wReal-Time Face Recognition Web App

The project will make the jetson device to run as an server that will be running an API for handling the face recognition model and that can be accessed through other devices .

### 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 fastapi uvicorn face\_recognition opencv-python Pillow jinja2 python-multipart

**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()

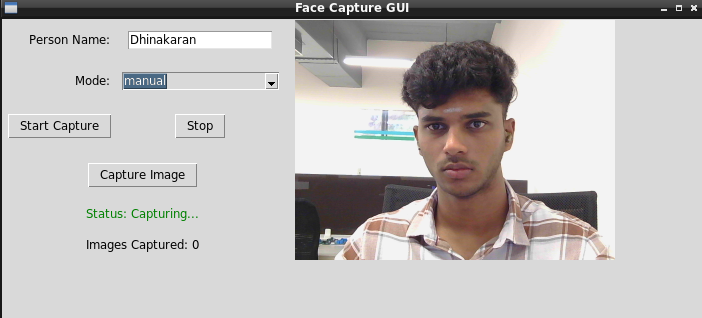
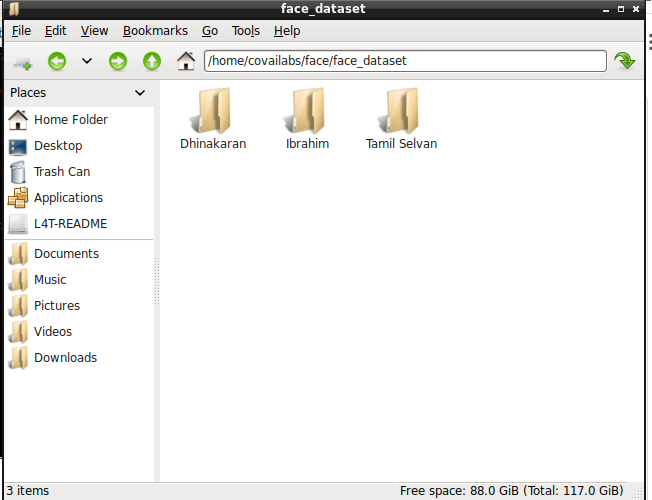
app = FaceCaptureApp(root)

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**

**Step 4:**

**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])

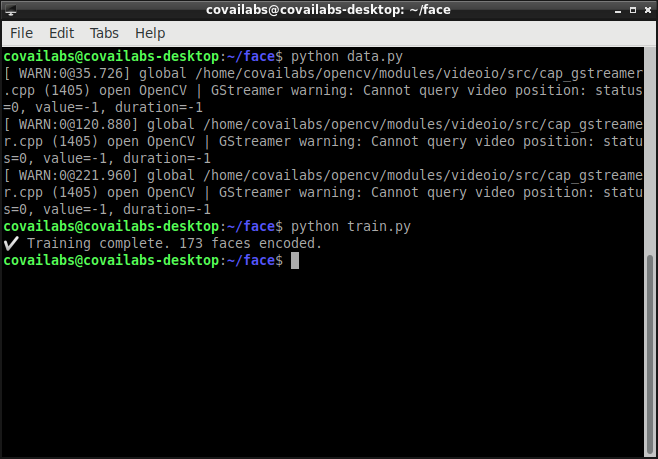
known\_names.append(person\_name)

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**

**Step 5 :**

**Create an new file named main.py and paste the following server code in it**

from fastapi import FastAPI, UploadFile, File, Request, HTTPException

from fastapi.responses import JSONResponse, HTMLResponse

from fastapi.staticfiles import StaticFiles

from fastapi.templating import Jinja2Templates

import face\_recognition

import numpy as np

import pickle

import cv2

# Load your encodings

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

    data = pickle.load(f)  # expects dict: {"encodings": [...], "names": [...]}

app = FastAPI()

app.mount("/static", StaticFiles(directory="static"), name="static")

templates = Jinja2Templates(directory="templates")

@app.get("/", response\_class=HTMLResponse)

async def index(request: Request):

    return templates.TemplateResponse("index.html", {"request": request})

