华南理工大学

South China University of Technology

# The Experiment Report of *Machine Learning*

**SCHOOL:** SCHOOL OF SOFTWARE ENGINEERING

**SUBJECT:** SOFTWARE ENGINEERING


*Author:*
Xiujun Zhou
Jieqiong Liu

*Supervisor:*
Mingkui Tan

*Student ID:*
201830664237
201830661465

*Grade:*
Undergraduate


November 30, 2020

# Face Detection Based on Neural Network

***Abstract*—In this experiment we used Multi-task cascaded convolutional neural network, namely, P-Net, R-Net and O-Net to complete the task of face detection. We read the code to understand the basic theory of face detection using neural network and the training process of MTCNN.**

## I. Introduction

FACE detection is one of classical problems in computer vision and it is challenging due to different poses, illuminations and occlusions. Recently, due to the fast iterating of deep learning methods, many face detection solutions use convolutional neural network to deal with these problems well. Due to its excellent ability to extract features and the satisfying results in most cases, MTCNN [1] is one of the most popular architectures which has been applied to a wide range of practical use. Followed by AdaBoost algorithm for face detection, in this experiment, we used MTCNN as a neural network solution for the same task.

## II. Methods and Theory

### A. *Introduction of MTCNN*

MTCNN (Multi-Task Cascaded Convolutional Neural Network) combined the task of face alignment and face detection together using a unified structure, it can be divided into 3 main stages: Proposal Network (P-Net), Refinement Network (R-Net) and Output Network (O-Net).

In the first stage, P-Net produced candidate windows, then on the second stage, R-Net refined the candidates. Finally in the third stage, O-Net produces the bounding box which contains faces that have been detected. The 3 stages are illustrated as the picture below, which is taken from original paper[1].

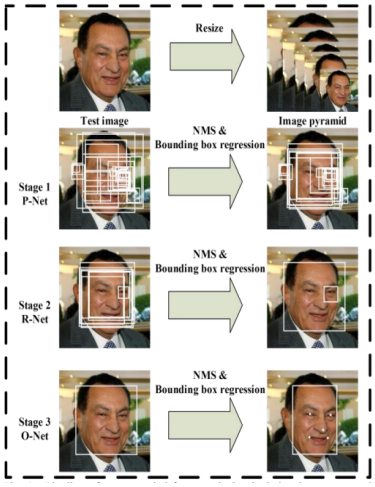


Fig. 1: MTCNN's 3 stages

*1) P-Net:*

The first stage, the original image is shrunk to different scales, and then the images of different scales are input to P-Net, in order to detect faces of different sizes and realize multi-scale target detection, P-Net outputs the predicted face probability and the predicted border coordinate offset of each small square in the convolutional layer of the last layer.

The predicted face probability can be 1 or 0. Thus, the learning objective is formulated as a two-class classification. We use cross-entropy loss as our loss function: Cross-Entropy Loss is :

$$H_y(y) := -\Sigma_i y_i' log(y_i) \tag{1}$$

The loss function in P-Net is:

$$L_i^{det} = -(y_i^{det} log(p_i) + (1 - y_i^{det})(1 - log(p_i))) \tag{2}$$

$P_i$ is the output of the network, which represents the probability of a picture being a human face. $Y_i^{det}$ means the ground-truth label.

With the data obtained above as input, a small square with a probability greater than the threshold is first extracted for preliminary filtering. The square containing the humnan face can be regardes as a candidate, our next work is to find the most suitable candidates under the current situation. Non-maximal suppression (NMS) was performed for merging highly overlapped candidates. NMS is the ratio of the area of the face in the image to the actual area of the face frame (IOU).

The image obtained by NMS is then returned by bounding box regression, which compares the position of the face frame to be tested with the actual face frame, and adjusts the position of the face frame to make it close to the position of the real face frame (the alignment), thus generating boxes and boxes alignment.

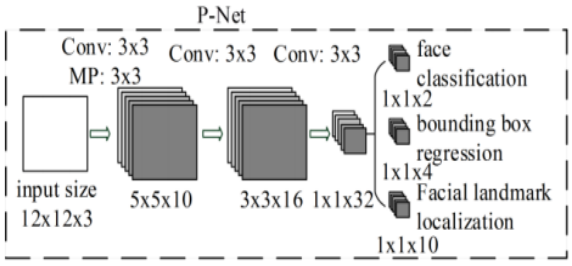


Fig. 2: Structure of P-Net

*2) R-Net:*

R-Net is a further filter for face frames and adjusts the border position.

