Face Recognition Application Based on MTCNN and FaceNet

Abstract – **This implementation was to design a face recognition application that can capture people's face images and store them as our database for the second time recognition to distinguish if I already have these people registered. Generally, it has two stages: firstly, the pre-processing of the captured face images with the approach "Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks (MTCNN)" which can detect faces correctly in the captured images and process them properly for the next stage. Secondly, the recognition of the pre-processed faces with the approach "FaceNet: A Unified Embedding for Face Recognition and Clustering" which can distinguish if they are the people who are already registered in the database or are strangers. Besides, I test the application on *CASIA-WebFace* and *CMU Face Images data set, and* change the brightness of face images to simulate the large visual variations of real world environment, which proves this face recognition application has a good accuracy and robustness.**

Index Terms – Face Detection, Face Alignment, Face Recognition

I. INTRODUCTION

Images of faces, represented as high-dimensional pixel arrays, often belong to a manifold of intrinsically low dimension. Face recognition and computer vision research in general, has witnessed a growing interest in techniques that capitalize on this observation and apply algebraic and statistical tools for extraction and analysis of the underlying manifold. [1] Moreover, face detection and alignment are essential to many face applications, such as face recognition and facial expression analysis. However, the large visual variations of faces, such as occlusions, large pose variations and extreme lightings, impose great challenges for these tasks in real world applications. [2] In the past few years, I have seen a lot of attempts in these two tasks. They all have some pros and cons.

For detection, Viola and Jones proposed a cascade face detector using Haar-Like features and AdaBoost to train cascaded classifiers [3]. Although it can achieve good performance with real-time efficiency, it may also degrade significantly in real-world applications with larger visual variations of human faces even with more advanced features and classifiers. Besides the deformable part models (DPM) can achieve remarkable performance in face detection but they are in need of high computational expense and expensive annotation in training. Recently, inspired by the good performance of CNNs, some CNNs based face detection methods have been proposed. Among them, the most representative ones are Yang et al.'s method [4] which trains deep CNNs for facial attribute recognition to obtain high response in face regions which yield candidate windows of faces but time costly in practice, and Li et al.'s method [5] which exploits cascaded CNNs for face detection but requires bounding box calibration from face detection with extra computational expense and ignores the inherent correlation between facial landmarks localization and bounding box regression. For alignment, the state-of-art can be divided into two branches: regression-based methods and template fitting approaches. Zhang et al. [6] proposed to use facial attribute recognition as an auxiliary task to enhance face alignment performance using deep convolutional neural network.

However, most of the available face detection and face alignment methods ignore the inherent correlation between these two tasks. [2] Although there are some attempts like Chen et al.'s [7] and Zhang et al.'s [8] methods, there are still some limitations in accuracy and performance. On the other hand, mining hard samples in training is critical to strengthen the power of detector. But for traditional hard sample mining, it is usually in an offline manner, which significantly increases the manual operations and is time consuming.

Therefore, as the pre-processing stage for face recognition, I use the framework proposed by Zhang et al. [2], which integrates face detection and alignment using unified cascaded CNNs by multi-task learning and takes advantage of a new online hard sample mining strategy in the learning process, to get better performance on accuracy and runtime.

The proposed CNNs consist of three stages. In the first stage, it produces candidate windows quickly through a shallow CNN. Then, it refines the windows to reject a large number of non-faces windows through a more complex CNN. Finally, it uses a more powerful CNN to refine the result and output facial landmarks positions.

For recognition, there is a vast of corpus of face verification and recognition talking from traditional ways to novel methods using neural network. At the very beginning, there ire some face recognition methods such as PCA(Principal Component Analysis) Eigenface and LDA(Linear Discriminant Analysis) FisherFace. These traditional ways have drawbacks like low-accuracy and low-robust although they are easy to implement. Thus, people proposed ways to improve the recognition performance, for example, combining the PCA with machine learning methods. Yuan et al. [9] proposed a new fusion method using PCA to extract facial images' global features, using LDP operator to extract local texture features and then applying SVM to complete classification and recognition.

Zhenyao et al.'s [10] approach transfers the face images with a complex set of variations to the canonical views. A facial component-based CNN is used to extract facial features.

Methods of using PCA for dimensionality reduction and SVM for classification are also combined in his approach.

However, with the dimension reduction thinking of PCA, many details will be lost simultaneously. And also, SVM performs better in binary classification problem, if I need to classify the images to a large number of classes, it will be time-consuming and complicated.

Then, currently people pay more attention to deep learning methods, Previous face recognition approaches based on deep network use a classification layer trained over a set of known face identities and then take an intermediate bottleneck layer as a representation used to generalize recognition beyond the set of identities used in training [11]. However, the downsides of this approach can not be ignored: to get a high accuracy one has to hope that the bottleneck representation generalizes ill to new faces, and also, the representation size per face always results in large computation demands.

Florian et al. [11] offer some new ideas in paper FaceNet： A Unified embedding for Face Recognition and Clustering to realize the recognition. The core idea of this system is mapping face images of people into a compact euclidean space, then, the similarity of faces can be compared by calculating the distance between different 128 dimensional face embeddings.

II. RELATED WORKS

Online Hard Sample Mining: Instead of conducting traditional hard sample mining after original classifier had been trained, I do online hard sample mining in face classification task to be adaptive to the training process. In particular, I sort the loss computed in the forward propagation phase from all samples and select the top 70% of them as hard samples in each mini-batch. Then I only compute the gradient from the hard samples selected in the backward propagation phase, which means ignoring the easy samples that are less helpful to strengthen the detector while training.

Smaller Filters: Till now, multiple CNNs have been designed for face detection. However, their performance might be limited due to the following reasons: The lack of diversity of some filters may limit them to produce discriminative description of features; Compared to other multi-class objection detection and classification tasks, face detection is a challenge binary classification task (face or non-face), so it may need less numbers of filters but more discrimination of them. In order to solve these problems, the authors reduce the number of filters and change the 5x5 filter to a 3x3 filter to reduce the computing while increase the depth to get better performance.

Non-maximum Suppression: One of the problem when doing the object detection is that your algorithm may find multiple detections of the same objects. So rather than detecting an object once, it might detect it multiple times. Non-maximum suppression is a way for you to make sure that your algorithm detects each object only once. Here in MTCNN, it means to find the candidate window which matches the face most perfectly.

Algorithm:

Each output prediction is: $\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \end{bmatrix}$

where (bx, by, bh, bw) means the bounding box (or candidate window) of a object, p_c means the probability that the object belongs to class c.

1) You should discard all the bounding boxes (or candidate windows) with p_c less than or equal to a threshold, here $p_c \leq 0.6$;
2) You repeatedly pick the box with the largest p_c and output that as a prediction for that there is an object belongs to class c;
3) You discard any remaining box with high overlap ($IoU \geq 0.5$) with the box you output in the previous step.

You keep doing this while there is still any remaining boxes (or candidates) that you have not yet processed until you have taken each of the boxes and either output it as a prediction, or discarded it as having too high overlap (IoU) with one of the boxes that you have output as your predicted position for one of the detected objects.

Intersection Over Union (IoU): As a function to tell if your object detection algorithm is working ill, the IoU computes the size of intersection over the size of union of two bounding boxes (candidate windows), like Fig. 2.1 shows. By convention, the low compute division task will judge that your answer is correct if the IoU is greater than 0.5. And if the predicted and the ground-truth bounding boxes overlapped perfectly, the IoU would be one, because the intersection would equal to the union. If you want to be more stringent, you can set a larger threshold, like the IoU is greater than equal to 0.6 or some other number to tell if the two bounding boxes are close to each other enough. So this is a method to map localization to accuracy where you count up the number of times an algorithm correctly detects and localizes an object where you could use a definition like this of whether or not the objects is correctly localized.

