Simple Face Detection, Classification and Alignment

Part I: Face Detection

Over last ten years, face recognition has become a hot topic in computer vision, which has been widely applied in unlock phone(iPhoneX), security camera, some digital make-up application(Meitu) and can be compared to other biometrics such as fingerprint or eye iris recognition system.

A general statement of the face recognition problem can be formulated as follows: Given still or video images of a scene, identify or verify one or more persons in the scene using a stored database of faces.

Originally, I was intended to implement machine learning algorithm on detection of clinical prostate tissue, then I realized it would be difficult for a starter. I started with the face recognition project which has been well-studied with oceans of open sources and tutorial that I can learn and follow., and I would like to get familiar with OpenCv library, which will benefit the further study.

OpenCV stands for open source computer vision, a library of programing functions mainly aimed at real-time computer vision. OpenCV uses Haar feature-based cascade classifiers to detect object. Initially, the algorithm extracts features from the input image. Haar features(Fig.1) shown in the below image, they are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle. But among all these features we calculated, most of them are irrelevant. We use Adaboost to select the best features. For each feature, it finds the best threshold which will classify the faces to positive and negative. Obviously, there will be errors or misclassifications. We select the features with minimum error rate, which means they are the features that most accurately classify the face and non-face images. The final classifier is a weighted sum of these weak classifiers. It is called weak because it alone can't classify the image, but together with others forms a strong classifier. The concept of cascade of classifier is a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. Instead of applying all features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

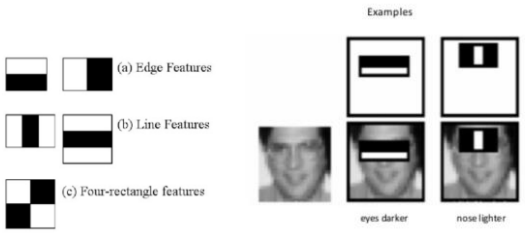

Fig.1 Haar features

In that case, we can easily detect the faces in any arbitrary image or a real time video using just 20 lines of code. Fig.2 and Fig.3 is the example for finding iron-man in an Infinity

War poster, it returns 8 faces by Haar cascade.

Fig.2 Original Poster Fig.3 8 faces founded by haar cascade

Part II: Face classification

After knowing how the face detection work, I decided to write my own code based on the paper I presented 'Deep learning in Metastatic Breast Cancer' to implement the CNN algorithm.

I started with a pretty small database 'the Olivetti faces dataset'. As described on the original website: There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement). The image is quantized to 256 grey levels and stored as unsigned 8-bit integers; the loader will convert these to floating point values on the interval [0, 1], which are easier to work with for many algorithms. The "target" for this database is an integer from 0 to 39 indicating the identity of the person pictured; however, with only 10 examples per class, this relatively small dataset is more interesting from an unsupervised or semi-supervised perspective. The original dataset consisted of 92 x 112, while the version available here consists of 64x64 images.

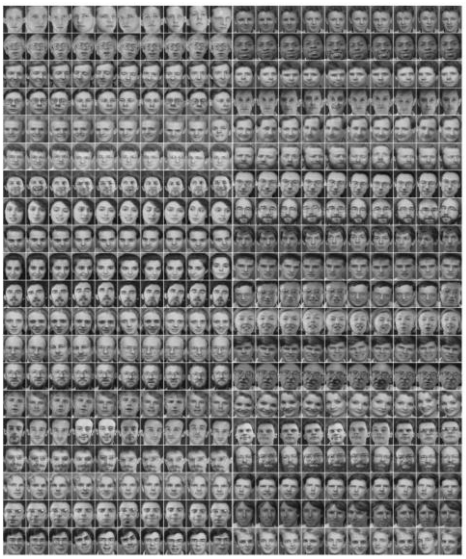

Fig.4 Olivetti-faces

I would like to build the neural network based on LeNet5. Inputs are the images, C1 is convolution layer, S2 is pooling layer, C3 is another convolution layer, S4 is another pooling layer, then fully connected to C5, C5 to F6, which can be seen as two hidden layers, logistic regression as classification in the end, the output of the model will be the 0-39 labels.
See details in CNN_face_detection_train.py

