**CODE**

**The below py codes are for preprocessing we can run individually**

**Colorfiltering.py**

import cv2

import os

# ✅ Fix the file path (Use raw string OR double backslashes)

image\_path = r"F:\LM'S\Main Project\Image\_Detection\train\images\tomato.jpg"

output\_folder = r"F:\LM'S\Main Project\Image\_Detection\preprocessed\_images\color\_filtered"

# ✅ Create output folder if it doesn't exist

os.makedirs(output\_folder, exist\_ok=True)

# ✅ Check if file exists

if not os.path.exists(image\_path):

print(f"Error: Image not found at {image\_path}")

else:

image = cv2.imread(image\_path)

# ✅ Check if image loaded successfully

if image is None:

print("Error: Unable to load image. Check file format or path.")

else:

# Convert to HSV

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

# ✅ Save processed image

output\_path = os.path.join(output\_folder, "color\_filtered.jpg")

cv2.imwrite(output\_path, hsv)

# ✅ Show output

cv2.imshow("Color Filtered", hsv)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Flipping.py**

import cv2

import os

# ✅ Define paths

input\_path = r"F:\LM'S\Main Project\Image\_Detection\train\images\tomato.jpg"

output\_folder = r"F:\LM'S\Main Project\Image\_Detection\preprocessed\_images\flipped"

# ✅ Create output folder

os.makedirs(output\_folder, exist\_ok=True)

# ✅ Check if file exists

if not os.path.exists(input\_path):

print(f"Error: Image not found at {input\_path}")

else:

image = cv2.imread(input\_path)

# ✅ Check if image loaded successfully

if image is None:

print("Error: Unable to load image.")

else:

# Flip image

flipped = cv2.flip(image, 1) # Horizontal flip

# ✅ Save and show result

output\_path = os.path.join(output\_folder, "flipped.jpg")

cv2.imwrite(output\_path, flipped)

cv2.imshow("Flipped", flipped)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Normalization.py**

import cv2

import numpy as np

import os

# ✅ Define paths

input\_path = r"F:\LM'S\Main Project\Image\_Detection\train\images\tomato.jpg"

output\_folder = r"F:\LM'S\Main Project\Image\_Detection\preprocessed\_images\normalized"

# ✅ Create output folder

os.makedirs(output\_folder, exist\_ok=True)

# ✅ Check if file exists

if not os.path.exists(input\_path):

print(f"Error: Image not found at {input\_path}")

else:

image = cv2.imread(input\_path)

# ✅ Check if image loaded successfully

if image is None:

print("Error: Unable to load image.")

else:

# Normalize pixel values

normalized = image / 255.0

normalized\_img = (normalized \* 255).astype(np.uint8) # Convert back for saving

# ✅ Save and show result

output\_path = os.path.join(output\_folder, "normalized.jpg")

cv2.imwrite(output\_path, normalized\_img)

cv2.imshow("Normalized", normalized\_img)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Resizing.py**

import cv2

import os

# ✅ Define paths

input\_path = r"F:\LM'S\Main Project\Image\_Detection\train\images\tomato.jpg"

output\_folder = r"F:\LM'S\Main Project\Image\_Detection\preprocessed\_images\resized"

# ✅ Create output folder

os.makedirs(output\_folder, exist\_ok=True)

# ✅ Check if file exists

if not os.path.exists(input\_path):

print(f"Error: Image not found at {input\_path}")

else:

image = cv2.imread(input\_path)

# ✅ Check if image loaded successfully

if image is None:

print("Error: Unable to load image.")

else:

# Resize image

resized = cv2.resize(image, (128, 128))

# ✅ Save and show result

output\_path = os.path.join(output\_folder, "resized.jpg")

cv2.imwrite(output\_path, resized)

cv2.imshow("Resized", resized)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Thresholding.py**

import cv2

import os

# ✅ Define paths

image\_path = r"F:\LM'S\Main Project\Image\_Detection\train\images\tomato.jpg"

output\_folder = r"F:\LM'S\Main Project\Image\_Detection\preprocessed\_images\thresholded"

os.makedirs(output\_folder, exist\_ok=True)

# ✅ Check if file exists

if not os.path.exists(image\_path):

print(f"Error: Image not found at {image\_path}")

else:

image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

# ✅ Apply Adaptive Thresholding

thresholded = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,

cv2.THRESH\_BINARY, 11, 2)

