Face Detection in Jetson nano

This project is a real-time face recognition system implemented in Python using the face\_recognition, OpenCV, and dlib libraries. It loads known faces from a dataset and identifies them via webcam input. Ideal for security systems, attendance systems, or personal use.

**How the System Works**

1. **Dataset Preparation**:  
   Add clear images of each person into known\_faces/<name>/.
2. **Encoding Faces**:  
   During runtime, the script loads all images and encodes facial features.
3. **Real-time Recognition**:  
   Captures frames from webcam → Detects faces → Compares with known encodings → Displays results.

### Step by step instructions :

**Step 1:**

Create an new repository and redirect to that repository

**Step 2:**

Install the following libraries and packages

pip install face\_recognition opencv-python numpy pillow

sudo apt-get install cmake

pip install dlib==19.22.99

**Step 3:**

**Connect an USB cam or CSI camera in the jetson nano and run the following program**

import cv2

import os

import tkinter as tk

from tkinter import ttk

from tkinter import messagebox

from PIL import Image, ImageTk

class FaceCaptureApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Face Capture GUI")

self.root.geometry("700x300") # Smaller window

# Variables

self.person\_name = tk.StringVar()

self.mode = tk.StringVar(value="auto")

self.save\_path = ""

self.count = 0

self.frame\_count = 0

self.capture\_running = False

self.frame = None

# GUI Layout

self.setup\_gui()

# OpenCV

self.cap = None

def setup\_gui(self):

# Input for name

tk.Label(self.root, text="Person Name:").grid(row=0, column=0, padx=5, pady=5, sticky="e")

tk.Entry(self.root, textvariable=self.person\_name).grid(row=0, column=1, padx=5, pady=5)

# Mode dropdown

tk.Label(self.root, text="Mode:").grid(row=1, column=0, padx=5, pady=5, sticky="e")

ttk.Combobox(self.root, textvariable=self.mode, values=["auto", "manual"], state="readonly").grid(row=1, column=1, padx=5, pady=5)

# Buttons

tk.Button(self.root, text="Start Capture", command=self.start\_capture).grid(row=2, column=0, padx=5, pady=5)

tk.Button(self.root, text="Stop", command=self.stop\_capture).grid(row=2, column=1, padx=5, pady=5)

# Capture Button (manual mode only)

self.capture\_btn = tk.Button(self.root, text="Capture Image", command=self.capture\_image)

self.capture\_btn.grid(row=3, column=0, columnspan=2, padx=5, pady=5)

self.capture\_btn.config(state="disabled")

# Status label

self.status\_label = tk.Label(self.root, text="Status: Idle", fg="blue")

self.status\_label.grid(row=4, column=0, columnspan=2)

# Image count

self.count\_label = tk.Label(self.root, text="Images Captured: 0")

self.count\_label.grid(row=5, column=0, columnspan=2)

# Video panel

self.video\_panel = tk.Label(self.root)

self.video\_panel.grid(row=0, column=2, rowspan=6, padx=10)

def start\_capture(self):

name = self.person\_name.get().strip()

if not name:

messagebox.showwarning("Input Error", "Please enter a name.")

return

self.save\_path = os.path.join("face\_dataset", name)

os.makedirs(self.save\_path, exist\_ok=True)

self.cap = cv2.VideoCapture(0)

if not self.cap.isOpened():

messagebox.showerror("Error", "Could not open webcam.")

return

# Smaller resolution

self.cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 320)

self.cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 240)

self.capture\_running = True

self.count = 0

self.frame\_count = 0

self.status\_label.config(text="Status: Capturing...", fg="green")

if self.mode.get() == "manual":

self.capture\_btn.config(state="normal")

self.root.after(10, self.video\_loop)

def stop\_capture(self):

self.capture\_running = False

if self.cap:

self.cap.release()

self.status\_label.config(text="Status: Stopped", fg="red")

self.video\_panel.config(image='') # Clear video feed

self.capture\_btn.config(state="disabled")

def capture\_image(self):

if self.frame is not None:

face\_img = cv2.resize(self.frame, (224, 224))

img\_name = os.path.join(self.save\_path, f"{self.count}.jpg")

cv2.imwrite(img\_name, face\_img)

self.count += 1

self.count\_label.config(text=f"Images Captured: {self.count}")

self.status\_label.config(text=f"Saved: {img\_name}", fg="blue")

def video\_loop(self):

if not self.capture\_running:

return

ret, frame = self.cap.read()

if not ret:

return

self.frame = frame.copy()

# Auto capture

if self.mode.get() == "auto" and self.frame\_count % 5 == 0:

self.capture\_image()

self.frame\_count += 1

# Convert image to RGB and resize to match preview size

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = Image.fromarray(rgb\_frame).resize((320, 240))

imgtk = ImageTk.PhotoImage(image=img)

self.video\_panel.imgtk = imgtk

self.video\_panel.configure(image=imgtk)

self.root.after(10, self.video\_loop)

