Learning-Based Face Recognition with User Interface

Tsun-Hsuang Wang
102061210
johnsonwang0810@gmail.com

Meng-Li Shih
102061240
shihsml@gmail.com

Yi-Lun Wu
102061122
cooolallen@gmail.com

Abstract

Face recognition becomes more popular in our daily lives. However, how to implement this system in real time with limited resources could be a challenging issue. Our project focuses on integrating face detection, face recognition and user interface to a real-time face recognition application on laptop. We apply MTCNN [1] on face detection which achieves average precision 0.82 on WIDER FACE medium set. Previous research like FaceNet [2, 3] has about 98% accuracy on LFW dataset and provides us an approach to real-time face recognition. In addition, PyQt5 gives us an efficient way to construct a friendly user interface in a very short time. With our system, the recognition accuracy achieves 98.67%, and each frame takes about 0.5 second for whole process. We believe this system can make contributions in various applications.

1. Introduction

Face recognition is a task that human performs routinely and effortlessly in our daily lives. It has several advantages, such as the face can be captured in a covert manner compared to fingerprint and Iris. Also, the powerful computers and embedded computing system (e.g. Nvidia Tk1) are ubiquitous, benifiting to face recognition and making it become a popular application nowadays (e.g. Tag suggestion by Facebook).

Traditionally, face recognition is a cascade feature extracting system, and every component is hand-crafted. After deep learning becomes popular, people rethink this technique. They use learning-based model with large human face dataset to extract more representative features. As a result, these features let face recognition performs better.

In this project, we use learning-based model to extract face features and to recognize who you are. Also, we create an elegant user interface in order to provide an awesome user experience. By interacting with our UI, you can add/remove your face in our database with just a few clicks.

2. Review of previous work

2.1. Face Detection

Previous face detection technique use hand-crafted features. In the early days, in order to detect face in real time, several techniques such as Viola-Jones face detector [4] are proposed. Later, cascade models [5] using several weak feature frameworks are proposed to boost the accuracy. Recently, as learning based model become popular, people start to cascade learning based model(ex. Auto-Encoder) [6] to further improve the performance. Our framework is more general in that we can adopt a CNN-based face alignment method to achieve joint face alignment and detection, and we use CNN to learn more robust features for faces.

2.2. Face Recognition

There are abundant works in face verification and recognition. We only name a few that is much more related to FaceNet. Sun et al. [7, 8] propose a compact and therefore relatively cheap to compute network. They use an ensemble of 25 of these network, each operating on a different face patch. Taigman et al. [9] propose a multi-stage approach that aligns faces to a general 3D shape model. A multi-class network is trained to perform the face recognition task on over four thousand identities. Zhenyao et al. [10] employ a deep network to warp faces into a canonical frontal view and then learn CNN that classifies each face as belonging to a known identity. For face verification, PCA on the network output in conjunction with an ensemble of SVMs is used.

3. Approach

We implement this project on the Python3 platform with OpenCV, TensorFlow and PyQt5 packages. Python3 is a platform with plenty of open source packages so that designer can create their project very efficiently. TensorFlow is a package for machine learning and deep learning. We use it to build up the process of detection and recognition. OpenCV mainly support functions about camera, mouse event and multi-media holder. On the other hand,

PyQt5 create an excellent user experience by constructing a friendly user interface. With this package, we can show the database in the form of table widget so that user can understand the situation with only a glance.

3.1. MTCNN

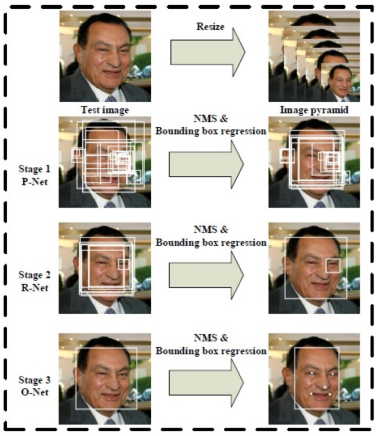

Figure 1. MTCNN Pipe Line.

