Hiding from the Eyes of the City

Ariana Freitag, Raymond Lee, Dhvanil Shah, Nithilam Subbaian

15 February 2020

1 Abstract

As facial recognition systems proliferate across the globe, so will the adversarial attacks against these systems. Adversarial attacks are designed to fool a recognition system into either not recognizing someone, or tricking it into thinking it is an image of something else, a contemporary form of identity camouflage. We imagine a world in where a recognition system will quickly evolve to outperform an adversarial attack, which in turn will evolve to outsmart the other. To demonstrate the effect of camouflaging from facial recognition, we develop an interactive exhibit with live cameras that implements a facial recognition affected by disguises that allow visitors to "camouflage".

2 Introduction

In collaboration with Prof. Ben Aranda and Cooper Union Architecture Seminar, "Hiding from the Eyes of the City" is an installation that implements facial recognition technology in real-time where people can explore degrees of recognition and anonymity using inconspicuous objects in the space. A person is registered into the model by having their picture taken. They can play with different objects to "disguise" their face from the facial recognition algorithm.

To break down the terminology associated with facial recognition, face Recognition includes both face identification and face verification/authentication. Face identification is identifying a person based on the image of a face, it is a 1:N matching and face verification/authentication is validating a claimed identity based on the image of a face, it is a 1:1 matching.

We are implementing Facial Recognition by identifying a face from an image/video frame and comparing the facial features from face with faces within a database.

Currently, facial recognition is used to unlock phones, create smarter advertisements, identify faces for tagging on social media, diagnose diseases, track attendance, payment verification, etc. Facial recognition is also used in many public spaces as well to track the movement and behavior of people. With recent events in media regarding the implementation of facial recognition by companies and governments it is clear that there is a lot more to be learned about the facial recognition and counter-attacks.

3 Relevant Work

Dlib and Facenet are the two algorithms that we used in the implementation of facial recognition for the Exhibition.

In the initial face registration, the Dlib model is used for the exhibit visiter to interact one on one with the facial recognition model. For the crowd live feed, the facenet model is used, as there has now been more time for the model to train with the new face that was just added during registration.

4 System Architecture

The system is split into two parts. In the first part the user will register their face and name at a booth, and then interact with a camera that shows their face identified as the name they entered. In the second part, the user will enter an open space with video cameras where they will be able to see a live crowd feed of all persons within the space. In the live feed, their face will be detected unless they are wearing the given disguise.

Here are the two system architecture implementations we designed in the Fall 2019 Semester. At the end we decided to use design two, which we implemented in the Houghton and Peter Cooper Brue Beer Exhibition.

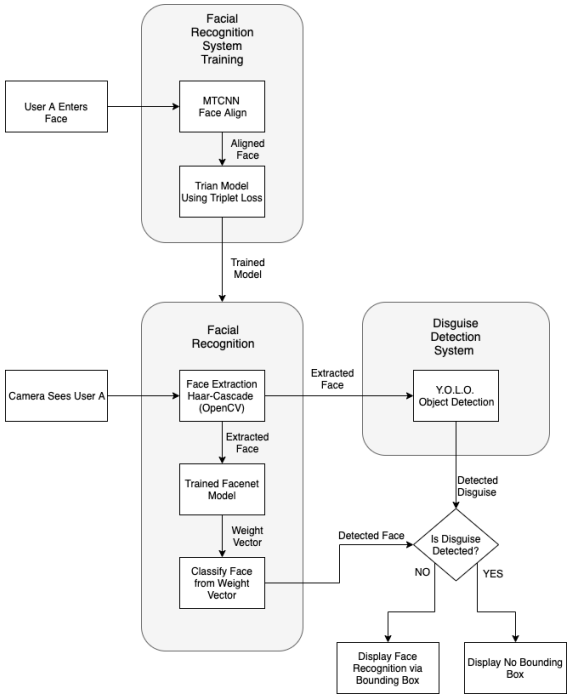


Figure 1: Design One

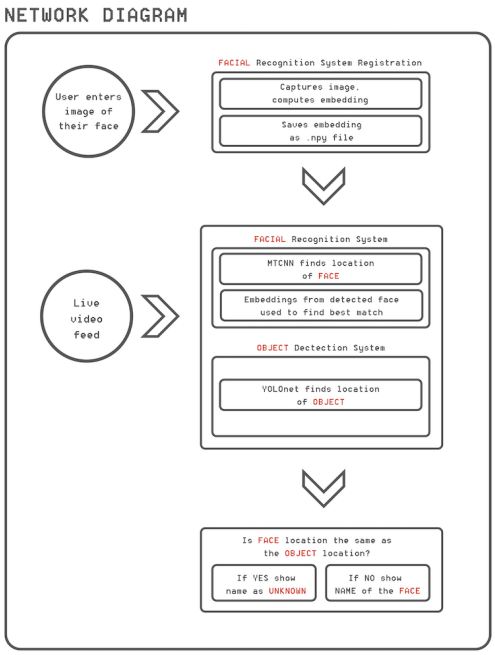


Figure 2: Design Two

For the Spring 2020 Semester, we will be implementing a different architecture that implements a Counter Attack.