@app.post("/recognize\_face")

async def recognize\_face(image: UploadFile = File(...)):

    contents = await image.read()

    nparr = np.frombuffer(contents, np.uint8)

    img = cv2.imdecode(nparr, cv2.IMREAD\_COLOR)

    if img is None:

        raise HTTPException(status\_code=400, detail="Invalid image")

    rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

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

    encodings = face\_recognition.face\_encodings(rgb, boxes)

    results = []

    for (box, encoding) in zip(boxes, encodings):

        matches = face\_recognition.compare\_faces(data["encodings"], encoding)

        name = "Unknown"

        if True in matches:

            matchedIdxs = [i for (i, b) in enumerate(matches) if b]

            counts = {}

            for i in matchedIdxs:

                matched\_name = data["names"][i]

                counts[matched\_name] = counts.get(matched\_name, 0) + 1

            name = max(counts, key=counts.get)

        top, right, bottom, left = box

        results.append({

            "name": name,

            "box": [left, top, right - left, bottom - top]

        })

    return JSONResponse(content={"results": results})

**Step 6:**

**Then create an new folder inside the project directory and name it as templates**

**Inside that folder create an new file index.html and paste the following code in it**

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>

  <title>Face Recognition</title>

  <style>

    body {

      font-family: 'Segoe UI', sans-serif;

      display: flex;

      flex-direction: column;

      align-items: center;

      padding: 1rem;

      background-color: #f5f7fa;

    }

    #dropZone {

      border: 3px dashed #888;

      border-radius: 10px;

      padding: 2rem;

      width: 90%;

      max-width: 600px;

      text-align: center;

      background: #fff;

      margin-bottom: 1rem;

      transition: background 0.3s;

    }

    #dropZone.dragover {

      background: #e0f0ff;

    }

    canvas {

      border: 2px solid #ccc;

      border-radius: 8px;

      max-width: 100%;

    }

    #results {

      margin-top: 1rem;

      background: #fff;

      padding: 1rem;

      border-radius: 10px;

      width: 100%;

      max-width: 600px;

    }

    #spinner {

      display: none;

    }

    button {

      margin-top: 1rem;

      padding: 0.5rem 1rem;

      font-size: 1rem;

      cursor: pointer;

      border: none;

      border-radius: 8px;

      background-color: #007bff;

      color: white;

    }

  </style>

</head>

<body>

  <h1>Face Recognition</h1>

  <div id="dropZone">Drag and drop an image here or click to upload</div>

  <input type="file" id="fileInput" style="display:none;" accept="image/\*" />

  <button id="webcamBtn">📸 Use Webcam</button>

  <video id="video" autoplay playsinline style="display:none; margin-top:1rem; border-radius:10px;"></video>

  <canvas id="webcamCanvas" style="display:none;"></canvas>

  <button id="snapBtn" style="display:none;">📷 Capture</button>

  <div id="spinner">🔄 Detecting faces...</div>

  <canvas id="canvas" width="400"></canvas>

  <div id="results"></div>

  <script>

    const dropZone = document.getElementById("dropZone");

    const fileInput = document.getElementById("fileInput");

    const canvas = document.getElementById("canvas");

    const ctx = canvas.getContext("2d");

    const resultDiv = document.getElementById("results");

    const spinner = document.getElementById("spinner");

    const webcamBtn = document.getElementById("webcamBtn");

    const video = document.getElementById("video");

    const webcamCanvas = document.getElementById("webcamCanvas");

    const snapBtn = document.getElementById("snapBtn");

    const MAX\_WIDTH = 600;

    dropZone.addEventListener("click", () => fileInput.click());

    dropZone.addEventListener("dragover", e => {

      e.preventDefault();

      dropZone.classList.add("dragover");

    });

    dropZone.addEventListener("dragleave", () => dropZone.classList.remove("dragover"));

    dropZone.addEventListener("drop", e => {

      e.preventDefault();

      dropZone.classList.remove("dragover");

      handleFile(e.dataTransfer.files[0]);