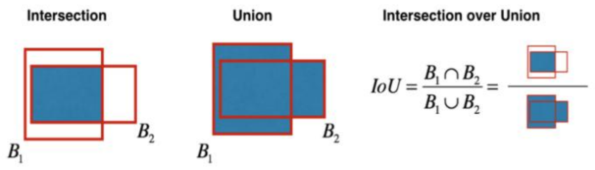


Fig. 2.1. The diagram of IoU.

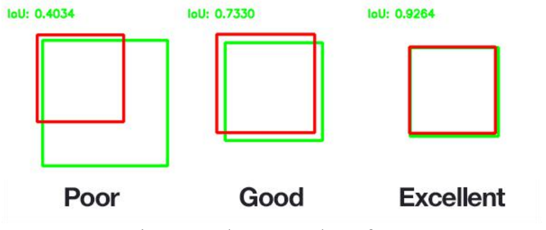


Fig. 2.2. The examples of IoU.

Inception Architecture: It is a common view that the most straightforward way to improve the performance of a deep neural network is to increase the size of this network. This includes not only increasing the depth- the number of network levels - but also its width: the number of units per layer. But this simple solution has two main drawbacks: Larger sizes usually means more parameters, which increase the probability of the happening of over-fitting situations , especially if the training set has a limited set of labels. Another disadvantage of evenly increasing network size is the significant increase in computing resource usage.

Inception structure has two main idea, one is the repetition of similar convolutional modules, which can also be understood as finding the optimal part of structure and repeating it spatially. The second idea is to 'cool down' where the amount of computation will increase significantly, and remain sparse in most parts of the network. For example, add additional 1*1 convolutional layers as a dimension reduction module to remove the convolution bottleneck.

III. ARCHITECTURE AND METHODS

This face recognition is a two-stage framework: the first stage detect faces in the captured images and does the pre-processing on these faces, especially reshapes them into a acceptable size for next stage. And the second stage network compares the distance between the person to be recognized and the faces in database to give us a value under the appropriate threshold if someone is a stranger or not. Actually, it is a MTCNN combined with FaceNet. Fig. 3.1 illustrates the framework of this two-stage network.

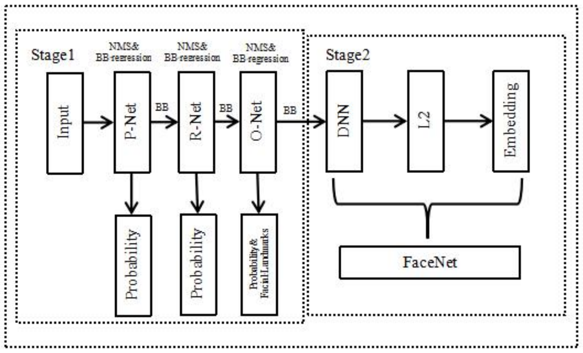


Fig. 3.1. Two-stage framework for face recognition.

A. MTCNN

1) Overall Framework:

Given an image, it is initially resized into different scales to build an image pyramid, which is the input of the following three-stage cascaded framework. Fig. 3.2 illustrates the framework of this three-stage networks.

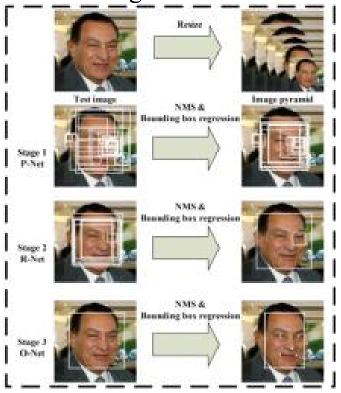

Resize
Test image
Image pyramid
NMS & Bounding box regression
Stage 1 P-Net
NMS & Bounding box regression
Stage 2 R-Net
NMS & Bounding box regression
Stage 3 O-Net

Fig. 3.2. The framework of MTCNN.

Stage 1: The authors exploit a fully convolutional network, the Proposal Network (P-Net) to obtain the candidate windows and their bounding box regression vector. Then they use the estimated bounding box regression vectors to calibrate the candidates and the non-maximum suppression to merge highly overlapped candidates.

Stage 2: All the candidates are fed to the next stage: Refine Network (R-Net). It further rejects a large number of false candidates, performs calibration with bounding box regression and the non-maximum candidate merge.

Stage 3: The final refinement is done in Output Network (O-Net) similar to the second stage. At the same time, this network will output 5 facial landmarks: eyes, noise, mouth corners to give more details.

2) Loss Function:

Although I make use of the parameters in the pre-trained model directly from the paper, the training part is still necessary to understand. The authors leverage three tasks to train the CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization. So it has three loss functions.

- Face classification

The learning objectives is formulated as a binary classification problem. For each sample x_i, the cross-entry loss is used:

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

Pi is the probability produced by the network indicating a sample being a face or not. The notation $y_i^{det} \in \{0,1\}$ denotes the ground-truth label.

- Bounding box regression

The learning objective is formulated as a regression problem. For each candidate window, the model predicts the offset between if and the nearest ground truth (i.e., the bounding boxes' left top, height and width). So here for each sample, the Euclidean loss is used:

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2 \qquad (2)$$

$\hat{y}_i^{box}$ is the regression vector produced by the network and y_i^{box} is the ground-truth coordinate, $\hat{y}_i^{box}, y_i^{box} \in R^4$.

- Facial landmark localization

Similar to the bounding box regression, facial landmark detection is formulated as a regression problem using the Euclidean loss:

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2 \qquad (3)$$

$\hat{y}_i^{landmark}$ is the facial landmark's coordinate (i.e., the left eye, right eye, nose, left mouth corner and right mouth corner) produced by the network and $y_i^{landmark}$ is the ground-truth coordinate, $\hat{y}_i^{landmark}, y_i^{landmark} \in R^{10}$.

- Multi-source training

Since different tasks are employed in each CNNs, there are different types of training images in the learning process, such as face, non-face and partially aligned face. In this case, some of the loss function ((1)-(3)) are not used. For example, for the sample of background region, $L_i^{\det}$ is only used, and the other two losses are set as 0. Therefore, a sample type indicator can be used to integrate the three loss functions above. Then the overall learning target can be formulated as:

$$\min \sum_{i=1}^{N} \sum_{j \in \{\det, box, landmark\}} \alpha_j \beta_i^j L_i^j \qquad (4)$$

$$P-Net, R-Net(\alpha_{\det}=1, \alpha_{box}=0.5, \alpha_{landmark}=0.5)$$

$$O-Net(\alpha_{\det}=1, \alpha_{box}=0.5, \alpha_{landmark}=1)$$

N here is the number of training samples. α_j denotes on the task importance. In P-Net and R-Net, the face classification is more important while in O-Net facial landmark is as significant as face classification for more accuracy.

$\beta_i^j \in \{0,1\}$ is the sample type indicator. Stochastic gradient descent is used to trained the CNNs.

3) How to implement it in Tensorflow:

- Building the MTCNN model

The whole MTCNN model consists of three networks: P-Net, R-Net and O-Net, so I have to build them separately first. It is worth noting here that MTCNN uses Parametric ReLU (PReLU) activation function in P-Net to add the non-linearity between layers.

$$PReLU:$$

$$f(x) = \begin{cases} x & if \quad x > 0 \\ \alpha \cdot x & otherwise \end{cases}$$

In PReLU, the coefficient α is trained along with the other neural network parameters.

Here the authors of FaceNet already trained the parameters of all layers in these three networks ill in CASIA-Webface data set and store them in three .npy files: det1.npy, det2.npy, det3.npy, so i can use these parameters directly to rebuild the networks.

Firstly, I should start a "Graph" in the main function and the whole model is built in this "Graph". Then I start a "session" to run the function to build the three networks. (More coding details are in the script file: align_dataset_mtcnn.py)

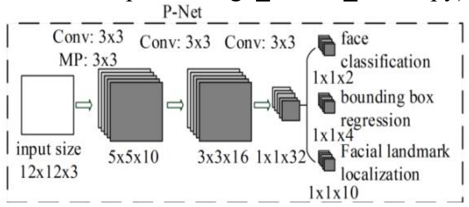


Fig. 3.3. The architecture of P-Net.

