Adaptive Home Surveillance through Computer Vision

Andrew Ingson Chase Carbaugh
University of Maryland Baltimore County
CMSC 491 Final Report
Spring 2021, Group 21

Abstract-
We present an application that serves as a prototype home surveillance system that identifies known individual's faces and adaptively incorporates new individuals for classification. Our application will work by first detecting a face, then classify the face as one of potentially many known individuals or as unknown. If the face is determined to be unknown it will collect frames from the video feed of the unknown individual as long as their face is still being detected. The classifier is then updated to include the new individual and it will be retrained with the new individual's images included. The application notifies the user of the new person and if they are known to the user they can be named.

1. INTRODUCTION

Home security systems have been becoming increasingly more common, most notably with doorbell cameras in the form of the Ring. The base Ring captures video when motion is detected in front of the camera and saves the clip for the user to watch later if it is of interest to them. We imagine a system that works similarly to a doorbell camera with the addition of facial recognition. The system would identify residents of the home and adaptively include newly encountered individuals into the identification system. If you wanted to expand your model to include those outside of your immediate household, you would need to gather images of them and manually incorporate them into the classifier. We seek to automate this process such that all the user needs to do after a new person is encountered by the camera is to give them a name.

While it may just be helpful for the homeowner to be aware of familiar people that are in their home, family members, children, friends, roommates etc. It may also serve a security purpose to incorporate unknown individuals that come to the home in the model as well. Many cases of breaking and entering and theft in which the perpetrators will scope out the home, sometimes this is done at a distance but sometimes a ruse is used to get up close to the home or even enter the home. If their face can be captured during this ruse and incorporated into the model, if the camera captures their face during the crime, the person can be identified as the one that used the ruse and that extra information can be used in finding them.

2. RELATED WORK

Face detection and identification are processes that have already been solved, the tutorial outlined by Rosebock [1] aided us in the process by which we can detect faces. However Rosebock uses OpenCV while we use tensorflow in our project. We utilize an mtcnn to detect faces, the technical process by which this is accomplished is outlined by Zhang et al [4]. In order to perform the classification of the faces, the faces needed to be converted into 128-dimensional embeddings that can represent it, to do this we followed the outline of the Inception Resnet-v2 convolutional neural network outlined by Szegedy et. al [2]. We referenced work done by Alemi [3] in implementing the Inception resnet-v4 model in Tensorflow v1, our project uses a pre-trained version of this model in Tensorflow v2 due to the time and resource constraints that would inhibit us from training a model from scratch. We reference Sonami's [5] work on utilizing pre trained models for face recognition in order to incorporate this into our project.