Multi-task convolutional neural network (MTCNN) is a cascaded CNN, which follows coarse-to-fine approach frequently used in computer vision. This can be found in the above figure that there are crowded bounding boxes in stage 1, and after each stages, bounding boxes are refined and number of bounding boxes decreases.

Preprocessing: We build a multi-scale image pyramid before input to multi-stage the convolution neural network (CNN). Such preprocessing is common in object detection tasks in order to search for objects, in our case, faces, in different scales.

Stage 1: The first stage CNN will propose many bounding box regression vectors and corresponding confidence by feeding multi-scale RGB images. Hence, we call it Proposal Network (P-Net). Next, we perform Non-maximum suppression(NMS) to merge highly overlapped bounding boxes.

Stage 2: The second stage CNN will refine bounding boxes proposed by P-Net. Hence, we call it Refine Network (R-Net). According to the paper, large amount of false candidate will not pass R-Net. Next, we perform NMS to further reduce the quantity of bounding boxes.

Stage 3: The third stage CNN will output accurate bounding boxes and corresponding face alignment. Hence, we call it Output Network (O-Net). Note that we do not use NMS after that since the bounding boxes is already very accurate.

To train MTCNN detector, there are 3 tasks to be leveraged: face/non-face classification, bounding box regression, and facial landmark localization.

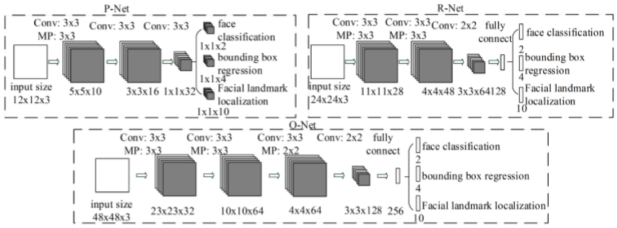

Figure 2. MTCNN Structure. Each cascade component (P-Net, R-Net, and Q-Net) is a simple CNN, which benefits to fast inference. Note that P-Net is fully convolutional, so it can be fed with images of arbitrary size.

1) Face classification: the learning objective is a binary classification problem, face and non-face. This can be done by minimizing cross entropy loss,

$$(L_i)^{det} = -(y_i^{det}\log(p_i)) + (1 - y_i^{det})(1 - \log(p_i)),$$

where p_i is the probability indicates a corresponding bounding box being a face. The notation $y_i^{det} \in \{0, 1\}$ denotes the ground truth label.

2) Bounding box regression: the learning objective is to find the correct coordinates of the four corners of the bounding box. This can be formulated as a regression problem, and we employ Euclidean loss for each sample x_i,

$$L_i^{box} = \left\| \hat{y_i}^{box} - y_i^{box} \right\|_2^2,$$

where $\hat{y_i}^{box}$ is the regression output from the Network and y_i^{box} is the tuple of ground-truth coordinates(left, top, height, width).

3) Facial landmark localization: similar to bounding box regression, facial landmark localization is also a regression problem. However, our objective now is not to detect faces but to detect five facial landmarks. This can further improve the accuracy of detection. Euclidean loss is used for each landmark,

$$L_i^{landmark} = \left\| \hat{y_i}^{landmark} - y_i^{landmark} \right\|_2^2,$$

where $\hat{y_i}^{landmark}$ is the facial landmarks coordinate output from the Network and $y_i^{landmark}$ is the tuple of ground-truth coordinates(left eye, right eye, nose, left mouth corner, and right mouth corner).

Another thing to be remarked, the training process of MTCNN utilize online hard sample mining, i.e. MTCNN only use 70% hard sampling mining in the back propagation in each mini branch. The reason is that the easy samples are less useful to improve the efficiency of training. The experiments show that this approach has better performance than traditional hard sampling mining.

