Code:

import face\_recognition

import cv2

from datetime import datetime, timedelta

import numpy as np

import platform

import pickle

known\_face\_encodings = []

known\_face\_metadata = []

def save\_known\_faces():

with open("known\_faces.dat", "wb") as face\_data\_file:

face\_data = [known\_face\_encodings, known\_face\_metadata]

pickle.dump(face\_data, face\_data\_file)

print("Known faces backed up to disk.")

def load\_known\_faces():

global known\_face\_encodings, known\_face\_metadata

try:

with open("known\_faces.dat", "rb") as face\_data\_file:

known\_face\_encodings, known\_face\_metadata = pickle.load(face\_data\_file)

print("Known faces loaded from disk.")

except FileNotFoundError as e:

print("No previous face data found - starting with a blank known face list.")

pass

def running\_on\_jetson\_nano():

# To make the same code work on a laptop or on a Jetson Nano, we'll detect when we are running on the Nano

# so that we can access the camera correctly in that case.

# On a normal Intel laptop, platform.machine() will be "x86\_64" instead of "aarch64"

return platform.machine() == "aarch64"

def get\_jetson\_gstreamer\_source(capture\_width=1280, capture\_height=720, display\_width=1280, display\_height=720, framerate=60, flip\_method=0):

return (

f'nvarguscamerasrc ! video/x-raw(memory:NVMM), ' +

f'width=(int){capture\_width}, height=(int){capture\_height}, ' +

f'format=(string)NV12, framerate=(fraction){framerate}/1 ! ' +

f'nvvidconv flip-method={flip\_method} ! ' +

f'video/x-raw, width=(int){display\_width}, height=(int){display\_height}, format=(string)BGRx ! ' +

'videoconvert ! video/x-raw, format=(string)BGR ! appsink'

)

def register\_new\_face(face\_encoding, face\_image):

"""

Add a new person to our list of known faces

"""

known\_face\_encodings.append(face\_encoding)

known\_face\_metadata.append({

"first\_seen": datetime.now(),

"first\_seen\_this\_interaction": datetime.now(),

"last\_seen": datetime.now(),

"seen\_count": 1,

"seen\_frames": 1,

"face\_image": face\_image,

})

def lookup\_known\_face(face\_encoding):

metadata = None

# If our known face list is empty, just return nothing since we can't possibly have seen this face.

if len(known\_face\_encodings) == 0:

return metadata

face\_distances = face\_recognition.face\_distance(known\_face\_encodings, face\_encoding)

# Get the known face that had the lowest distance (i.e. most similar) from the unknown face.

best\_match\_index = np.argmin(face\_distances)

if face\_distances[best\_match\_index] < 0.65:

metadata = known\_face\_metadata[best\_match\_index]

metadata["last\_seen"] = datetime.now()

metadata["seen\_frames"] += 1

if datetime.now() - metadata["first\_seen\_this\_interaction"] > timedelta(minutes=0.02):

metadata["first\_seen\_this\_interaction"] = datetime.now()

metadata["seen\_count"] += 1

return metadata

def main\_loop():

if running\_on\_jetson\_nano():

video\_capture = cv2.VideoCapture(get\_jetson\_gstreamer\_source(), cv2.CAP\_GSTREAMER)

else:

video\_capture = cv2.VideoCapture(0)

# Track how long since we last saved a copy of our known faces to disk as a backup.

number\_of\_faces\_since\_save = 0

while True:

# Grab a single frame of video

ret, frame = video\_capture.read()

# Resize frame of video to 1/4 size for faster face recognition processing

small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_small\_frame = small\_frame[:, :, ::-1]

# Find all the face locations and face encodings in the current frame of video

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

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

# Loop through each detected face and see if it is one we have seen before

# If so, we'll give it a label that we'll draw on top of the video.

face\_labels = []

for face\_location, face\_encoding in zip(face\_locations, face\_encodings):

# See if this face is in our list of known faces.

metadata = lookup\_known\_face(face\_encoding)

# If we found the face, label the face with some useful information.

if metadata is not None:

time\_at\_door = datetime.now() - metadata['first\_seen\_this\_interaction']

face\_label = f"At door {int(time\_at\_door.total\_seconds())}s"

# If this is a brand new face, add it to our list of known faces

else:

face\_label = "New visitor!"