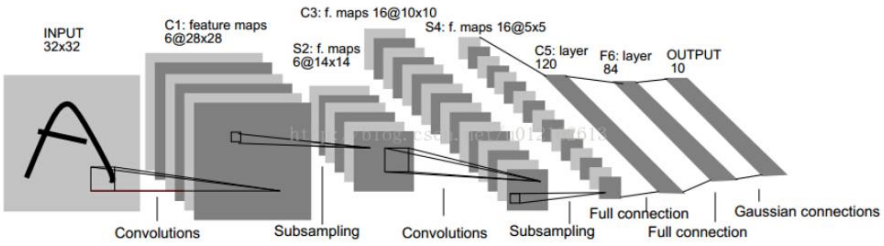

Fig.5 Neural network architecture

Now I will talk about set the parameters including learning rate, batch size, maximum epoch, number of convolution kernel, pooling size. First, I adjusted learning_rate with other parameters fixed. When learing_rate=0.1, the validation error is always 97.5%, it could be too high that it skips the optimal; then setting learning_rate as 0.01, when I trained 60 epoch, validation error downs to 5%, test error downs to 15%; again set learning_rate=0.05, when I trained 36 epoch, validation error downs to 2.5%, test error downs to 5%. The validation and test set only has 40 pictures respectively, which saying only one and two images are classified wrong, I set learning_rate=0.05 ad last. Second, I regulated the batch size which is best be a factor of 40, such as 1, 2, 5, 10, 20, 40. When I tried batch_size=1,2,5,10, validation error has been 97.5%, I think may be the sample categories covering ratio is too small, finally I set batch_size=20 and the corresponding validation error=10%. Third, I set the maxmimum epoch as 500 based on batch size. Fourth, I set pooling size=(2,2) since each image of the dataset only has 57*47 pixels. Fifth, I set number of kernel=[20,50] based on several attempts as record: nkerns=[20,50], when epoch=36, validation error=2.5%, test error=5%; nkerns=[5,10], when epoch=38, validation error=5%, test error=7.5%; nkerns=[10,30], when epoch=38, validation error=5%, test error=7.5%.
For the last part, I put the whole 400 images into test, run CNN_face_detection_test.py, we get the fo
llowing results with five images misclassified.
Best validation score of 5.000000 % obtained at iteration 304, with test performance 7.500000 %
The code for file CNN_face_detection_test.py ran for 2.17m
picture: 89 is person 8, but mis-predicted as person 34
picture: 178 is person 17, but mis-predicted as person 37
picture: 189 is person 18, but mis-predicted as person 6
picture: 299 is person 29, but mis-predicted as person 39
picture: 368 is person 36, but mis-predicted as person 20

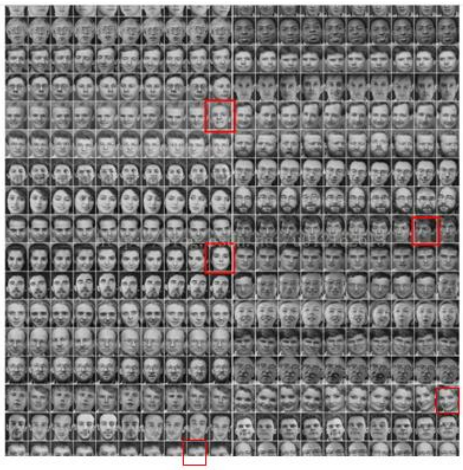

Fig.6 test result with five images misclassified

Part III: Face Alignment

As mentioned earlier, using Haar cascade to detect face is an old school. Now, many of people are using MTCNN, stands for multi-tasks CNN to do the face detection and face alignment. Fig.7 is the overall pipeline for the framework, given an image, initially resize it to different scales to build an image pyramid, which is the input of the following three-stage cascaded framework. In the first stage, it produces candidate windows and their bounding box regression vectors quickly through a shallow CNN called proposal network(P-Net). In the second stage, all candidates are fed to another CNN, called refined network(R-Net), which further rejects a large number of false candidates. At last stage, it uses a more powerful CNN called output network(O-Net) to refine the result and output facial landmarks positions. In each stage, also using the estimated bounding box regression vectors to calibrate the candidates and employing non-maximum suppression(NMS) to merge highly overlapped candidates. The pipeline just gives you a general idea, I will talk about the three stage, bounding box regression and NMS later.

