Facial Expression Recognition System

Rui Zhang,Yiji Zhang

December 14, 2016

1 introduction for graphical user interface

This section is the introduction of our graphical user interface. We will introduce the function of every icon by using both texts and pictures to introduce clearly.

First of all, figure 1.1 is our main interface. In this window, users have three choices which include collect data, testing and filter/save. We will introduce these three functions separately in more details below.

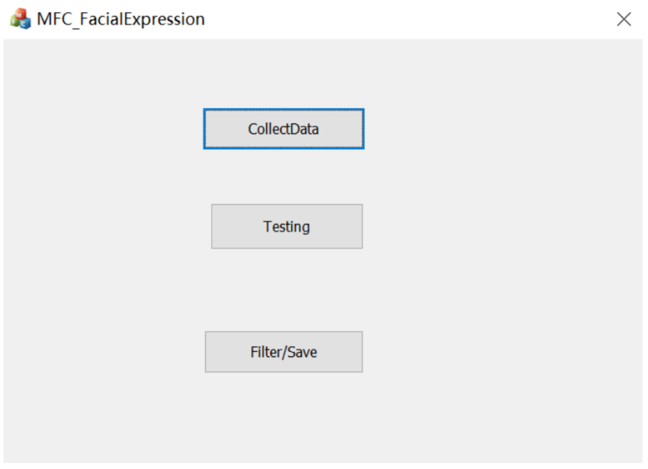


figure 1.1 main interface

If users click collect data, it will pop-up a new window shown in figure 1.2. In this window, there are four icons which both represent the expression. As you can see, they are angry, happy, neutral and surprise. In our project, we only record, train and test data of these four expressions. No matter what you click on these four icons, the

following procedure and windows that popped-up are totally similar. So I only choose the icon happy to introduce.

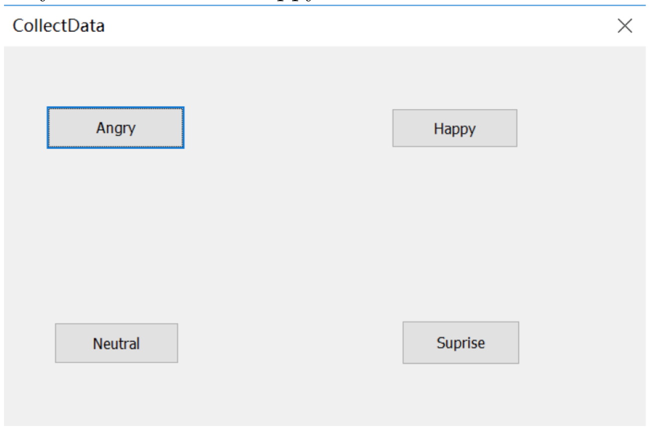


figure 1.2 collect data window

After users click happy, they will jump to a new interface for capturing and recording image data that shown in figure 1.3.

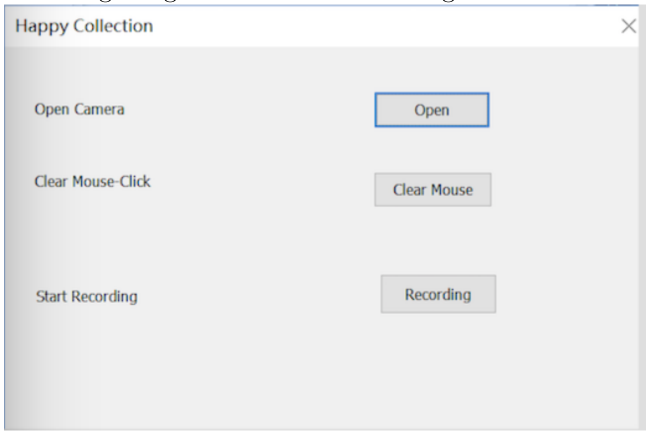


figure 1.3 happy collection window

Firstly, you need to click open camera to capture facial expression image. We will capture 200 pictures for one expression and the whole process lasts about 20 seconds. Secondly, in order to ensure that we can capture the effective area for our facial expression, we need to adjust manually by clicking and dragging frame. As you can see in figure 1.4, red points determine the edge of the frame. For consistency of every image captured, we will set up one of the points at ear position.