# Grab the image of the the face from the current frame of video

top, right, bottom, left = face\_location

face\_image = small\_frame[top:bottom, left:right]

face\_image = cv2.resize(face\_image, (150, 150))

# Add the new face to our known face data

register\_new\_face(face\_encoding, face\_image)

face\_labels.append(face\_label)

# Draw a box around each face and label each face

for (top, right, bottom, left), face\_label in zip(face\_locations, face\_labels):

# Scale back up face locations since the frame we detected in was scaled to 1/4 size

top \*= 4

right \*= 4

bottom \*= 4

left \*= 4

# Draw a box around the face

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

# Draw a label with a name below the face

cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)

cv2.putText(frame, face\_label, (left + 6, bottom - 6), cv2.FONT\_HERSHEY\_DUPLEX, 0.8, (255, 255, 255), 1)

# Display recent visitor images

number\_of\_recent\_visitors = 0

for metadata in known\_face\_metadata:

# If we have seen this person in the last minute, draw their image

if datetime.now() - metadata["last\_seen"] < timedelta(seconds=10) and metadata["seen\_frames"] > 5:

# Draw the known face image

x\_position = number\_of\_recent\_visitors \* 150

frame[30:180, x\_position:x\_position + 150] = metadata["face\_image"]

number\_of\_recent\_visitors += 1

# Label the image with how many times they have visited

visits = metadata['seen\_count']

visit\_label = f"{visits} visits"

if visits == 1:

visit\_label = "First visit"

cv2.putText(frame, visit\_label, (x\_position + 10, 170), cv2.FONT\_HERSHEY\_DUPLEX, 0.6, (255, 255, 255), 1)

if number\_of\_recent\_visitors > 0:

cv2.putText(frame, "Visitors at Door", (5, 18), cv2.FONT\_HERSHEY\_DUPLEX, 0.8, (255, 255, 255), 1)

# Display the final frame of video with boxes drawn around each detected fames

cv2.imshow('Video', frame)

# Hit 'q' on the keyboard to quit!

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

save\_known\_faces()

break

# We need to save our known faces back to disk every so often in case something crashes.

if len(face\_locations) > 0 and number\_of\_faces\_since\_save > 100:

save\_known\_faces()

number\_of\_faces\_since\_save = 0

else:

number\_of\_faces\_since\_save += 1

# Release handle to the webcam

video\_capture.release()

cv2.destroyAllWindows()

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

load\_known\_faces()

main\_loop()

**Explanantion of Working and Code:**

Whenever a new person steps in front of the camera, it will register their face and start tracking how long they have been near your door. If the same person leaves and comes back more than 12 seconds later, it will register a new visit and track them again. You can hit ‘q’ on your keyboard at any time to exit.

The app will automatically save information about everyone it sees to a file called *known\_faces.dat*. When you run the program again, it will use that data to remember previous visitors. If you want to clear out the list of known faces, just quit the program and delete that file.

The code starts off by importing the libraries we are going to be using. The most important ones are OpenCV (called cv2 in Python), which we’ll use to read images from the camera, and face\_recognition, which we’ll use to detect and compare faces.

import face\_recognition  
import cv2  
from datetime import datetime, timedelta  
import numpy as np  
import platform  
import pickle

Next, we are going to create some variables to store data about the people who walk in front of our camera. These variables will act as a simple database of known visitors.

known\_face\_encodings = [ ]  
known\_face\_metadata = [ ]

This application is just a demo, so we are storing our known faces in a normal Python list. In a real-world application that deals with more faces, you might want to use a real database instead, but I wanted to keep this demo simple.

Next, we have a function to save and load the known face data. Here’s the save function:

def save\_known\_faces():  
 with open("known\_faces.dat", "wb") as face\_data\_file:  
 face\_data = [known\_face\_encodings, known\_face\_metadata]  
 pickle.dump(face\_data, face\_data\_file)  
 print("Known faces backed up to disk.")

