**CS5131 Project Report**

**Title: Facial Recognition Model using One-shot Learning as an alternative to boarding fingerprint scanners**  
  
**By: Choy Aik Lok & Dominic Cheong**

**Objective**This project intends to provide an alternative to the fingerprint scanners at our boarding school using facial recognition. The project only intends to cover the backend, and there will be no physical prototype. The hardware used for demonstration will be laptop webcam, and is more of a proof of concept rather than a fully working and accurate model.

**Development Processes and Tools Use**

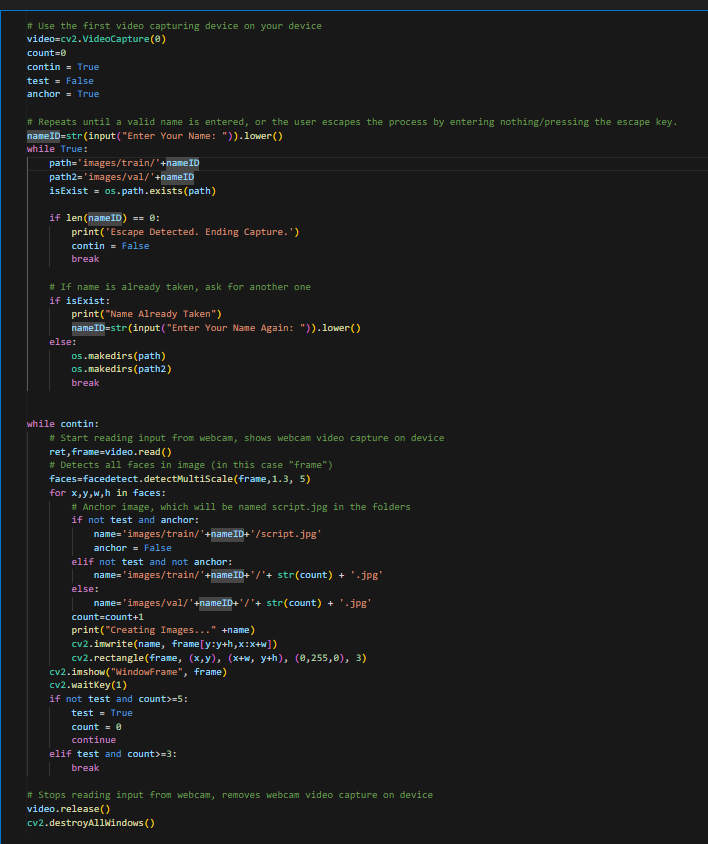
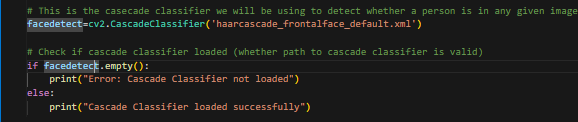
In order to run the notebook, just download the given libraries in the first cell and run the notebook. If there are issues with loading of files, just replace the given file path with the absolute file path to the corresponding files, replacing all backslashes (\) with forward flashes (/).

Tools used: OpenCV, Tensorflow, Tensorflow Keras, Matplotlib, GitHub for collaboration

Development Processes:

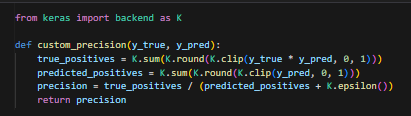
* Data Collection — Using the OpenCV library and the publicly available Cascade Classifier, images captured by the device camera can be processed by an algorithm to detect faces, draw a bounding box around them and store them in an appropriately named file.
* Model Architecture — The Tensorflow library allows for complex models to be built layer by layer to fulfil specific purposes. In this notebook, a simple transfer learning model was built from the existing mobilenetV2 model publicly available on the Tensorflow Keras website. In addition to that, a more complex triplet model was also built using guidelines from an online Kaggle notebook.
* Model Training — Both models are trained using Tensorflow’s .fit function. For the triplet model, a custom triplet loss function had to be built, also referencing the Kaggle notebook.
* Model Comparison — Both models were compared based on the accuracy and precision, which had to be calculated manually for the triplet loss function models. We also used a webcam to test how the models would predict different faces in a real-life scenario.