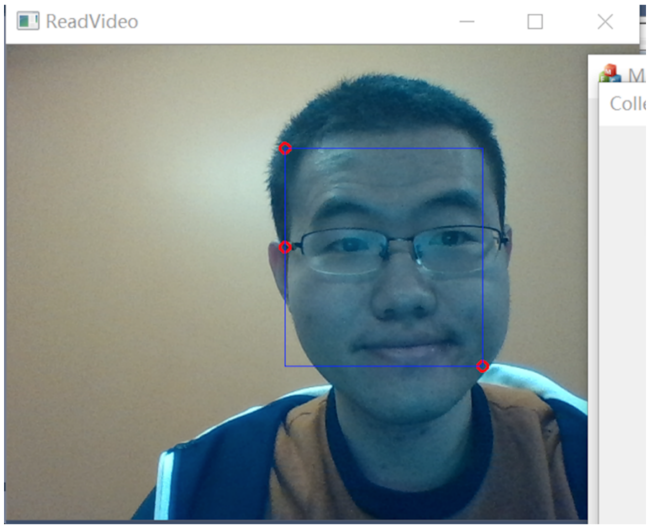
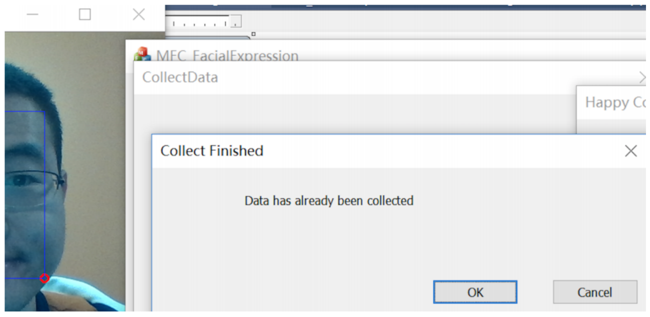


figure 1.4 frame for capturing image

If we want to reset these red points, we can click clear mouse icon in happy collection window shown in figure 1.3. Then these red points we selected before will be cleared instantly. Thirdly, after capturing image, we need to record them. So we can click recording icon in figure 1.3. After that, all images that we captured and processed by gray image algorithm will be saved in specific folder. Like all happy expression will be saved in a folder named Happy. When recording image has already been finished, a notification window shown in figure 1.5 will be popped-up. Above is the introduction of collecting data, then we will go to next part of filter/save icon



Collect Finished

Data has already been collected

figure 1.5 notification window

After capturing and recording our images, we need to process these images. In this project, we choose Gabor filter. As same as collecting data, we need to click filter/save icon in the main interface shown in figure 1.1. The new window is showed in figure 1.6. In this new windows, we sill use happy expression as example. We would like to process happy expression by Gabor filter, so we choose happy in figure 1.6. Then all happy expression images will be fetched to process by Gabor filter. These images will be saved automatically in specific folder. In this case, happy expression will be saved in folder GaborHappy.

Finally, we would like to introduce Testing icon in main interface shown in figure 1.1. The new window will be showed in figure 1.7. The function of this window is similar to figure 1.3.

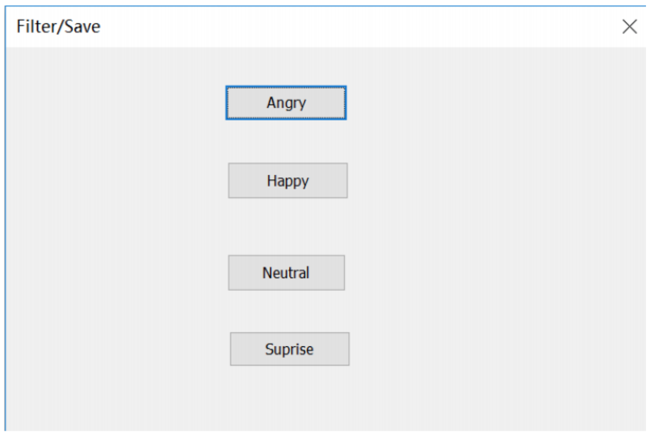


figure 1.6 Filter/Save window

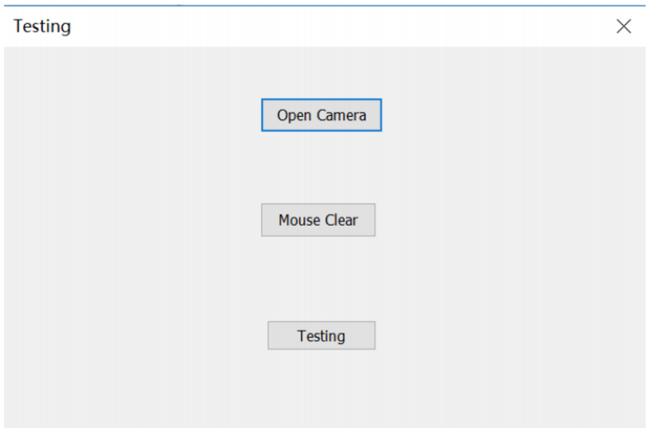


figure 1.7 Testing window

We need to click Open camera icon and Mouse clear icon to capture our testing image. The capturing window for testing image is showed in figure 1.8. When we have already collected testing image. then we