This writes the known faces to disk using Python’s built-in pickle functionality. The data is loaded back the same way, but I didn’t show that here.

I wanted this program to run on a desktop computer or on a Jetson Nano without any changes, so I added a simple function to detect which platform it is currently running on:

def running\_on\_jetson\_nano():  
 return platform.machine() == "aarch64"

This is needed because the way we access the camera is different on each platform. On a laptop, we can just pass in a camera number to OpenCV and it will pull images from the camera. But on the Jetson Nano, we have to use gstreamer to stream images from the camera which requires some custom code.

By being able to detect the current platform, we’ll be able to use the correct method of accessing the camera on each platform. That’s the only customization needed to make this program run on the Jetson Nano instead of a normal computer!

Whenever our program detects a new face, we’ll call a function to add it to our known face database:

def register\_new\_face(face\_encoding, face\_image):  
 known\_face\_encodings.append(face\_encoding) known\_face\_metadata.append({  
 "first\_seen": datetime.now(),  
 "first\_seen\_this\_interaction": datetime.now(),  
 "last\_seen": datetime.now(),  
 "seen\_count": 1,  
 "seen\_frames": 1,  
 "face\_image": face\_image,  
 })

First, we are storing the face encoding that represents the face in a list. Then, we are storing a matching dictionary of data about the face in a second list. We’ll use this to track the time we first saw the person, how long they’ve been hanging around the camera recently, how many times they have visited our house, and a small image of their face.

We also need a helper function to check if an unknown face is already in our face database or not:

def lookup\_known\_face(face\_encoding):  
metadata = None  
  
 if len(known\_face\_encodings) == 0:  
 return metadata  
  
 face\_distances = face\_recognition.face\_distance(  
 known\_face\_encodings,   
 face\_encoding  
 )  
  
 best\_match\_index = np.argmin(face\_distances)  
  
 if face\_distances[best\_match\_index] < 0.65:  
 metadata = known\_face\_metadata[best\_match\_index]  
 metadata["last\_seen"] = datetime.now()  
 metadata["seen\_frames"] += 1  
  
 if datetime.now() - metadata["first\_seen\_this\_interaction"]   
 > timedelta(minutes=5):  
 metadata["first\_seen\_this\_interaction"] = datetime.now()  
 metadata["seen\_count"] += 1  
  
 return metadata

We are doing a few important things here:

Using the face\_recogntion library, we check how similar the unknown face is to all previous visitors. The face\_distance() function gives us a numerical measurement of similarity between the unknown face and all known faces— the smaller the number, the more similar the faces.

If the face is very similar to one of our known visitors, we assume they are a repeat visitor. In that case, we update their “last seen” time and increment the number of times we have seen them in a frame of video.

Finally, if this person has been seen in front of the camera in the last five minutes, we assume they are still here as part of the same visit. Otherwise, we assume that this is a new visit to our house, so we’ll reset the time stamp tracking their most recent visit.

The rest of the program is the main loop — an endless loop where we fetch a frame of video, look for faces in the image, and process each face we see. It is the main heart of the program. Let’s check it out:

def main\_loop():  
 if running\_on\_jetson\_nano():  
 video\_capture =   
 cv2.VideoCapture(  
 get\_jetson\_gstreamer\_source(),   
 cv2.CAP\_GSTREAMER  
 )  
 else:  
 video\_capture = cv2.VideoCapture(0)

The first step is to get access to the camera using whichever method is appropriate for our computer hardware. But whether we are running on a normal computer or a Jetson Nano, the video\_capture object will let us grab frames of video from our computer’s camera.

So let’s start grabbing frames of video:

while True:  
 # Grab a single frame of video  
 ret, frame = video\_capture.read()  
  
 # Resize frame of video to 1/4 size  
 small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)  
  
 # Convert the image from BGR color  
 rgb\_small\_frame = small\_frame[:, :, ::-1]

Each time we grab a frame of video, we’ll also shrink it to 1/4 size. This will make the face recognition process run faster at the expense of only detecting larger faces in the image. But since we are building a doorbell camera that only recognizes people near the camera, that shouldn’t be a problem.