3.2. Face Net

In our face recognition system, which needs to compare and recognize arbitrary faces, the nave classifier scheme doesnt work since there is no way to define categories to be classified as. Alternatively, we resort to a general transformation over faces and later on applying matching or clustering algorithm. This is exactly how FaceNet can be used. FaceNet projects images of faces to an embedding space following Euclidean metric structure, allowing us to later on perform very simple comparing algorithm on those face embedding.

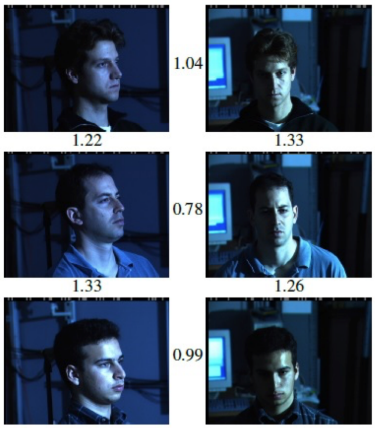

Figure 3. Numbers among the above images are the distances between embedding of the faces outputted from FaceNet. Faces in each row have the same identity. We can see that threshold at about 1.1 can classify every pairs correctly.

There are 2 stage in training FaceNet. It is first bootstrapped as a classifier that aims to recognize $N = 2,622$ different identities. Fixed-size images are fed into a deep convolutional neural network, and the network will output a probability distribution over the N classes, indicating the confidence of input face image being recognized as certain identities. This leads to the objective as softmax log-loss

$$E(\phi_1) = -\sum_t \log\left(\frac{e^{<e_{c_t},\phi_1(x)>}}{\sum_{q=1}^{N} e^{<e_q,\phi_1(x)>}}\right),$$

where ϕ_1 is our model in the first training stage, x is the input image, and e_c denotes the one-hot label of class c. This stage simply makes the network pre-trained and learning to extract image features, meanwhile, significantly speeding up and easing training process.

The second stage is to train the general transformation from face images to embedding space. The final layer, i.e. output layer, of the pre-trained network is taken away and replaced by a D-dimensional fully connected layer, D may not be equal to N, aiming to output the embedding $\phi_2(x)$. In order to enforce the embedding space to take on Euclidean metric structure, the network is trained to minimize the triplet loss $E(\phi_2) =$

$$\sum_{(a,p,n)\in T} (\alpha - \|\phi_2(x_a) - \phi_2(x_n)\|_2^2 + \|\phi_2(x_a) - \phi_2(x_p)\|_2^2)_+,$$

where $\phi_2$is our model in the second training stage, is the collection of all triplets, α is margin that is enforced between positive and negative pairs. Also, the embedding is constrained to live on D-dimensional hypersphere, i.e. $\|\phi_2(x)\|_2 = 1$. A triplet contains an anchor face image , a positive face image $p \neq a$ but has the same identity with a, and a negative face image $n \neq a$, having different identity from a. Here, we want to maximize distance between embedding of distinct identities $\phi_2(x_a)$ and $\phi_2(x_n)$, and minimize distance between embedding of the same identities $\phi_2(x_a)$ and $\phi_2(x_p)$. In order to enhance discriminative power of the embedding, triplets are chosen to keep challenging the model, i.e. x_n is a hard negative example, and x_p is a hard positive example. Finally, the actual deep network structure can be arbitrary. In our work, we use inception-residual modules. You can find more details of the architecture in our implementation posted on github. [11]

Figure 4. Model structure of FaceNet consists of a mini-batch input layer, from which triplets are chosen, and a deep CNN followed by normalization, which results in the face embedding, and finally followed by the triplet loss during training.

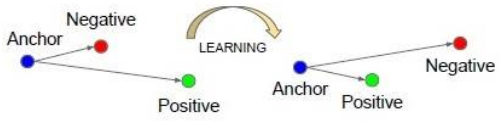

Figure 5. Learning objective of triplet loss is to increase the distance between face embedding of different identities and lower the distance between face embedding of the same identities.