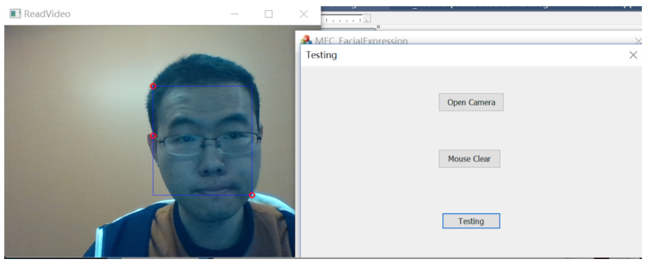
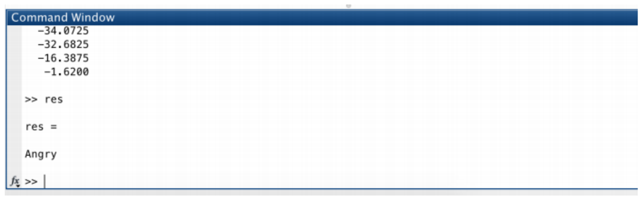


figure 1.8 Testing window

click Testing icon. After that, gray image of this expression that we collected earlier will be processed by Gabor filter automatically and it will be saved in Gabor test folder. By the way, we only select one of all testing images randomly to execute.

If we want to see the result of which expression it belongs to, we need to execute our Matlab code of different algorithm. In this project, we use least square, perceptron and SVM algorithm to analyze data. After execution, we need to input res in command windows. Then we can see a result as shown in figure 1.9.



```
Command Window
  -34.0725
  -32.6825
  -16.3875
   -1.6200

>> res

res =

Angry

fx >>
```

figure 1.9 the result for test

2 image processing

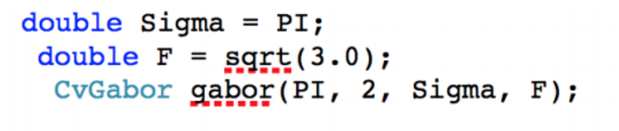
2.1 Gray Image

We can use OpenCV's function to convert normal pictures to Gray Image to reduce size of pictures

2.2 Gabor filter

In image processing, the Gabor function is a linear filter for edge extraction. The frequency and direction of the Gabor filter are similar to the human visual system. It is found that the Gabor filter is suitable for texture expression and separation. In the spatial domain, a twodimensional Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave.

We achieve Gabor filter in OpenCV. Here is the link of reference we use for Gabor filter. Link: http://blog.csdn.net/lichengyu/article/ details/20743877 . In our project, we set the Gabor parameter as shown in figure 2.3.1.



```
double Sigma = PI;
 double F = sqrt(3.0);
  CvGabor gabor(PI, 2, Sigma, F);
```

figure 2.3.1 parameter of Gabor filter

The format of gray images is 200*200. After Gabor filter, the images will become 200*200*3. Gray image and processed image by Gabor filter have been shown in figure 2.3.2. We will use these Gabor images for all algorithms.

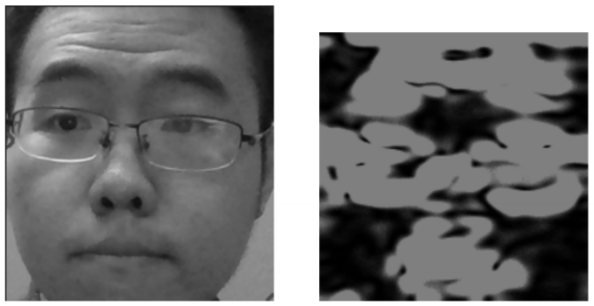
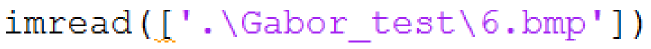


figure 2.3.2 gray images and Gabor images

3 introduction of Matlab code

3.1 data processing

The first part of our Matlab code before three algorithms is data processing. We use imread function of Matlab to turn all Gabor filter images into data.

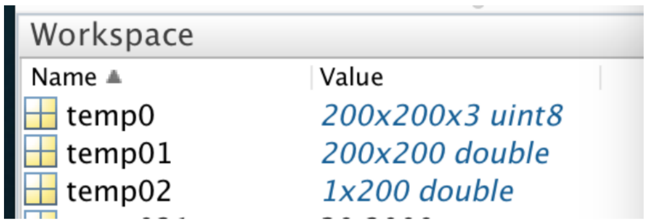


```
imread(['.\Gabor_test\6.bmp'])
```

figure 2.1.1 imread function

3.1.1 load data

The data processing for these four expressions are all the same, so we only use angry to introduce. As you can see in figure 3.1.2, variable temp0 represents the initial data we obtain from Gabor filter images. There is no doubt that it is so difficult if we use this kind of 200*200*3 data to execute our algorithm. So we need to design a piece of code to simplify them. Our Matlab code is shown in figure 3.1.3. Firstly, we need to calculate the mean for every pixel in z-axi