3. METHODOLOGY

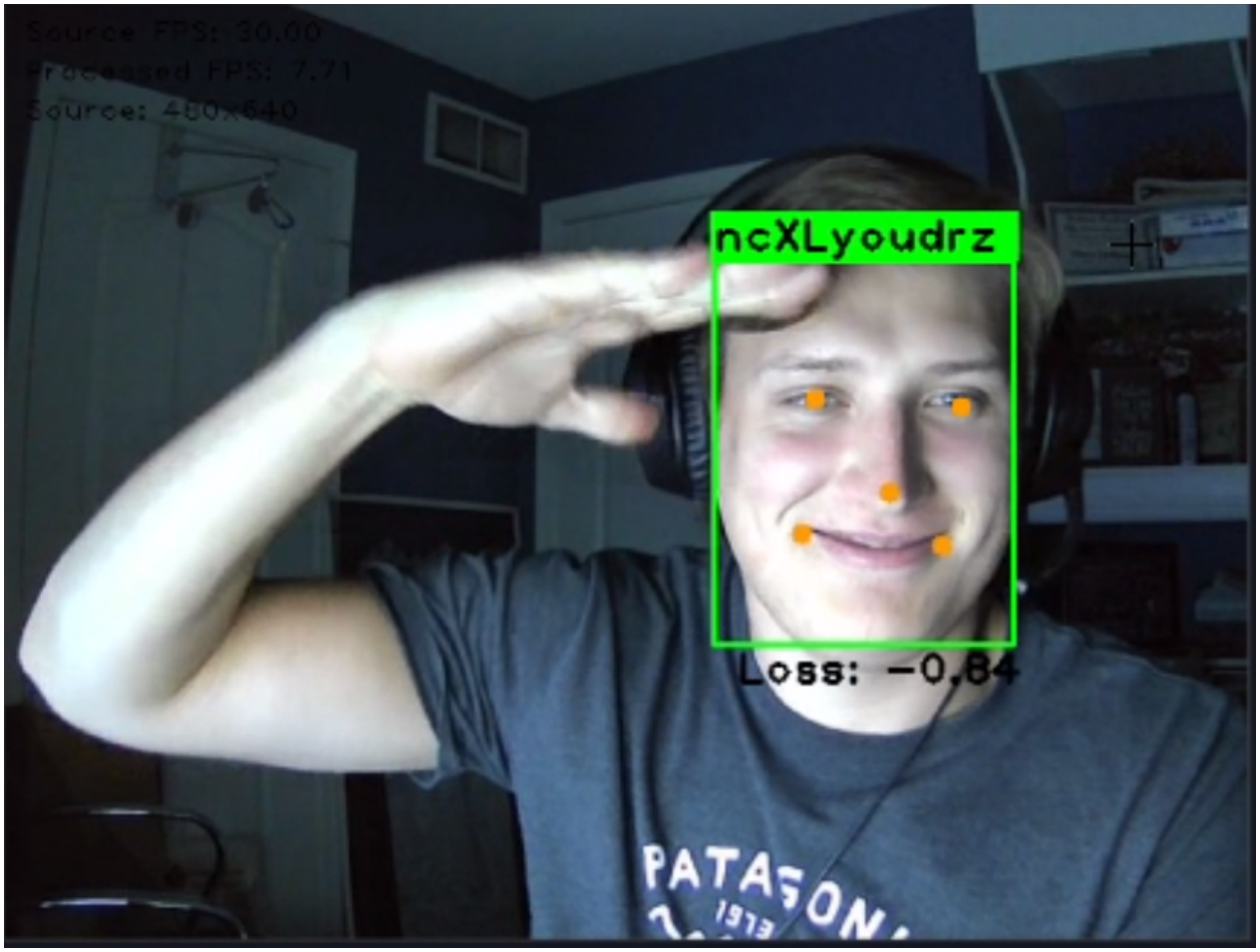

Figure 1. Identification of one the authors using proposed technique.

