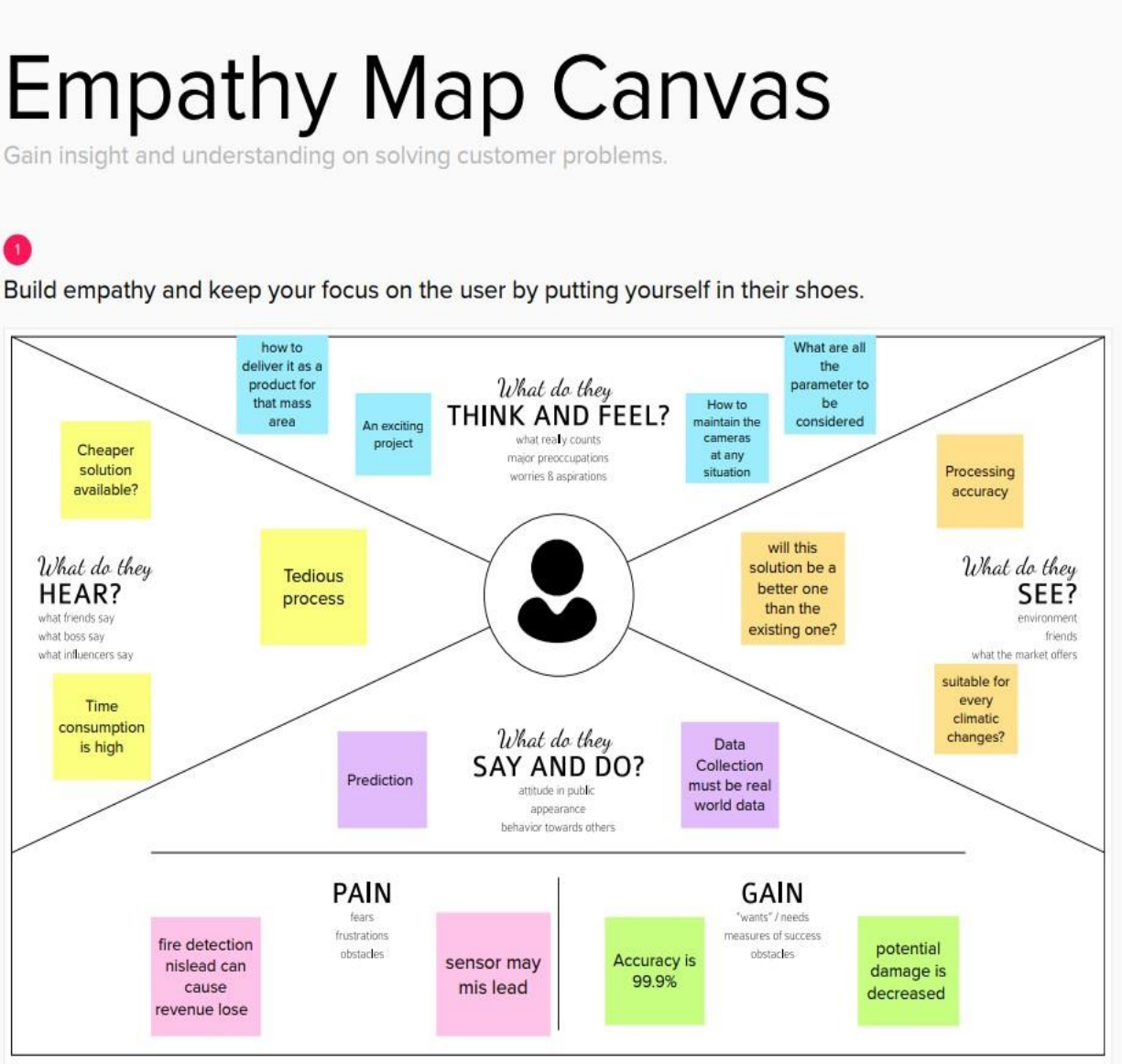


Fig 3.1 Empathy map

3.2 IDEATION AND BRAINSTORMING

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity. Both brainstorming and ideation are processes invented to create new valuable ideas, perspectives, concepts and insights, and both are methods for envisioning new frameworks and systemic problem solving.

The Ideation chart for Inventory management system for retailers is shown in Table 3.2.

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Table 3.2 Ideation and Brainstorming

