# APPLIED DATA SCIENCE FINAL PROJECT
# FACIAL KEYPOINTS DETECTION

Yuezhi Wang (yw2586) Wenkai Pan (wp2191)
Min Shi (ms4786) Mi Xiong (mx2152)

## Abstract

Facial keypoints such as the center of eyes and nose tips are important features for many different tasks in automatic face processing. The detection and localization of facial keypoints is usually performed interactively or it is not very precise. The goals of our project are to develop an accurate model to accurately measure facial features, tracking characteristic edge and output key points of the facial objects to be searched. The whole searching and analyzing process of our project is based on a dataset from a Kaggle competition and is principally inclined to use general methods like Trees, Regularization regression and Principal Components Analysis (PCA).

**Keywords:** Facial Keypoints Detection, Face Recognition, Decision Trees, Dimension Reduction, Principal Components Analysis

## 1. Introduction

Facial keypoints detection is a computer technology that finds the locations and sizes of human face main organs like mouth, nose and eyes in an image or a sequence of images. As one of the most important problems to be solved in human face processing, locating and extracting the key points from a whole human face with and without background is currently a very active research due to its potential applications like automatic face feature recognition, security system, video conferencing, intelligent human-computer interaction. During last decade, a large amount of methods have been proposed for face detection and facial keypoints detection. These methods can be generally divided into two categories: feature based approaches and image-based approaches (or named classification-based approaches).
The goal of our project is to detect the location of both eyes and nose on a human face based on the information of a grey scale picture.

## 2. Data Processing

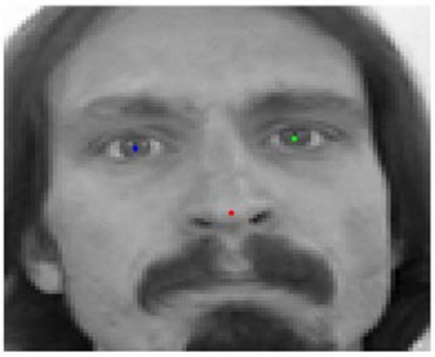
In this project, we mainly use dataset gained from Kaggle[i]. Since it is an open competition, the result of the test data is not given. So we split the training data into two parts, and the second 1000 observations are used as test data. Our goal is to detect the locations of eyes and nose.

### 2.1 Training Data

Training data contains 6049 training images. And for each image there are image ID and 6 variables corresponding to the target facial keypoints' x and y coordinates: Left eye center, Right eye center, and Nose tip. The image variable 'image' contains 9216 Grey Values from 0 to 255, which can output a 96*96 pixels image with key points marked (figure 2.1).

### 2.2 Testing Data

Test data contains 1000 test images with image IDs and 'image' variable with 9216 Grey Values from 0 to 255, which output a 96*96 image (figure 2.2).



*(figure 2.1 Training data image with keypoint)* *(figure 2.2 Test data image)*

## 2.3 Exploratory analysis of the data

Based on common sense, human faces are more or less similar. If the pictures in training data are presented as the one shown above, it is simpler to locate the feature. So we need to see how variable our data is. One good way is to plot out the locations of all the nose tips (figure 2.3).