In the second stage, the candidate boxes aligned by P-Net network is taken as the input of R-Net, which outputs the

face score and border coordinate offset of each image (P-Net's output is the score of a certain area of the image, while R-Net's output is the score of the whole image). Next, in order to select better candidates, it excludes images whose scores are smaller than the threshold. It also use NMS non-maximal inhibitory to calibrate.

Bounding box regression is the last step to predict the offset between the candidate window and its nearest ground truth. MTCNN models this task as a regression problem, with Euclidean loss for each sample $x_i$:

$$l_i^{box} = ||\hat{y_i^{box}} - y_i^{box}||_2^2 \tag{3}$$

$\hat{Y_i^{box}}$ means the regression target and $y_i^{box}$ means the ground-truth coordinate. Both of them are $R^4$, for the fact that there are 4 coordinates (left top, right top, height, width). After that we can get boxes and boxes_align.

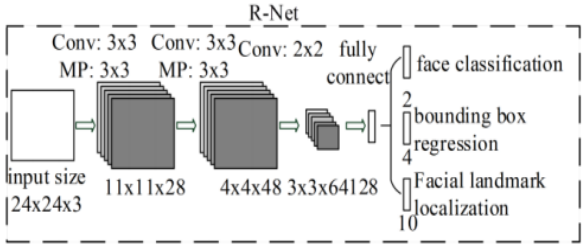


Fig. 3: Structure of R-Net

*3) O-Net:*
The third stage is similar to the basic flow of the second stage. It is modeled as a regression task as well as box regression.But its outputs include the prediction and position adjustment of 5 key points of faces.

We express the Euclidean Loss in a similar way:

$$l_i^{landmark} = ||\hat{y_i^{landmark}} - y_i^{landmark}||_2^2 \tag{4}$$

$\hat{Y_i^{landmark}}$ is the facial landmark's coordinate from network and $y_i^{landmark}$ is the ground truth. At last, it returns boxes_align and landmark_align.

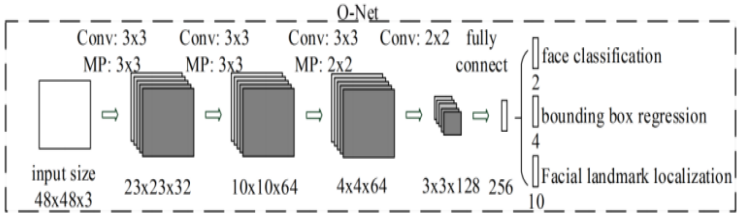


Fig. 4: Structure of O-Net

*4) Multi-training Loss:*
Since the author apply 3 different networks in total, it is necessary to find a general loss function for the architecture. MTCNN neglects some of loss functions in specific situation by a variable named type indicator $\beta \in 0, 1$. The overall learning target is represented as:

$$min\Sigma_{i=1}^{N}\Sigma_{j\in det,box,landmark}\alpha_j\beta_i^j L_i^j \tag{5}$$

Where $\alpha$ and $\beta$ are set different values in order to make each network work the best.$\beta_i^j \in 0, 1 is the sample type indicator$. In this case, we use stochastic gradient descent (SGD) to train CNNs. The detailed parameter configuration is shown as follows:

TABLE I: Table parameter setting for MTCNN

| Network | $\alpha_{det}$ | $\alpha_{box}$ | $\alpha_{landmark}$ |
|---|---|---|---|
| P-Net | 1 | 0.5 | 0.5 |
| R-Net | 1 | 0.5 | 0.5 |
| O-Net | 1 | 0.5 | 1 |

## III. EXPERIMENTS

### A. *Dataset and Environment*

We used WiderFace as our training set for P-Net and R-Net when dealing with the face classification and face bounding box regression during P-Net, R-Net and O-Net stages. Besides, we used training set from [2] for face feature regression during the O-Net stage.

We trained the model with Intel(R) Core(TM) i7-8550U CPU, 1.80GHz. Due to the limitation of conditions, we didn't use GPUs to train our model.

