CPS803 - Group 11 - Project Report

Khoa Tran
Computer Science
Ryerson University
Toronto, Canada
khoa.leminh.tran@ryerson.ca

Imran Remtulla
Computer Science
Ryerson University
Toronto, Canada
iremtulla@ryerson.ca

Akif Imtiaz
Computer Science
Ryerson University
Toronto, Canada
akif.imtiaz@ryerson.ca

I. INTRODUCTION

The need for capable facial recognition systems is increasing every day, as new applications for the technology are found - from security and surveillance to self-driving cars. One unique requirement for facial recognition that sets it apart from other AI problems is the relatively small datasets it is expected to work with; anyone who needs a facial recognition system to recognize them would not be willing to provide more than a few images. With this, the computer is expected to learn how to accurately identify that person under a wide range of lighting conditions, facial expressions, hairstyles, etc. That type of machine learning - where a model uses a small number of examples to cover a wide range of inputs - is referred to as few-shot learning, and was the focus of this project.

In short, a dataset of labeled face images was used, with only a few training images per label. Then, image augmentation was used to effectively create more training data. Then, different models were created and trained on the training data. The models were then used to make predictions for the test data, and various performance metrics were recorded. The models were then refined and experimented with to achieve the best results. This process is elaborated on further below.

II. PROBLEM STATEMENT AND DATASET

The main problem to be solved was to create a system that could be trained on images of thousands of individuals, with only a few images per person, and then be capable of recognizing the person in a new image with a high degree of accuracy.

To achieve this, the publicly available CelebA dataset of face images of over 253 different people was used. For each person, 5 images were picked at random for the training dataset, and 3 were picked for the test dataset.

Image augmentation was performed on the training dataset, where 9 new images were derived from each original image.

The original train dataset was then split into train and evaluation ('eval') sets, at a 70/30 ratio. The eval set was meant to be used to evaluate model performance during training.

Aside from the train and test dataset, an 'other' dataset was also prepared. This contained images of people that did not exist in the training dataset, and so could never be identified by any trained model. The purpose of this dataset was to ensure that the models could correctly predict when an image had an unknown face in it, rather than reporting a false positive for any existing label. This is important since identifying strangers is a critical feature for any security system that uses facial recognition. During training, this 'other' set was to be appended to the eval set and used to tune the models.

Finally, the images in all datasets needed to be turned into a standardized numerical format that the models could work with. To do this, two pre-trained models were used as described in the following sections. Any images that failed to go through this process were discarded.

At this point, the dataset was ready to be used. It had the train, test, and eval (+ 'other') subsets with 7023, 900, and 3724 images respectively, and consisted of face embeddings for the faces in the original images.

III. METHODS AND MODELS

A. Data Processing

a. Image Augmentation

The training set was augmented to provide more training material. This involved manipulating the existing images in various ways such as rotating, flipping, resizing, adjusting the brightness, and adding noise. This helped increase the amount of training data and also helped the models to better recognize faces under various different conditions (such as the ones listed above). The augmentation allowed each original training image to have an additional 9 augmented images, thus helping to increase the overall accuracy of the model. In total, after augmentation, the number of training examples increased to 12,630 data points for the training + eval set (including those that would be discarded later after failing the face detection step). This contributed to the problem statement that computers are expected to accurately recognize a person under various different lighting conditions and features with only a few reference images of that person's face.

Original image

Flipped image

Grey Image

Noise Added

b. MTCNN

First, the MTCNN facial extraction model was used to identify the region of each image that contained a face. MTCNN (Multi-Task Cascaded Convolutional Networks) is a highly accurate face detection model that was developed in 2016 and is freely available. It uses three stages of convolutional neural networks to perform its task. With it, the face in each image could be cropped and then resized to 160 by 160 pixels. Any image in which MTCNN could not find a face was ignored.

An example of MTCNN's ability

c. FaceNet

FaceNet is a Google-developed ML system from 2015 that "directly learns a mapping from face images to a compact Euclidean space where distances directly correspond to a measure of face similarity." Put simply, the FaceNet model was used to extract numerical feature mappings, called face embeddings, from each image. A face embedding is a 128-element numerical array that captures information about the unique or interesting features of the face. These can be directly compared with each other or can be used to train models and make predictions.

B. Models

a. Brute Force Euclidean Distance

This first system was the simplest possible approach. It does not involve training. For prediction, the new 'test' face embedding is simply compared to every single face embedding from the training dataset by measuring the Euclidean distances between them. The training image with the shortest Euclidean distance from the test image is found, and the person/label from that training image is identified as the predicted identity for the test face. If the shortest distance is still above a certain threshold, then the identity of the new test face is declared to be unknown. This threshold is the only parameter of this system; it was experimented with using the eval set to find the optimal value.