What is bounding box regression, as shown in Fig.7, green box is the ground truth while red box is the proposal region, we calculate their intersection over union(IoU)=area of overlap/ area of union, it turns out the IoU is smaller than 0.5, if our threshold is 0.6, it can be considered as detecting the plane correctly. While bounding box regression is used here to calibrate the proposal window. We want to find a mapping such that the red box will become the blue box which is more accurate as the ground truth. And the mapping is a function of shifting and scaling.

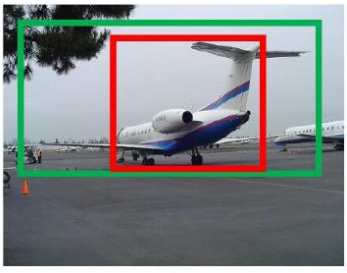

Fig.7 ground truth and region proposal

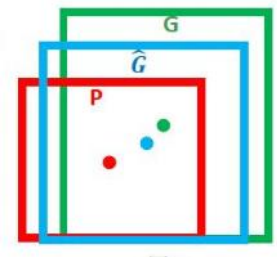

Fig.8 map with shifting and scaling

Non-maximum suppression(NMS) is a way to eliminate the miscellaneous bounding boxes. For example, the P-Net will return lots of bounding boxes on the same face, then we go through each bounding box and consider the bounding box as challenger in a boxing match, other boxes in this iteration are warriors who want to beat the challenger, if their IoU with the challenger is larger than a threshold, meaning they represent the same face, they can't beat the challenger, the challenger will win; if their IoU with the challenger is smaller than a threshold, meaning they could represent two different faces, they will stay. In the next iteration, the remain warrior will be the next challenger.

Fig.9 is the architecture for the three stage, P-Net only has the convolution layers, but R-Net and O-Net both have fully connected layer.

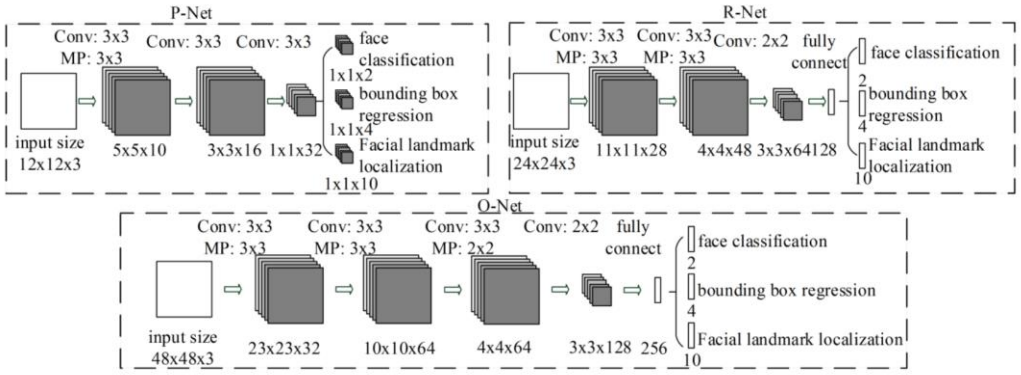

Fig.9 P-Net, R-Net, O-Net

We leverage three tasks to train the CNN network: face/non-face classification, bounding box regression, and facial landmark localization by loss function describes as following.

1) Face classification: The learning objective is formulated as a two-class classification problem. For each sample x_i, we use the cross-entropy loss:

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

where p_i is the probability produced by the network that indicates a sample being a face. The notation $y_i^{det} \in \{0,1\}$ denotes the ground-truth label.

2) Bounding box regression: For each candidate window, we predict the offset between it and the nearest ground truth (i.e., the bounding boxes' left top, height, and width). The learning objective is formulated as a regression problem, and we employ the Euclidean loss for each sample x_i:

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

where $\hat{y}_i^{box}$ regression target obtained from the network and y_i^{box} is the ground-truth coordinate. There are four coordinates, including left top, height and width, and thus $y_i^{box} \in \mathbb{R}^4$.