3.3. Matching

After obtaining embedding of an unidentified face, we then compare it with all identified embedding saved in our database. We have tried nearest neighbor algorithm and K-nearest neighbor algorithm, and we found that nearest neighbor is good enough and is faster. Thus, we use nearest neighbor algorithm for face embedding matching in our system. Besides, we have thought of an extra constraint that

faces with the same identity may not exist in the same image. To cope with this issue, Munkres algorithm may be a good solution but we leave this to future exploration.

3.4. User Interface

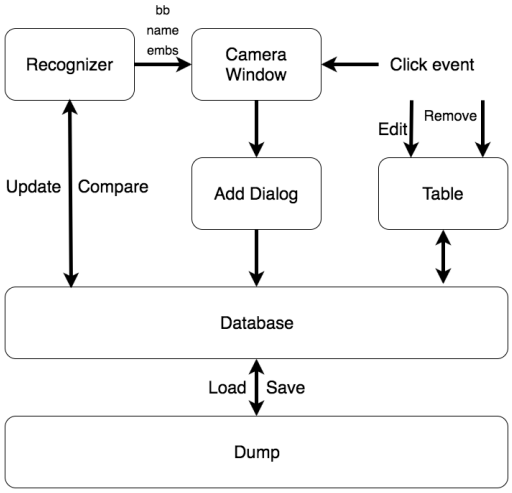

Figure 6. Flow Chart of User Interface

As for user interface, the recogniser will send the information of bounding box, identity name and corresponding features to the Camera Window(held by OpenCV2).

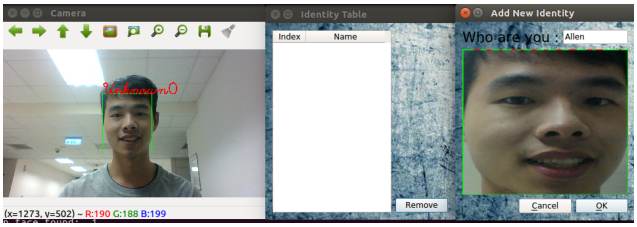

Figure 7. When user click the bounding box, the Add Dialog pop up and wait for user typing his name.

Add Identity

When user click on the window, it will capture the mouse event and check if the coordination is local in bounding box or not. If yes, Add Dialog will pop out with a bounding box cropping image and corresponding features. When user finish editing his name and click on the ok button, the pair of name and features will be updated into both database and table. If user enter a name which is already exist in the database, it will build up a new prototype under the same name and increase number of prototype of this identity.

Edit name

The Table Dialog will send out a signal to database when user editing the name cell, this signal will update the name in the database and dump to the disk at the same time.

Remove Identity

The Remove Button in the Table Dialog will send out a remove signal when user click on it and delete the selected row in the table, update the database and dump database to the disk.

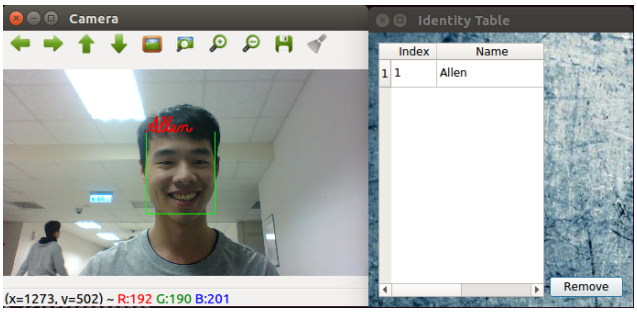

Figure 8. After user completing edition, our system can recoginize this new identity

4. Experiments

4.1. t-SNE visualization tool:

As mentioned before, the embedding of FaceNet have the properties of metric structure. However, we can not visualize them in high dimension. Instead, we use a tools called t-SNE to reduce the dimension of embedding from 1792 dimension to 2 dimension.