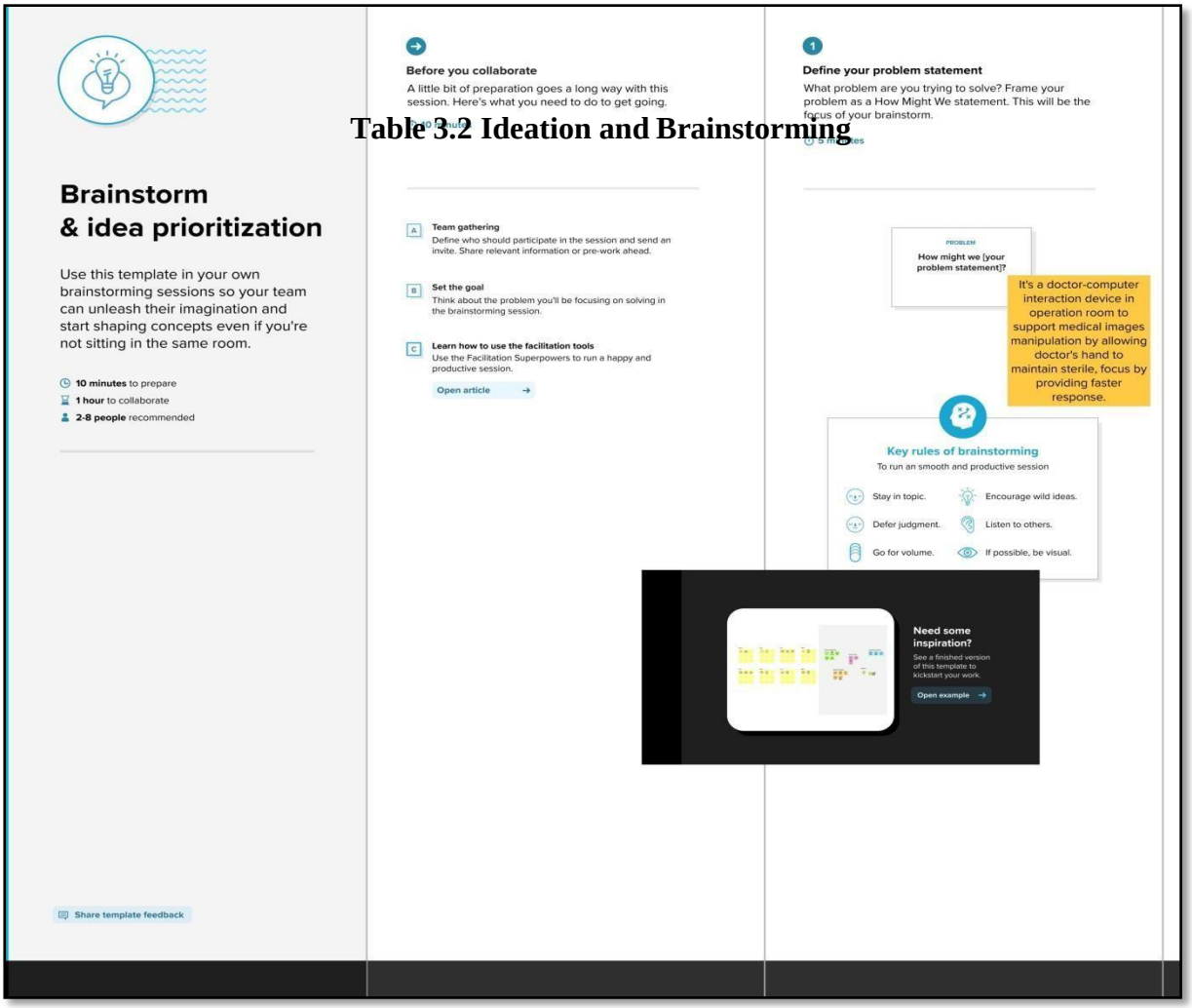
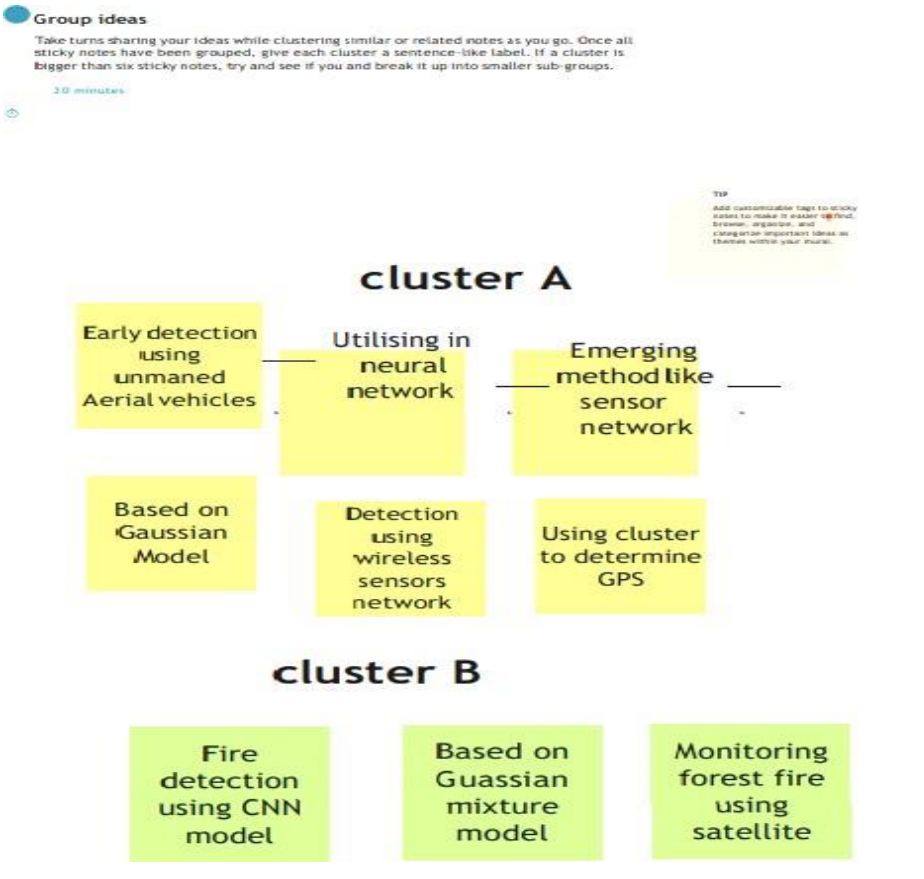
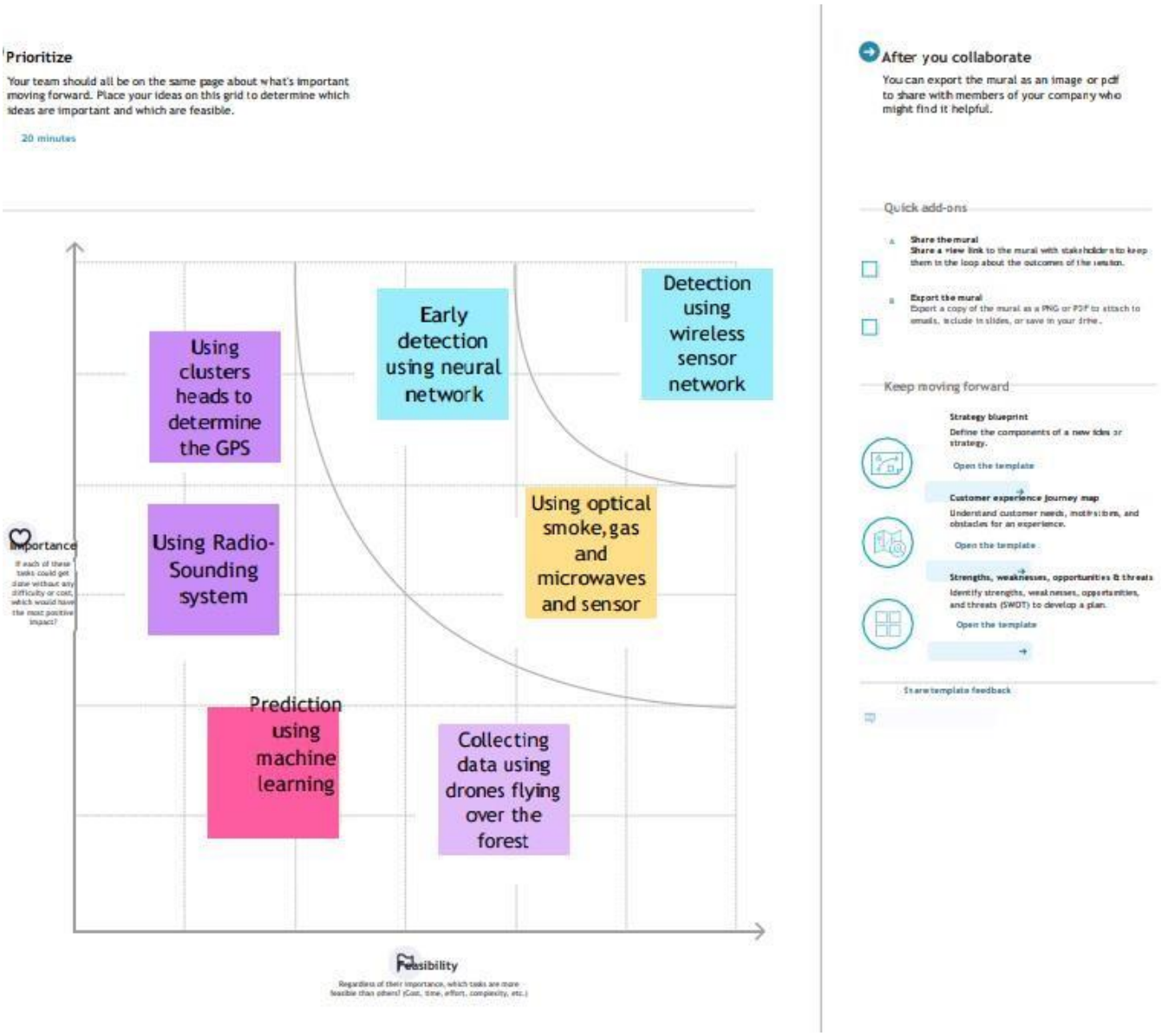