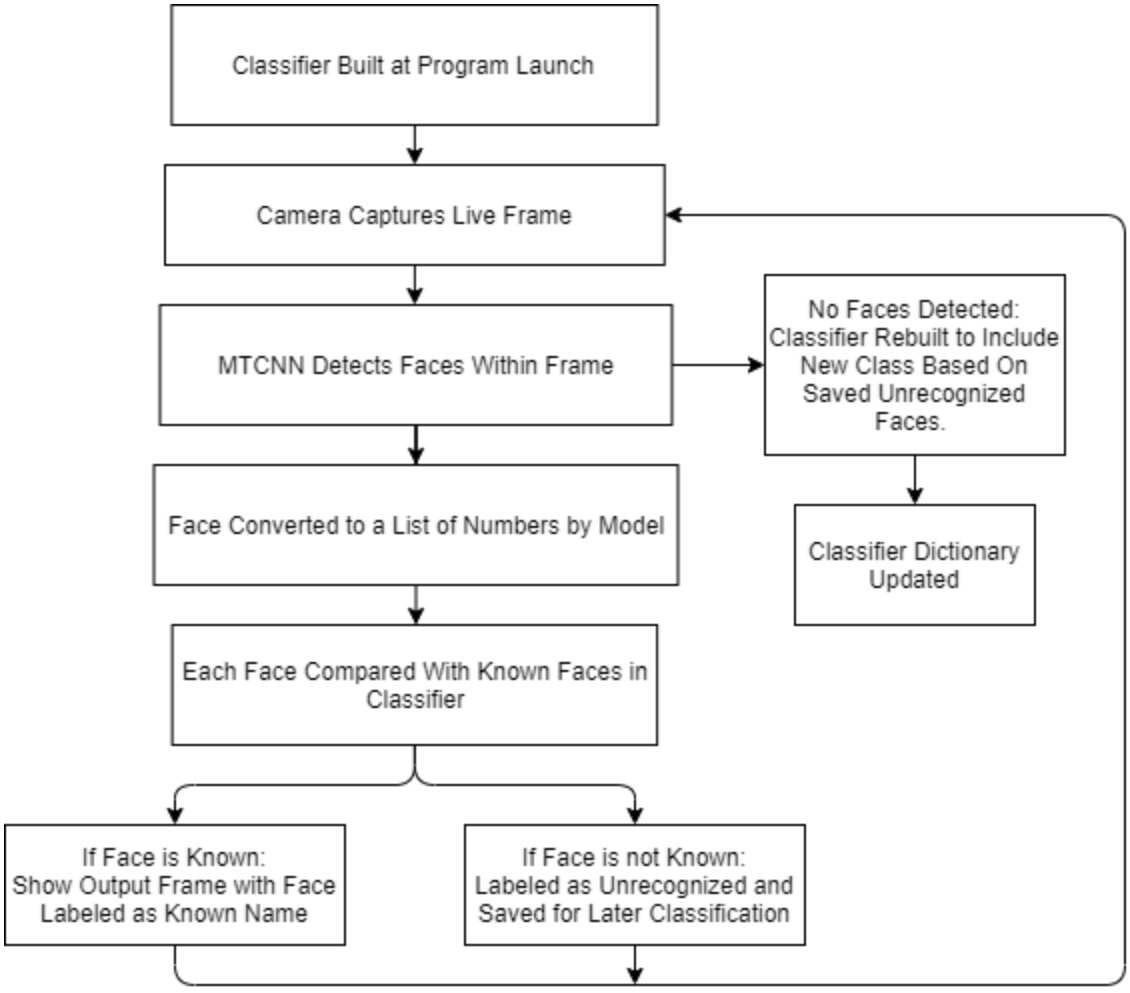

Figure 2. Workflow diagram of technique.

The first stage of implementation was to be able to create a set of possible classes that we could assign faces to. To accomplish this, we use mtcnn to produce an array of points representing all the possible faces in the input images. Based on these data points, we construct a sub image of only the detected face and perform a series of preprocessing steps on that image to convert it to the datatype and size needed to predict with our pre-trained model. After a set of predictions has been made for each image in a class's input directory, the predictions are summed, normalized, and stored in a dictionary that will be used in the primary classification stage. This process repeats for each set of images in the input dataset. The next stage of implementation was to create an application to classify faces from a camera in real time. Similar to the generation step, we use mtcnn to get all the faces present in an image. Each face in the image is passed through to model to give us a tensor representing the face. This tensor is compared to each tensor in our classification dictionary using the Cosine Similarity loss function. If the loss value between a given class and an input face is below a certain threshold, the application will label the face as belonging to that class, as shown in Figure 1. Dots show the location of facial landmarks. If there are more than 1 class that results in a loss below the threshold, then the image will be classified as the class with the best loss value. If no matching

class is found, a new class directory is added to the input directory and the live image is saved to it. At a predetermined interval, the classifier will be automatically regenerated to allow the application to dynamically adapt to new faces over time. As seen in Figure 2, this cycle then repeats for each new frame from the camera. Our dataset will come in two phases. In the classifier generation phase, our dataset will be at least 1 picture of each of the input faces. In the primary classification phase, our dataset will be live images gathered from a camera attached to the host machine. Team members will approach the project incrementally by breaking down the project into smaller, standalone assignments.