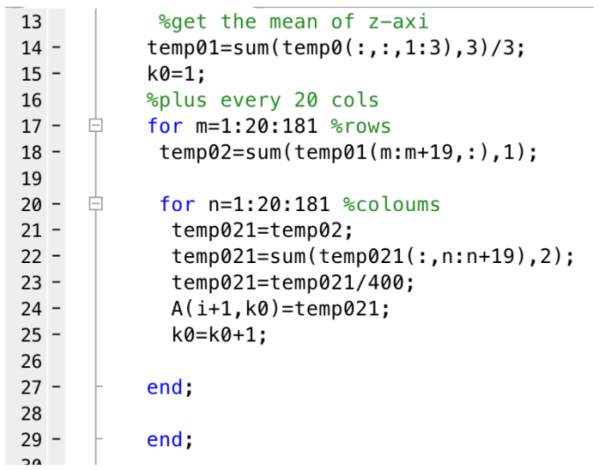


Workspace

Name ▲	Value
temp0	200x200x3 uint8
temp01	200x200 double
temp02	1x200 double

figure 3.1.2 variables for data processing

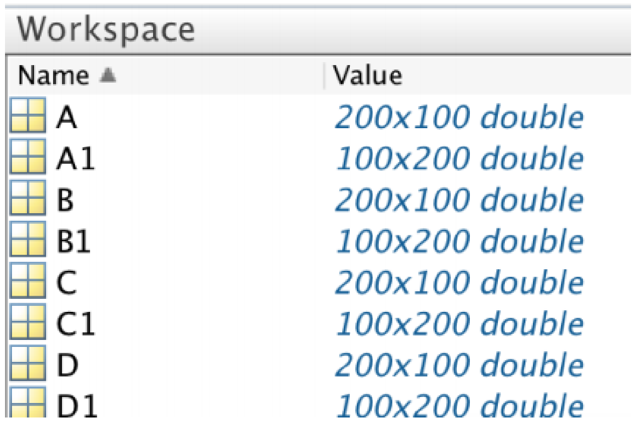
then our data will become 200*200 format. Secondly, we calculate the sum of every 20 rows and columns, then we can get average through divided them by 400. After that, our data will become 10*10. Finally, it is easy to turn our data from 10*10 into 1*100.



```
 %get the mean of z-axi
temp01=sum(temp0(:,:,1:3),3)/3;
k0=1;
%plus every 20 cols
for m=1:20:181 %rows
 temp02=sum(temp01(m:m+19,:),1);

 for n=1:20:181 %coloums
  temp021=temp02;
  temp021=sum(temp021(:,n:n+19),2);
  temp021=temp021/400;
  A(i+1,k0)=temp021;
  k0=k0+1;

end;

end;
```

figure 3.1.3 simplifying data

After that, we can collect 200 data like 1*100 format and we use a matrix to represent them. As you can see in figure 3.1.4, variable A, B, C, D represent respectively expression angry, happy, neutral and surprise. They are all 200*100 matrix. We will use these processed datas to execute least square, perceptron and SVM algorithm.



Workspace

Name	Value
A	200x100 double
A1	100x200 double
B	200x100 double
B1	100x200 double
C	200x100 double
C1	100x200 double
D	200x100 double
D1	100x200 double

figure 3.1.4 ultimate processed data

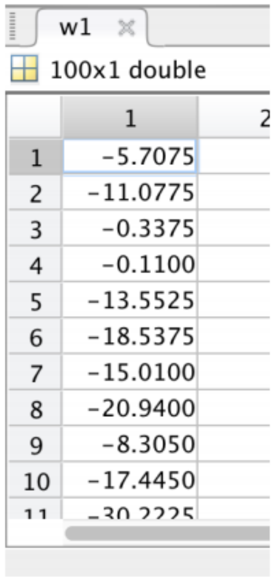
3.2 algorithm

3.2.1 perceptron

For perceptron algorithm, the most critical method is we need to iterate W by determining whether the data belongs to specific expression. We use the part of expression angry to explain. The number of our data set is 800, so we have 800 data to train this model. First of all, we set an initial w1 as w1=[ones(1,100)] . Then we select one data from data set randomly. We need to judge whether this expression data belongs to angry. We use w1 * x11 as our discriminant, w1 is the parameter we want to train and x11 represent this data. If this discriminant is less than 0, we determine this data does not belong to angry. So we need to do iteration, w1=w1+x11* t(n). By the way, t(n) equals to 1 when this data belongs to angry. If not, t(n) equals to -1. On contrary, if this discriminant is greater than 0, this data belongs to angry and we dont do anything about w1. Because we have 800 datas, we need to perform 800 rotations. The whole code for this rotation is shown in figure 3.2.11.

figure 3.2.11 code of iteration for w1

When we finish this rotation, we can get a variable w1 shown in figure 3.2.12 which has been trained by perceptron algorithm.

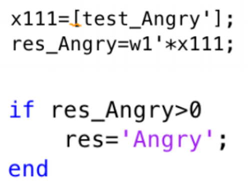

w1

100x1 double

	1	2
1	-5.7075	
2	-11.0775	
3	-0.3375	
4	-0.1100	
5	-13.5525	
6	-18.5375	
7	-15.0100	
8	-20.9400	
9	-8.3050	
10	-17.4450	
11	-30.2225	

figure 3.2.12 variable w1 of angry

Finally, we use this variable w1 to carry out test. We need to process our testing data as same as what we do in training. Variable x111 represents one of testing data for angry. As you can see in figure 3.2.13, variable res-angry equals to w1*x111. If res-angry is greater than 0, we can draw a conclusion that this expression data belongs to angry.