Table 3.2


Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
Akash G
Based on Gaussian mixture model
Emerging methods like LoRaWAN Sensor Networks
Image processing
Fire Dection Using CNN Model
Mani Bharathi B
Collecting Data Using Satellite Image
Monitoring the forest Using satellites
Implementing Ground Level Sensor for data
Deep Learning can be used
Balaji R
Detection using wireless sensor network
Using microwave sensor
Using Cluster Heads to determine the GPS
Using Optical sensor and Digital camera
Manikandan V
Prediction using machine learning
Early dedection using unmaned Aerial Vechicle
Utilising Neural network
Using radio Acoustic Sounding system


Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.
20 minutes
cluster A
Early detection using unmaned Aerial vehicles
Utilising in neural network
Emerging method like sensor network
Based on Gaussian Model
Detection using wireless sensors network
Using cluster to determine GPS
cluster B
Fire detection using CNN model
Based on Guassian mixture model
Monitoring forest fire using satellite

Step-3: Idea Prioritization



3.3 PROPOSED SOLUTION

The proposed solution for Inventory management system for retailers is shown in table 3.3

S.No.	Parameter	Description
1.	**Problem Statement (Problem to be solved)**	The problem faced by the retailers is that they do not have any system to record and keep their inventory data. It is difficult for the owner to record the inventory data quickly and safely because they only keep it in the logbook and not properly organized.
2.	**Idea / Solution description**	We aim to design an Inventory Management system which is used to manage the inventory details and aims to save for the future investments. User can track the stocks sold and yet to be sold and can visualize it. The Application will notify the user when a stock is about to complete. Our web application will monitor user's stock by tracking the received SMSs from the user's mobile.
3.	**Novelty / Uniqueness**	Retailers get notified when the stock is about to get over and intimates the user to buy more stock. Providing Key Performance Indicator for Analyzing stock. Demand based advanced stockpre-order.
4.	**Social Impact / Customer Satisfaction**	Encourages user to track Stock availability and increase profit. It helps to make a better budget that he will have A financial control.
5.	**Business Model (Revenue Model)**	The low- c o s t requirement for designing this proposed model makes it more reliable and user-friendly.
6.	**Scalability of the Solution**	With efficient usage of IBM cloud, this proposed model will be able to handle a large number of user data. This makes a huge number of users to easily Access and efficiently use it.

Table 3.3 Proposed Solution

3.4 PROBLEM SOLUTION FIT:

Project Design Phase-I - Solution Fit Template

Project Title: Emerging Methods for Early Detection of Forest Fires **Team Id :PNT2022TMID37993**

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

In order to protect the forest resources, which are essential for supporting life on Earth, from sudden fire and smoke outbreaks. The forest management group does require this gadget. in places at risk of fire.

6. CUSTOMER CONSTRAINTS CC

The devastation is caused by greenhouse gases and changes in the climate. The human tendency to consume resources greedily is another important contributing cause to forest fires

5. AVAILABLE SOLUTIONS AS

For the purpose of detecting forest fires, existing systems use optical sensors. The sensors alert the office of forest management when a fire is spotted. In addition, satellites are utilised to find IR rays seen in forested areas

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

By releasing a lot of carbon dioxide, carbon monoxide, and fine particulate matter into the environment, the main issue is weather and climate. As a result, air pollution can lead to a variety of health problems, such as respiratory and cardiovascular disorders.

9. PROBLEM ROOT CAUSE RC

The following are some rationales
1. Lightning, a natural occurrence
2. Man-made causes: cigarettes, naked flames, and electric sparks
Therefore, ongoing care and observation are required to protect natural resources in order to save lives.

7. BEHAVIOUR BE

When fire is detected the sytem which is implemented to monitor the forests sets the alarm to ring, that is it gives the signal through which fire management team and the forest committee tries to call off the fire. Thus, the aim is to recognise the fire as early as possible to prevent spread of fire which will cause further damage and it'll become difficult to control.

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

Due to the existence of a great deal of dry grass all around and the possibility of the campfire remaining scorched, the uncontrolled behaviour toward burned cigarettes can spread.

4. EMOTIONS: BEFORE / AFTER EM

Since the variables that affect a wildfire's course and intensity are erratic and subject to alter at any time, they can be very stressful. People who have experienced wildfires may experience severe anxiety and mood swings.

10. YOUR SOLUTION SL

We have presented a method to detect forest fires early using CCTV camera surveillance, which can detect fire in both indoor and outdoor activities, in order to reduce these losses. In order for the forest management office to stop the damage brought on by the fire, immediate alarms must be given to them

8. CHANNELS of BEHAVIOUR CH

Online detection: As a result, the chatbot or the API can connect over the internet to provide you with information on the forest's present condition.
Offline Detection: As a result, the forest managers can notify surrounding residential areas or raise awareness through the media (news, radio).

Identify strong TR & EM

Fig 3.4 Problem Solution fit

CHAPTER 4

REQUIREMENT ANALYSIS

Requirements analysis is very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and Non-functional requirements.

4.1 FUNCTIONAL REQUIREMENTS

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements. The below table 4.1 shows the Functional Requirements for the cloud Based Inventory management System.

FR No.	Functional Requirement (EPIC)	Sub Requirement (Story/ Sub Task)
FR-1	User Registration	Registration through Gmail
FR-2	User Confirmation	Confirmation Via Email Confirmation Via OTP
FR-3	User Login	Login using credentials
FR-4	User Search	Search for Info on forest fire occurrence
FR-5	User Profile	User is alerted if there is a forest fire occurrence in their surroundings

Table 4.1 Functional Requirements for the cloud Based Inventory management System

4.2 NON-FUNCTIONAL REQUIREMENTS

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements. The below table 4.2 shows the Non-Functional Requirements for the cloud Based Inventory management System

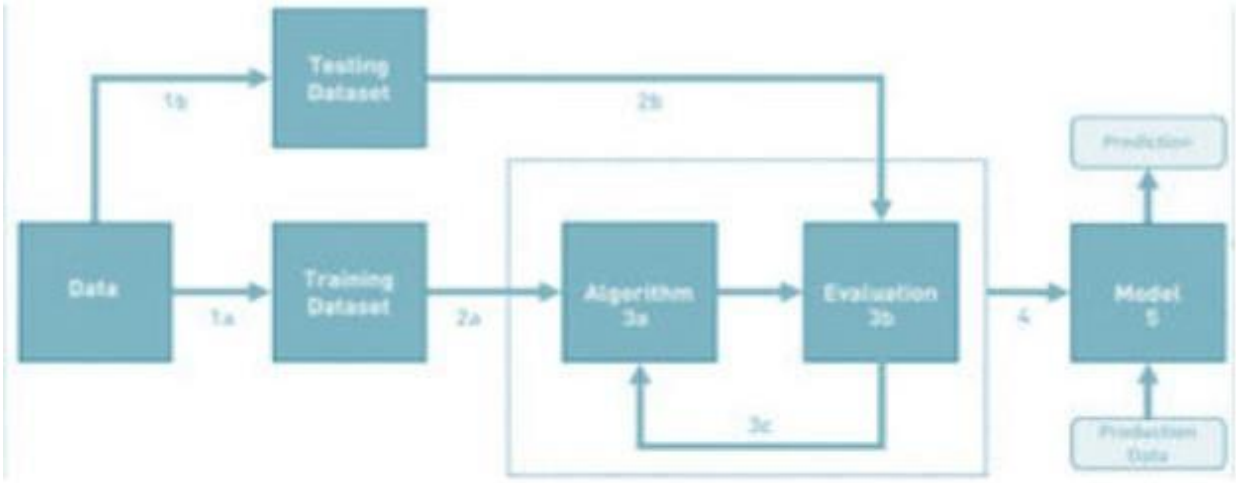
N-FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Alerts according to the user location
NFR- 2	**Security**	Instant live feed with alert of the situation
NFR- 3	**Reliability**	The prediction of the forest fire is 87% accurate
NFR- 4	**Performance**	The feed and the alert message an immediate action without a lag.