5 Exhibition Flow

1. User registers face to be used in Facial Recognition in Booth, and takes note of the identification number assigned to their face.
    - Once the user gives consent, an image of their face is taken, and it is run through dlib to locate the face in the image, and calculate the embedding associated with the particular face.

Figure 3: Face Registration Booth

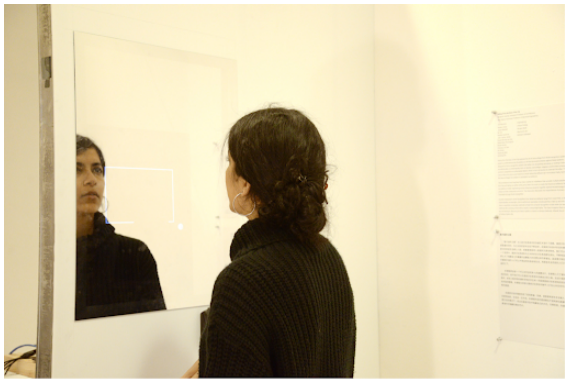


Figure 4: Visitor registering face into system

2. User selects through an assortment of disguises to experiment with in attempting to camouflage from the facial recognition.
    - In the meantime, the embedding along with the corresponding identification number from the prior step is then sent through a LAN connection to the Nvidia Jetsons running the facial recognition algorithm.
    - Refer to Appendix figure 17 and 16 for more examples of disguises used in the system

Figure 5: Applying face paint

Figure 6: Visitors with disguises of face paint and sticker

3. User observes the detection of their faces in a live video feed, which shows a rectangular identification box around their face with their given identification number, and then attempts to remove the identification box by using their chosen disguises.

- The facial recognition algorithm compares the faces detected in the live video feed to the stored embeddings, and displays the identification number of the embedding that best matches the face in the live video feed. Simultaneously, in order to simulate counter attacks, the object detection algorithm work to identify disguises, and remove the detection box if the disguise is near the identified face.

Figure 7: Visitor sees their face identified with given registration number

Figure 8: Display of Setup in Houghton Gallery

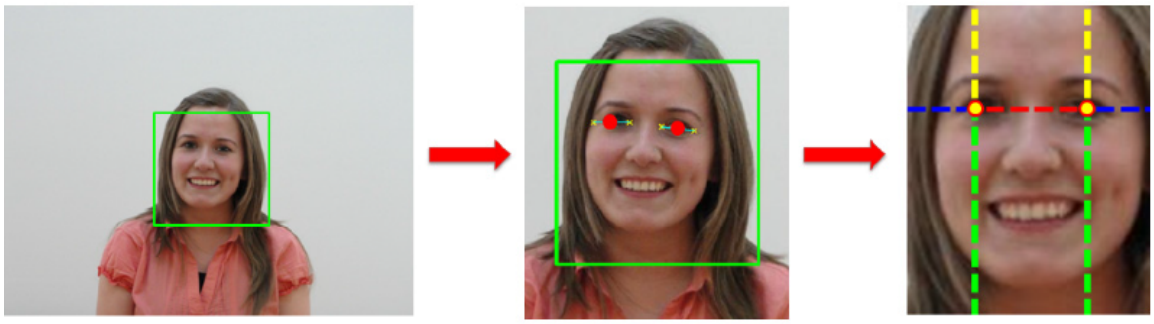
Figure 9: Vistors being detected at Houghton Gallery Exhibition

6 Exhibition Technology

6.1 Part One: Registration

When a user walks to the booth, they will be prompted to position their face so the camera can take a picture of their face. Then the user will be prompted to select from a list of names (to prevent the user from entering profanity). This name will then be designated as the name associated to the photo just taken, and this "feature" and "label" will be saved in a folder, and sent to the dlib model and Multi-task Cascaded Convolutional Networks algorithm (MTCNN) algorithm.