4. RESULTS

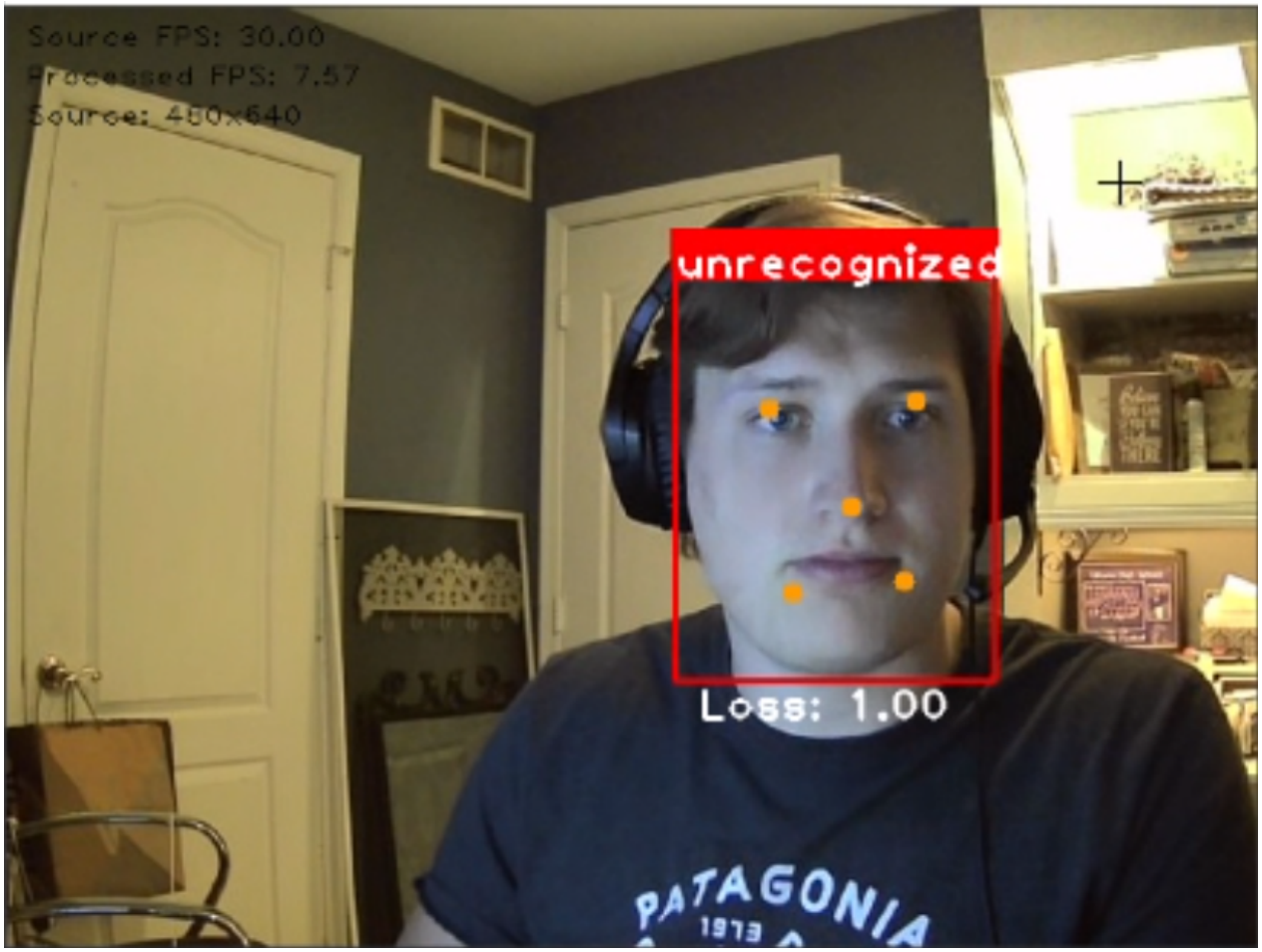

Figure 3.1. Face is detected but does not closely resemble any known class so it is labeled "unrecognized".