Secondly, I create a "variable_scope" with prefix "pnet". The input for P-Net is a (None, None, None, 3) placeholder. Then according to the architecture shown in Fig. 3.3, I use Keras to build all layers in Tensorflow backend. The output are (H, W, 2) face classification and (H, W, 4) bounding box regression.

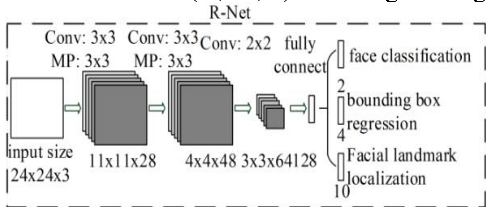


Fig. 3.4. The architecture of R-Net.

Thirdly, I create a "variable_scope" with prefix "rnet". The input for R-Net is a (None, 24, 24, 3) placeholder, building layers with the architecture shown in Fig. 3.4, and the output are (H, W, 2) face classification and refined (H, W, 4) bounding box regression.

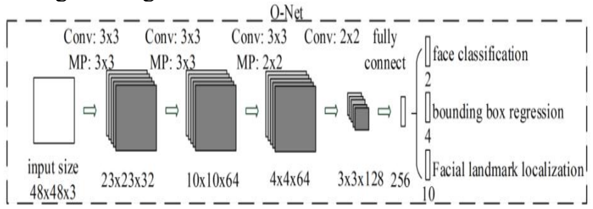


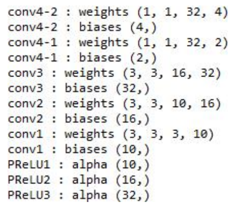
Fig. 3.5. The architecture of O-Net.

Finally, I create a "variable_scope" with prefix "onet". The input for O-Net is a (None, 48, 48, 3) placeholder, building layers with the architecture shown in Fig. 3.5, and the output are (H, W, 2) face classification and refined (H, W, 4) bounding box regression as ill as (H, W, 10) facial landmarks localization. (More coding details are in the script file: detect_face.py)

- Loading parameters

Till now, I already rebuild the framework of the whole networks, then i need to load the pre-trained parameters to make it capable of dealing with specific problems. These parameters are stored in files: det1.npy (P-Net), det2.npy (P-Net), det3.npy (P-Net). So take P-Net for example, what I have to do is to find the names and parameters of each layer in det1.npy, then pass these parameters to corresponding layers i built above according to their names.

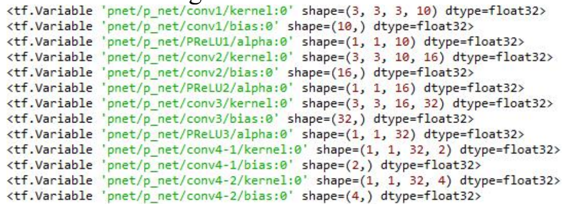
Firstly, let's print out the data in det1.npy. The results are shown in Fig. 3.6.



```
conv4-2 : weights (1, 1, 32, 4)
conv4-2 : biases (4,)
conv4-1 : weights (1, 1, 32, 2)
conv4-1 : biases (2,)
conv3 : weights (3, 3, 16, 32)
conv3 : biases (32,)
conv2 : weights (3, 3, 10, 16)
conv2 : biases (16,)
conv1 : weights (3, 3, 3, 10)
conv1 : biases (10,)
PReLU1 : alpha (10,)
PReLU2 : alpha (16,)
PReLU3 : alpha (32,)
```

Fig. 3.6. The data in det1.npy.

Secondly, let's print out the variables i rebuild in P-Net. The results are shown in Fig. 3.7.



```
<tf.Variable 'pnet/p_net/conv1/kernel:0' shape=(3, 3, 3, 10) dtype=float32>
<tf.Variable 'pnet/p_net/conv1/bias:0' shape=(10,) dtype=float32>
<tf.Variable 'pnet/p_net/PReLU1/alpha:0' shape=(1, 1, 10) dtype=float32>
<tf.Variable 'pnet/p_net/conv2/kernel:0' shape=(3, 3, 10, 16) dtype=float32>
<tf.Variable 'pnet/p_net/conv2/bias:0' shape=(16,) dtype=float32>
<tf.Variable 'pnet/p_net/PReLU2/alpha:0' shape=(1, 1, 16) dtype=float32>
<tf.Variable 'pnet/p_net/conv3/kernel:0' shape=(3, 3, 16, 32) dtype=float32>
<tf.Variable 'pnet/p_net/conv3/bias:0' shape=(32,) dtype=float32>
<tf.Variable 'pnet/p_net/PReLU3/alpha:0' shape=(1, 1, 32) dtype=float32>
<tf.Variable 'pnet/p_net/conv4-1/kernel:0' shape=(1, 1, 32, 2) dtype=float32>
<tf.Variable 'pnet/p_net/conv4-1/bias:0' shape=(2,) dtype=float32>
<tf.Variable 'pnet/p_net/conv4-2/kernel:0' shape=(1, 1, 32, 4) dtype=float32>
<tf.Variable 'pnet/p_net/conv4-2/bias:0' shape=(4,) dtype=float32>
```

Fig. 3.7. The variables in the rebuilt P-Net

Clearly, now I can pass the parameters in det1.npy to corresponding variables in P-Net, so as the other two.

- How to get the output tensors of each networks

I have already successfully rebuild the three cascaded networks and loaded their parameters. Then I should feed input images to get the output tensors. Here in P-Net, our output tensors are bounding box regression from pnet/conv4-2 and face classification from pnet/prob1. So I should run a session from the tensor "pnet/input:0" in Graph to get "pnet/conv4-2/BiasAdd:0" and "pnet/prob1:0" as the output. R-Net and O-Net are almost the same process. So far, the whole MTCNN networks has been preliminarily rebuilt.

- Data flow

I have already known how I rebuild the framework of MTCNN. Then let's see some details used in this joint face detection and alignment method through data flow from an input image to the output.

P-Net:

I can find there is no fully-connected layer in this network, so when I input a at least 12x12x3 image, what I obtain from P-Net is a 16-dimensional feature map rather than a vector. But in our implementation, the 10-dimensional landmarks are not output. Here I set the minimum size of the input image as 20x20, because although the minimum size P-Net can handle is 12x12, as I said above, I have to build a image pyramid first as the input as ill as some conventional image processing work. I build the image pyramid according to the scales I get from:

$$scale = \frac{12.0}{20} \cdot factor^{n}, (factor = 0.709)$$

Then I input the image pyramid to the P-Net and get two outputs:

Bounding box regression vector:

[-0.04171251   -0.03393787   -0.05021905   0.14131135]

Face classification: [0.956438   0.04356182]

Bounding box:

Then I have to generate the bounding box. From the feature maps I get from P-Net, I find the points which are classified as human faces (probability > threshold 0.6), and compute their coordinates in the original inputs. Because I use 12x12 block sliding with stride 2 in the original inputs to get our feature maps, a point in the feature maps equals to a 12x12 block in the original inputs. So the coordinates of the starting point and the ending point of this block is:

$$q1 = np.fix((bb * stride + 1) / scale)$$

$$q2 = np.fix((bb * stride + cell\_size) / scale)$$

The output of this function are 9-dimensional vectors, whose first four values are the coordinates of the block in the original inputs, the fifth value is the probability of face classification, and the last four are the bounding box regression vector.

NMS:

Then I do the non-maximum suppression (NMS) under different scale and bounding box regression calibration. NMS is easy to operate using IoU, and I do the bounding box regression calibration by adding the offset (height, width) to the coordinate of the left top point (x, y) to get the calibrated bounding box regression vector. Then I adjust the the bounding box into a square and cutting off the redundant parts. Finally, I can get the coordinates of bounding boxes.