As the dlib model trains fast for the addition of one image, the user will be able to view the detection of their face in the monitor that they just registered their face on. At the same time MTCNN algorithm works to align the faces to be fed into the facial recognition model for training. The diagram below demonstrates face alignment.



(a) (b) (c)

We will use the aligned face to train the facenet facial recognition model. Once the model is trained we will export it to the device running the Facial Recognition.

6.2 Part Two: Inferencing

The device running the facial recognition will use a camera to capture a live video of the exhibition space. We will use OpenCV to extract faces from the video feed. These captured faces will be fed into (1) our facial recognition model and (2) our disguise detection system. The facial detection model will assign weights to a face and a classifier will identify a person based on the assigned weights. At the same time, our disguise detection system will try to identify if a person has a disguise on using the You Only Look Once (Y.O.L.O.) object detection model. Then, our algorithm will take the two outputs (from 1 and 2) and only display the person's identity if a disguise is not present.

7 Software Configuration

7.1 FaceNet

FaceNet: A Unified Embedding for Face Recognition and Clustering published by Florian Schroff, Dmitry Kalenichenko, James Philbin introduces a method for facial verification and recognition that scales. FaceNet maps images of a faces to a 128 dimensional Euclidean space, in which distances correspond to how similar

faces are. A deep convolutional neural network is used to optimize the FaceNet embeddings presented as feature vectors. In training, triplets of non-matching face patches from a new online triplet mining method is used, allowing FaceNet to achieve state of the art results. The publication introduces the new concepts of harmonic embeddings and harmonic triplet loss that allows for the comparison of different face embeddings from different compatible networks.

Figure 10: Overall Architecture of FaceNet

As shown in figure 10, The network consists of a batch input layer to a deep CNN and then L2 normalization which creates an embedding, which is then fed into triplet loss during training. Triplet loss is used to learn the similarity of two images from their embeddings. Triplet Loss is visualized in the figure 11 , in which positive is an image that contains the same face as the the image chosen as anchor, and negative is an image that contains a different face. Triplet loss minimises the distance between the anchor and positive, and maximize the difference between the anchor and negative, as visualized by the length of the arrows.

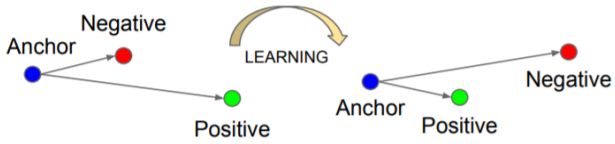


Figure 11: Visualization of Triple Loss

To find the distance between two points in the Euclidean space, the following formula is used as shown in figure 12.

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Figure 12: Distance between two points in Euclidean Space

To find the triplet loss with inputs of an anchor, positive and negative, the following formula is used, as shown in figure 13. Alpha represents a hyperparameter that controls the desired difference between the difference between d(anchor, positive) and d(anchor, negative). Ideally, the difference is significant in order to differentiate between an image with same identity and image with different identity to the anchor. The FaceNet algorithm attempts to minimize this loss by making sure it is always less than 0, meaning that the distance between the positive and the anchor is less than the distance between the negative and anchor. For

example, if you had a training set of 30k images that contained pictures of 1k different people, then from the 30k pictures, triplets containing an anchor, positive, and negative are selected, and gradient descent is used to train with a cost function based on this triplet loss. Note that this requires multiple pictures of the same person while training.

$$Loss = \sum_{i=1}^{N} \left[\|f_i^a - f_i^p\|_2^2 - \|f_i^a - f_i^n\|_2^2 + \alpha \right]_+$$

Figure 13: Equation for Triple Loss

If the anchor, positive, and negative are chosen randomly, then d(Anchor, positive) + alpha will often be less than d(anchor, negative) as for randomly chosen pictures the difference between images of the same person will easily be less than the difference between two images of different people. Therefore the neural network will not be learning much from the data. To improve learning, triplets should be chose so that d(Anchor, positive) and d(Anchor, negative) are close, so that the network is forced to pushed d(Anchor, positive) apart from d(Anchor, negative) through gradient descent.

Standard deep learning algorithms require training on large datasets. If a new image is to be added the model, the model needs to be retrained on all the images. However, as is the case with facial recognition algorithms, sometimes it is not feasible due to time constraints, to retrain the entire model for the addition of one image. In these cases, one shot learning allows for the addition of a new image without complete retraining. The model is initially trained on a smaller dataset, and new data can be added without having to retrain the model on all the data. Analogous to one shot learning, humans are capable of recognizing roses, having been provided one image of a rose.