Naturally, the brute-force method of comparing every training sample to the test sample means this system is very slow. The next approach is an optimized version of this Euclidean distance comparison and is much faster.

b. Using K-D Tree With Euclidean Distance Metric

A K-D tree is a data structure that is used to organize points in a k-dimensional space. This makes it ideal to store face embeddings, as they are 128-dimensional vectors. While building a K-D tree takes time, that initial investment pays off later as the structure can be navigated very efficiently. The samples in the training dataset can be arranged in a K-D tree

according to their Euclidean distances relative to each other. Then instead of comparing the Euclidean distance of the new test sample with every single training sample (as in the previous section), the K-D tree can be used to efficiently search through the training set and drastically reduce the number of comparisons needed. It can be thought of as searching through a sorted binary tree, except in k-dimensions instead of 2. Similar to the previous approach, the only parameter to change is the threshold at which the prediction is "unknown".

As will be explored further below, the results from the previous approach are identical to those from this method, but the latter is several times faster.

c. Stacked SVM

Support Vector Machines are famous for their ability to fit high dimensional data with limited examples, therefore an SVM was a natural choice for this project. The previous methods used a threshold value to decide when a sample was from the 'other' dataset, meaning they did not exist in the training set and were, therefore, unknown. To play a similar role in this method, a second SVM was introduced to process the output probability distribution of the first SVM. The overall system structure is shown below:

The first model, SVM A, was trained to classify examples. The second model, SVM B, was trained to study the softmax-like probability distribution output of SVM A to determine if the data point is from the 'other' dataset or not. If its answer is yes, the example is concluded to be unknown; if not, it will be pushed back to model A for class determination.

To decide the viability of the proposed method, we have visualized the entropy output from SVM A. The hypothesis is that if the label is from the 'other' dataset, SVM A output will be distributed more evenly than the softmax-like output. In other words, the model will be more uncertain.

The above graph shows that the hypothesis is true. For the unseen labels, there is a high density of around 5.3, whereas the remaining samples are spread out in the lower unit.

IV. RESULTS AND DISCUSSIONS

A. Exploratory Data Analysis

After augmentation, within 12,630 total examples, MTCNN was able to detect 10,75 faces, around 85%; the remaining face images were not used. Most of the face-detection errors were from the results of image augmentation. This shows that the image augmentation process used could be improved in the future. The following is the error distribution.

	Agu1	Agu2	Agu3	Agu4	Agu5
Count	312	290	295	302	288

	Blur	Bright	Grey	Noise	None
Count	1	339	27	20	4

To understand more about face embeddings, we decided to find the centroid of each label. This was achieved by averaging all the vectors of a person to find their spatial center. This provides the opportunity to study which faces are most similar, which are the most different by measuring the distances from one to another.

Label 1286 Label 9212

According to the FaceNet model, these were the 2 most similar people in our data set, with a distance of 5.22884 from each other. The resemblance is notable by eye.

Label 171 Label 9047

These were the 2 most different people, with a distance of around 15. We might expect that these people would have opposite genders and/or different skin colours, but these two have the same gender and skin colour. This gives us an insight that the model only focuses on facial features, not other visual factors.

A dimensional reduction technique called tsne was employed to visualize a sample of 100 labels from the training set:

It is clear that some clusters have formed, indicating that different boundaries can be drawn for classification.

Interestingly some outliers are gathered in the middle; this noise is possibly caused by the image augmentation.

B. Model Performance

a. Brute Force Euclidean Distance

After repeated trial and error, the threshold value of 10 was found to be optimal. With this threshold, the system had an accuracy of about 84.77% on the eval dataset, and the time taken to make all the predictions was over 530 seconds.

On the test dataset, an accuracy of about 71.11% was achieved in 122.44 seconds.

It should be noted that an analysis of the training dataset showed that the two labels that were closest to each other (the two different people in the dataset who looked the most alike) had a Euclidean distance from each other of only 5.13 on average. It is likely that errors made by the system involved these individuals or others that looked like. This is also true for the next K-D tree model.

We consider the Brute Force model as a baseline. It achieved good accuracy with zero training time, but a terribly high classification time. Inspired by KNN, we decided to employ a K-D tree to optimize this process.

b. K-D Tree With Euclidean Distance

Since this optimized approach is very fast, predictions were made for the eval dataset using every threshold value between 1 and 20. As with the previous approach, the optimal threshold was found to be 10, and the highest accuracy achieved on the eval dataset was 84.77%. The time taken was only about 6.87 seconds.