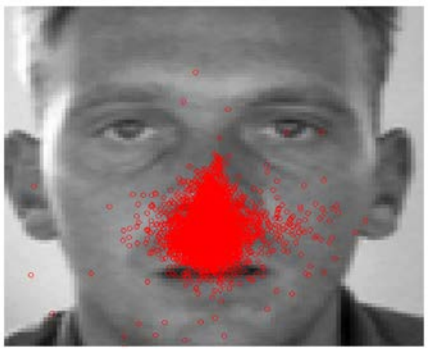


*Figure 2.3 The location of nose in the training data*

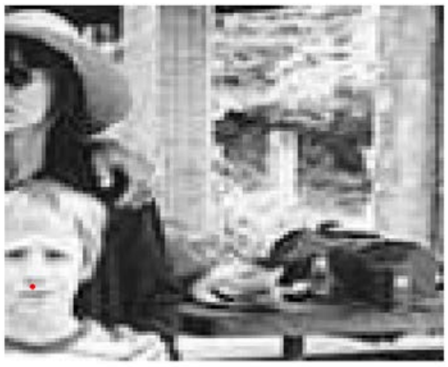


*Figure 2.4 One of the outliers*

Most nose points are located in the center of an image as we expected, but there are certain outliers that deserves further investigation, as they could be errors. One extreme outlier is shown in figure 2.4, which clearly is not an error. This indicates we cannot treat outliers as normal data, as the background information beyond a face could cause inaccuracy. So we need to eliminate the effect of the noise information as much as we can.
Also we can see check the covariance matrix of the training data. The diagonal elements can reflect the variation of the feature. We can see that the variation of feature nose_tip_y is obviously larger than the others which makes the feature nose harder to predict.

| **Left_eye_center_x** | **Left_eye_center_y** | **Right_eye_center_x** |
|---|---|---|
| 11.408 | 9.566 | 8.756 |
| **Right_eye_center_y** | **Nose_tip_x** | **Nose_tip_y** |
| 9.010 | 17.508 | 33.414 |

## 2.4 Noise Elimination

As we shown above, the main problem of local and data-driven keypoints detection is that there is too much local image structure in complex real world images. Therefore, it is careless to give any interpretation to local features without considering their context. Thus, our approach starts by eliminating background noise. We search the MATLAB tool box for help and managed to determinr the boundary of each face. Then we set the pixel value outside the face area to 0 to realize background noise elimination. The result can be shown as following:

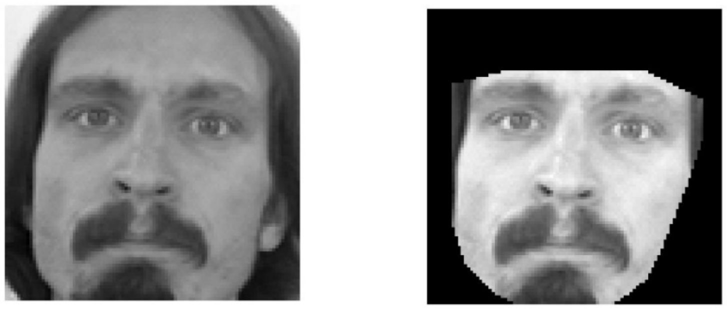


*Figure 2.5: The results of background noise elimination*

## 3. Methodology

We have come up with several methods to extract the features. We use the accuracy of a very simple method as the baseline to assess the models we built. The baseline method (Naive Mean Method) is simplistic as it utilized a quite general method which didn't take into consideration of individual difference. The test error of the Naïve Mean Method is 19.53.

### 3.1 Key point Box matching method

Assumption: In predicting problem for one specific key point, we should highlight those pixels that are around that key point.
Motivated by this assumption, we design our new strategy of using "Image Box": Instead of utilizing all information of a picture, we use small box to extract out matching area in picture for one specific key point. This method is a two-step strategy: first calculate a mean key point and pick the best match of the mean key point to all possible small boxes in one test data.To further illustrate this method, we first focus on one single key point, i.e. nose tip.
For each picture in training dataset, we extract a 21*21 (after trial and error) square box around nose tip. After we got all 6049 21*21 boxes, we calculate the average of all these 6049 matrices.The result is presented below.