In order to make a model capable of one shot learning, a similarity function is used in which the image that needs to be classified is input along with each face in the database of the model. The pair of images that produces the lowest difference, is likely the correct label for the image. In order to add a new image to the model, the new image is simply added to the database, and there is no need to retrain the model.

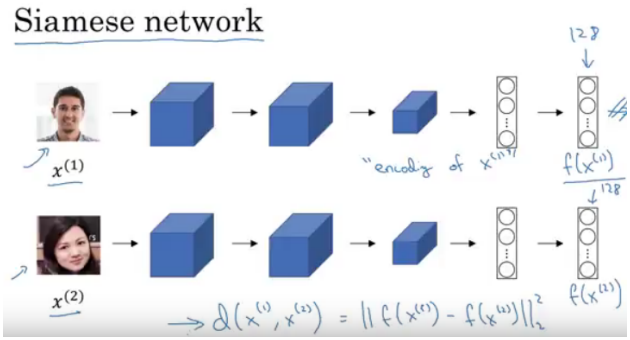


Figure 14: Illustration of siamese network producing encodings

One shot learning is implemented using a siamese network, in which there are two identical fully connected CNNs that each take in a different image as shown in figure 15. The output of a typical CNN is uses a softmax to achieve classification; however, for one hot encoding, the output of the CNN is a 128 dimensional encoding of the input image. Each CNN outputs an encoding that represents the inputted image. The network is then optimized so that the loss of any two encodings are small if the two input images represent the same face, but large if the two input images represent different faces.

In the implementation of the FaceNet model, MTCNN is used to detect faces and align them, and FaceNet is used to obtain the embeddings for the faces. MTCNN is further described by the paper https://kpzhang93.github.io/MTCNN_face_detection_alignment/paper/spl.pdf.

8 Hardware Configuration

The team has decided that the hardware of choice for this project is the NVIDIA TX2. The TX2 is a device built specifically for running machine learning models. First, we plan on having a single device connect to all of the booths where users will register their faces. Second, we wish to have one device per camera for the cameras that surveil the exhibition space.

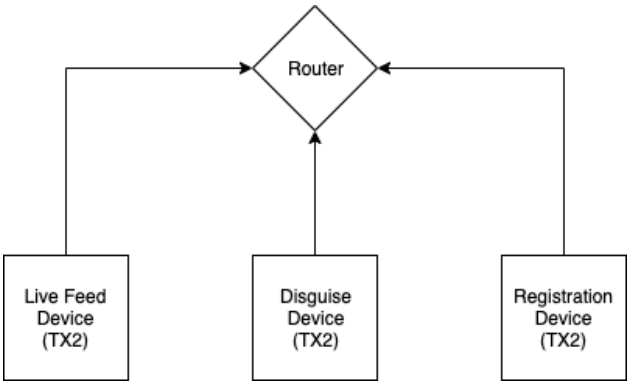


Figure 15: Illustration of siamese network producing encodings

For the Houghton Gallery exhibition and Peter Cooper Brue Beet exhibiton we used SFTP to transfer facial embeddings from UI to running model with a CISCO router and achieved very low latency in transferring models.

8.1 Training the Models Using a Single Device

Training the models using a single device will be beneficial to us because we will not have to worry about multiple individually trained models with different faces on each model. Using a single device for training allows us to have all faces trained on one model.

9 Discussion

In light of all the issues in communication and coordination faced in dealing with the curators from the biennial, we learned a lot about facial recognition technology and the communication required to plan an event at this scale. Although we were not able to travel to China as planned, we were able to develop a working exhibit that simulated an adversarial attack which we displayed at both the Houghton Gallery and Peter Cooper Brue Beer Exhibition. For next semester, we are excited to learn more about adversarial attacks, and hopefully construct an exhibit featuring an adversarial attack.

10 Acknowledgments

Thank you to Prof. Keene and Prof. Aranda for the help and coordination with the exhibition and thank you to Prof. Sable for the support in EE senior projects.

Thank you to all the work and collaboration with the architects from Cooper Union in completion of this project: Risako Arcari, Jesse Bassett, Bo Cai, Natalie Dechime, Jan Carlos Javier, Kayla Mones de Oca, Stella Blue Porzungolo, Doosung Shin, Daniel Smith, Cheung Lun Jeremy Son, Ain Son, Qicheng Wu.