### B. *Model Training*

*1) Preparations:* After we got the source code of MTCNN, we run test_image.py to test the correctness of our network configuration. The process is shown as follows:

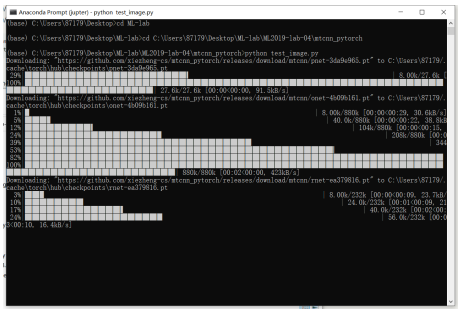

Fig. 5: Data loading

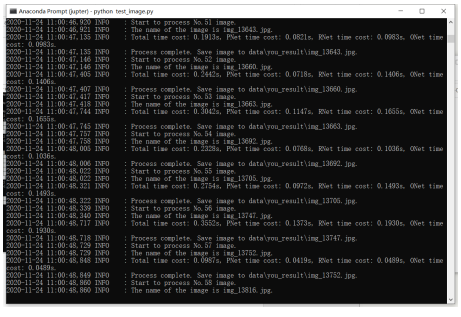

Fig. 6: Output during testing

Finally, we got the results of the pre-trained model. From the figure we can find that MTCNN has a satisfying output on the samll testing set.

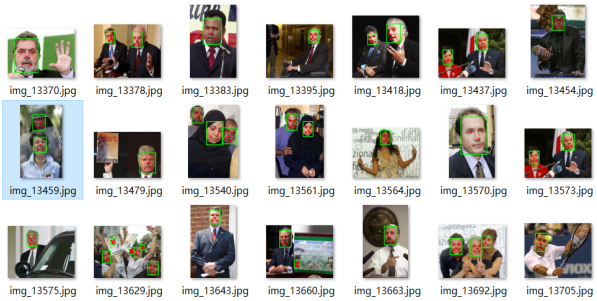


Fig. 7: Final output results

‘

*2) Training:* Additionally, to optimize the performance of training, MTCNN employs a strategy which ignore the easy samples to improve the performance. The inputs and outputs of each network are listed as follows.

Listing 1: **P-Net inputs and outputs.py**

```
def detect_pnet(self, im):
    Get face candidates through pnet
    Parameters:
    ----------
    im: numpy array, input image array

    Returns:
    -------
    boxes: numpy array
    detected boxes before calibration
    boxes_align: numpy array
    boxes after calibration
```

Listing 2: **R-Net inputs and outputs.py**

```
def detect_rnet(self, im, dets):
    Get face candidates using rnet
    Parameters:
    ----------
    im: numpy array
        input image array
        dets: numpy array
        detection results of pnet
    Returns:
    -------
    boxes: numpy array
        detected boxes before calibration
        boxes_align: numpy array
        boxes after calibration with
        whose scores are smaller than threshold
            are discarded
```

Listing 3: **O-Net inputs and outputs.py**

```
def detect_onet(self, im, dets):
    Get face candidates using onet

    Parameters:
    ----------
    im: numpy array
        input image array
    dets: numpy array
        detection results of rnet
    Returns:
    -------
    boxes_align: numpy array
        boxes after calibration
    landmarks_align: numpy array
        landmarks after calibration
```

### C. *Results*

After finishing the training, there are respectively 50 models of the three networks, P-Net network, R-Net network, O-Net network.

To test the results of the three nets, we use a function called detect_face(). This function read a image, At first, it calls another function detect_pnet(), which uses image as input to get boxes and boxes_align through pnet; the second, it calls function detect_rnet(), which uses image and boxes_align from the first step as inputs to get another boxes and boxes_align through rnet; the third, it calls function detect_onet(), which uses image and boxes_align from the second step as inputs to get boxes_align and landmark_align through onet. At last, it returns boxes_align, landmark_align. Now, we can use the return object to draw the images to get a face with the landmarks in a box.

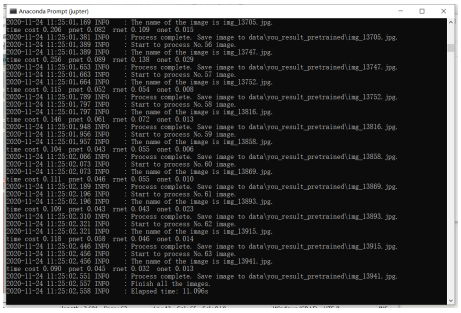

Fig. 8: Training process

We find that when the faces are distinct, the results are correct. In most instances, the results of the detection are correct. And as the MTCNN is able to exploit the inherent correlation between face detection and face alignment, it performs well in both tasks. From the following figure we can find that even with a face which is similar to its surrounding environment, the network is able to detect it with correct alignment.