R-Net:

The input of R-Net are all the bounding boxes generating in the P-Net. I resize them into 24x24 at first, then get rid of those whose probability of face classification < 0.7, and then refine the remaining bounding boxes with NMS and calibration. Here with a fully-connected layer, the output of this network are 16-dimensional vectors, including 4-dimensional bounding box regression vectors and 2-dimensional face classification, still no landmarks.

O-Net:

The input of O-Net are the bounding boxes from R-Net. I resize them into 48x48, then do the final NMS and bounding box calibration, outputting 16-dimensional vectors, which are 2-dimensional face classification, 4-dimensional bounding box regression and 10-dimensional facial landmark localization. Here the 10-dimensional landmark values are the offsets of bounding boxes' width and height, the first five for x last five for y.

B. FaceNet

The method of FaceNet is based on learning a Euclidean embedding per image with only 128-bytes using a deep convolutional network. Once the embedding is achieved, tasks like face verification, recognition and clustering can be done easily.

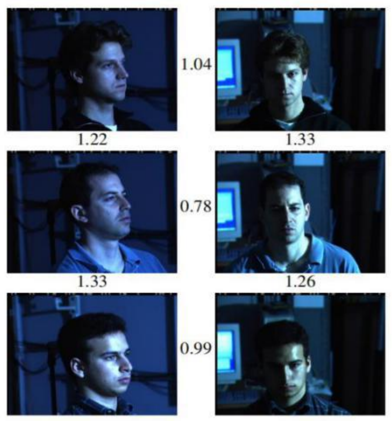


Fig. 3.8. Distance between face images with illumination and pose differences

The two most important parts to mention here are the Model Structure and the Triplet Loss.

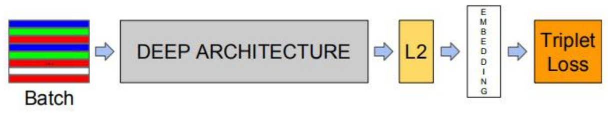
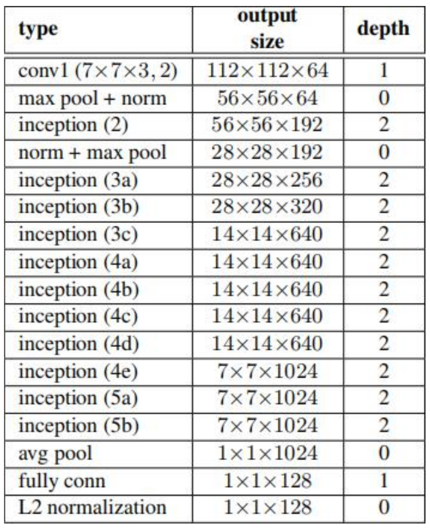


Fig. 3.9. Model structure

The model structure of FaceNet is shown in Figure. 3.9, the input contains a batch of images, then image matrix pass through the deep neural network architecture which was inspired by Zeiler&Fergus [12] and zegedy et al.'s [13] GoogLeNet style Inception models, and also the network has a fully connected layer as the next to last layer with the output size 1*1*128. Thirdly, an L2 normalization is implemented. After these three process, I can get the face embedding with 128-dimension. Next, this is followed by the triplet loss during training.



type	output size	depth
conv1 (7×7×3, 2)	112×112×64	1
max pool + norm	56×56×64	0
inception (2)	56×56×192	2
norm + max pool	28×28×192	0
inception (3a)	28×28×256	2
inception (3b)	28×28×320	2
inception (3c)	14×14×640	2
inception (4a)	14×14×640	2
inception (4b)	14×14×640	2
inception (4c)	14×14×640	2
inception (4d)	14×14×640	2
inception (4e)	7×7×1024	2
inception (5a)	7×7×1024	2
inception (5b)	7×7×1024	2
avg pool	1×1×1024	0
fully conn	1×1×128	1
L2 normalization	1×1×128	0

Fig. 3.10. Details of part of the inception structure wit the input size 224*224

Considering the triplet loss function, it is a loss function which is more suitable for face verification since the triplet loss tries to enforce a margin between each pairs of faces from one person to all other faces. As shown in Figure. 3.11, the triplet loss has three elements: Anchor, Positive, Negative. 'Anchor' and 'Positive' can be viewed as pictures from the same person, and 'Anchor' and 'Negative' can be viewed as pictures from different person. The aim of training is to minimize the distance between Anchor and Positive, and maximum the distance between Anchor and Negative.

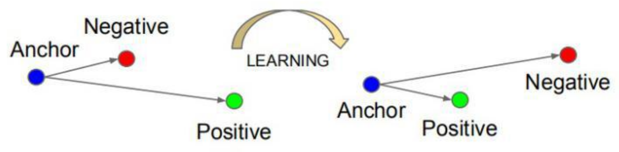


Fig. 3.11. Triplet loss training

**Picture source of Fig.3.8~3.11: Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering.*

For more details, here I use an equation to denote the process:

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \qquad (3\text{-}1)$$

$$\forall\, (f(x_i^a), f(x_i^p), f(x_i^n)) \in \mathcal{T} \qquad (3\text{-}2)$$

Here α represents the margin between positive and negative pairs, Γ represents the aggregate of all possible triplets.

Then I can get the loss function which is needed to be minimized in the following training:

$$L = \sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+ \qquad (3\text{-}3)$$

However, for the training, if I use one picture in the training set as the anchor and compare all the remaining pictures in the training set in sequence to find the distance between positive and negative pairs, and then repeat this process to all pictures in the set, the requirement of computation will be huge. Moreover, generating all possible triplets will lead to one situation: a lot of triplets satisfy the equation (3-1) easily, that means this kind of triplets only offer tiny contributes of the convergence of the triplet loss and make the training effect become bad, thus one task to solve is trying to find hard triplets. For the hard triplets mining, I have two ways:

1) Generate triplets offline every n steps, using the most recent network checkpoint and do the computation on a subset of data. [11]
2) Generate triplets online. This can be done by selecting the hard positive/negative exemplars from within a mini-batch.

Authors of FaceNet focus a novel online triplet mining method. In their experiments, firstly, they choose around 40 images each identity per mini-batch to build the training set;

secondly, they put some randomly selected negative face images into each mini-batch. In addition, instead of picking the hardest positive, they use all anchor-positive pairs in a mini-batch while still selecting the hard negative since anchor-positive method was found to have a slightly faster convergence speed. [11]

IV. APPLICATION

The working progress of the face recognition application is shown in Fig.4.1. The team of FaceNet offers two official pre-trained model online. One is 20170511-185253, another is 20170512-110547. Both of these two models using the Inception RexNet v1 architecture. The 20170511-185253 model was trained on CASIA-WebFace, and it has a 98.7% accuracy performance on LFW database. The 20170512-110547 model was trained on MS-Celeb-1M, and it has a 99.2% accuracy performance on LFW database. In our application, i use the model 20170512-110547.