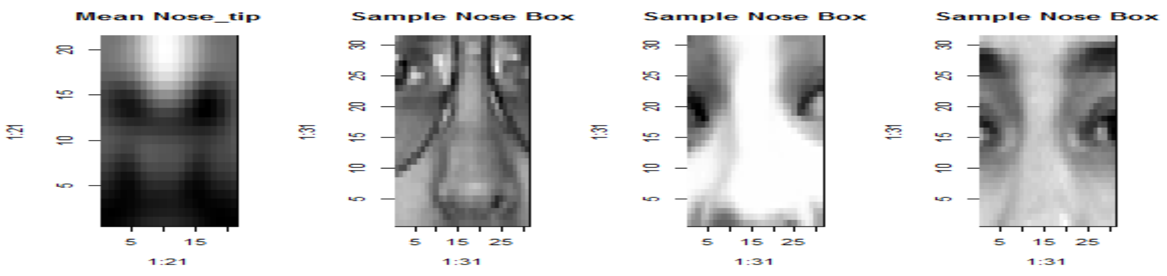


*Figure 3.1: Results of Nose boxes.*

Now we can use this Mean Nose tip box to "search" for the same keypoint in the test images. Given a test image we need to try all its possible 21*21 boxes, and see which one best matches the mean-nose box. Our measure to determine the best match is correlation with the mean-nose box. The coordinate with the highest score of correlation with that mean-nose box is considered to be our predicted location of nose.
We can easily apply the same idea to other key points.
Compared with Naive Mean method baseline, this feature box extraction method achieves a prediction error score of 3.702. It's an improvement and the result makes sense.

### 3.2 Combine PCA with KNN

Assumption: If two pictures are "close" enough, then the location of key points of the two faces are very similar. Inspired by this idea, we would like to detect the "nearest neighbor" of each test picture in the training datasets, and immediately assign the coordinates of that nearest training picture to the test picture.
However, it is hard to measure the distance. If we choose Euclid Distance measure, there still exists severe problem about "huge" data. Here we have altogether 7049 pictures, each represented by a 96*96 matrix (or a vector with length 9216). To calculate distance matrix for this dataset with such huge number of dimensions is a disaster.

Dimension reduction is something covered in basic machine learning topic. We immediately turn to PCA (Principle Component Analysis)technique.
To do PCA on a 7049*(96*96) matrix still suffer typical “n<<p” problem. To solve this, we apply a simple strategy to compress each picture from 96*96 to 24*24. We take the average of each 4*4 box in original picture and replace initial 16 pixels with one value. This method ends up with a very satisfying result. After all, we are trying to reduce dimension from 96*96 to something like 30 or less, this initial step with 24*24 is reasonable.
The next step is to implement PCA on training set and use the first 30 scores as a vector to represent data. It includes creating the mean face from the training set, subtracting the mean face, and running *prcomp()* in R on the resulting image matrix. Project the testing data onto the first 30 loadings so that it is also represented by the first 30 scores. Figure 3.2 is the “Eigenfaces” (eigenvalues).

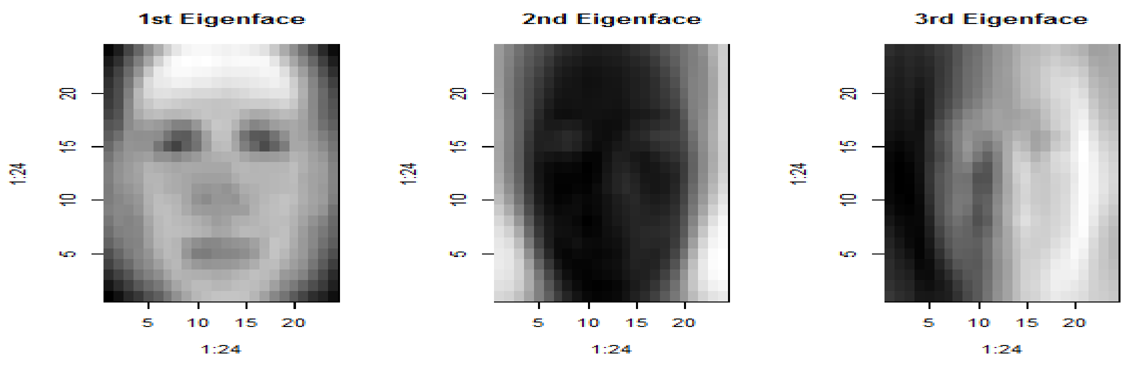


*Figure 3.2: The first three “eigenfaces” of the data*

Next step is to use 1-NN classification in the space of the first 30 scores to identify the subject for each testing observation.
Let’s look at some of the matched pairs given by our PCA&1NN method (see Figure 3.3).