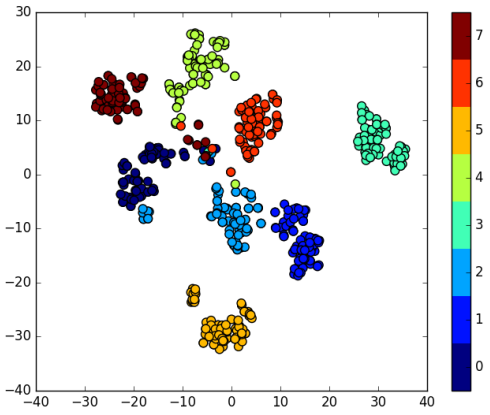

Figure 9. Dimension-reduced embedding from different identities. We can see that embedding of each identity face become a group themselves which means FaceNet really project face images to embedding following Euclidean metric structure.

Visualize the effect of glasses and image resolution:

Different appearance, e.g. Wearing a glasses or not, will affect the embedding. As we can see, embedding of faces

wearing glasses become a group and embedding of original faces become another group. Also, the resolution will have effect on the embedding and the reason will be discussed in the next part.

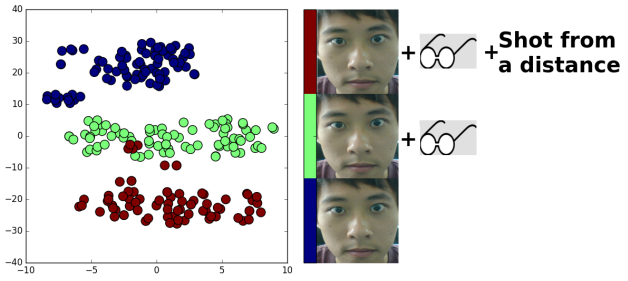

Figure 10. Dimension-reduced embedding from different identities. We can see that embedding of each identity face become a group themselves which means FaceNet really project face images to embedding following Euclidean metric structure.

Compare different identities and different appearance:

By visualizing the embedding above, we get two interesting results. First, even if the certain face images are taken from different appearance (ex. with/without glasses, low/high resolution), their embedding are still close to each other in comparison to other identities. Second, as the image resolution becomes lower, face embedding get closer even though they are from different identities. The reason is intuitively simple. As we degrade the image resolution, important information on the face lost and the faces become hard-to-tell. In other words, the points representing embedding of faces (no matter with the same of different identities) get closer.

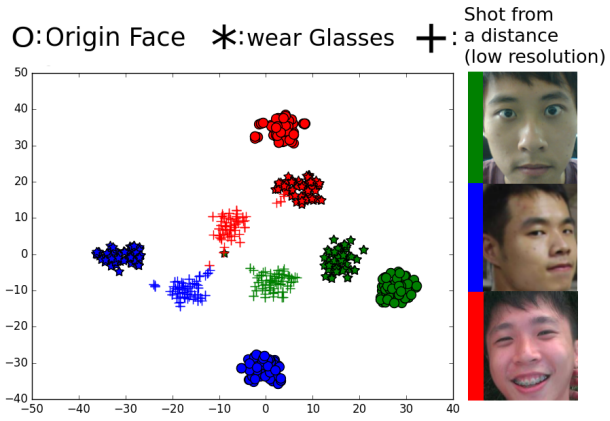

Figure 11. Dimension-reduced embedding from different appearance with different identities. Color indicates the identity, Circle indicates origin face, Star indicates the face with glasses, and Cross indicates the low resolution image.

4.2. Distance Histogram and Threshold

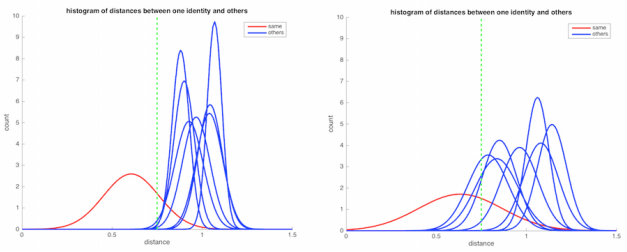

Figure 12. Histogram of distances between one embedding A and the others. Red curve is histogram fit of distances between embedding A and other embedding of the same identities. Blue curve is histogram fit of distances between embedding A and other embedding of different identities. Green dashed line is threshold.