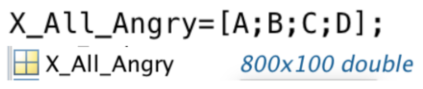


```
x111=[test_Angry'];
res_Angry=w1'*x111;

if res_Angry>0
    res='Angry';
end
```

figure 3.2.13 testing

subsubsection least square

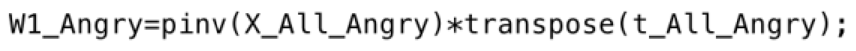
For least square, we have the different method to obtain w1. We still use the Matlab code of angry to explain. The data processing is totally similar with perceptron. So we do not introduce these code of data processing in this section. As you know from above, A, B, C, D are all 200*100 matrix. In this algorithm, we set a new variable X-All-Angry equals to the combination of A, B, C and D shown in figure 3.2.21. So this variable is a 800*100 matrix and we use it to train model.



```
X_All_Angry=[A;B;C;D];
X_All_Angry        800x100 double
```

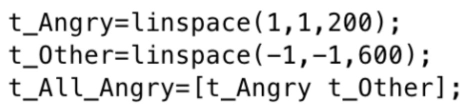
figure 3.2.21 Variable X-All-Angry

It is easier than perceptron to calculate variable W1-Angry by least square. We just need to multiply the pseudoinverse matrix of variable X-All-Angry by transpose of t-AllAngry. This equation is shown in figure 3.2.22. Variable t-All-Angry represents which expression the whole data set respectively belongs to. If it belongs to angry, t equals to 1. If not, t equals to -1. The calculation of t-All-Angry is be shown in figure 3.2.23. Then we can use this W1-Angry to test our data.



```
W1_Angry=pinv(X_All_Angry)*transpose(t_All_Angry);
```

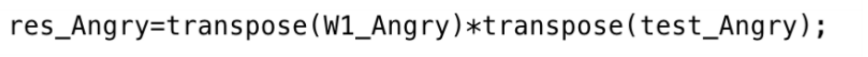
figure 3.2.22 variable W1-Angry



```
t_Angry=linspace(1,1,200);
t_Other=linspace(-1,-1,600);
t_All_Angry=[t_Angry t_Other];
```

figure 3.2.23 variable t-All-Angry

Variable test-Angry represents one of testing data which has been processed. So we can use this discriminant shown in figure 3.2.24.



```
res_Angry=transpose(W1_Angry)*transpose(test_Angry);
```

figure 3.2.24 variable res-Angry

If the value of variable res-Angry is greater than 0, the testing expression image belongs to angry.

3.2.2 SVM algorithm

The third algorithm is SVM. We want to use SVM algorithm to maximize the minimum of the margin. The discriminant is shown in figure 3.2.31. We write the activation as: a(x,w) = wT(x) + w0 to emphasize offset w0. If we supposed that tn*a(Xn,w)=1, what we do is that minimizing the value of

$$||w||^2$$

. After classification, we hope the result multiplied by tn is greater than a fixed value and this value is greater than 0 and near 1. So we use equation 1-z to represent the result, z is greater than 0. Then we can optimize w by minimizing the value of

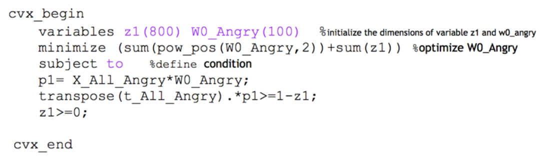
$$||w||^2 + z$$

.

$$\text{maximize} \min_{n=1,\ldots,N} t_n \frac{a(\mathbf{x}_n, \tilde{\mathbf{w}})}{||\mathbf{w}||}$$

figure 3.2.31

We use Matlab code for angry to explain SVM algorithm which has been shown in figure 3.2.32.



```
cvx_begin
    variables z1(800) W0_Angry(100)   %initialize the dimensions of variable z1 and w0_angry
    minimize (sum(pow_pos(W0_Angry,2))+sum(z1)) %optimize W0_Angry
    subject to   %define condition
    p1= X_All_Angry*W0_Angry;
    transpose(t_All_Angry).*p1>=1-z1;
    z1>=0;

 cvx_end
```