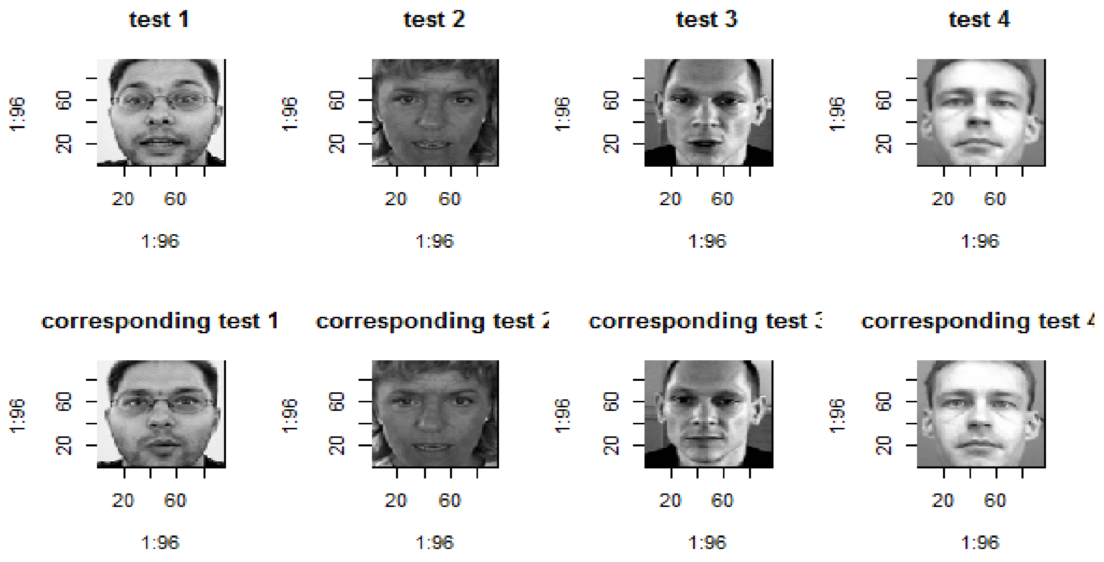


*Figure 3.3: Matching result of PCA.*

Visualization indicates that our combined method works perfectly as it can exactly picks out the same person in training data corresponding to test data! We have gained some very interesting insight about this dataset: This dataset might be originally from so-called “facial detection” problem, which is interested in identifying the same person’s picture in training data. We can definitely say that our method did a very good job in this problem. The differences of two pictures are merely slight different pose of that person or different angle of camera. It seems like magic and we are very confident that PCA retain very useful information of the initial data so that the two corresponding pictures are really close enough.

However, our main concern lies in whether this method has good performance of our interested question in terms of prediction error? Unfortunately, the answer is not so satisfying. Our test error has an average performance of 5.654, compared with the feature box method in 3.1. However, we also notice that the error in the first 4 test data is 1.531, 1.370, 1.684 and 1.193 which is quite satisfying not only compared with the error in this method but also with baseline. This gives us the message that our method has a good performance on “facial detection” problem. If the same person is detected, the prediction error of key point detection is satisfying. If there is no such same person matched in training data, it seems to be not that reasonable to utilize the nearest neighbor to give key points’ location. The reason is that PCA still fetch out those general feature of the whole picture. In order to focus our attention on “key points”, we still need to extract the key feature of that specific box of pixels. This is the motivation of our following method, which applies PCA on keypoints area.

## 3.3 Combine PCA on features and Key point matching method:

After trying to detect whole faces with principle component analysis, we are also inspired to directly detect facial features like eyes, nose, or mouth with this method. And the “eigeneyes””eigennoses” can also be used to do matching as presented in 3.1. In this part, we use the noise-eliminated data.