3) Facial landmark localization: Similar to the bounding box regression task, facial landmark detection is formulated as a regression problem and we minimize the Euclidean loss:

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

where $\hat{y}_i^{landmark}$ is the facial landmark's coordinate obtained from the network and $y_i^{landmark}$ is the ground-truth coordinate. There are five facial landmarks, including left eye, right eye, nose, left mouth corner, and right mouth corner, and thus $y_i^{landmark} \in \mathbb{R}^{10}$.

4) Multi-source training: Since we employ different tasks 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 functions (i.e., Eq. (1)-(3)) are not used. For example, for the sample of background region, we only compute L_i^{det}, and the other two losses are set as 0. This can be implemented directly with a sample type indicator. Then the overall learning target can be formulated as:

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

where N is the number of training samples. α_j denotes on the task importance. We use ($\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 0.5$) in P-Net and R-Net, while ($\alpha_{det} = 1, \alpha_{box} = 0.5, \alpha_{landmark} = 1$) in O-Net for more accurate facial landmarks localization. $\beta_i^j \in \{0,1\}$ is the sample type indicator. In this case, it is natural to employ stochastic gradient descent to train the CNNs.

Wider face dataset is a face detection benchmark dataset, they have 32,203 images and label 393,703 faces with a high degree of variability in scale, pose and occlusion. CelebA is a large-scale face attributes dataset with 10,177 number of identities, 202,599 number of face images with 5 landmark locations and 40 binary attributes annotations per image. Since

wider face dataset doesn't have the five landmark to do the face alignment and celebA dataset only has one face in each image, we have to combine this two dataset. Basic idea is to use wider face to do face detection and celebA to do face alignment. To reduce the computation, I only chose 2,000 images from each dataset, and 300 images in each dataset for testing. While we have joint perform face detection and alignment, here we use four different kinds of data annotation in the training process: 1)Negatives: Regions that the IoU ration less than 0.3 to any ground truth faces; 2)Positives: IoU above 0.65 to a ground truth; 3)Part faces: IoU between 0.4 to 0.65 to a ground truth; 4)Landmark faces: faces labeled 5 landmarks' positions. Negatives and positives are used for face classification tasks, positives and part faces are used for bounding box regression, and landmark faces are used for facial landmark localization. I set the minimum face size=15, minimum detection size(the pyramid)=12, scaling factor=sqrt(0.5), get 11 scales: ['0.80', '0.57', '0.40', '0.28', '0.20', '0.14', '0.10', '0.07', '0.05', '0.04', '0.02'], IoU threshold for bounding box calibration=[0.6, 0.7, 0.8], IoU threshold for NMS=[0.5, 0.6, 0.7]. We get the following receiver operating characteristic curve depending on the IoU with ground truth, AOC(attestation of compliance)=0.6154. The result is not very good compared with many sources online. I only used nearly one tenth of the dataset and there lots of parameters or thresholds need to calibrate. Mtcnn_step.ipynb is an example I demonstrated in the presentation written in pytorch: P-Net returns 2453 bounding boxes, reducing to 1869 after NMS and calibration; R-Net returns 92 bounding boxes, reducing to 76 after NMS and calibration; O-Net returns 33 bounding boxes, reducing to 11 after NMS and calibration. Compared to 8 faces the old school method Haar cascade using in OpenCV, I think the result is not bad.

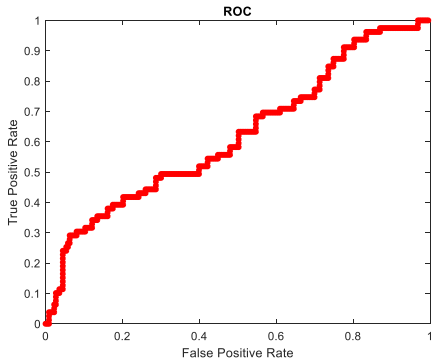

Fig.10 ROC curve