Data Collection Process

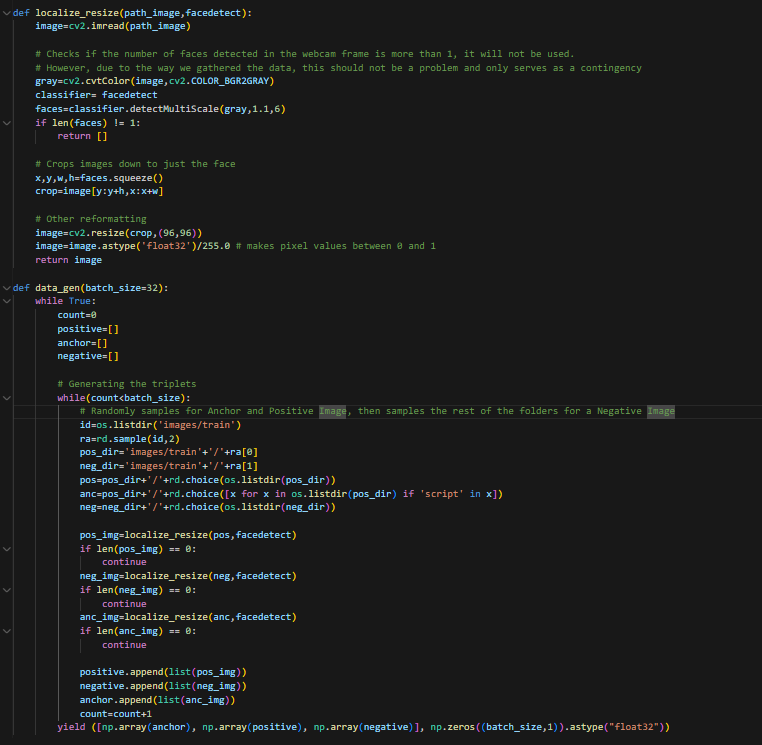


We load up a cascade classifier. We then ask the user to input a valid name until they either exit the cell or they enter in a name not already in use. It then checks for any faces using the cascade classifier in the video capturing device loaded, and if there is, draw a rectangle around the face to show that a face has been detected on the screen, as well as cropping the images down to just the face and adding it into the input image folders.

Custom Precision Function

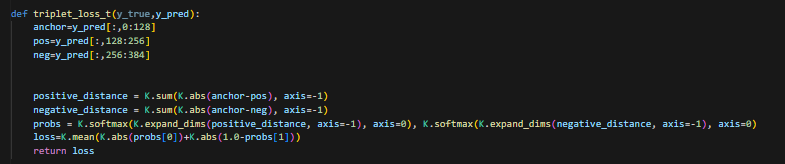


Data Generation Function and Image Resize Functions

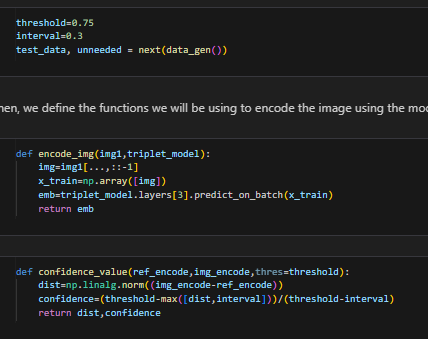


The resize function checks if there is only 1 face in the images, crops the image down to just the face, resizes the image to 96x96 and divides the pixel values by 255 so it is between 0 and 1. Most of the reformatting in the resize function is already done during data collection and only serves as a contingency. The data generator function generates 32 batches of data by default, and each batch returns a list of 3 numpy arrays, those being arrays of the anchor images, positive images and negative images in that order, as well as another numpy array of zeros which is meant to be the labels (x and y value input).