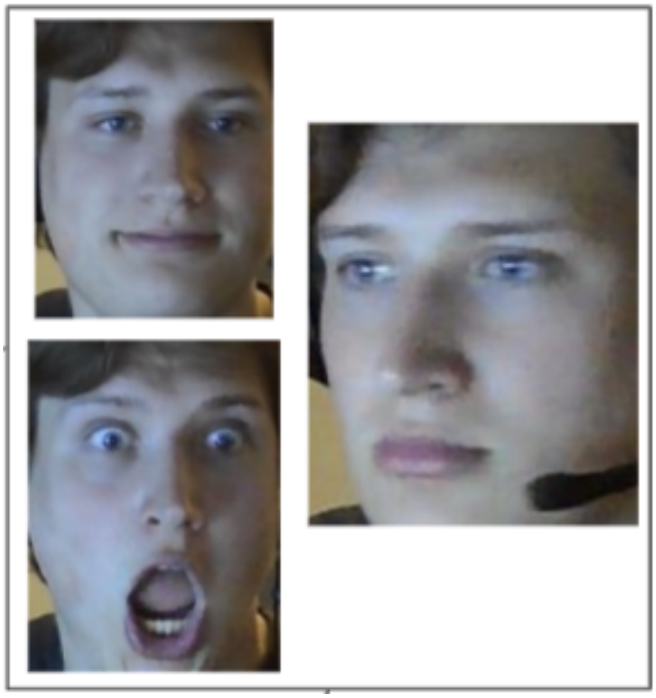

Figure 3.2. Frames where the unrecognized face is detected are cropped to contain only the face and are saved.