# ✅ Save and show result

output\_path = os.path.join(output\_folder, "adaptive\_thresholded.jpg")

cv2.imwrite(output\_path, thresholded)

cv2.imshow("Adaptive Thresholding - High Contrast", thresholded)

cv2.waitKey(0)

cv2.destroyAllWindows()

**Training the codes based on the dataset for application**

* Training the model based on Random Forest Algorithm

import os

import cv2

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.model\_selection import train\_test\_split

def load\_data(image\_dir, label\_dir):

X, y = [], []

for img\_name in os.listdir(image\_dir):

if img\_name.endswith('.jpg') or img\_name.endswith('.png'):

img\_path = os.path.join(image\_dir, img\_name)

label\_path = os.path.join(label\_dir, img\_name.replace('.jpg', '.txt').replace('.png', '.txt'))

if not os.path.exists(label\_path):

print(f"Skipping {label\_path}: label file not found")

continue

with open(label\_path, 'r') as f:

lines = f.readlines()

if len(lines) == 0:

print(f"Skipping {label\_path}: empty label file")

continue

line = lines[0].strip()

if not line:

print(f"Skipping {label\_path}: empty line inside label file")

continue

try:

class\_id = int(line.split()[0])

except (IndexError, ValueError):

print(f"Skipping {label\_path}: malformed label line")

continue

img = cv2.imread(img\_path)

if img is None:

print(f"Skipping {img\_path}: unable to read image")

continue

img = cv2.resize(img, (100, 100))

hist = cv2.calcHist([img], [0, 1, 2], None, [8, 8, 8], [0, 256, 0, 256, 0, 256])

hist = cv2.normalize(hist, hist).flatten()

X.append(hist)

y.append(class\_id)

return X, y

# Dataset paths

base\_path = r"F:/LM'S/Main Project/Image\_Detection"

datasets = ['train', 'valid', 'test']

X\_total, y\_total = [], []

for split in datasets:

image\_dir = os.path.join(base\_path, split, 'images')

label\_dir = os.path.join(base\_path, split, 'labels')

X, y = load\_data(image\_dir, label\_dir)

X\_total.extend(X)

y\_total.extend(y)

X\_total = np.array(X\_total)

y\_total = np.array(y\_total)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_total, y\_total, test\_size=0.2, random\_state=42)

# Train Random Forest

rf = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf.fit(X\_train, y\_train)

# Evaluation

y\_pred = rf.predict(X\_test)

print("\n=== Random Forest Results ===")

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1-Score:", f1\_score(y\_test, y\_pred))

* Training the model based on CNN

import os

import cv2

import numpy as np

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

def load\_data(image\_dir, label\_dir):

X, y = [], []

for img\_name in os.listdir(image\_dir):

if img\_name.endswith('.jpg') or img\_name.endswith('.png'):

img\_path = os.path.join(image\_dir, img\_name)

label\_path = os.path.join(label\_dir, img\_name.replace('.jpg', '.txt').replace('.png', '.txt'))

if not os.path.exists(label\_path):

continue

with open(label\_path, 'r') as f:

lines = f.readlines()

if len(lines) == 0:

continue

line = lines[0].strip()

if not line:

continue

try:

class\_id = int(line.split()[0])

except (IndexError, ValueError):

continue

img = cv2.imread(img\_path)

if img is None:

continue

img = cv2.resize(img, (100, 100))

img = img / 255.0 # Normalize

X.append(img)

y.append(class\_id)

return X, y

# Dataset paths

base\_path = r"F:/LM'S/Main Project/Image\_Detection"

datasets = ['train', 'valid', 'test']

X\_total, y\_total = [], []

for split in datasets:

image\_dir = os.path.join(base\_path, split, 'images')

label\_dir = os.path.join(base\_path, split, 'labels')

X, y = load\_data(image\_dir, label\_dir)

X\_total.extend(X)

y\_total.extend(y)

X\_total = np.array(X\_total)

y\_total = np.array(y\_total)

y\_total = to\_categorical(y\_total, num\_classes=2) # convert to one-hot

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_total, y\_total, test\_size=0.2, random\_state=42)

# CNN Model

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(100, 100, 3)),

MaxPooling2D((2, 2)),

Conv2D(64, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Conv2D(128, (3, 3), activation='relu'),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(2, activation='softmax')

])

model.compile(optimizer=Adam(learning\_rate=0.001), loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

# Train CNN

model.fit(X\_train, y\_train, epochs=10, batch\_size=32, validation\_split=0.1)