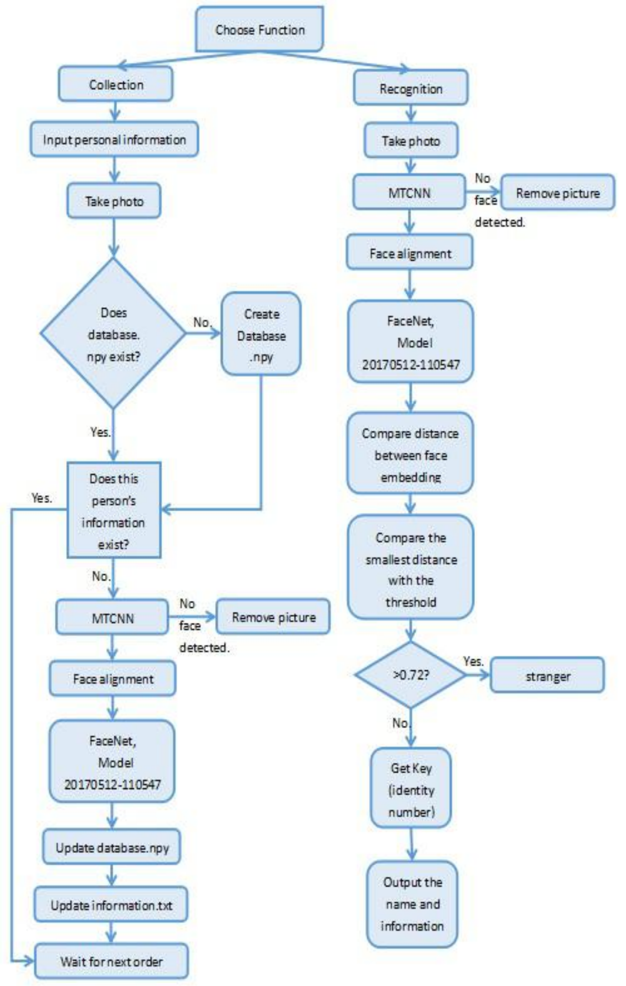

Choose Function
Collection
Input personal information
Take photo
Does database. npy exist?
No.
Create Database .npy
Yes.
Does this person's information exist?
Yes.
No.
MTCNN
No face detected.
Remove picture
Face alignment
FaceNet, Model 20170512-110547
Update database.npy
Update information.txt
Wait for next order
Recognition
Take photo
MTCNN
No face detected.
Remove picture
Face alignment
FaceNet, Model 20170512-110547
Compare distance between face embedding
Compare the smallest distance with the threshold
>0.72?
Yes.
stranger
No.
Get Key (identity number)
Output the name and information

Fig. 4.1. Working progress flow chart

The application contains two functions: collection and recognition.

For the collections function, firstly, the personal information including last name, first name, birthday, student number will be input. Then, the program will start to take photo of the person, at the meantime, instructions asking the person to look straight, turn right, turn left,etc will be given. Thirdly, there is some code to check whether the 'database file' exists or not, if not, the file database,npy will be created. Otherwise, the program will check whether this person's face information is already in the database.npy. If not, it will start to process the face images which have been saved in a temporary folder. During the process, I use MTCNN and FaceNet. In the first step, images encounters face detection failure will be removed. Secondly, face alignment will be done by the MTCNN, face in each image will be cut and stored in a new folder named resize_anchor. Thirdly, all images in the resize-anchor folder will be treated as the input to the FaceNet network (here I import the pre-trained model 20170512-110547.pb) to get the 128-dimension face embedding, Then, all data of face embedding will be written in the database.npy and personal information will be written in information.txt with the form of dictionary.

For the recognition part, I will take one picture of the person. Then process the face image with the MTCNN to get face area image and with FaceNet network to get the 128 dimension face embedding. Next, I compare the distance between the test embedding with all embeddings in the database and sort all distances from small to large. For the smallest distance, it will be compared with the threshold 0.72. If larger, the outcome 'stranger' will be output, otherwise, I will use the top ten numbers and letters of the smallest-distance image's name as a key to find the corresponding value which refers to personal information in the dictionary named 'information' and output it as the result.

V. EXPERIMENTAL RESULTS AND EVALUATIONS

All face images used in experiments are from CASIA-Webface data set and CMU Face Images data set. In the experimental part I do five different tests to evaluate the performances of our application in the real world environment, to see the robustness of the recognition system when there are brightness changes, expression changes and face incompleteness. In addition, I also record the program's running time.

A. Test with a Batch of Images

In this test, I use part of the CASIA-Webface. The total identities are 200 person, and for each person, I randomly pick 7 face images(250*250) to build our database(ex.Fig.5.1 (1)~(7)). In the recognition part, I take one image which is different from images in database for each of the 200 identity from the corresponding person's file folder in the CASIA-Webface(ex.Fig.5.1 (8)). The purpose of this test is to evaluate the accuracy and speed of our face recognition application under the circumstance which is close to real daily life scene.

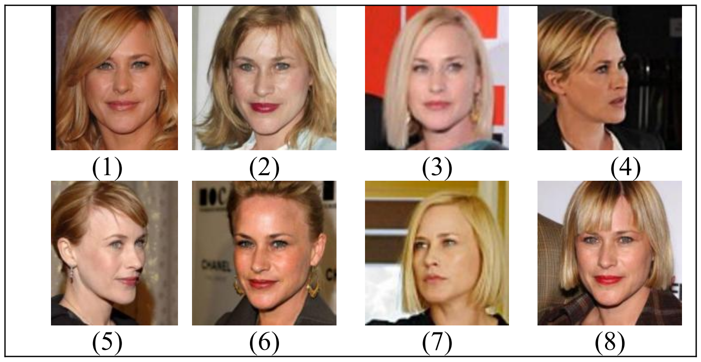


Fig. 5.1. Examples for facial images in the database and the one used to test

Since our 'the same or not the same judgement' is decided by the distance-threshold, I set two different threshold to verify the effect.

- Threshold =0.7

After 200 times recognition, the result shows 21 test images were mistakenly recognized as stranger and 1 image was recognized incorrectly. Thus the recognition accuracy is **89%.**

- Threshold =0.72

I use the same database and test pictures to run the code again, I set the threshold as 0.72. This time, I have one incorrect recognition which is the same as the mistake mentioned above, but I have no incorrect 'stranger' judgement this time, thus the accuracy is **99.5%**.

Some of the test results are shown below.

1) Incorrect recognition:

picture:350001.jpg(test image)
recognized as identity:533004.jpg,
distance:0.6748525018163373

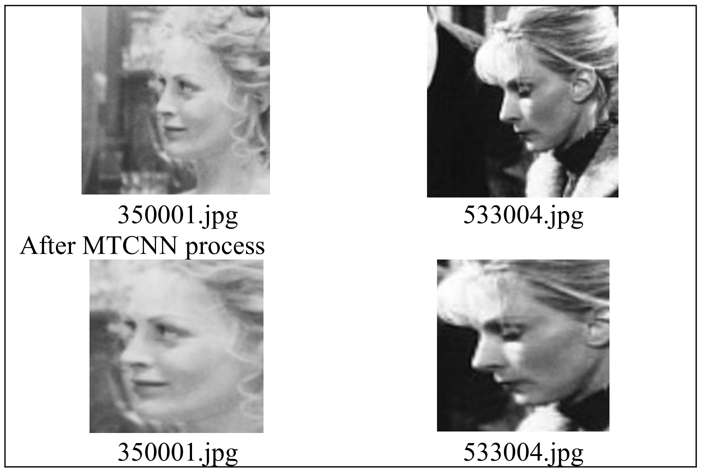


Fig. 5.2. Incorrect recognition image pair

2) Image pair with the smallest distance:

picture:346001.jpg(test image)
recognized as identity:346008.jpg,
distance:0.3408540498521112

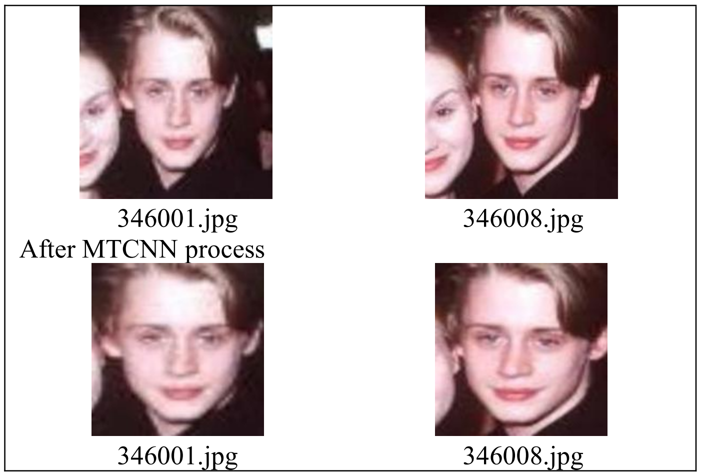


Fig. 5.3. Image pair with the smallest distance

3) Image pair with the largest distance:

picture:296001.jpg(test image),
recognized as identity:296008.jpg,
distance:0.7193053995039862