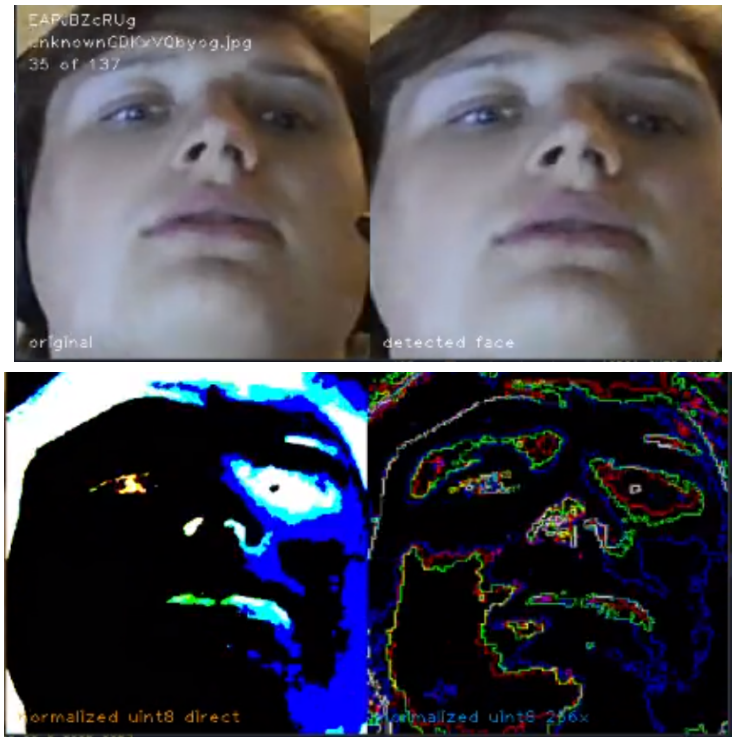

Figure 3.3. Preprocessing the input images of the new face during classifier construction. Input image data types are converted from numpy's unit8 to a normalized float64.

```
ndarray:(128,) [-4.48114909e-02  8.66886452e-02
-1.28114417e-01 -3.44409794e-02, -4.96775992e-02
1.18752919e-01  3.86295654e-02  1.01980194e-01,
1.04480378e-01 -1.70499738e-03 -4.22314228e-03
-1.04925325e-02, -6.67548925e-02  1.40124857e-01
1.01650976e-01  2.11938452e-02,  4.46094647e-02
-6.45223558e-02 -1.73819080e-01 -1.40554315e-04,
-4.37860079e-02 -2.81159002e-02 -9.21628848e-02
-1.34357931e-02,  2.74036638e-02 -5.02553657e-02
1.21222258e-01 -6.39320537e-03, -4.72588055e-02
1.02293834e-01  1.22750849e-02  2.12923735e-02,
-1.68112982e-02  8.20450187e-02  2.06089504e-02
-1.01611443e-01, -9.58155021e-02  1.66714459e-03
1.26340136e-01 -4.22063097e-02, -9.43656191e-02
-3.73117626e-02 -1.70304865e-01 -1.01191856e-01,
3.54005136e-02 -1.30364016e-01  2.55141966e-02
-7.77054429e-02,  2.11757794e-01  3.87608409e-02
-1.12184219e-01 -5.32621518e-02,  1.21137023e-01
-1.13985334e-02  2.62084175e-02 -7.29190335e-02,
2.86295414e-02 -8.37607533e-02  8.01518187e-02
4.68068905e-02, -1.50732875e-01  2.05303...
```

Figure 3.4. Output embeddings from the model representing the unknown face.