# Evaluation

y\_pred = model.predict(X\_test)

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true = np.argmax(y\_test, axis=1)

print("\n=== CNN Results ===")

print(classification\_report(y\_true, y\_pred\_labels, target\_names=['ripe', 'unripe']))

# Save model

model.save(r"F:\LM'S\Main Project\Image\_Detection\models\tomato\_cnn\_model.keras")

* Integrating CNN with app.py for website

from flask import Flask, render\_template, request

import cv2

import numpy as np

import os

from tensorflow.keras.models import load\_model

app = Flask(\_\_name\_\_)

model = load\_model(r"F:\LM'S\Main Project\Image\_Detection\models\tomato\_cnn\_model.h5") # Load trained CNN model

UPLOAD\_FOLDER = 'static/uploads'

os.makedirs(UPLOAD\_FOLDER, exist\_ok=True)

@app.route('/')

def index():

return render\_template('index.html', prediction=None, image\_path=None)

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

def upload():

if 'file' not in request.files:

return render\_template('index.html', prediction="No file part", image\_path=None)

file = request.files['file']

if file.filename == '':

return render\_template('index.html', prediction="No file selected", image\_path=None)

file\_path = os.path.join(UPLOAD\_FOLDER, file.filename)

file.save(file\_path)

return render\_template('index.html', image\_path=file\_path, prediction=None)

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

def predict():

image\_path = request.form['image\_path']

img = cv2.imread(image\_path)

img = cv2.resize(img, (100, 100))

img = img / 255.0

img = np.expand\_dims(img, axis=0)

pred = model.predict(img)

label = np.argmax(pred)

class\_names = ['ripe', 'unripe']

result = class\_names[label]

return render\_template('index.html', prediction=result, image\_path=image\_path)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

* Line Plot and Bar Plotting

import numpy as np

import matplotlib.pyplot as plt

# ✅ Define the models and their metrics

models = ["Random Forest (RF)", "CNN with MobileNet"]

metrics = ["Accuracy", "Precision", "Recall", "F1-Score"]

# ✅ Performance data (as per your provided table)

data = {

"Random Forest (RF)": [88.5, 86.2, 84.8, 85.5],

"CNN with MobileNet": [92.3, 91.5, 91.1, 91.1]

}

# ✅ Convert data into a NumPy array

values = np.array([data[model] for model in models])

# ✅ Colors for each metric

colors = ["blue", "green", "red", "purple"]

# ✅ Create the Bar Chart

plt.figure(figsize=(8, 5))

bar\_width = 0.2 # Width of bars

x = np.arange(len(models)) # X-axis positions

for i, metric in enumerate(metrics):

plt.bar(x + i \* bar\_width, values[:, i], width=bar\_width, color=colors[i], label=f"{metric} (%)")

plt.xlabel("Models")

plt.ylabel("Performance Metrics (%)")

plt.title("Comparison of Machine Learning and Deep Learning Models")

plt.xticks(x + bar\_width, models) # Adjust x-ticks to align

plt.legend()

plt.grid(axis="y", linestyle="--", alpha=0.7)

plt.show()

# ✅ Create the Line Chart

plt.figure(figsize=(8, 5))

for i, metric in enumerate(metrics):

plt.plot(models, values[:, i], marker="o", linestyle="-", color=colors[i], label=f"{metric} (%)")

* Home page (home.html in templates)

plt.xlabel("Models")

plt.ylabel("Performance Metrics (%)")

plt.title("Performance Comparison of ML and DL Models")

plt.legend()

plt.grid(True, linestyle="--", alpha=0.7)

plt.show()

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Tomato Ripeness Detector</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='style.css') }}">

</head>

<body>

<h1>Tomato Ripeness Detection</h1>

<form action="/upload" method="post" enctype="multipart/form-data">

<input type="file" name="file">

<button type="submit">Upload</button>

</form>

{% if image\_path %}

<h3>Uploaded Image:</h3>

<img src="{{ image\_path }}" alt="Tomato Image" width="300">

<form action="/predict" method="post">

<input type="hidden" name="image\_path" value="{{ image\_path }}">

<button type="submit">Predict</button>

</form>

{% endif %}

{% if prediction %}

<h2>Prediction: {{ prediction }}</h2>

{% endif %}

</body>

</html>

* Style.css

body {

font-family: Arial, sans-serif;

background-color: #f8f8f8;

text-align: center;

padding: 40px;