Table 4.2 Non-Functional Requirements of cloud-based Inventory management System

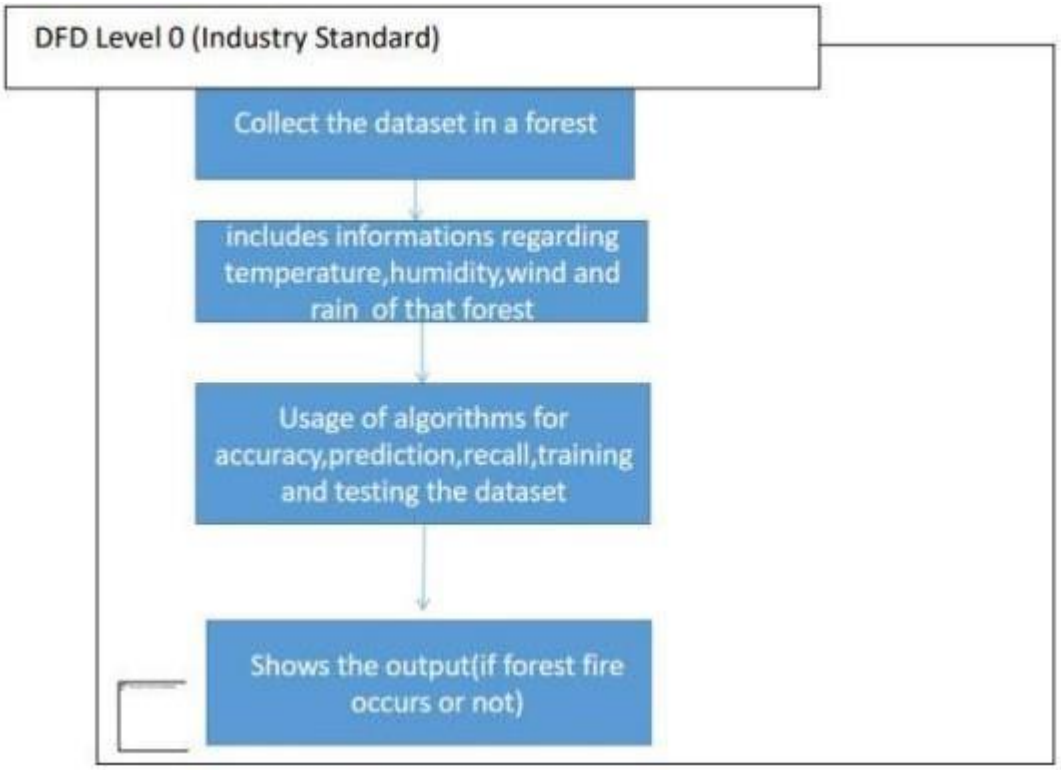
CHAPTER 5

PROJECT DESIGN

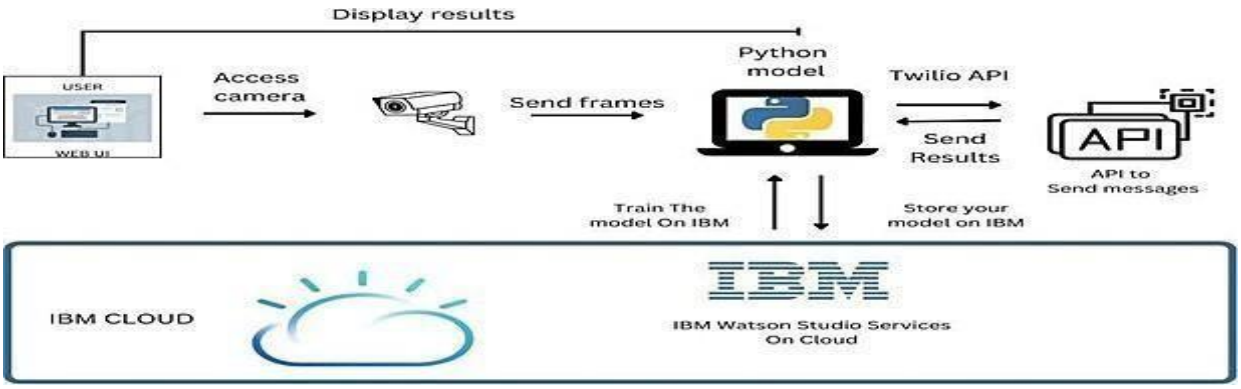
5.1 DATA FLOW DIAGRAMS

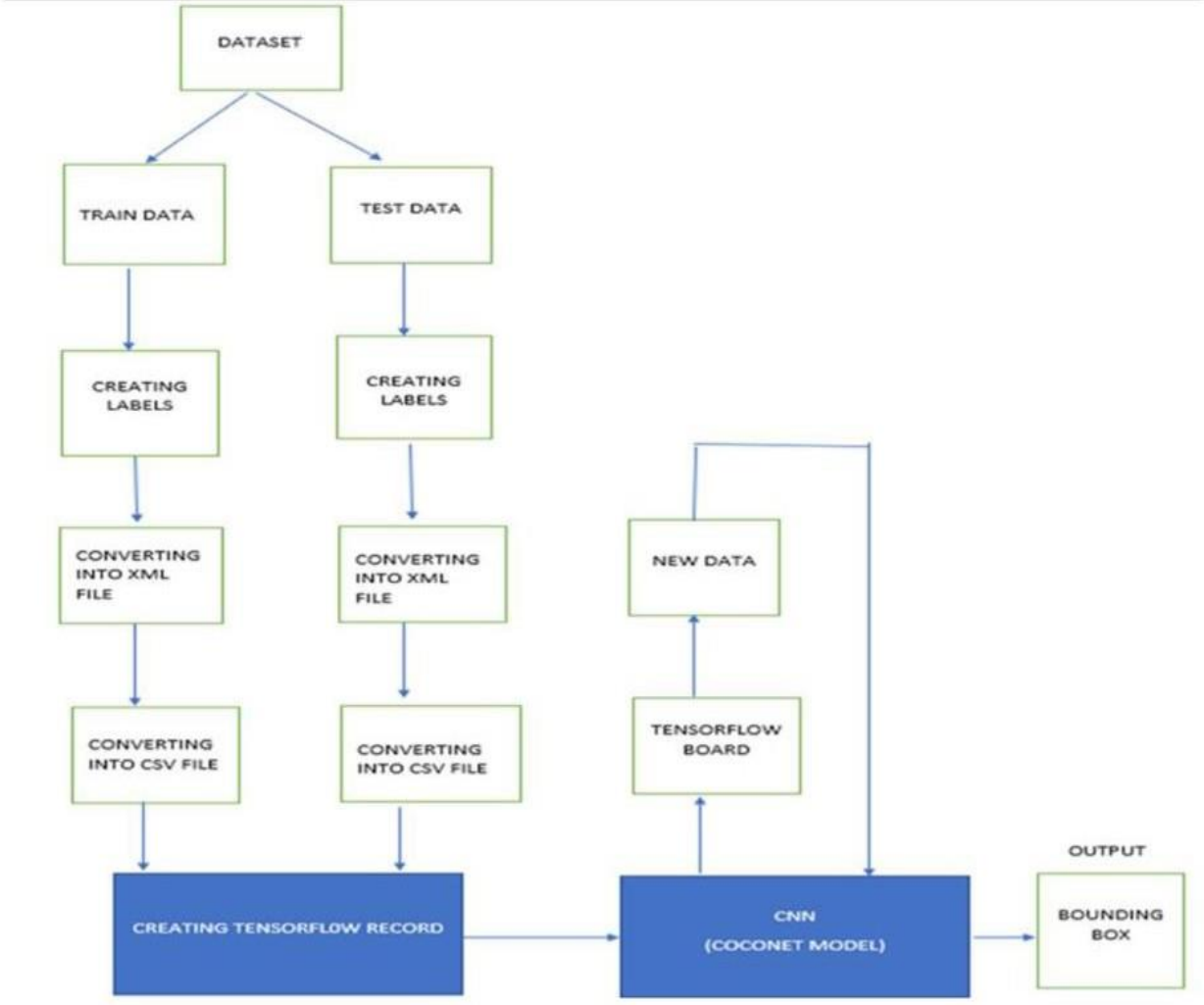


.


DFD Level 0 (Industry Standard)
Collect the dataset in a forest
includes informations regarding temperature,humidity,wind and rain of that forest
Usage of algorithms for accuracy,prediction,recall,training and testing the dataset
Shows the output(if forest fire occurs or not)

5.2 SOLUTION ARCHITECTURE





S.No	Component	Description	Technology
1	User Interface	The uses uses the console to access the interface	Python/HTML ,CSS , Javascript and react.Js
2	Input	Video Feed	Web Camera/Video on a site
3	Conversion	Video inputted is converted into Frames	Frame Converter
4	Feeding the Model	The Frames are sent to the Deep leaning model	Our Model
5	Dataset	Using best set and train set , train the model	Data set film Cloud Storage , Database
6	Cloud Database	he model is trained in the cloud moiré precise with detections moiré images can be added late on.	IBM Cloud ant ,Python Flask.
7	Infrastucture (Server / Cloud), API	Application Deployment on Local System / Cloud Local ,Cloud Server Configuration , Twilio API to send messages	Java/python ,React.Js ,JavaScript ,HTML ,CSS ,IBM Cloud ,OPEN CV ,Anaconda Navigator ,Local.

Table-2: Application Characteristics:

S.No	Characteristics	Description	technology
1	Open-Sluice Frameworks	Python Flask framework is used	Technology of Open source framework
2	Security Implementations	Mandatory Access Control (MAC) and Preventative Security Control is used	e.g. SHA-256, Encryptions, IAM Controls, OWASP etc.
3	Scalable Architecture	High scalability with 3-tier architecture	Web server – HTML ,CSS ,JavaScírpt Application server – Python , Anaconda Database server –IBM DB2
4	Availability	Use of load balancingto distribute traffic across servers	IBM load balancer
5	Performance	Enhance the performance by using IBM CDN	IBM Content Delivery Netwok

USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story /ask	Acceptance criteria	Prioirty	Release