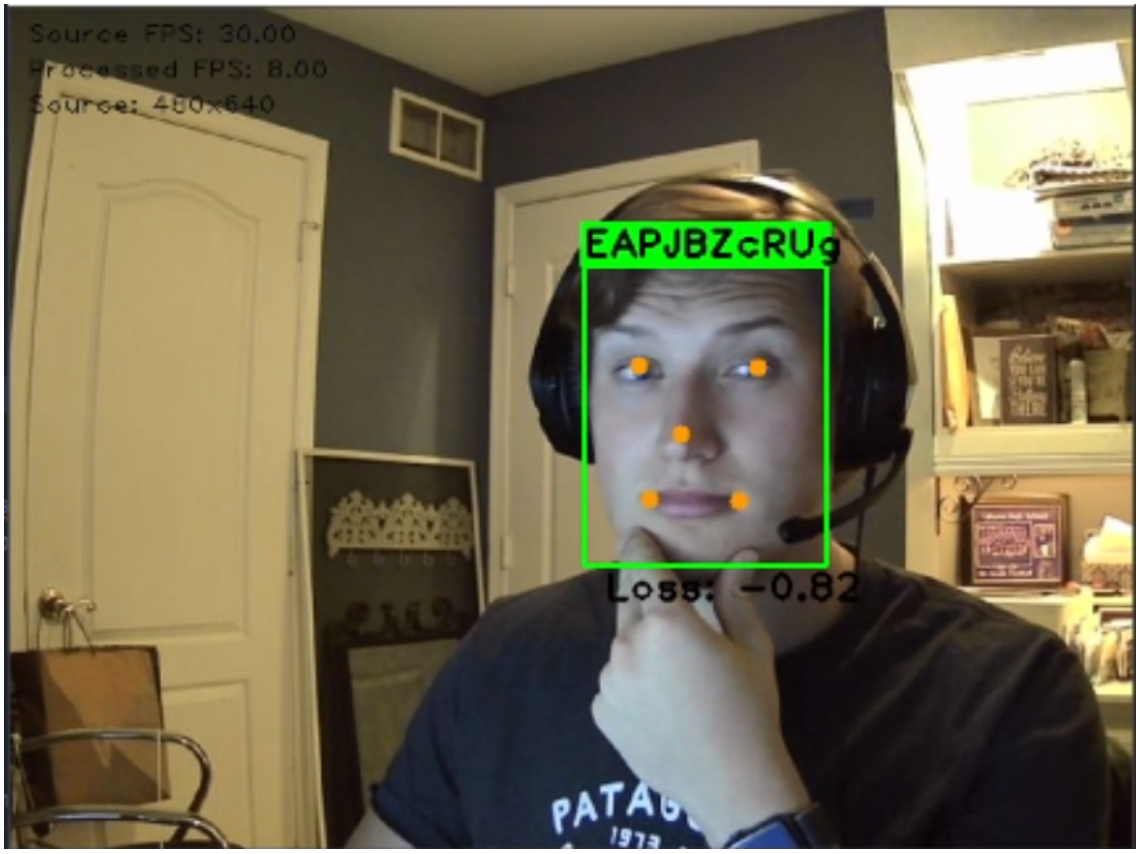

Figure 3.5. Unrecognized face is now detected and correctly classified.

Figure 3.6. Unrecognized face is renamed after user input.

In Figure 3.1 we can see the presence of an unrecognized face in front of the camera is detected, and labeled as such. While the face is present the application saves snapshots of the face to the input directory with a random generated name. Figure 3.2 represents a sample of the images that are captured while an unknown face is present. After a period of inactivity where the application has not detected any faces, a thread is spawned in the background where the classifier is rebuilt with the newly added face, we can see this process in Figure 3.3. The training process results in a 1 dimensional numpy array containing 128 64-bit float embeddings. These embeddings are a numerical representation of the target face, as seen in Figure 3.4. Once the classifier is rebuilt, it notifies the host thread and the host's identification classifier dictionary is updated with the fresh one. In Figure 3.5, the face is now recognized with a loss value of -0.82. Since we use Cosine Similarity as our loss function, loss values range from -1 to 1 where -1 represents absolute similarity to the class and 1 represents absolute dissimilarity from the class. Figure 3.6 demonstrates the result of renaming a once unknown class, as the heretofore unknown face has now been named *Bigfoot*.

5. CONCLUSION

We set out to create a doorbell security system that is capable of detecting faces, identifying known and unknown faces, and automating the process of incorporating unknown faces into the model. We have shown that we can in fact detect faces, identify known faces, and seamlessly incorporate new faces such that the user only needs to provide the new person's name. We have accomplished this by using a Multi-task Cascaded Convolutional Network library to detect faces, an pre-weighted Inception Resnet v-2 convolutional neural network to create face embeddings which we use to identify the faces as someone that is known or as unknown. If the faces are

unknown we have created the system to capture the unknown face and incorporate it into the classifier.

6. DISCUSSION

If this technique were to be delivered as a product, we believe that there is room for future work and improvements. As we have implemented our technique, we have the number of pictures that can be saved to a class set to an arbitrary cap, but ideally once it hits that maximum number of images, new images would overwrite old images to keep the classifier as up to date as possible. Additionally, our implementation is currently limited by the hardware required to run it. In order to achieve performance in excess of 10 frames per second at 480p, a high end processor was required. If future optimization were performed, the application would be able to run at a higher frame resolution allowing it to detect faces at a greater distance from the camera.

REFERENCES

[1] Rosebrock, A. (2018, September 24). OpenCV face recognition. Retrieved April 21, 2021, from https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

[2] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, & Alex A. Alemi (2016). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *ICLR 2016 Workshop*.

[3] Alemi, A. (2016, August 31). Improving inception and image classification in TensorFlow. Retrieved April 21, 2021, from https://ai.googleblog.com/2016/08/improving-inception-and-image.html

[4] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint Face Detection and Alignment Using Multi-task Cascaded Convolutional Networks. *IEEE Signal Processing Letters, 23(10), 1499-1503.*

[5] Somani, S. (2021, January 31). Face recognition using tensorflow pre-trained model & opencv. Retrieved April 21, 2021, from https://swastiksomani.medium.com/face-recognition-using-tensorflow-pre-trained-model-opencv-91184efa4aaf