However, with disturbing factors like small or misty faces, somber environment, the reflective glasses or even something

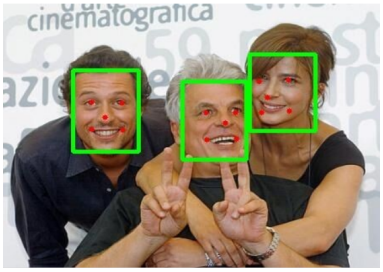


Fig. 9: Correct detection

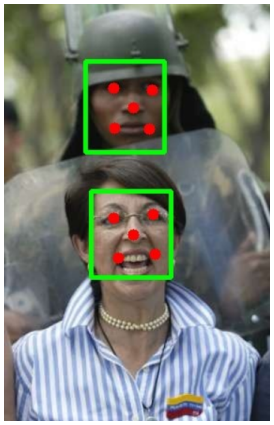

Fig. 10: Correct detection

similar to eyes or other landmarks, the results have some deviations or errors. One of typical incorrect outputs are as follows. In the figure, the ear of a children is classified to human faces incorrectly.

Fig. 11: Incorrect detection

Listing 4: **detect.py**

```
def detect_face(self, img):
    ''' Detect face over image '''
    boxes_align = np.array([])
    landmark_align = np.array([])
    t = time.time()
    # pnet
    if self.pnet_detector:
        boxes, boxes_align = self.detect_pnet(img)
        if boxes_align is None:
            return np.array([]), np.array([])
        t1 = time.time() - t
        t = time.time()
    # rnet
    if self.rnet_detector:
        boxes, boxes_align = self.detect_rnet(img,
            boxes_align)
        if boxes_align is None:
            return np.array([]), np.array([])
        t2 = time.time() - t
        t = time.time()
    # onet
    if self.onet_detector:
        boxes_align, landmark_align = self.
            detect_onet(img, boxes_align)
        if boxes_align is None:
            return np.array([]), np.array([])
        t3 = time.time() - t
        t = time.time()
        print("time cost " + '{:.3f}'.
            format(t1 + t2 + t3) + '  pnet {:.3f}
             rnet {:.3f}  onet {:.3f}'.
            format(t1, t2))
return boxes_align, landmark_align
```

## IV. Conclusion

In this experiment, we understand the basic theory of face detection using neural network and the processes of a multi-task cascaded convolutional networks.

We have a better understanding of how 3 different CNNs cooperate and interact with each other to give a final output. What's more, when reading papers, we found something interesting about the overall loss function and a novel training strategy which ignores less important samples. This is an improvement in order to strengthen the detector, which from our perspective has something similar to the method used by AdaBoost, both of which changes the weight of training data points. But the main difference is that AdaBoost complete this operation in an iterative manner while this strategy is simpler.

Additionally, we use it in practice. We find the factors of face detection, including illumination, the number of faces, the definition of faces, the angle of faces or something similar to faces in the environment. Moreover, the threshold value of the network also influences the final result. All these factors will result in a deviation or error when mark the landmark and draw the face's position.

Early face detection methods used artificial feature extraction, training classifier, face detection. For example, the OpenCV source code of the face detector is the use of HAAR features for face detection. The disadvantage of this method is that the detection effect is not ideal when the environment changes strongly, such as low light condition and incomplete face.

There are other face detection algorithms inherited from general target detection algorithms. For example, FASTER-RCNN is used to detect human face, whose results are

satisfying and is able to adapt to environmental changes and incomplete faces. However, its time consumption is high. MCTNN is a cascade structure of a convolutional neural network, it outperforms other state-of-art networks in both time consumption and performance.

However, we also find some of its disadvantages. Firstly, the detection speed of MTCNN has a great relationship with the number of faces to be detected. When the number of faces increases, both the detection time and deviation or error rate increase significantly. Secondly, the time consumption is unacceptable when we are only equipped with CPU or a low performance GPU. The third is the angle of the faces or something on the face like hair can influence it, when face alignment is carried out on the side face, the position of feature points is not accurate. These disadvantages above are remained to solved.

## V. REFERENCES

[1] Zhang K, Zhang Z, Li Z, et al. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks[J]. IEEE Signal Processing Letters, 2016, 23(10):1499-1503.

[2] Sun Y, Wang X,Tang X.Deep Convolutional Network Cascade for Facial Point Detection[C]. Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013.