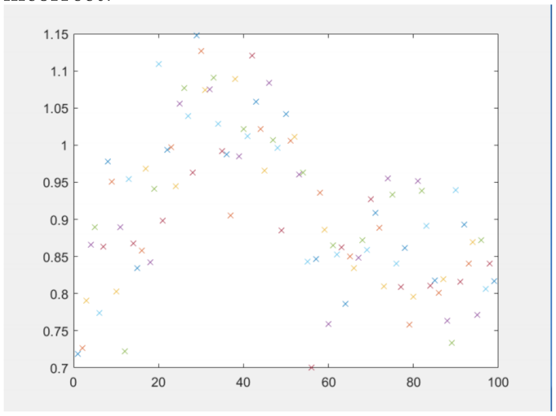
figure 3.2.32 Matlab code for SVM

4 testing of three algorithm

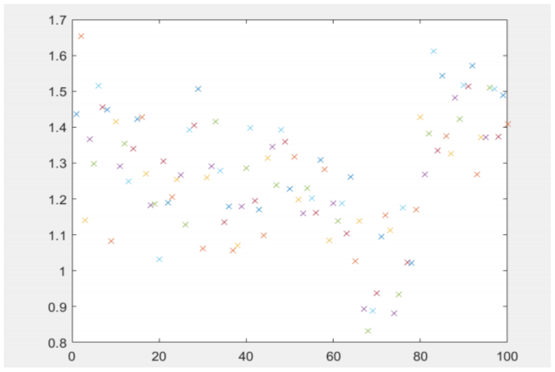
In this section, we will show the testing results about the accuracy of three algorithms. For every algorithm, we choose 100 new images for every expression to test. For example, we only choose 100 angry images to test part of algorithm for training angry. So, the result can be expected that accuracy approaches 100 percent if our algorithm is accurate enough.

4.1 least square

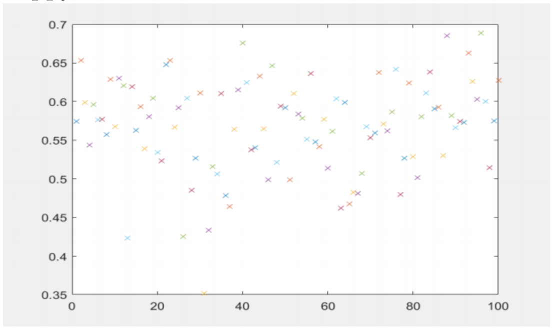
There are four pictures in figure 4.1.1, each picture shows the result of accuracy for every expression respectively. X-axis represents the number of each picture, the sum is 100. Y-axis represents the result of calculation. If its value is greater than 0, it means this image belongs to this expression and our classification is correct. Oppositely, if the value is less than 0, classification for this image is incorrect.



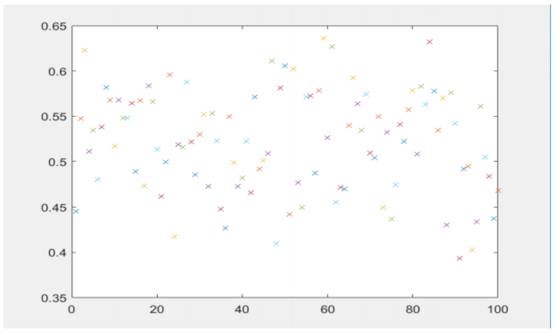
angry



happy



neutral



surprise
figure 4.1.1 testing result for least square

As you can see, every point locates above the X-axis. It means every expression image can be recognized and classified correctly. So at least for these 400 testing pictures, least square algorithm is highly accurate.

But after we do more tests, we will find that result is inaccurate in some situation. We think the reason is that if there is an obvious difference between the distance from camera to our face when we train and test data, it will cause result become inaccurate. For example, we want to test the algorithm of surprise. If we keep the distance from camera to our face a little bit longer when we capture testing images, the result is shown in figure 4.1.2. As this figure show, the result will no longer be kept at 100 percent.

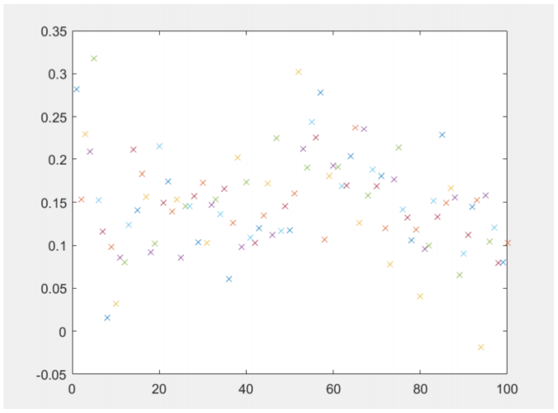


figure 4.1.2 more testing result

4.2 perceptron

There are all testing pictures for perceptron in this section. From figure 4.2.1 to 4.2.4, pictures respectively represent the testing result for angry, happy, neutral and surprise. We do numerous tests and only show part of pictures to explain.