In the first place, we can utilize the training images to derive the eye images and non-eye images. Through observation, we can find that the eyes in training image are basically contained in a 21*11 pixel sub-image. Therefore we take the single pixel that contains the hand-picked mark as the center pixel. And expand to left and right for 10 pixels respectively and up and down for 10 pixels respectively. It is worth noting that we need to discard the eyes that are not completely contained in the training image, in our case, centered too close to the edge of the photos.

### 3.3.1 EigenEyes and EigenNoses:

The following pictures show the EigenEyes and EigenNoses that we got from PCA. Because we choose 21*11=231 dimensional features, there are 231 orthogonal Eigen features. We can sort them using their scores. It’s clear that eigen features with high score are more important as they contain the majority of information. We can see important Eigen features usually can be recognized by human. If we plot the score, first 10 Eigen features will stand out because their score are much higher than the rest. So we can basically ‘rebuild’ any features without large loss of information merely with a few Eigen features.

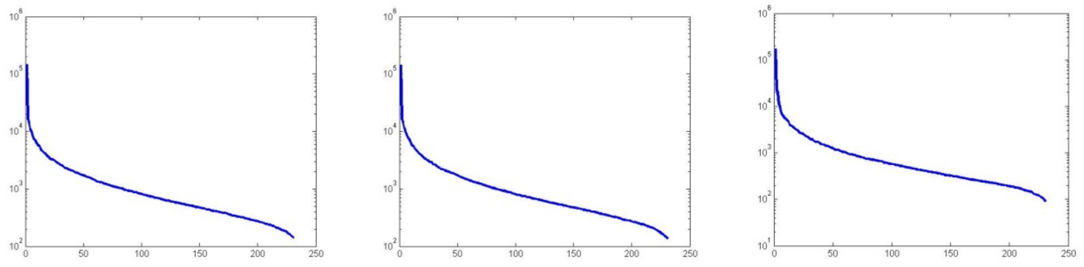


*Figure 3.4: Sorted Scores for Eigen features of left eyes, right eyes and noses.*

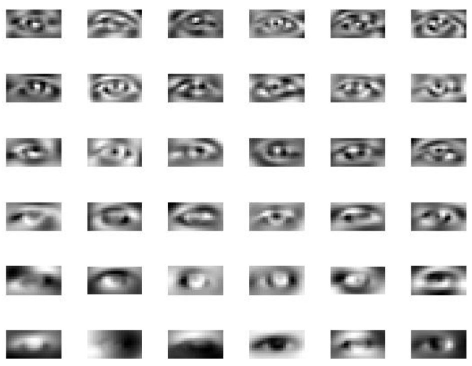


*Figure 3.5: Most important eigenvectors for left eyes. (Bottom Ones are most important)*

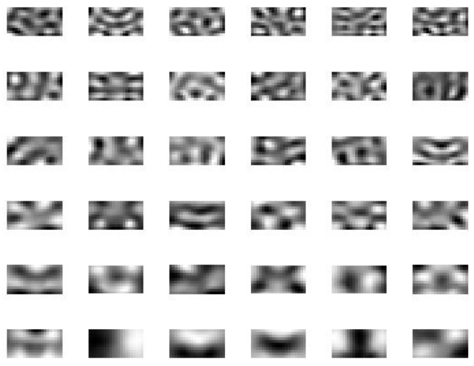


*Figure 3.6: Most important eigenvectors for noses. (BottomOnes are most important.)*

### 3.3.2 Detecting eyes and noses in a whole photo

To detect eyes or noses on a real photo, we need to first ensure the faces are of about the same size as our training photos. For example, we can't directly compare an eye of 200*100 pixels large to our Eigen features which is 21*11 pixels. Here we use the methods mentioned before to extract the outline of the face in the photos. First we can use the outline to estimate the size of face and adjust it to be close in size with our training faces. Secondly we can just skip evaluating the possibility of being a specific feature at points outside a face.
We take each 21*11 sub-pictures in the original 96*96 photo and make projections onto our most important Eigen features. In detail, we calculate the inner product of each sub-picture and 9 most important Eigen features, then add all the squares together, finally take the square root as the 'possibility' of the sub-picture being the specific feature on the face. After that we just choose the sub-picture with highest 'possibility' to become our detected feature.