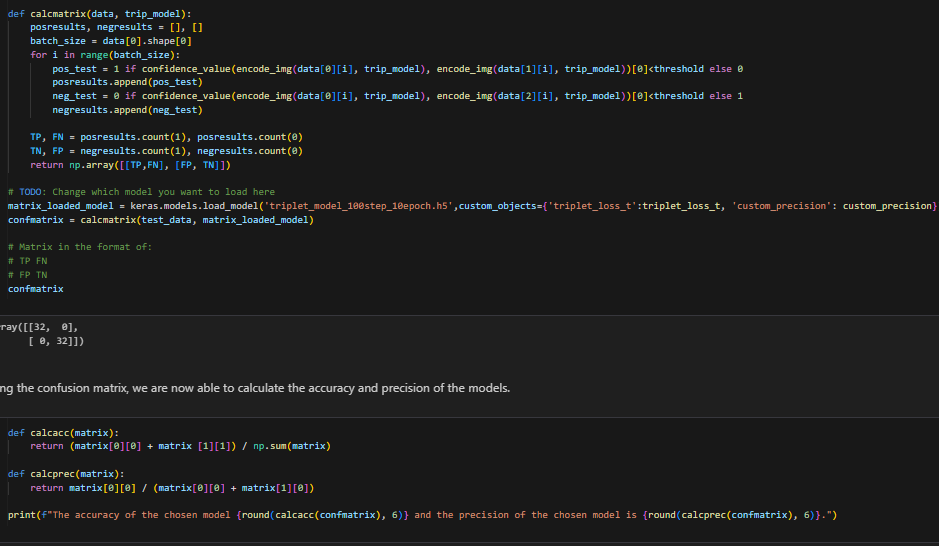
Custom Triplet Loss Function



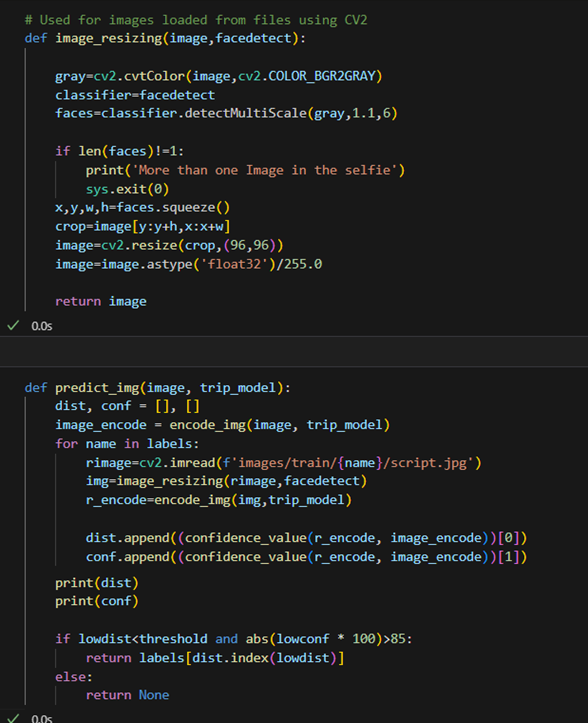
Encoding image and Calculate Distance between 2 Faces + Confidence Value Functions



Calculate Confusion Matrix & Calculate Accuracy and Precision Functions

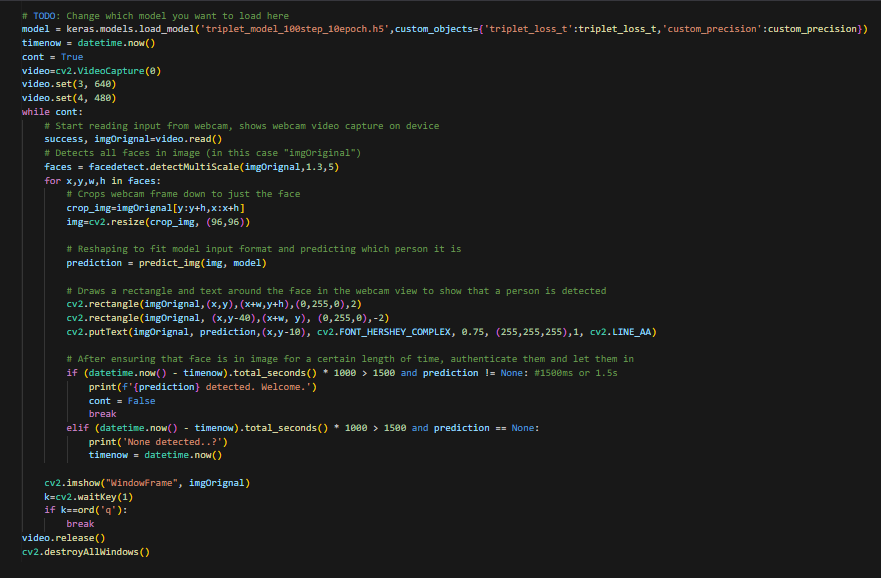


Resizing Image loaded from files and Predict Image Person Functions



The resizing image function is the same as the one above. The predict function encodes the image input, then reads every anchor image for each person in the folder, encodes it and compares it with the input image using the confidence value function. The function then finds the person where the encoded image input and the encoded anchor image has the smallest distance, and if the confidence value is above 85%, returns it, else returns None.

Testing of Models using Webcam



Loads up a trained model and starts reading webcam view. When there is a person in the webcam detected by the cascade classifier, crops the image down to 96x96 and uses it as input for the predict\_img() function. Then, draws a rectangle around person and labels it based on the output of predict image. The loop is broken out of and the person is authenticated (in this case by means of a print statement) after 1500ms of the person’s face being in the camera. Also ends prematurely if the q key is pressed. For the transfer learning models, initializes model to be the transfer learning model, and instead of using the custom predict image function, simply uses model.predict() and returns the index with the highest value, then gets the label using the index.