We compute distances from one embedding of identity A (a prototype) to many other embedding of arbitrary identities (including A), and plot the histogram. We use Gaussian distribution to fit the discrete histogram bins. By simple thresholding, our system can discriminate identity from other different identities. However, if we only take one embedding as prototype, accuracy may be extremely fluctuated by the embedding chosen. This can be observe by comparing left and right graph of Figure 12, which are plotted with respectively a good prototype and a bad prototype.

	TP	TN	FP	FN	Precision	Recall
General Case	74	559	1	6	98.67%	92.5%
Exception	51	489	71	29	41.8%	63.75%

Table 1. True Positive, True Negative, False Positive, and False Negative table

We conduct experiments on 8 different identities with each identity having 80 samples. An embedding is randomly chosen and used as prototype for matching. Then, we test our system with all other samples. Performance of recognizing a specific identity from the others is shown as Table 1. The table shows the performance of our system in general case. Also in some rare case, our system may not work well due to bad prototype choices. Note that the performance can be further boosted if we add more prototypes.

4.3. Time Comsuption

Since we use CPU to run our network, the number of faces existing in a frame has a huge impact over inference time of our system. Inference time of MTCNN does not vary a lot with the number of faces existing. However, inference time of FaceNet is theoretically linear to the number of faces. Accordingly, using GPU for inference in FaceNet will extremely alleviate this problem.

# of faces	time per frame(second)
0	0.08
1	0.45
2	0.82
3	1.15
4	1.54
5	1.88

Table 2. Inference time with different number of faces in a frame.

4.4. User Experience Feedback

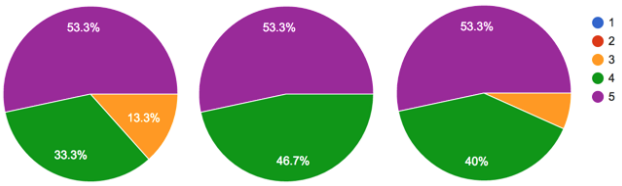

Figure 13. User experience feedback on user interface, recognition performance, and overall satisfaction, corresponding to pie chart from left to right. From 1 to 5 refers to bad to good.

We collect user experience feedback from 15 people, with each of them first plays around with our system and fill in a form. The result is shown in Figure 13. Besides, we get addition suggestions, e.g. can get much fancier in UI, speed of the system is yet to be improved, face images get distorted in the pop-up window while adding new identity.

5. Conclusion

For face detection and recognition tasks, our approach using MTCNN and FaceNet both achieve fairly great performance, as shown in our experimental results. However, there are still some problems to be coped with. The speed of our system still have a lot of room for improvement. Accelerating flow of the entire system still has a long way to go. Inference should not be done for every frame. Pure image processing approach does not fully exploit the characteristic of videos. Taking temporal information into consideration may reduce inference time and accomplish better recognition accuracy. Besides, from user feedback, we still have to work further on the design of our user interface. By and large, our project may be valuable and worth-seeing as a prototype of an for-sale product.

References

[1] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. *arXiv preprint arXiv:1604.02878*, 2016.

[2] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[3] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, volume 1, page 6, 2015.

[4] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE, 2001.

[5] Dong Chen, Shaoqing Ren, Yichen Wei, Xudong Cao, and Jian Sun. Joint cascade face detection and alignment. In *European Conference on Computer Vision*, pages 109–122. Springer, 2014.

[6] Jie Zhang, Shiguang Shan, Meina Kan, and Xilin Chen. Coarse-to-fine auto-encoder networks (cfan) for real-time face alignment. In *European Conference on Computer Vision*, pages 1–16. Springer, 2014.

[7] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1988–1996. Curran Associates, Inc., 2014.

[8] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[9] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[10] Zhenyao Zhu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Recover canonical-view faces in the wild with deep neural networks. *arXiv preprint arXiv:1404.3543*, 2014.

[11] Yi-Lun Wu Tsun-Hsuang Wang, Meng-Li Shih. Face detection and recognition system with user interface. `https://github.com/cooolallen/Face-recognition-with-User-interface`, 2017.