    });

    fileInput.addEventListener("change", e => handleFile(e.target.files[0]));

    function showSpinner(show) {

      spinner.style.display = show ? "block" : "none";

    }

    async function handleFile(file) {

      const img = new Image();

      const reader = new FileReader();

      reader.onload = e => {

        img.src = e.target.result;

      };

      reader.readAsDataURL(file);

      img.onload = async () => {

        const scale = Math.min(MAX\_WIDTH / img.width, 1);

        canvas.width = img.width \* scale;

        canvas.height = img.height \* scale;

        ctx.clearRect(0, 0, canvas.width, canvas.height);

        ctx.drawImage(img, 0, 0, canvas.width, canvas.height);

        const blob = await new Promise(resolve => canvas.toBlob(resolve, "image/jpeg"));

        const formData = new FormData();

        formData.append("image", blob, "upload.jpg");

        showSpinner(true);

        const response = await fetch("/recognize\_face", {

          method: "POST",

          body: formData,

        });

        const data = await response.json();

        showSpinner(false);

        ctx.strokeStyle = "red";

        ctx.lineWidth = 2;

        ctx.font = "16px 'Segoe UI'";

        ctx.fillStyle = "red";

        resultDiv.innerHTML = data.results.length

          ? "<h2>Detected Faces:</h2><ul>"

          : "<p style='color:gray;'>No faces detected.</p>";

        for (const face of data.results) {

          const [x, y, w, h] = face.box;

          ctx.strokeRect(x, y, w, h);

          ctx.fillText(face.name, x + 2, y - 8);

          resultDiv.innerHTML += `<li><strong>${face.name}</strong> at (x:${x}, y:${y})</li>`;

        }

        if (data.results.length) resultDiv.innerHTML += "</ul>";

      };

    }

    webcamBtn.addEventListener("click", async () => {

      try {

        const stream = await navigator.mediaDevices.getUserMedia({ video: true });

        video.srcObject = stream;

        video.style.display = "block";

        snapBtn.style.display = "inline-block";

      } catch (err) {

        alert("Webcam access denied or unavailable.");

      }

    });

    snapBtn.addEventListener("click", async () => {

      const width = video.videoWidth;

      const height = video.videoHeight;

      webcamCanvas.width = width;

      webcamCanvas.height = height;

      const webcamCtx = webcamCanvas.getContext("2d");

      webcamCtx.drawImage(video, 0, 0, width, height);

      const tracks = video.srcObject.getTracks();

      tracks.forEach(track => track.stop());

      video.style.display = "none";

      snapBtn.style.display = "none";

      const scale = Math.min(MAX\_WIDTH / width, 1);

      canvas.width = width \* scale;

      canvas.height = height \* scale;

      ctx.clearRect(0, 0, canvas.width, canvas.height);

      ctx.drawImage(webcamCanvas, 0, 0, canvas.width, canvas.height);

      const blob = await new Promise(resolve => canvas.toBlob(resolve, "image/jpeg"));

      const formData = new FormData();

      formData.append("image", blob, "webcam.jpg");

      showSpinner(true);

      const response = await fetch("/recognize\_face", {

        method: "POST",

        body: formData,

      });

      const data = await response.json();

      showSpinner(false);

      ctx.strokeStyle = "red";

      ctx.lineWidth = 2;

      ctx.font = "16px 'Segoe UI'";

      ctx.fillStyle = "red";

      resultDiv.innerHTML = data.results.length

        ? "<h2>Detected Faces:</h2><ul>"

        : "<p style='color:gray;'>No faces detected.</p>";

      for (const face of data.results) {

        const [x, y, w, h] = face.box;

        ctx.strokeRect(x, y, w, h);

        ctx.fillText(face.name, x + 2, y - 8);

        resultDiv.innerHTML += `<li><strong>${face.name}</strong> at (x:${x}, y:${y})</li>`;

      }

      if (data.results.length) resultDiv.innerHTML += "</ul>";

    });