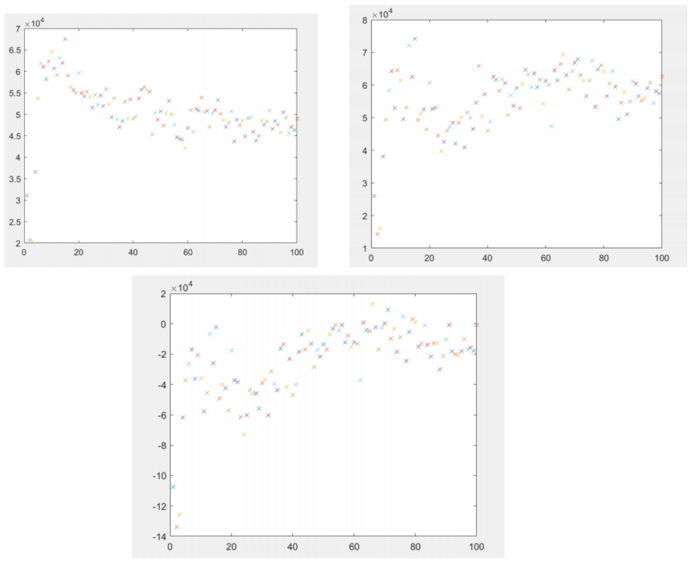


figure 4.2.1 testing result of angry

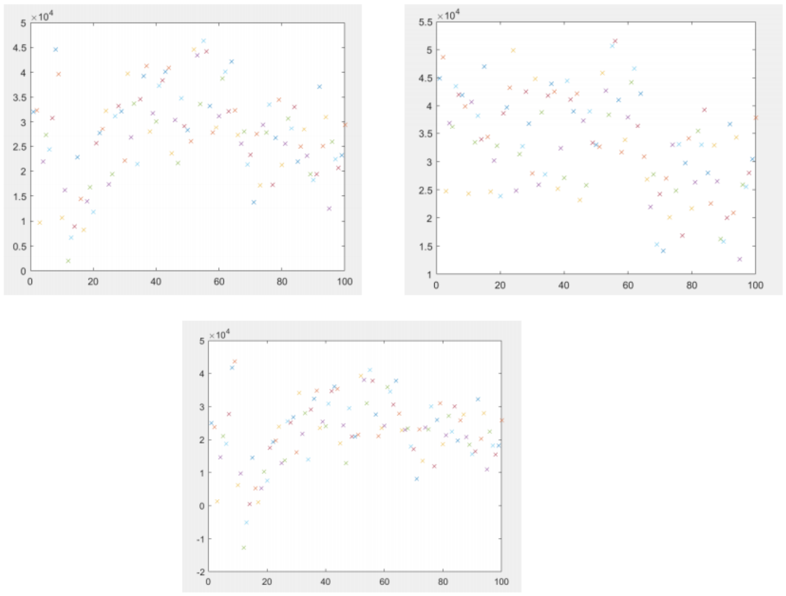


figure 4.2.2 testing result of happy

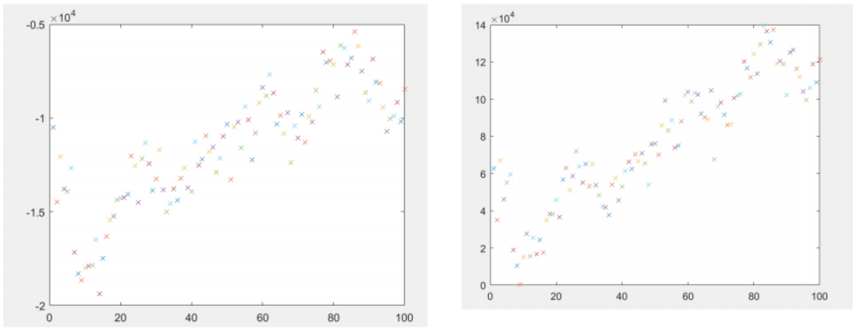


figure 4.2.3 testing result of neutral

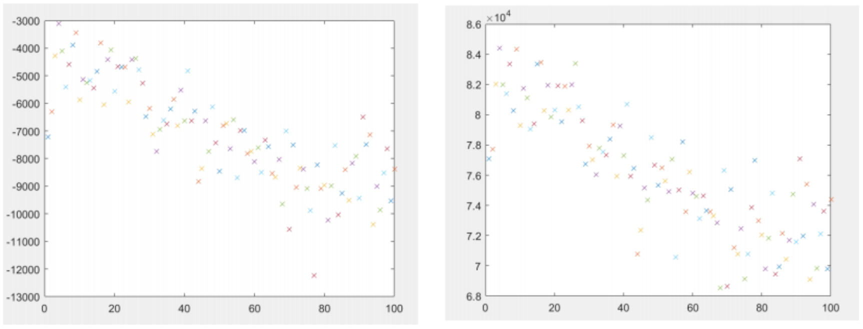


figure 4.2.4 testing result of surprise

We keep the same testing method and evaluation standard with least square algorithm. As you can see from these pictures, perceptron algorithm can keep 100 precent accuracy in some pictures, all points locate above the X-axis. But in some situation almost all point locate below the X-axis, it means almost all expression can not be classified correctly.

Obviously, the accuracy of perception algorithm always fluctuates in our testing process compared with least square algorithm. Because for this algorithm, we need to use our data set to iterate W. And when we design our code, we choose to select data randomly from data sets. It means for every training process, the training data order is different. So every whole iteration process is different and it will cause the W that has been iterated must have the error.

4.3 SVM

In this section, we not only carry out test for SVM, but also compare the accuracy between least square and SVM by using the same testing data. Firstly, there are four testing results for different expression shown in figure 4.3.1.