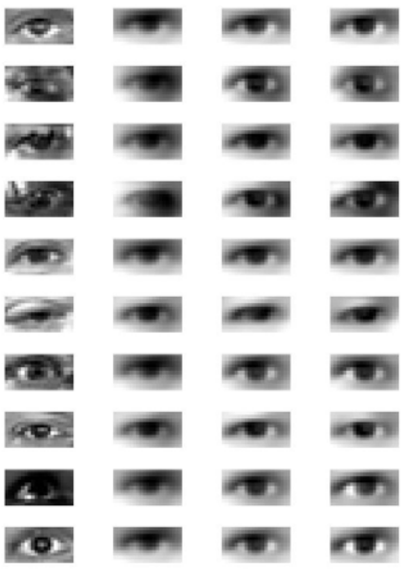


*Figure 3.7: Rebuilding left eyes from original photos with 3, 6 and 9 Eigen features respectively from left to right.*

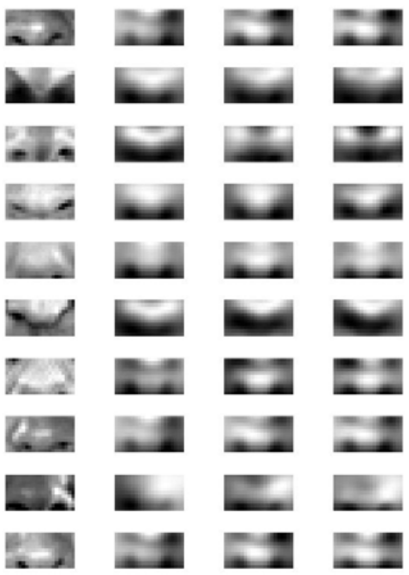


*Figure 3.8: Rebuilding noses from original photos with 3, 6 and 9 Eigen features respectively from left to right.*

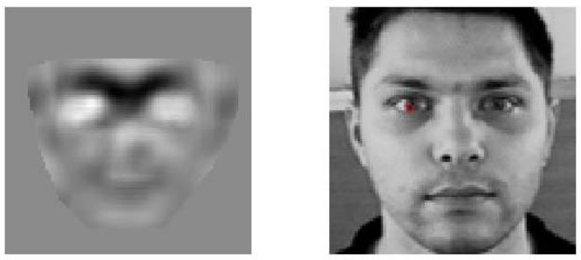


*Figure 3.9: Left picture represents the possibility of each pixel to be the center of left eye. The red circle of the right picture represents the left eye center we detected.*

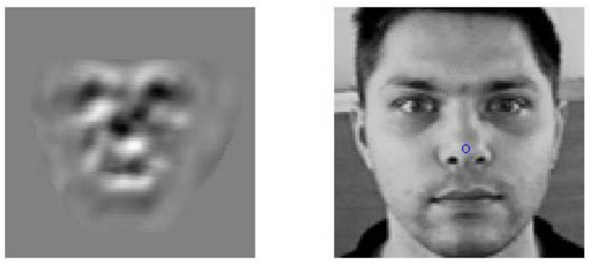


*Figure 3.10: Left picture represents the possibility of each pixel to be the center of nose. The blue circle of the right picture represents the nose center we detected.*

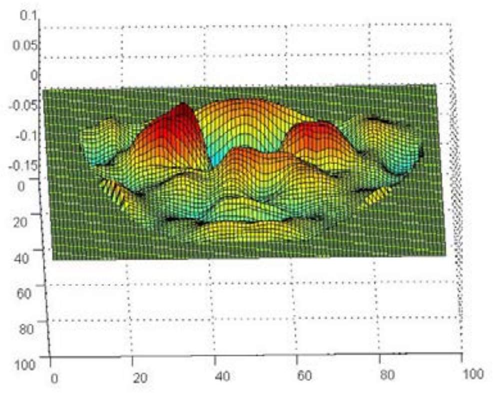


*Figure 3.11: The possibility of each pixel to be the center of left eye. (3D mesh) Higher terrain means larger possibility.*