Environmentalist	Collect the data	USN-1	As an Environmentalist,it is necessary to collect the data of the forest which includes temperature, humidity,wind and rain of the forest	It is necessary to collect the light data else the prediction may become wrong	High	Sprint-1

		USN-2	Identify algorithms that can be used for prediction	To collect the algorithm to identify The accuracy level of Each algorithms	Medium	Spíint-2
		USN-3	Identify the accuracy of each algorithms	Accuracy of each algorithm- calculated so that it is easy to obtain the most accurate output	High	Spírnt-2
		USN-4	Evaluate the Dataset	Data is evaluated before processing	Medium	Spíint-1
		USN-5	Identify accuracy,precision,recall of Each algorithms	These values are important for obtaining the light output	High	Spíint-3
		USN-6	Outputs from each algorithm are obtained	It is highly used to predict the effect and to take precaution aey measures	High	Spíint-4

CHAPTER 6
PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

Sprint planning is an event in scrum that kicks off the sprint. The purpose of sprint planning is to define what can be delivered in the sprint and how that work will be achieved. Sprint planning is done in collaboration with the whole scrum team.

The below table 6.1 shows the Sprint Planning and estimation for Inventory Management system for Retailers

Sprint Planning & Estimation

SPRINT	FUNCTIONAL REQUIREMENT	USER STORY NUMBER	USER STORY / TASK	STORY POINTS	PRIORITY
Sprint-1	Saving the Model USN	USN-1	As a developer saving the model developed for estimation of fire 10 high	10	High
Sprint-2	Video Analysis	USN-2	.	10	Medium
Sprint-3	Twilio Message Service	USN-3	As a user, I can log in to the authorized account by Entering the registered email and password.	3	Low

Sprint-4	Train Model on cloud	USN-5	Application Deployment on Local System/ Cloud Local Server Congifuration: Cloud Server Configuration:and to train the deep learning model in IBM Cloud	10	Medium

Table 6.1: Sprint Planning and estimation

6.2 SPRINT DELIVERY SCHEDULE

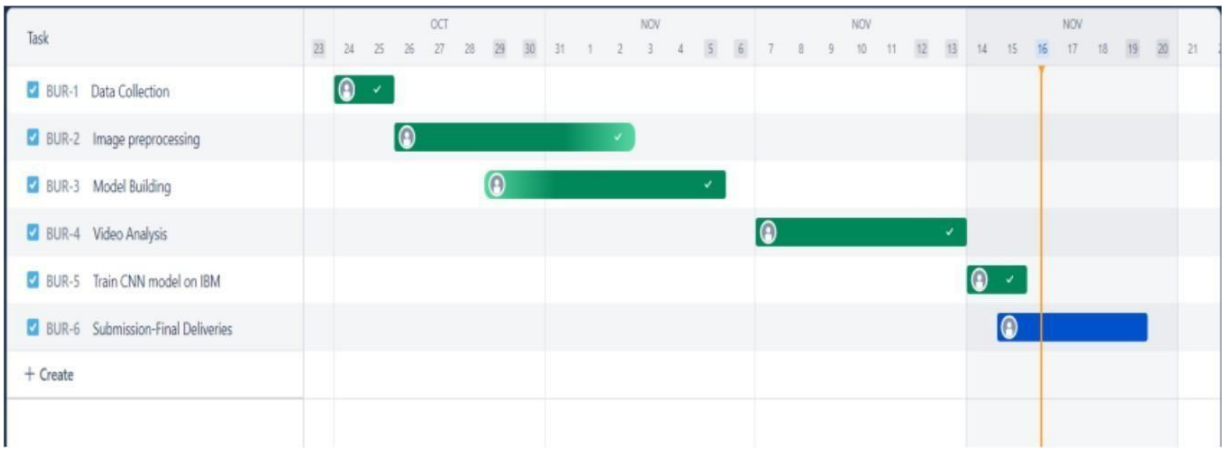
SPRINT	TOTAL STORY POINTS	DURATION	SPRINT START DATE	SPRINT END DATE (PLANNED)	STORY POINTS COMPLETED (AS ON PLANNED END DATE)	SPRINT RELEASE DATE (ACTUAL)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