}

h1 {

color: #333;

margin-bottom: 20px;

}

form {

margin: 20px auto;

}

input[type="file"] {

padding: 10px;

margin-bottom: 10px;

}

button {

padding: 10px 20px;

background-color: #28a745;

color: #fff;

border: none;

border-radius: 5px;

cursor: pointer;

transition: background-color 0.3s ease;

}

button:hover {

background-color: #218838;

}

img {

margin-top: 20px;

border: 2px solid #ddd;

border-radius: 10px;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

h3, h2 {

color: #555;

}

.result {

margin-top: 20px;

padding: 15px;

background: #fff;

display: inline-block;

border-radius: 8px;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

* Integrating the Image detection code which is based on CNN with Raspberry Pi for hardware

import argparse

import sys

import time

import cv2

import mediapipe as mp

from mediapipe.tasks import python from mediapipe.tasks.python import vision

from utils import visualize from picamera2 import Picamera2

Global variables to calculate FPS

COUNTER, FPS = 0, 8

START TIME = time.time()

picam2= Picamera2()

picam2.preview\_configuration.main.size = (640,480)

picam2.preview\_configuration.main.format = "RGB888"

picam2.preview\_configuration.align()

picam2.configure("preview")

picam2.start()

def run(model: str, max\_results: int, score\_threshold: float, camera\_id: int, width: int, height: int) -> None:

row\_size = 50

left\_margin = 24

text color = (0, 0, 0)

#black

font size=1

font\_thickness = 1

fps\_avg\_frame\_count = 10

detection frame = None

detection result list = []

def save\_result(result: vision.ObjectDetectorResult, unused\_output\_image: mp.Image, timestamp\_ms: int):

global FPS, COUNTER, START\_TIME

if COUNTER % fps\_avg\_frame\_count == 0:

FPS=fps\_avg\_frame\_count/ (time.time() - START\_TIME)

START\_TIME = time.time()

detection\_result\_list.append(result)

COUNTER += 1

base\_options = python. BaseOptions(model\_asset\_path-model)

options=vision.ObjectDetectorOptions(base\_options=base\_options,

running\_mode=vision. RunningMode. LIVE\_STREAM,

max\_results=max\_results, score\_threshold=score\_threshold, result\_callback=save\_result)

detector =vision.ObjectDetector.create\_from\_options(options)

while True:

im= picam2.capture\_array()

image=cv2.resize(im, (640,480))

image = cv2.flip(image, -1)

rgb\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

mp\_image = mp.Image(image\_format=mp.ImageFormat.SRGB, data=rgb\_image)

detector.detect\_async(mp\_image, time.time\_ns() // 1\_000\_000)

fps\_text = 'FPS= {:.1f}'.format{FPS}

text\_location= (left\_margin, row\_size)

current frame = image

cv2.putText(current\_frame, fps\_text, text\_location. cv2.FONT\_HERSHEY\_DUPLEX,

font\_size, text\_color, font\_thickness, cv2.LINE\_AA)

if detection\_result\_list:

print(detection\_result\_list)

current\_frame =visualize{current frame, detection result\_list[0]}

detection\_frame= current\_fram

detection\_result\_list.clear()

if detection\_frame is not None:

cv2.imshow('object\_detection', detection\_frame)

if cv2.waitKey(1) == 27:

break

detector.close()

cap.release()

cv2.destroyAllWindows()

def main():

parser=argparse.ArgumentParser(formatter\_class=argparse.ArgumentDefaultsHelpFormatter)

parser.add argument(

'--model',

help='Path of the object detection model',

required=False,

default ='tomato.flite')

parser.add argument(

'--maxResults',

help ='Max no of detection results',

required=False,

default='tomato.tflite')

parser.add argument(

'--maxResults',

help=' Max no of detection results',

required=False,

default=5)

parser.add\_argument (

'--scoreThreshold',

help='The score threshold of detection results',

required=False

type=float,

default=0.25)

parser.add\_argument(

'--camerald', help='Id of camera.', required=False, type=int, default=0)

parser.add\_argument(

'--frameWidth',

help='Width of frame to capture from camera.',

required=False,

type=int,

default=640)

parser.add\_argument(

'--frameHeight',

help='Height of frame to capture from camera',

required=False,

type=int,

default=480)

args=parser.parse\_args()

run(args.model, int(args.maxResults),

args.scoreThreshold, int(args.camerald), args.frameWidth, args.frameHeight)

if \_name=='main\_':

main()