References

[1] Redmon, Joseph. "Joseph Chet Redmon." Survival Strategies for the Robot Rebellion, pjreddie.com/.

[2] Kumar, Satyam. "Face Recognition Using OpenFace." Medium, Medium, 6 May 2019, medium.com/@krsatyam1996/face-recognition-using-openface-92f02045ca2a.

[3] Geitgey, Adam. "Ageitgey/face_recognition." GitHub, 3 Dec. 2019, github.com/ageitgey/face_recognition.

[4] "Face Recognition." Electronic Frontier Foundation, 5 Sept. 2019, www.eff.org/pages/face-recognition.

[5] "An Update About Face Recognition on Facebook." About Facebook, 7 Nov. 2019, about.fb.com/news/2019/09/update-face-recognition/.

[6] OpenFace, cmusatyalab.github.io/openface/.

[7] Jose, Edwin, et al. "Face Recognition Based Surveillance System Using FaceNet and MTCNN on Jetson TX2." Research Gate, 2019, www.researchgate.net/publication/333660229_Face_Recognition_based_Surveillance_System_Using_FaceNet_and_MTCNN_on_Jetson_TX2.

11 Appendix: Code Highlights

11.1 Sticker for Shenzhen Biennial

Figure 16: Sticker for ShenZhen Bienniale, Camouflage Pattern with Vibrant colors to mimic the appearance of an adversarial attack

11.2 Sticker for Shenzhen Biennial

Figure 17: Collection of images used to train model for Peter Cooper Brue Beer Exhibition