The following table shows the sprint works assigned to the members along with the priority and story points assigned with the functional requirements with regards to user story

Table 6.2: Sprint Planning done

Reports from JIRA

Buildown Chart:



CHAPTER 7

CODING & SOLUTION

FEATURE 1:

./TRAIN \

```
import json
import os
import tensorflow as tf
from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util
tf.logging.set_verbosity(tf.logging.INFO)
flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.  Note that even if '
                     'set to False (allowing ops to run on gpu), some ops may '
                     'still be run on the CPU if they have no GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and training summaries.')
flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are ignored')
flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config file.')
FLAGS = flags.FLAGS
@tf.contrib.framework.deprecated(None, 'Use object_detection/model_main.py.')
```

```
def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir, 'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
      for name, config in [('model.config', FLAGS.model_config_path),
                           ('train.config', FLAGS.train_config_path),
                           ('input.config', FLAGS.input_config_path)]:
        tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                      overwrite=True)
  model_config = configs['model']
  train_config = configs['train_config']
  input_config = configs['train_input_config']
  model_fn = functools.partial(
      model_builder.build,
      model_config=model_config,
      is_training=True)
  def get_next(config):
    return dataset_builder.make_initializable_iterator(
        dataset_builder.build(config)).get_next()
  create_input_dict_fn = functools.partial(get_next, input_config)
  env = json.loads(os.environ.get('TF_CONFIG', '{}'))
  cluster_data = env.get('cluster', None)
  cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
  task_data = env.get('task', None) or {'type': 'master', 'index': 0}
  task_info = type('TaskSpec', (object,), task_data)
  # Parameters for a single worker.
  ps_tasks = 0
  worker_replicas = 1
  worker_job_name = 'lonely_worker'
  task = 0
  is_chief = True
```

```
  master = ''
  if cluster_data and 'worker' in cluster_data:
    # Number of total worker replicas include "worker"s and the "master".
    worker_replicas = len(cluster_data['worker']) + 1
  if cluster_data and 'ps' in cluster_data:
    ps_tasks = len(cluster_data['ps'])
  if worker_replicas > 1 and ps_tasks < 1:
    raise ValueError('At least 1 ps task is needed for distributed training.')
  if worker_replicas >= 1 and ps_tasks > 0:
    # Set up distributed training.
    server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc',
                             job_name=task_info.type,
                             task_index=task_info.index)
    if task_info.type == 'ps':
      server.join()
      return
    worker_job_name = '%s/task:%d' % (task_info.type, task_info.index)
    task = task_info.index
    is_chief = (task_info.type == 'master')
    master = server.target
  graph_rewriter_fn = None
  if 'graph_rewriter_config' in configs:
    graph_rewriter_fn = graph_rewriter_builder.build(
        configs['graph_rewriter_config'], is_training=True)
  trainer.train(
      create_input_dict_fn,
      model_fn,
      train_config,
      master,
      task,
      FLAGS.num_clones,
      worker_replicas,
      FLAGS.clone_on_cpu,
      ps_tasks,
      worker_job_name,
      is_chief,
      FLAGS.train_dir,
      graph_hook_fn=graph_rewriter_fn)
if __name__ == '__main__':
  tf.app.run()
```

FEATURE 2:

SOURCE CODE

App.py

```
from flask import Flask, render_template, url_for, request, redirect, session,
make_response

import sqlite3 as sql

from functools import wraps

import re

import ibm_db

import os

from sendgrid import SendGridAPIClient

from sendgrid.helpers.mail import Mail

from datetime import datetime, timedelta

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=815fa4db-dc03-4c70-869a-
a9cc13f33084.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30367;SECURI
TY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=gkx49901;PWD=kvWC
sySl7vApfsy2", '', '')

app = Flask(_name_)

app.secret_key = 'jackiechan'


def rewrite(url):

    view_func, view_args = app.create_url_adapter(request).match(url)

    return app.view_functions[view_func](**view_args)




def login_required(f):

    @wraps(f)
```

```
    def decorated_function(*args, **kwargs):
        if "id" not in session:
            return redirect(url_for('login'))
        return f(*args, **kwargs)
    return decorated_function


@app.route('/')
def root():
    return render_template('login.html')


@app.route('/user/<id>')
@login_required
def user_info(id):
    with sql.connect('inventorymanagement.db') as con:
        con.row_factory = sql.Row
        cur = con.cursor()
        cur.execute(f'SELECT * FROM users WHERE email="{id}"')
        user = cur.fetchall()
    return render_template("user_info.html", user=user[0])


@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ''
```

```
    if request.method == 'POST':
        un = request.form['username']
        pd = request.form['password_1']
        print(un, pd)
        sql = "SELECT * FROM users WHERE email =? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, un)
        ibm_db.bind_param(stmt, 2, pd)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['loggedin'] = True
            session['id'] = account['EMAIL']
            userid = account['EMAIL']
            session['username'] = account['USERNAME']
            msg = 'Logged in successfully !'

            return rewrite('/dashboard')
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg=msg)


@app.route('/signup', methods=['POST', 'GET'])
def signup():
    mg = ''
    if request.method == "POST":
```

```
        username = request.form['username']
        email = request.form['email']
        pw = request.form['password']
        sql = 'SELECT * FROM users WHERE email =?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.execute(stmt)
        acnt = ibm_db.fetch_assoc(stmt)
        print(acnt)

        if acnt:
            mg = 'Account already exits!!'

        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            mg = 'Please enter the avalid email address'
        elif not re.match(r'[A-Za-z0-9]+', username):
            ms = 'name must contain only character and number'
        else:
            insert_sql = 'INSERT INTO users
(USERNAME,FIRSTNAME,LASTNAME,EMAIL,PASSWORD) VALUES (?,?,?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, username)
            ibm_db.bind_param(pstmt, 2, "firstname")
            ibm_db.bind_param(pstmt, 3, "lastname")
            # ibm_db.bind_param(pstmt,4,"123456789")
            ibm_db.bind_param(pstmt, 4, email)
            ibm_db.bind_param(pstmt, 5, pw)
            print(pstmt)
```

```
            ibm_db.execute(pstmt)
            mg = 'You have successfully registered click login!'
            message = Mail(
                from_email=os.environ.get('MAIL_DEFAULT_SENDER'),
                to_emails=email,
                subject='New SignUp',
                html_content='<p>Hello, Your Registration was successfull. <br><br> Thank you for choosing us.</p>')

            sg = SendGridAPIClient(
                api_key=os.environ.get('SENDGRID_API_KEY'))

            response = sg.send(message)
            print(response.status_code, response.body)
            return render_template("login.html", meg=mg)

    elif request.method == 'POST':
        msg = "fill out the form first!"
    return render_template("signup.html", meg=mg)


@app.route('/dashboard', methods=['POST', 'GET'])
@login_required
def dashBoard():
    sql = "SELECT * FROM stocks"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    stocks = []
```

```
    headings = [*dictionary]
    while dictionary != False:
        stocks.append(dictionary)
        # print(f"The ID is : ", dictionary["NAME"])
        # print(f"The name is : ", dictionary["QUANTITY"])
        dictionary = ibm_db.fetch_assoc(stmt)

    return render_template("dashboard.html", headings=headings, data=stocks)

@app.route('/addstocks', methods=['POST'])
@login_required
def addStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            quantity = request.form['quantity']
            price = request.form['price']
            total = int(price) * int(quantity)
            insert_sql = 'INSERT INTO stocks
(NAME,QUANTITY,PRICE_PER_QUANTITY,TOTAL_PRICE) VALUES (?,?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.bind_param(pstmt, 2, quantity)
            ibm_db.bind_param(pstmt, 3, price)
            ibm_db.bind_param(pstmt, 4, total)
            ibm_db.execute(pstmt)
```

```
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))


@app.route('/updatestocks', methods=['POST'])
@login_required
def UpdateStocks():
    if request.method == "POST":
        try:
            item = request.form['item']
            print("hello")
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE stocks SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
            if field == 'PRICE_PER_QUANTITY' or field == 'QUANTITY':
                insert_sql = 'SELECT * FROM stocks WHERE NAME= ?'
```

```
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
            dictonary = ibm_db.fetch_assoc(pstmt)
            print(dictonary)
            total = dictonary['QUANTITY'] * dictonary['PRICE_PER_QUANTITY']
            insert_sql = 'UPDATE stocks SET TOTAL_PRICE=? WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, total)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))



@app.route('/deletestocks', methods=['POST'])
@login_required
def deleteStocks():
    if request.method == "POST":
        print(request.form['item'])
        try:
            item = request.form['item']
            insert_sql = 'DELETE FROM stocks WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
```

```
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
            return redirect(url_for('dashBoard'))



@app.route('/update-user', methods=['POST', 'GET'])
@login_required
def updateUser():
    if request.method == "POST":
        try:
            email = session['id']
            field = request.form['input-field']
            value = request.form['input-value']
            insert_sql = 'UPDATE users SET ' + field + '= ? WHERE EMAIL=?'
            pstmt  = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, email)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e

        finally:
            # print(msg)
```

```
        return redirect(url_for('profile'))



@app.route('/update-password', methods=['POST', 'GET'])
@login_required
def updatePassword():
    if request.method == "POST":
        try:
            email = session['id']
            password = request.form['prev-password']
            curPassword = request.form['cur-password']
            confirmPassword = request.form['confirm-password']
            insert_sql = 'SELECT * FROM  users WHERE EMAIL=? AND PASSWORD=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.bind_param(pstmt, 2, password)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
            if curPassword == confirmPassword:
                insert_sql = 'UPDATE users SET PASSWORD=? WHERE EMAIL=?'
                pstmt = ibm_db.prepare(conn, insert_sql)
                ibm_db.bind_param(pstmt, 1, confirmPassword)
                ibm_db.bind_param(pstmt, 2, email)
                ibm_db.execute(pstmt)
        except Exception as e:
            msg = e
        finally:
```

```
            # print(msg)
            return render_template('result.html')


@app.route('/orders', methods=['POST', 'GET'])
@login_required
def orders():
    query = "SELECT * FROM orders"
    stmt = ibm_db.exec_immediate(conn, query)
    dictionary = ibm_db.fetch_assoc(stmt)
    orders = []
    headings = [*dictionary]
    while dictionary != False:
        orders.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template("orders.html", headings=headings, data=orders)


@app.route('/createOrder', methods=['POST'])
@login_required
def createOrder():
    if request.method == "POST":
        try:
            stock_id = request.form['stock_id']
            query = 'SELECT PRICE_PER_QUANTITY FROM stocks WHERE ID= ?'
            stmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(stmt, 1, stock_id)
            ibm_db.execute(stmt)
```

```
            dictionary = ibm_db.fetch_assoc(stmt)
            if dictionary:
                quantity = request.form['quantity']
                date = str(datetime.now().year) + "-" + str(
                    datetime.now().month) + "-" + str(datetime.now().day)
                delivery = datetime.now() + timedelta(days=7)
                delivery_date = str(delivery.year) + "-" + str(
                    delivery.month) + "-" + str(delivery.day)
                price = float(quantity) * \
                    float(dictionary['PRICE_PER_QUANTITY'])
                query = 'INSERT INTO orders
(STOCKS_ID,QUANTITY,DATE,DELIVERY_DATE,PRICE) VALUES (?,?,?,?,?)'
                pstmt = ibm_db.prepare(conn, query)
                ibm_db.bind_param(pstmt, 1, stock_id)
                ibm_db.bind_param(pstmt, 2, quantity)
                ibm_db.bind_param(pstmt, 3, date)
                ibm_db.bind_param(pstmt, 4, delivery_date)
                ibm_db.bind_param(pstmt, 5, price)
                ibm_db.execute(pstmt)
        except Exception as e:
            print(e)


        finally:
            return redirect(url_for('orders'))



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

```
def updateOrder():
    if request.method == "POST":
        try:
            item = request.form['item']
            field = request.form['input-field']
            value = request.form['input-value']
            query = 'UPDATE orders SET ' + field + "= ?" + " WHERE ID=?"
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))



@app.route('/cancelOrder', methods=['POST'])
@login_required
def cancelOrder():
    if request.method == "POST":
        try:
            order_id = request.form['order_id']
            query = 'DELETE FROM orders WHERE ID=?'
            pstmt = ibm_db.prepare(conn, query)
            ibm_db.bind_param(pstmt, 1, order_id)
            ibm_db.execute(pstmt)
```

```
        except Exception as e:
            print(e)

        finally:
            return redirect(url_for('orders'))



@app.route('/suppliers', methods=['POST', 'GET'])
@login_required
def suppliers():
    sql = "SELECT * FROM suppliers"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    suppliers = []
    orders_assigned = []
    headings = [*dictionary]
    while dictionary != False:
        suppliers.append(dictionary)
        orders_assigned.append(dictionary['ORDER_ID'])
        dictionary = ibm_db.fetch_assoc(stmt)

# get order ids from orders table and identify unassigned order ids
    sql = "SELECT ID FROM orders"
    stmt = ibm_db.exec_immediate(conn, sql)
    dictionary = ibm_db.fetch_assoc(stmt)
    order_ids = []
    while dictionary != False:
        order_ids.append(dictionary['ID'])
```

```
        dictionary = ibm_db.fetch_assoc(stmt)


    unassigned_order_ids = set(order_ids) - set(orders_assigned)
    return
render_template("suppliers.html",headings=headings,data=suppliers,order_ids=unassigned
_order_ids)


@app.route('/updatesupplier', methods=['POST'])
@login_required
def UpdateSupplier():
    if request.method == "POST":
        try:
            item = request.form['name']
            field = request.form['input-field']
            value = request.form['input-value']
            print(item, field, value)
            insert_sql = 'UPDATE suppliers SET ' + field + "= ?" + " WHERE NAME=?"
            print(insert_sql)
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, value)
            ibm_db.bind_param(pstmt, 2, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e


        finally:
            return redirect(url_for('suppliers'))
```

```
@app.route('/addsupplier', methods=['POST'])
@login_required
def addSupplier():
    if request.method == "POST":
        try:
            name = request.form['name']
            order_id = request.form.get('order-id-select')
            print(order_id)
            print("Hello world")
            location = request.form['location']
            insert_sql = 'INSERT INTO suppliers (NAME,ORDER_ID,LOCATION)
VALUES (?,?,?)'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, name)
            ibm_db.bind_param(pstmt, 2, order_id)
            ibm_db.bind_param(pstmt, 3, location)
            ibm_db.execute(pstmt)

        except Exception as e:
            msg = e

        finally:
            return redirect(url_for('suppliers'))