# Run the GUI

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

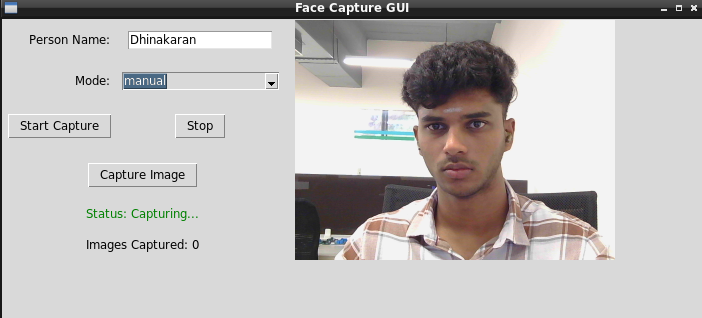
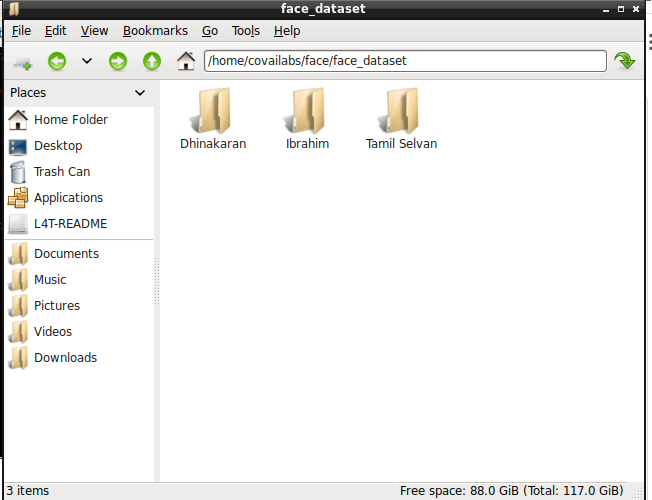
app = FaceCaptureApp(root)

root.mainloop()

**While running the program please ensure that you’re inside the desired project repo then on successful running of the program the GUI will open**



**Enter the person’s name and select the model then start capturing the image the collected images will be stored under separate folders**

**Then we can start training the model with the collected data for training the model use this code**import os

import cv2

import face\_recognition

import pickle

DATASET\_PATH = "face\_dataset"

ENCODINGS\_FILE = "face\_encodings.pkl"

known\_encodings = []

known\_names = []

for person\_name in os.listdir(DATASET\_PATH):

person\_path = os.path.join(DATASET\_PATH, person\_name)

if not os.path.isdir(person\_path):

continue

for image\_name in os.listdir(person\_path):

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

image = cv2.imread(image\_path)

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

encodings = face\_recognition.face\_encodings(rgb\_image)

if len(encodings) > 0:

known\_encodings.append(encodings[0])

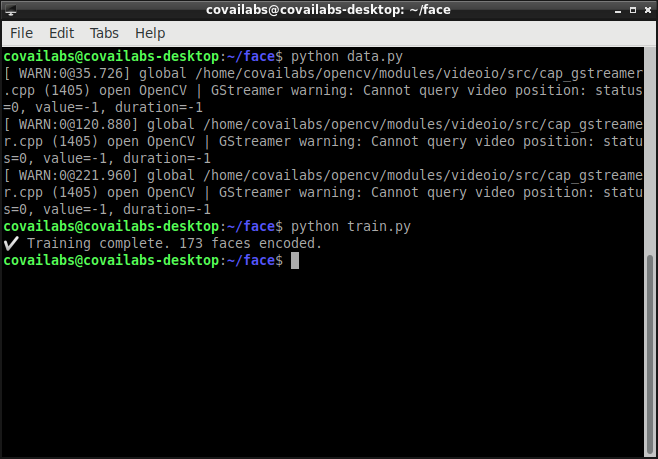
known\_names.append(person\_name)

data = {"encodings": known\_encodings, "names": known\_names}

with open(ENCODINGS\_FILE, "wb") as f:

pickle.dump(data, f)

print(f"✅ Training complete. {len(known\_names)} faces encoded.")



**When the training is completed a new file with .pkl extension will be created in your repository that will contains the encodings of the training data**

Then run the following program for detection  
  
import cv2

import face\_recognition

import pickle

import threading

# Load trained face encodings

with open("face\_encodings.pkl", "rb") as f:

data = pickle.load(f)

# Extract the encodings and names

known\_faces = data["encodings"]

known\_names = data["names"]

# Start video capture from the webcam

video\_capture = cv2.VideoCapture(0)

# Set the resolution to a lower value for faster processing (e.g., 640x480)

video\_capture.set(3, 640) # Width

video\_capture.set(4, 480) # Height

print("📷 Starting webcam... Press 'q' to quit.")

frame\_skip\_rate = 2 # Number of frames to skip (e.g., 2 means process every 2nd frame)

frame\_count = 0

# Function to process each frame in a separate thread

def process\_frame(frame):

global frame\_count

# Resize the frame to speed up face recognition

rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

# Detect faces in the frame

face\_locations = face\_recognition.face\_locations(rgb\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_frame, face\_locations)

# Loop through each face found in the frame

for (top, right, bottom, left), face\_encoding in zip(face\_locations, face\_encodings):

# Compare the detected face with known faces

matches = face\_recognition.compare\_faces(known\_faces, face\_encoding)

name = "Unknown"

# If a match is found, use the name of the matched person

if True in matches:

index = matches.index(True)

name = known\_names[index]

# Draw a rectangle around the face and display the name

cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)

cv2.putText(frame, name, (left, top - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.8, (0, 255, 0), 2)

# Display the resulting frame

cv2.imshow("Face Recognition", frame)

frame\_count += 1

if cv2.waitKey(1) & 0xFF == ord("q"):

return False # Exit loop

return True

while True:

ret, frame = video\_capture.read()

if not ret:

break

# Skip processing frames based on frame\_skip\_rate

if frame\_count % frame\_skip\_rate == 0:

# Process the current frame in a separate thread

if not process\_frame(frame):

break

# Continue to the next frame

frame\_count += 1

# Release the video capture and close the window

video\_capture.release()

cv2.destroyAllWindows()

**It will start the detection**