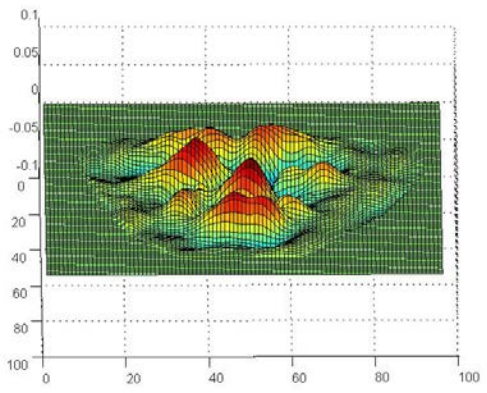


*Figure 3.12: The possibility of each pixel to be the center of nose. (3D mesh) Higher terrain means larger possibility.*

### 3.3.3 Results:

After trying many test photos, we find this method is efficient and accurate. But due to the huge amount of computation, we can't get a test error rate. So computational expensive is the major defeat of this method.

The following (Figure 3.13) is the results of eight randomly selected testing data (image 5, 55, 555, 5555 and 6, 66, 666, 6666).

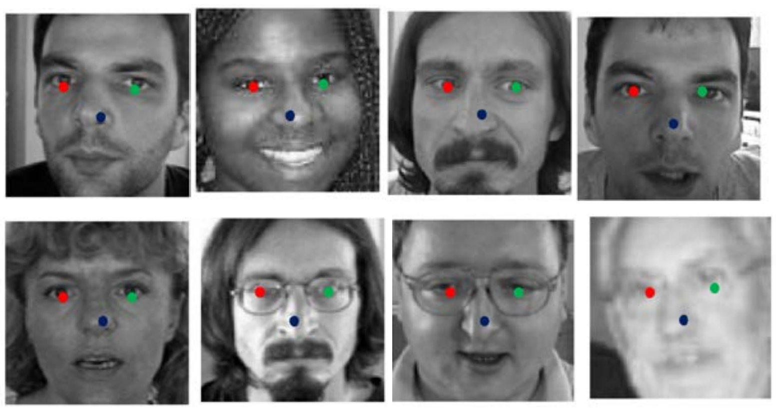


*Figure 3.13: The Results on 8 randomly chosen test image*

## 3.4 Mixture of trees with a shared pool of parts

Though great strides have been made in face detection and facial keypoints detection, the tasks have traditionally been approached as separate problems with a disparate set of techniques. For further digging, we studied an open source novel facial detection model. This model is a simple approach to encoding elastic deformation and three-dimensional structure by using mixtures of trees with a shared pool of parts. This model includes face detection, pose estimation and landmark localization by defining parts at some facial landmarks and using global mixtures to capture topological changes due to viewpoint. In this work, we developed five new trees standing for three new keypoints we are about to detect and merged them into the original pool.

**Tree structured part model:**

Write each tree $T_m = (V_m, E_m)$as linearly parameterized, tree-structured pictorial structure, where m indicates a mixture and $V_m \in V(V$: shared pool of parts). Write I for an image, and $l_i = (x_i, y_i)$ for the pixel location of part I. Score a configuration of parts $\mathrm{L} = \{l_i : i \in V\}$ as:

$$S(I, L, m) = App_m(I, L) + Shape_m(L) + \alpha^m \qquad (1)$$

$$App_m(I, L) = \sum_{i \in V_m} \omega_i^m \cdot \phi(I, l_i) \qquad (2)$$

$$Shape_m(L) = \sum_{ij \in E_m} a_{ij}^m dx^2 + b_{ij}^m dx + c_{ij}^m dy^2 + d_{ij}^m dy \quad (3)$$

After defining and adding five new trees standing for left eye center, right eye center and nose tip in to the original trees, we retrained by our testing data and output all predict coordinates to estimate the error. Results on some random chosen faces from testing data are summarized in Figure 3.14.