We also have to deal with the fact that OpenCV pulls images from the camera with each pixel stored as a Blue-Green-Red value instead of the standard order of Red-Green-Blue. Before we can run face recognition on the image, we need to convert the image format.

Now we can detect all the faces in the image and convert each face into a face encoding. That only takes two lines of code:

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

Next, we’ll loop through each detected face and decide if it is someone we have seen in the past or a brand new visitor:

for face\_location, face\_encoding in zip(  
 face\_locations,   
 face\_encodings): metadata = lookup\_known\_face(face\_encoding)  
  
 if metadata is not None:  
 time\_at\_door = datetime.now() -   
 metadata['first\_seen\_this\_interaction']  
 face\_label = f"At door {int(time\_at\_door.total\_seconds())}s"  
  
 else:  
 face\_label = "New visitor!"  
  
 # Grab the image of the the face  
 top, right, bottom, left = face\_location  
 face\_image = small\_frame[top:bottom, left:right]  
 face\_image = cv2.resize(face\_image, (150, 150))  
  
 # Add the new face to our known face data  
 register\_new\_face(face\_encoding, face\_image)

If we have seen the person before, we’ll retrieve the metadata we’ve stored about their previous visits. If not, we’ll add them to our face database and grab the picture of their face from the video image to add to our database.

Now that we have found all the people and figured out their identities, we can loop over the detected faces again just to draw boxes around each face and add a label to each face:

for (top, right, bottom, left), face\_label in   
 zip(face\_locations, face\_labels):  
 # Scale back up face location  
 # since the frame we detected in was 1/4 size  
 top \*= 4  
 right \*= 4  
 bottom \*= 4  
 left \*= 4  
  
 # Draw a box around the face  
 cv2.rectangle(  
 frame, (left, top), (right, bottom), (0, 0, 255), 2  
 )  
  
 # Draw a label with a description below the face  
 cv2.rectangle(  
 frame, (left, bottom - 35), (right, bottom),   
 (0, 0, 255), cv2.FILLED  
 )  
 cv2.putText(  
 frame, face\_label,   
 (left + 6, bottom - 6),   
 cv2.FONT\_HERSHEY\_DUPLEX, 0.8,   
 (255, 255, 255), 1  
 )

I also wanted a running list of recent visitors drawn across the top of the screen with the number of times they have visited your house:

To draw that, we need to loop over all known faces and see which ones have been in front of the camera recently. For each recent visitor, we’ll draw their face image on the screen and draw a visit count:

number\_of\_recent\_visitors = 0for metadata in known\_face\_metadata:  
 # If we have seen this person in the last minute  
 if datetime.now() - metadata["last\_seen"]   
 < timedelta(seconds=10): # Draw the known face image  
 x\_position = number\_of\_recent\_visitors \* 150 frame[30:180, x\_position:x\_position + 150] =  
 metadata["face\_image"] number\_of\_recent\_visitors += 1  
  
 # Label the image with how many times they have visited  
 visits = metadata['seen\_count']  
 visit\_label = f"{visits} visits" if visits == 1:  
 visit\_label = "First visit" cv2.putText(  
 frame, visit\_label,   
 (x\_position + 10, 170),   
 cv2.FONT\_HERSHEY\_DUPLEX, 0.6,   
 (255, 255, 255), 1  
 )

Finally, we can display the current frame of video on the screen with all of our annotations drawn on top of it:

cv2.imshow('Video', frame)

And to make sure we don’t lose data if the program crashes, we’ll save our list of known faces to disk every 100 frames:

if len(face\_locations) > 0 and number\_of\_frames\_since\_save > 100:  
 save\_known\_faces()  
 number\_of\_faces\_since\_save = 0  
else:  
 number\_of\_faces\_since\_save += 1

And that’s it aside from a line or two of clean up code to turn off the camera when the program exits.

The start-up code for the program is at the very bottom of the program:

if \_\_name\_\_ == "\_\_main\_\_":  
 load\_known\_faces()  
 main\_loop()

All we are doing is loading the known faces (if any) and then starting the main loop that reads from the camera forever and displays the results on the screen.