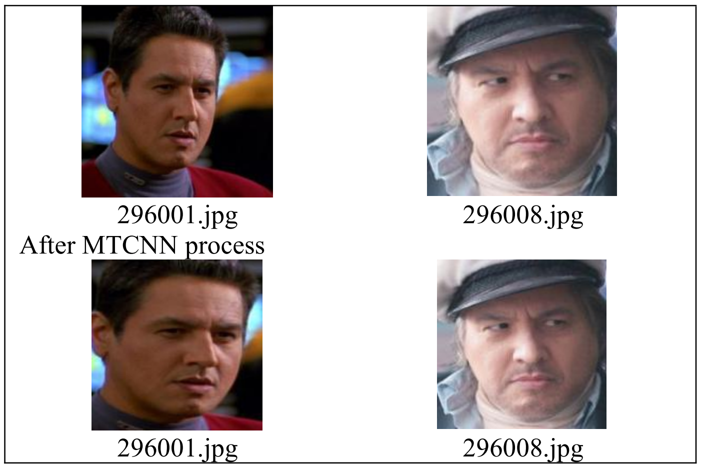


Fig. 5.4. Image pair with the largest distance

B. Recognition When Image Brightness Changes

In this test, I still use part of the CASIA-Webface. The total identities are 100 person this time, and for each person, I randomly pick 7 face images to build our database. In the recognition part, I take one image which is different from images in database for each of the 100 identity from the corresponding person's file folder in the CASIA-Webface, and change the picture brightness as 25%, 50%, 75%, 125%, 150%, 175% compared to the origin pictures respectively. Recognition threshold is set as 0.72. The purpose of this test is to evaluate the robust of the recognition system when there are light condition changes.

- 25% brightness

When the brightness of test images changes to 25%, in the 100 test images, I have 8 images in total in which human face can't be found. And 21 images are mistakenly recognized as 'stranger'. So the accuracy drops to **71%**.

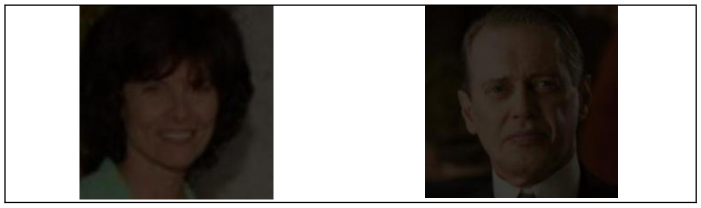


Fig .5.5. Example of test images with 25% brightness

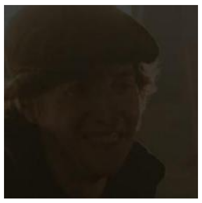


Fig. 5.6. Example of 'can't find face' test images with 25% brightness

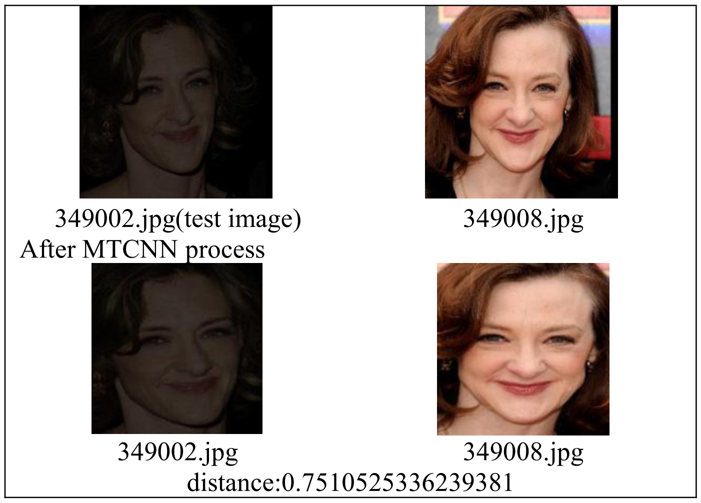


Fig. 5.7. Example of image pair which has been mistakenly recognized as 'stranger' with 25% brightness

- 50% brightness

When the brightness of test images changes to 50%, in the 100 test images, I have 3 images in total in which human face can't be found. And 13 images are mistakenly recognized as 'stranger'. So the accuracy is **84%**.

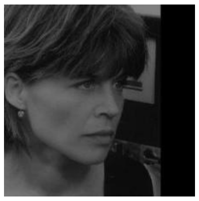


Fig. 5.8. Example of 'can't find face' test images with 50% brightness

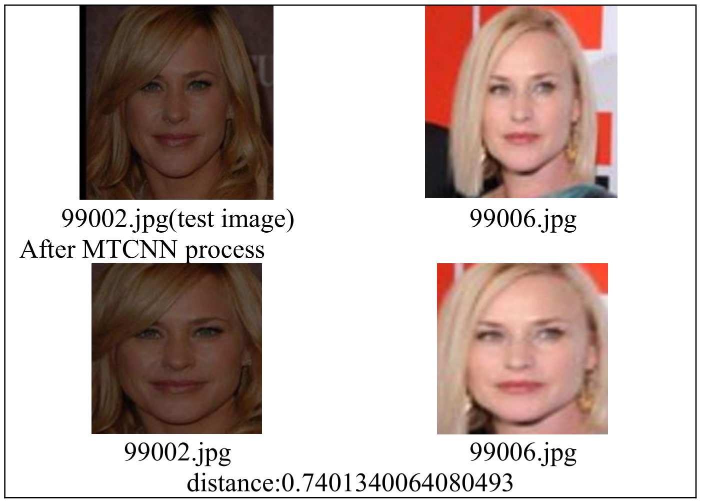


Fig. 5.9. Example of image pair which has been mistakenly recognized as 'stranger' with 50% brightness

- 75% brightness

When the brightness of test images changes to 75%, in the 100 test images, I have 1 image in which human face can't be found. And 4 images are mistakenly recognized as 'stranger'. So the accuracy is **95%**.

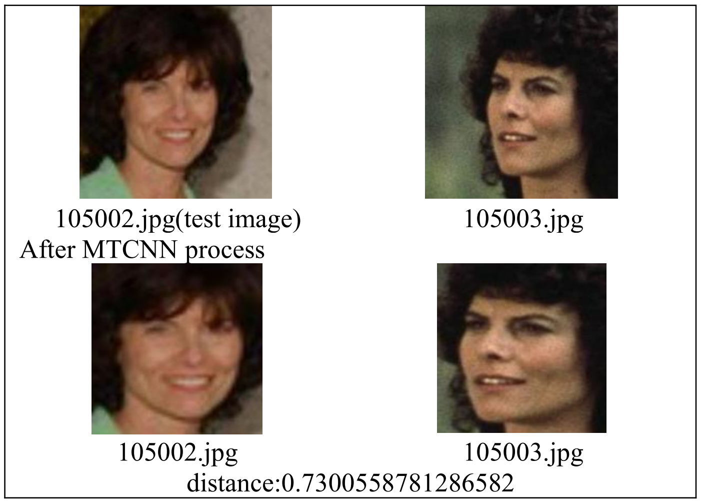


Fig. 5.10. Example of image pair which has been mistakenly recognized as 'stranger' with 75% brightness

- 125% brightness

When the brightness of test images changes to 125%, in the 100 test images, I have 1 image in which human face can't be found. And 8 images are mistakenly recognized as 'stranger'. Thus the accuracy is **91%**.