**Application of concepts and techniques**

This project has made use of the concepts of CNN, transfer learning, metrics to compare models (accuracy and precision), triplet loss function, as well as python libraries such as OpenCV and a lot of searching for references/models.

**Results and Findings**

We have found that one-shot learning is hard to implement. Even with such a complicated model and a reference that was referencing papers published on one-shot learning, the performance of our models is still very poor. We are unsure whether it is an issue with the way our data was collected (since they are all in the same place using the same conditions) or if the encodings and by extension the training process was flawed (most of the values in the encoding were 0, and we are unable to cross check with the original source since we are unable to run the notebook on Kaggle).

However, if we are able to iron out the issues, the triplet loss model is definitely viable to be used in real life, especially given the results of a properly trained model as seen in the original reference.

**Screenshots and Diagrams**

Transfer Learning Model

**A screenshot of a computer program

Description automatically generated**

Triplet Loss Model

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Triplet loss


Model Diagram for the embedding model is too large to be displayed here. You can find it under here: <https://www.kaggle.com/code/amankumarmallik/one-shot-learning-for-face-verification/output?select=Inception_one_shot.png>

**Limitations**

Tensorflow backwards compatibility issues caused issues when running code from the Kaggle notebook and loading the VGGface model. Many of the fixes aren’t 1-to-1 representations of the original code, or were workarounds that did not make use of libraries with such issues.

The assumption is made that the person does not change during runtime, for both the testing using webcam and data collection process. However, for our purposes, these are reasonable since the data collection is very fast and should not have more than 1 person in frame during the time period, and for testing, we only want to test what the models would predict given different faces. If the person wears different facial accessories, our model will have trouble recognising them.

**Conclusion and Recommendations**

Deep learning architecture is very useful in performing classification tasks and can be used in a variety of situations, as demonstrated by this project. However, the step-up from deep learning to one-shot learning is a difficult process.

In the future, it would be helpful to generate the embeddings and analyze the result to trace the source of error. Another useful but arduous task would be to study what each layer did to find the layer where errors originate from.

**Individual Reflections**

Dominic:

This project has been enlightening and has opened my eyes to the usefulness of AI and Machine Learning. In this project, I learnt how to use OpenCV, various TensorFlow libraries and the triplet model – triplet loss architecture. Not everything was smooth sailing, of course. Many errors arose due to input errors, which could be traced back to our data generation function. Fixing these errors sheds more insight onto how the model can be visualised, and how the model trains. Across this project, there were times where I felt frustrated, but when I pulled through, there were also moments of excitement when the model finally ran. Besides that, I could have improved the model's performance by optimizing the data fed into the model, as well as testing the triplet loss function manually to see how it performed.

Aik Lok:

For me, this project has been as insightful as it has been frustrating. I have learnt a great deal of things, namely the triplet loss function, the existence of one-shot learning as well as the usage of the OpenCV library. I have ran into many errors, them being the inability to run the Kaggle notebook on Kaggle due to tensorflow claiming the input shape needs to be in NCHW format (standing for number of inputs in a batch from the generator, number of channels for the images with RGB being 3 and B&W being1, height of image and width of image) for the shape of the input data for the triplet loss function model despite already changing the data generator function to output it in that format, the generator outputting the training data as a list which is no longer accepted as a format of training data for models on tensorflow, which had to be changed in order for it to work with the current version of tensorflow and keras no longer allowing specific imports when attempting to import VGGFace, to name a few. Trying to resolve or find workarounds for issues that should not have arisen, and were not expected especially when running the reference notebook on the site it was found, that being Kaggle, was definitely frustrating and took a lot of time. In the end, we did manage to work out most of the issues, so even though the model is definitely far from perfect, I am happy enough with the fact that we even managed to get the model training and running in the first place. I feel like with more time, we might have been able to figure out the issues with the model, but even then, I still have learnt quite a bit from this project, and the module by extension.

**References**

[1] <https://www.youtube.com/watch?v=lH01BgsIPuE>

[2] <https://www.kaggle.com/code/amankumarmallik/one-shot-learning-for-face-verification>

**Work Distribution Matrix**

|  |  |  |
| --- | --- | --- |
| **Work Description** | **Aik Lok** | **Dominic** |
| Ideation & Proposal | ✔ | ✔ |
| Data Collection & Method | ✔ |  |
| Model Type 1: MobileNet Transfer Learning CNN | ✔ |  |
| Model Type 2: Advanced CNN Model utilising Triplet Loss Function | ✔ | ✔ |
| Model Type 2: Data Generation for Triplet Loss Function |  | ✔ |
| Evaluation of Models | ✔ |  |
| Video | ✔ |  |
| Report | ✔ | ✔ |
| Finding References | ✔ | ✔ |