  </script>

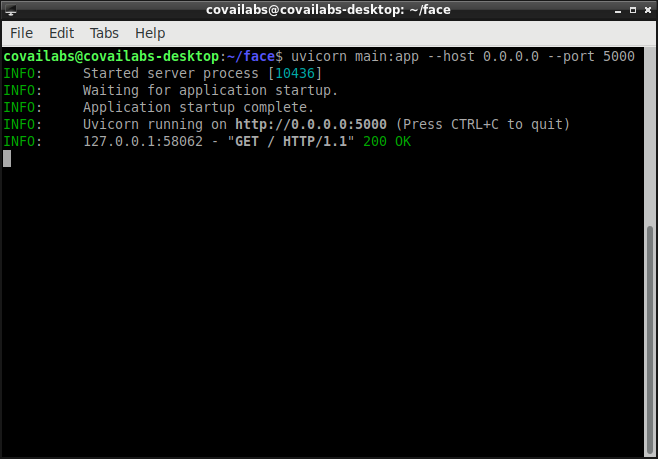
</body>

</html>

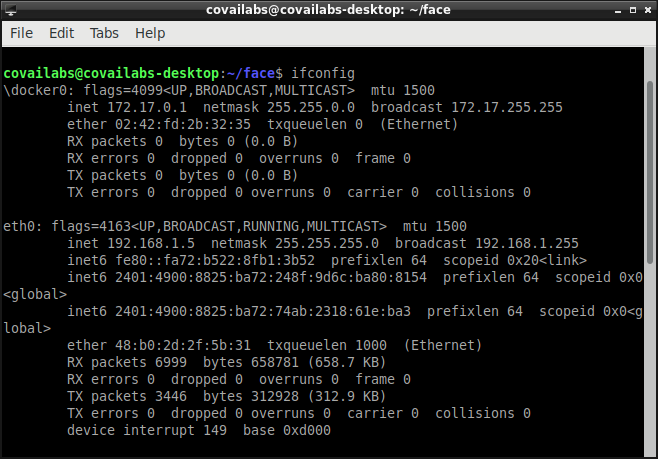
**Step 7:**

**Now run the backend main.py file using this command**

**Uvicorn main:app --host 0.0.0.0 –port 5000**

**It will show the output as**   


now find the IP of the jetson nano

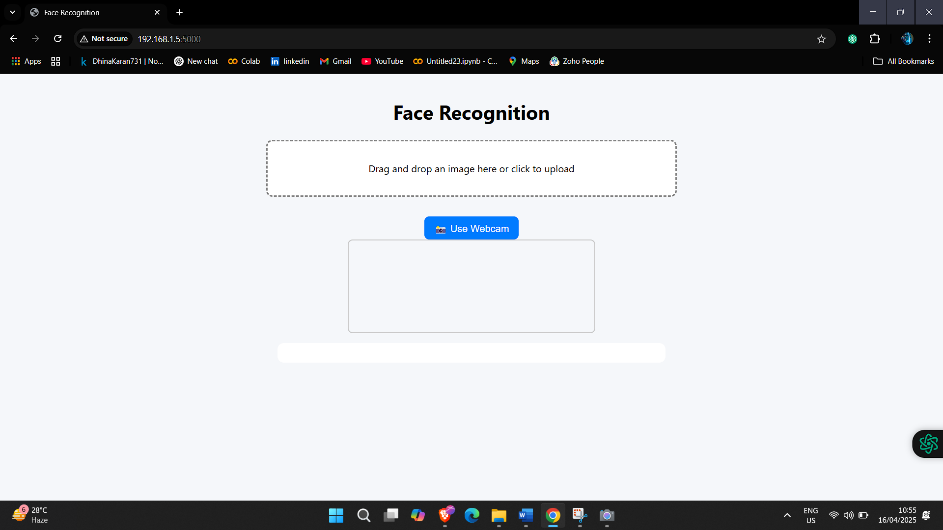


Then add the ip into the below url then search it on the browser ( both devices has to be connected on the same device )

http://<Jeston IP >:5000/



It will open the server on remote device



Select and upload the image for detecting faces in It