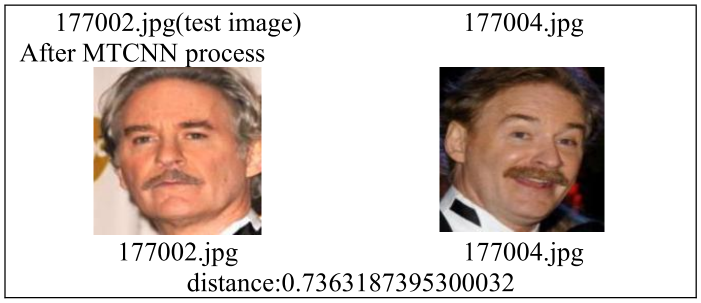


Fig. 5.11. Example of image pair which has been mistakenly recognized as 'stranger' with 125% brightness

- 150% brightness

When the brightness of test images changes to 150%, in the 100 test images, I have 5 images in total in which human face can't be found. And 20 images are mistakenly recognized as 'stranger'. So the accuracy is **75%**.

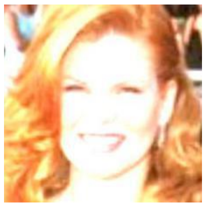


Fig. 5.12. Example of 'can't find face' test images with 150% brightness

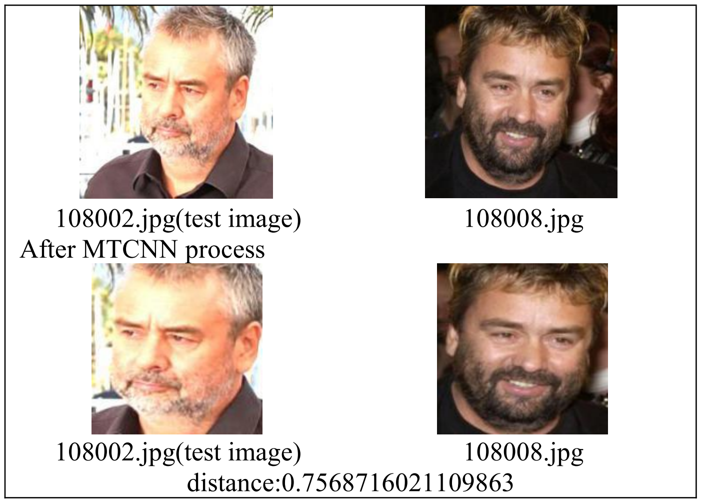


Fig. 5.13. Example of image pair which has been mistakenly recognized as 'stranger' with 150% brightness

- 175% brightness

When the brightness of test images changes to 175%, in the 100 test images, I have 13 images in total in which human face can't be found. And 33 images are mistakenly recognized as 'stranger'. So the accuracy drops to **54%**.

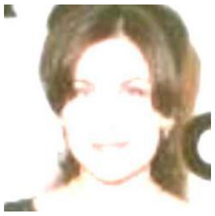


Fig. 5.14. Example of 'can't find face' test images with 175% brightness

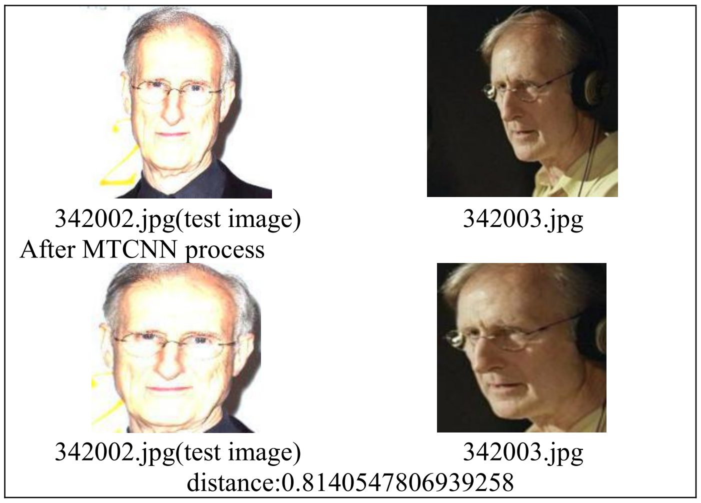


Fig. 5.15. Example of image pair which has been mistakenly recognized as 'stranger' with 175% brightness

Summary:

Brightness level	Accuracy
25%	71%
50%	84%
75%	95%
125%	91%
150%	75%
175%	54%

Table.5.1

In summary, 50% and 150% brightness-level test shows obvious drop in recognition accuracy. Performances of our application in 75% and 125% brightness-level condition are still not bad. The application shows more robust in dark condition.

C. Recognition When Expression Changes

In the third test, I use CMU face images data set. The total identities are 20 person in this test , and for each person, i choose four open-eyes face images(128*120) with four different expressions and four different postures (16 images for each identity)to build our database. In the recognition part, the threshold is set as 0.72. I use the expression of happy and neutral of each identity to make comparison with three other expressions in the database respectively. The purpose of this test is to evaluate whether different expressions will bring obvious influences to the recognition accuracy.

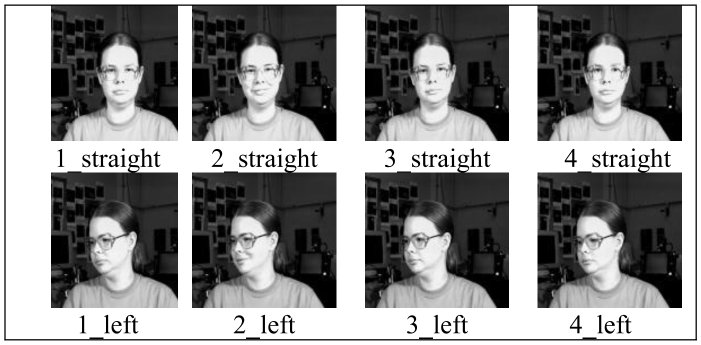


Fig. 5.16. CMU face images examples

When processing the 320 images database, 39 images have been removed for the reason of face detection failure.

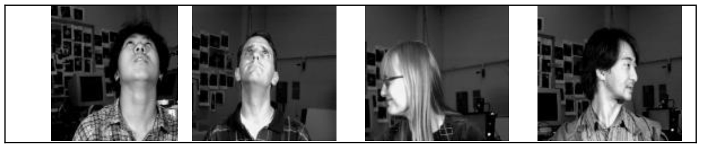
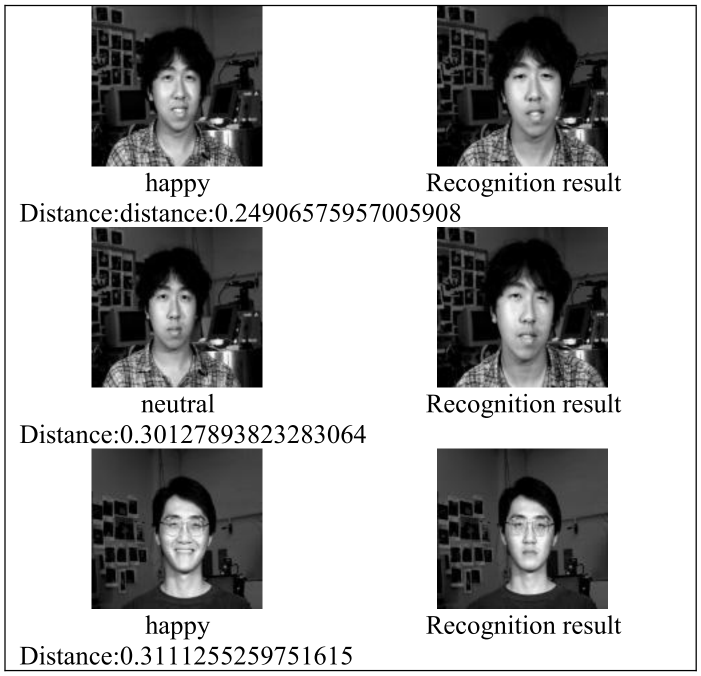