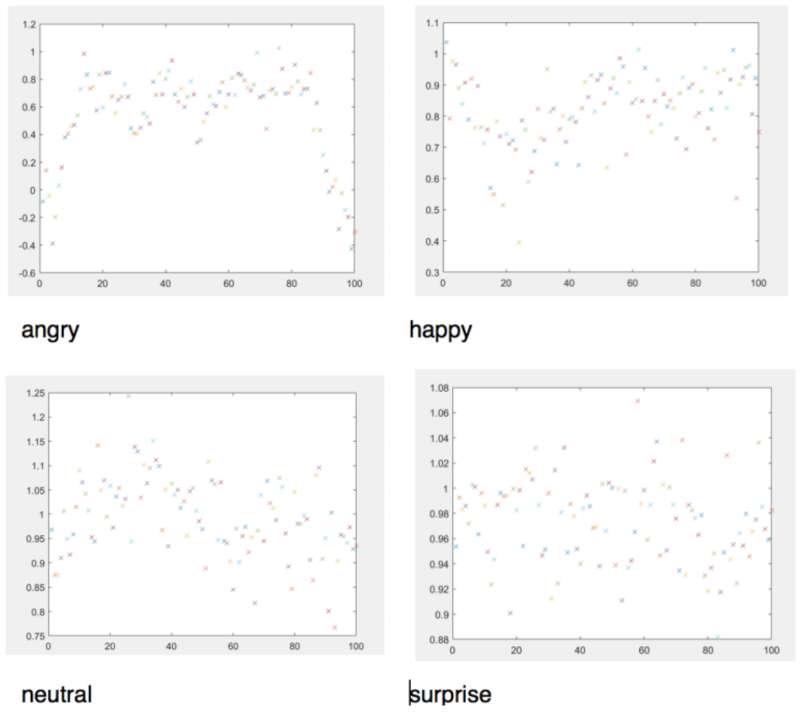


figure 4.3.1 testing result for SVM

In figure 4.3.2, we show the result of using the same data to test another algorithm least square. So these two algorithms can be compared through testing pictures.

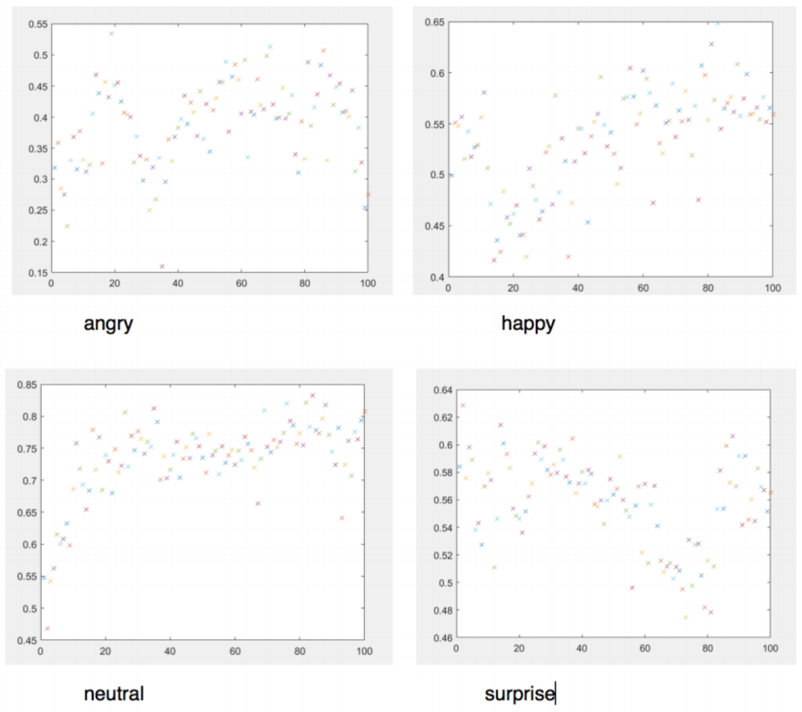


figure 4.3.2 testing results for least square by using same testing data

Basically, SVM can keep high accurate. In expression happy, neutral and surprise, the accuracy of classification is 100 percent. Only in expression angry, there is some error happened in classification. Compared these two algorithms from testing picture, they both can keep high accurate. But we can find that for same testing data, ordinate of testing points in SVM are greater than least square. It means variable res-angry, res-happy, res-neutral and res-surprise in SVM are greater than least square.

5 Reference

[Bishop, 2006] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
[Murphy, 2012] Murphy, K. P. (2012). Machine Learning. MIT Press.
[Olah, 2014a] Olah, C. (2014a). http://colah.github.io/posts/2014-07-Conv-Nets-Modular/.
[Olah, 2014b] Olah, C. (2014b). http://colah.github.io/posts/2014-07-NLP-

RNNs-Representations/.
[Olah, 2015] Olah, C. (2015). http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
[Jolly12191,2015]Jolly12191 http://blog.csdn.net/jolly12191/article/details/46428107