11.3 Planned Layout for Shenzhen Biennial

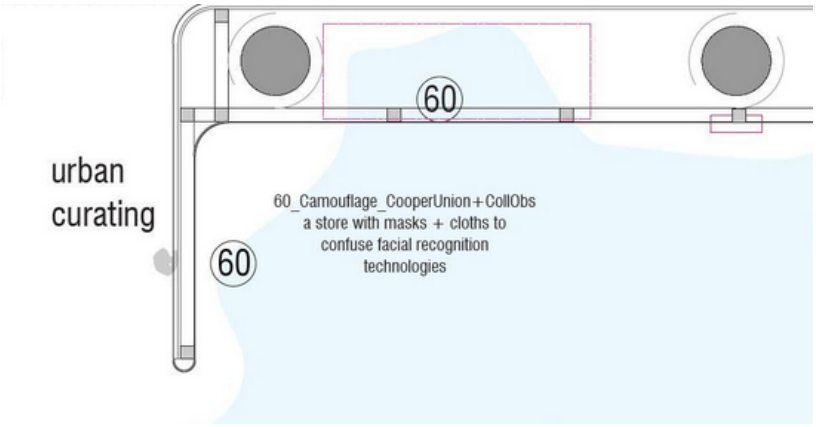


Figure 18: Birds Eye View - Exhibit Space

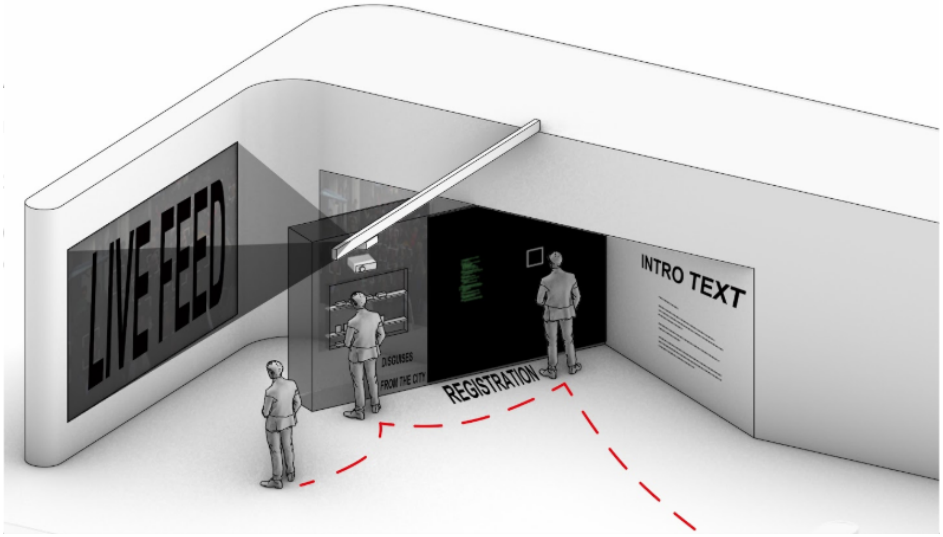


Figure 19: Exhibit Space

11.4 Materials

1. Nvidia Jetson TX2
2. Camera - TBD
3. Twisted Veins HDMI Cable 30 ft
4. Logitech HD Pro Webcam C920, Desktop or Laptop Webcam
5. iMBAPrice USB 3.0 Extender - 15 Feet SuperSpeed USB 3.0 A Male to USB 3.0 A Female Extension Cable
6. Monitor with Mirror for Registration
7. Disguise Objects:
    - Glitter
    - Colored face paint
    - Beer Brue Sticker
    - Disguise for Shenzhen Biennial, seen in appendix item 1
8. Projector for crowd feed demonstration
9. Exhibit Construction Material

11.5 Converting Each Face into an Encoding: encoding_maker.py

```
1
2 import face_recognition
3 import cv2
4 import numpy as np
5 import sys
6 import os.path
7 from os import path
8
9 try:
10     if (path.exists(sys.argv[1])):
11         print(sys.argv[1])
12         person_image = face_recognition.load_image_file(sys.argv[1])
13         person_name = sys.argv[2]
14         person_face_encoding = face_recognition.face_encodings(person_image)[0]
15         print("Person encoding:", person_face_encoding.shape)
16 except Exception as e:
17     print("No system argument provided, please enter an image file name, followed
       by person name")
18     quit()
19     raise
20
21 if(path.exists('encoding.npy') and path.exists('names.npy')):
22     original_arr_encoding = np.load('encoding.npy')
23     original_arr_encoding = original_arr_encoding.tolist()
24     person_face_encoding = person_face_encoding.tolist()
25     original_arr_encoding.append(person_face_encoding)
26
27     # Debug Statement
28     # print(original_arr_encoding)
29
30     original_arr_names = np.load('names.npy')
31     original_arr_names = np.append(original_arr_names, [person_name], axis=0)
32     np.save('encoding', original_arr_encoding)
33     np.save('names', original_arr_names)
34
35
36 else:
37     known_face_encodings = [person_face_encoding]
38     known_face_names = [person_name]
39     np.save('encoding', known_face_encodings)
40     np.save('names', known_face_names)
41
42     # Debug Statement
43     # print("no existing file, encodings", known_face_encodings)
44     # print("no existing file, names", known_face_names)
45     # print("Original arr length if file dont exist:", len(known_face_encodings))
```

11.6 Detecting Faces and Placing a Bounding Box: face_detec_mtcnn.py

```
import face_recognition
import cv2
import numpy as np
from mtcnn.mtcnn import MTCNN
from os import path
import time

detector = MTCNN()


# Get a reference to webcam #0 (the default one)
video_capture = cv2.VideoCapture(1)

face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

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 = frame
    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]

    np.load.__defaults__=(None, True, True, 'ASCII')

    while not path.exists('encoding.npy'):
        time.sleep(1)
    while True:
        try:
            known_face_encodings = np.load('encoding.npy')
            break
        except:
            continue

    while not path.exists('names.npy'):
        time.sleep(1)
    while True:
        try:
            known_face_names = np.load('names.npy')
            break
        except:
            continue
   #print(known_face_encodings.shape)
    #print(known_face_names)

```

```
    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        result_mtcnn = detector.detect_faces(rgb_small_frame)
        #print("MTCNN", result_mtcnn["box"])
        fl = []
        for face in result_mtcnn:
            bb = face['box']
            fl.append((bb[1], bb[0]+bb[2], bb[1]+bb[3], bb[0]))
        #print("MTCNN", fl)
        face_locations = fl
        #face_locations = face_recognition.face_locations(rgb_small_frame)
        #print("Face_Locs", type(face_locations))
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"

            # # If a match was found in known_face_encodings, just use the first one.
            # if True in matches:
            #     first_match_index = matches.index(True)
            #     name = known_face_names[first_match_index]

            # Or instead, use the known face with the smallest distance to the new face
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame

    for (top, right, bottom, left), name in zip(face_locations, face_names):
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        bounding_box = result_mtcnn[0]['box']
        cv2.rectangle(frame,
                      (bounding_box[0]*4, bounding_box[1]*4),
                      (bounding_box[0]*4+bounding_box[2]*4, bounding_box[1]*4 + bounding_box[3]*4),
                      (0,155,255),
```

```
                            2)
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255),
    cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255,
    255), 1)
    #cv2.imshow('SMALL_FRAME', small_frame)
    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

video_capture.release()
cv2.destroyAllWindows()
```