Fig. 5.17. Examples of images with face detection failure

But the accuracy of recognition are 100% for both of the happy and neutral expressions and there is no obvious differences on the range and distributions of embedding distance. Distance between happy expression test images and face images in database ranks from 0.249065~0.429170. Distance between neutral expression test images and face images in database ranks from 0.241889~0.435104. Expression changes do not bring apparent influences on recognition result.


happy
Recognition result
Distance:distance:0.24906575957005908
neutral
Recognition result
Distance:0.30127893823283064
happy
Recognition result
Distance:0.3111255259751615

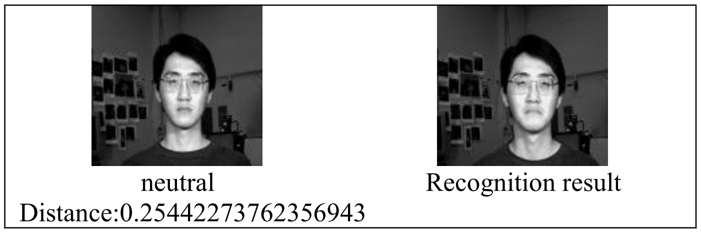


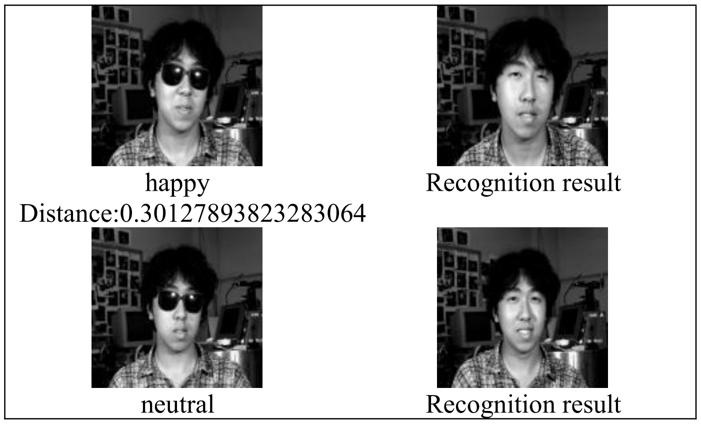
Fig. 5.18. Examples of face recognition results using images with different expressions

**all distance here is the distance between images after face alignment process*

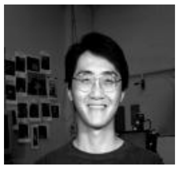
One thing to mention here is the high possibility of MTCNN alignment failure on the CMU face images (grey-level image). 67% of the failures happen on images with up posture, the remaining 33% failures happen on turning head postures. That is because the parameters i use to rebuild the MTCNN model are trained in CASIA-Webface data set which are 3-channel RGB images while the CMU data set are 1-channel grey images. Apparently, the parameters are not able to deal with images having such different features.

D. Recognition When Covering Part of Faces

In the fourth test, I use the same database as test C. In the recognition part, the threshold is set as 0.72. I use the expression of happy and neutral of each identity to make comparison with three other expressions in the database respectively, however, this time people in all test image are wearing glasses and this results in face-covering effect. The purpose of this test is to see the recognition accuracy when part of faces are covered. For the CMU images data set, the recognition accuracy remains 100%. Distance between happy expression test images with face covering effect and face images in database ranks from :0.241889~0.435234. Distance between neutral expression test images with face covering effect and face images in database ranks from 0.254422~0.439925. Since the distance ranges are similar to those in test C and the accuracy remains 100%, here I think that covering the eyes part, although may bring some negative influences on facial landmark searching, it does not bring apparent changes in recognition result.

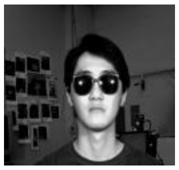
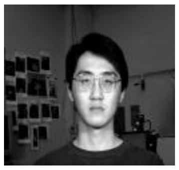


Distance:0.24906575957005908



happy Recognition result

Distance:0.3111253823283064

neutral Recognition result

distance:0.25442273762356943533

Fig. 5.19. Examples of face recognition results using images with face-covering effect

**all distance here is the distance between images after face alignment process*

E. PROGRAMME RUNNING TIME

In test one: TEST WITH A BATCH OF IMAGE, I record the running time of image processing and face recognition: A).The time used to process 1400 face images in the database is **669.872778** seconds, specific processes including: import the FaceNet model once, using a looping method to do face alignment on the 1400 images, store the resized image, applying FaceNet on each image and store every embedding in database.npy in sequence. B).The time used to do 200 times face recognition is **123.641737** seconds. Since in this test I put the 200 test image in the same test folder, specific processes including: import the FaceNet model once, using a looping method to do face alignment on the 200 images, store the resized image, applying FaceNet on each image and compare each test embedding with all 1400 embedding data in database.npy (repeat 200 times).

In addition, I also record C).the time of single image recognition based on the 1400 images database(embedding data for the 1400 image has already been stored). Time to process the single test image is **4.130753** seconds**,** specific processes including: import the FaceNet model once, do face alignment on the single test image, store the resized test image, applying FaceNet on the resized image to get the embedding. And time used to compare the single test embedding with all 1400 embedding data in database.npy is **0.145276** seconds**,** thus the total time is about 4.3 seconds.

VI. CONCLUSION

In this report, I have proposed a practical face recognition application based on MTCNN and FaceNet. Experimental results demonstrate that our application has a quite good performance in accuracy and runtime. Since I have done lots of tests to simulate the real world environment, the robustness can also be guaranteed.

As the first project I finished in combining deep learning with computer vision, I have benefited a lot. I learned not only the knowledge in face detection and recognition, but also the ideas and how to take advantage of them in our real life.

REFERENCES

[1] Jain, Anil K., and Stan Z. Li. Handbook of face recognition. New York: springer, 2011.

[2] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, “Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks,” Senior Member, IEEE, and Yu Qiao, Senior Member, IEEE.

[3] P. Viola and M. J. Jones, “Robust real-time face detection. International journal of computer vision,” vol. 57, no. 2, pp. 137-154, 2004.

[4] S. Yang, P. Luo, C. C. Loy, and X. Tang, “From facial parts responses to face detection: A deep learning approach,” in IEEE International Conference on Computer Vision, 2015, pp. 3676-3684.

[5] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, “A convolutional neural network cascade for face detection,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5325-5334.

[6] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, “Facial landmark detection by deep multi-task learning,” in European Conference on Computer Vision, 2014, pp. 94-108.

[7] D. Chen, S. Ren, Y. Ii, X. Cao, and J. Sun, “Joint cascade face detection and alignment,” in European Conference on Computer Vision, 2014, pp. 109-122.

[8] C. Zhang, and Z. Zhang, “Improving multiview face detection with multi-task deep convolutional neural networks,” IEEE Winter Conference on Applications of Computer Vision, 2014, pp. 1036-1041.

[9] Luo, Y., Zhang, T. and Zhang, Y., 2016. A novel fusion method of PCA and LDP for facial expression feature extraction. Optik-International Journal for Light and Electron Optics, 127(2), pp.718-721.

[10] Zhu, Z., Luo, P., Wang, X. and Tang, X., 2014. Recover canonical-view faces in the wild with deep neural networks. arXiv preprint arXiv:1404.3543.

[11] Schroff F, Kalenichenko D, Philbin J. Facenet: A unified embedding for face recognition and clustering. InProceedings of the IEEE conference on computer vision and pattern recognition 2015 (pp. 815-823).

[12] Zeiler MD, Fergus R. Visualizing and understanding convolutional networks. InEuropean conference on computer vision 2014 Sep 6 (pp. 818-833). Springer, Cham.

[13] Szegedy C, Liu W, Jia Y, Sermanet P, Reed S, Anguelov D. & Rabinovich, A.(2015). Going deeper with convolutions. InProceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).