On the test dataset, an accuracy of about 71.11% was achieved in 2.32 seconds.

The graph of accuracy on the eval dataset for each threshold between 1 and 20 is below:

c. Stacked SVM

Initially, both SVM A and SVM B showed signs of overfitting, with high accuracy on the training set and low accuracy on the eval set. By combining grid search and cross-validation, optimal hyperparameters were found for both models.

For SVM A, the radial basis function kernel was chosen with an L2 penalty of 10 and kernel coefficient gamma of 0.01. It took around 49 seconds to train. Model SVM B also uses the radial basis function kernel as a kernel with an L2 penalty of 1000 and gamma of 1.

In total, the time taken for both models to train was 50.68s. It achieved an accuracy of 82.01% on the validation set in 47s, and 67.88% on the test set in 11.72s.

d. Comparison of the Models

	BF	KD	SSVM
Time train total	0.00 s	0.12 s	50.68 s
Time per test	142.04 ms	2.33 ms	10.00 ms
Accuracy Eval	84.77 %	84.77 %	82.01 %
Accuracy Test	71.11 %	71.11 %	65.33 %

BF: Brute Force Euclidean Distance

KD: K-D Tree With Euclidean Distance

SSVM: Stacked SVM

Overall, the second method using the K-D tree was the most accurate, performing slightly better than the SVM. It was also the fastest method. For further real-time systems, the K-D tree may be suitable due to its reliable performance and rapid classification time.

Compared to other existing methods, our approach is not very accurate. However, with limited experience in the field, we wanted to tackle the few-shot learning problem using machine learning to understand more about the subject and its process. We also wanted to try innovative methods to solve the stranger detection problem. If the project is to be continued, we want to optimize the running time and implement a real-time system. A more powerful deep learning method like CNN could be used from the start, rather than pre-trained ML models.

V. IMPLEMENTATION AND CODE

This project was implemented in the Python programming language and various libraries as follows.

The Pillow library was used to load images, and the MTCNN face detection library was used to crop out the faces in each image. Each cropped face was then turned into a face embedding using the pre-trained FaceNet model and the Keras ML library. To avoid having to repeat this process later, the face embeddings for each dataset were saved to CSV files using the NumPy and Pandas libraries. During model training and testing, the face embeddings could be similarly loaded and used.

To make predictions using the brute-force Euclidean distance approach, the `norm` function from the NumPy library was used to calculate distances. For the K-D tree model, the implementation of a K-D tree from the scikit-learn library was used. And the SVM implementation from the scikit-learn library was used for the SVM model.

Finally, the Matplotlib library was used to plot graphs using the results after they were obtained.

VI. REFERENCES

Brownlee, J. (2020, June 10). *One-Shot learning for face recognition*. Machine Learning Mastery. Retrieved September 27, 2021, from https://machinelearningmastery.com/one-shot-learning-with-siamese-networks-contrastive-and-triplet-loss-for-face-recognition.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2016). Large-scale celebfaces attributes (celeba) dataset. CelebA Dataset. Retrieved December 3, 2021, from http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html.

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, *23*(10), 1499–1503. https://doi.org/10.1109/lsp.2016.2603342

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://doi.org/10.1109/cvpr.2015.7298682

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Gandhi, A. (2021, May 20). *Data augmentation: How to use deep learning when you have limited data*. AI & Machine Learning Blog. Retrieved October 4, 2021, from https://nanonets.com/blog/data-augmentation-how-to-

use-deep-learning-when-you-have-limited-data-part-2
.

Brownlee, J. (2020, August 23). *How to develop a face recognition system USING Facenet in Keras*. Machine Learning Mastery. Retrieved September 27, 2021, from https://machinelearningmastery.com/how-to-develop-a-face-recognition-system-using-facenet-in-keras-and-an-svm-classifier.

Tom, MarianD, & RockTheStar. (2014, November 7). *How to determine the optimal threshold for a classifier and generate ROC curve?* Cross Validated. Retrieved October 4, 2021, from https://stats.stackexchange.com/questions/123124/how-to-determine-the-optimal-threshold-for-a-classifier-and-generate-roc-curve.

Pandey, P. (2019, December 27). *A guide to machine learning in R for beginners: Logistic regression*. Medium. Retrieved October 4, 2021, from https://medium.com/analytics-vidhya/a-guide-to-machine-learning-in-r-for-beginners-part-5-4c00f2366b90.