@app.route('/deletesupplier', methods=['POST'])
@login_required
def deleteSupplier():
    if request.method == "POST":
```

```
        try:
            item = request.form['name']
            insert_sql = 'DELETE FROM suppliers WHERE NAME=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, item)
            ibm_db.execute(pstmt)
        except Exception as e:
            msg = e


        finally:
            return redirect(url_for('suppliers'))
@app.route('/profile', methods=['POST', 'GET'])
@login_required
def profile():
    if request.method == "GET":
        try:
            email = session['id']
            insert_sql = 'SELECT * FROM users WHERE EMAIL=?'
            pstmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(pstmt, 1, email)
            ibm_db.execute(pstmt)
            dictionary = ibm_db.fetch_assoc(pstmt)
            print(dictionary)
        except Exception as e:
            msg = e
        finally:
            # print(msg)
            return render_template("profile.html", data=dictionary)
```

```
@app.route('/logout', methods=['GET'])
@login_required
def logout():
    print(request)
    resp = make_response(render_template("login.html"))
    session.clear()
    return resp
if__name__== '_main_':
    app.run(debug=True)
```

CHAPTER 8
TESTING AND RESULTS

8.1 TEST CASE

```
import cv2
import numpy as np
from keras.preprocessing
import image from keras.models
import load_model from twilio.rest
import Client from playsound
import playsound
model=load_model('forest1.h5')
video=cv2.VideoCapture(0)
name=['forest','with fire']
while(True):
ret,frame=video.read()
cv2.imshow('frame',frame)
cv2.imwrite('image.jpg',frame)
img=image.load_img('train_set/forest/NoFire (1).bmp',target_size=(64,64))
x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
pred=model.predict(x)
index=np.argmax(pred)
if index==0:
account_sid=' AC805dad0282a5eaebfa3149365f1c605d
' auth_token=' a27b535245ca70f9c7b4875865a620e1
'client =Client(account_sid,auth_token)
message=client.messages \
```

```
.create(body='-------Fire is detected,Stay Alert !!!----------',
from_='+ 12232174143
',to='+916300780739')
print(message.sid)
print('Fire detected')
print("Alert Message sent!")
playsound('tornado-siren.mp3')
else:
print('No Danger')
cv2.imshow("image.jpg",frame)
if cv2.waitkey(2)&0xff == ord('q'):
break
video.release()
cv2.destroyAllWindows()
```

Fig 8.1: Sign Up page for Emerging Method for Early Detection of Forest Fire

Fig 8.2: Login Page for Emerging Method for Early Detection of Forest Fire

CHAPTER 9

PERFORMANCE RESULTS

9.1 PERFORMANCE METRICES

Fig 9.1 shows the performance of result

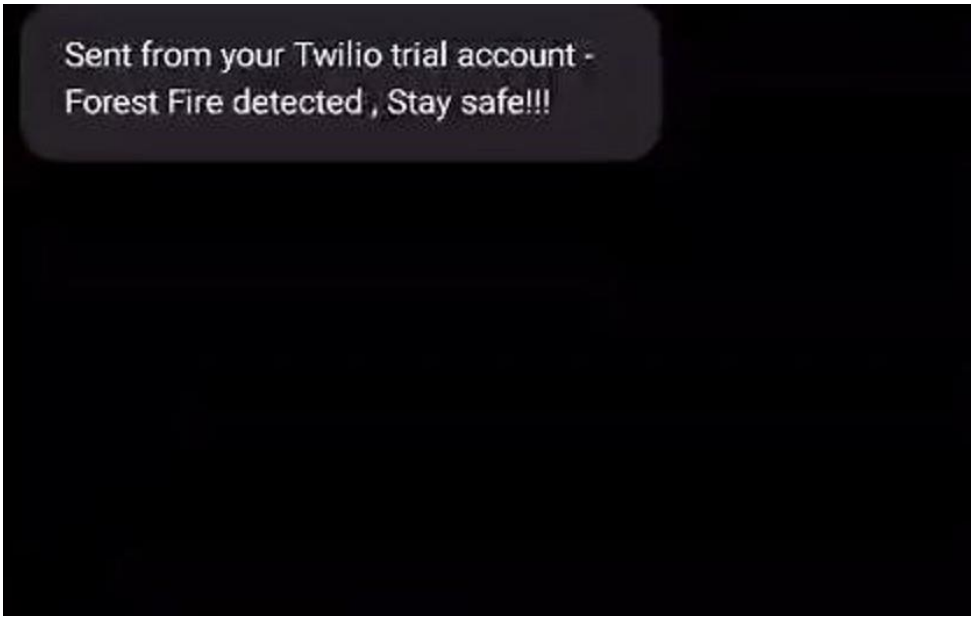


Fig 9.1: Performance results

CHAPTER 10

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- easily detect and Estimate the Forest Fire.
- Most Accurate
- Flexible Model which can give maximized outcome
- No Specific Requirements needed to implement the model

DISADVANTAGES

- Training model is time consuming process.
- Error in CV can cause damage to camera
- Access of camera are prohibited due to personal issues

CHAPTER 11

CONCLUSION

Thus we have constructed a model that can identify the effects of the forest fire and it can analyses the forest fire by advanced AI techniques and CNN Algorithm then the Prediction model is Checked and then the model is connected with Twilio account credentials of the Developer consisting of phone numbers of the persons in the surroundings of the people in the area of easy forest fire zone then an security sound alert system is developed to make a alert sound which is downloaded from internet then the entire model is deployed to the IBM Cloud account that we have created was made with the studies we have done

.

CHAPTER 12
FUTURE SCOPE

1. It can be developed as a Web or Android Application.
2. In future Alternate Advanced technologies can be implemented.
3. The Identification and tracking system can be implemented if possible.

GitHub: https://github.com/IBM-EPBL/IBM-Project-7722-1658896724

REFERENCES

1. Georgi Hristov; Jordan Raychev; Diyana Kinaneva; Plamen Zahariev , Published in: 2018 28th EAEEIE Annual Conference (EAEEIE).

2. Chi Yuan, Youmin Zhang, and Zhixiang Liu , Published in: 2015, Published by NRC Research Press.

3. Mohamed Hefeeda and Majid Bagheri , Published in: June 26, 2008.

4. PRIYADARSHINI M HANAMARADDI , Published in: January 2016.

5. Dr. Panagiotis Barmpoutis, Periklis Papaioannou, Dr. Kosmas Dimitropoulos, Dr. Nikos GRAMMALIDIS , Published in: 11 November 2020.

6. Vinay Chowdary , Mukul Kumar Gupta , Rajesh Singh, Published in:2018

7. Majid Bahrepour, Nirvana Meratnia, Paul Havinga , Published in: January 2008.

8. Dr.L.Latha , Published in: January 2015 9. P. Piccinini, S. Calderara, and R. Cucchiara ,Published in: September, 2006.