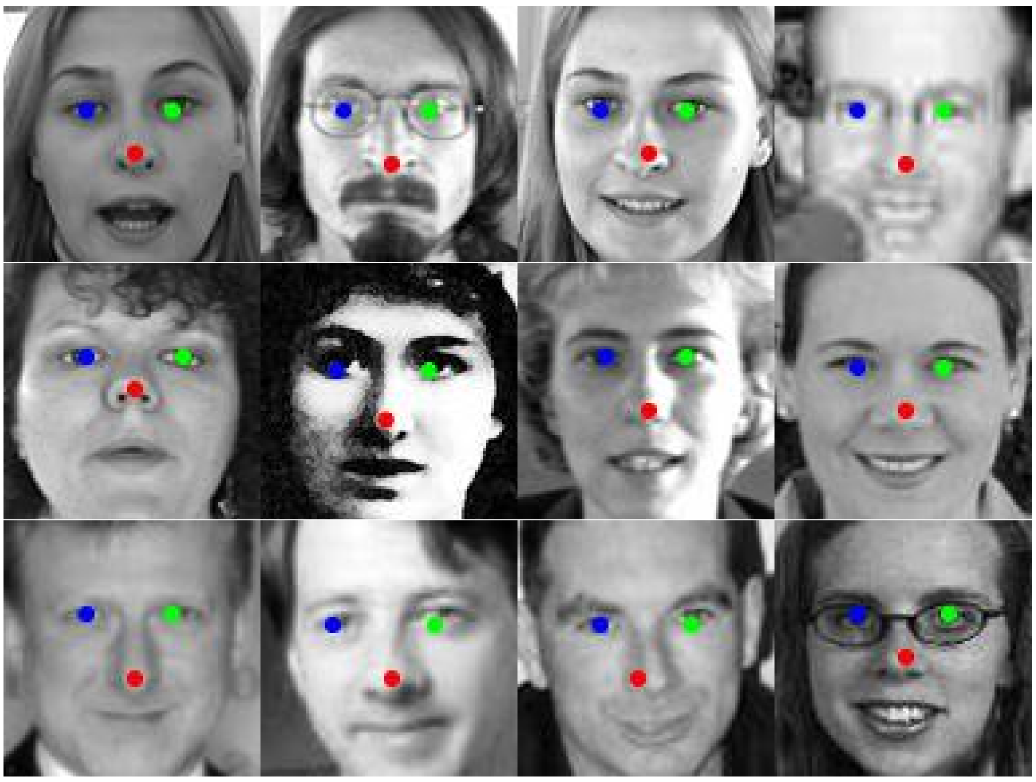


*Figure 3.14: Results on Test data.*

Through the visualization results we can see the model is surprisingly effective in capturing facial keypoints. Concerning of performance, the test errors of the model are left-eye-center-error: 3.5253; right-eye-center-error: 3.3748; nose-tip-error 4.4337, we conclude the new added trees are performing well.

## 4. Summary

In the final analysis, we have learned a lot about our data set and the methods that can be applied to achieve better results. In general, Principle Component Analysis is essential due to the high dimension. But solely extract the principle components of the whole picture can't predict precisely the location of certain feature. So a combination of PCA on features and match box searching is really a good approach.

Facial detection problem is really interesting and there are various ways to address it. What we focus here is in the statistical methods we have learned this semester. There is a lot of valuable research in computer science area that worth studying, for example Computer Vision. Just like we learned in the first class, data science is a combination of statistics, computer science and domain science.

The methods we applied in this project are not very advancing but the results are interesting. Nevertheless, we gained significant insight into the nature of image processing schemes and are now well-prepared to continue tackling related challenges in the facial feature detection area.