Hand-Written Character Recognition

MCEN 5125 Project #2

Hanwen Zhao

Abstract

In this report we will describe how to create classifiers for hand-written digits by linear programming (LP) optimization via CVX Matlab tool which create 45 hyper-planes. In addition, we will compare the performance with increased or decreased feature sizes.

1 Introduction

Numerical optimization is an extremely powerful tool for solving big and complex problem that would be difficult or costly to solve by conventional methods. In this report, we will solving a hand-written character recognition problem by formulate is as a Linear Programming(LP) problem in CVX modeling language so that we can use the state-of-the-art solvers that are available to solve it efficiently. In the following subsections, we will introduce the database we used for this project and the CVX modeling system for convex optimization. Then, we will describe how to formulate it as a LP problem and solve it in Matlab with CVX solvers. Last but not least, we will discuss the performance between different weighting parameter value and different feature sizes.

1.1 MNIST Database of Handwritten Digits

The training database we will use is the MNIST database of handwritten digits available at http://yann.lecun.com/exdb/mnist/. The MNIST database contains a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits in the database have been size-normalized and centered in a fixed-sized image. Professor Ruben has converted the MNIST database into a Matlab mat file for us to work with.

Figure 1: Example of database image

Figure 1 shows one of the 60,000 images in the MNIST database. Noticed that all the images have the same size with 28×28 pixels. One of the simplest ways to extract features from one image is to store every single pixels in the image as a feature. Therefore, there are 784 features for a 28×28 pixels image.

1.2 CVX Modeling System

CVX is a Matlab-based modeling system for convex optimization. It turns Matlab into higher computer language, allowing constrains and objectives to be specified using standard Matlab expression syntax [1],[2]. For example, assume we have the following convex optimization model:

$$\begin{array}{ll} \text{minimize} & \|Ax-b\|_2 \\ \text{subject to} & Cx=d \\ & \|x\|_\infty < e \end{array} \qquad (1)$$

With CVX tool installed, we can simply setup the LP problem as following:

```
cvx_begin
    variable x(n)
    minimize( norm( A * x - b, 2 ) )
    subject to
        C * x == d
        norm( x, Inf ) <= e
cvx_end
```

2 LP Formation

2.1 Binary Classification Example

In order to explain how classification works, let us start with a simple binary categories classifier. Assume that we have "The Zoo" with only two kinds of animals: Elephants and Giraffes[3].

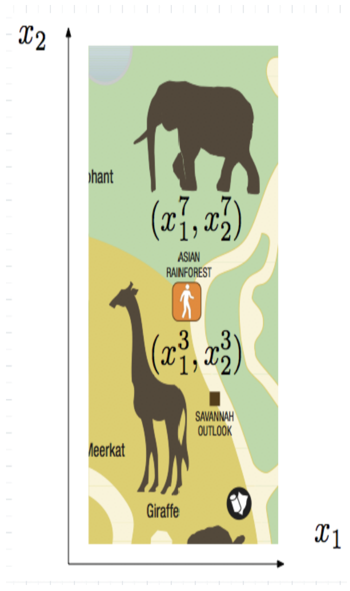

Figure 2: Example of "The Zoo" Classification[3].

As shown in Figure 2, the coordinate of the "unknown" animal i in the zoo: (x_1^i, x_2^i), we want to find a hyperplane that separate animals into their categories. Assume the hyperplane we found is defined as $a^T x - b = 0$.

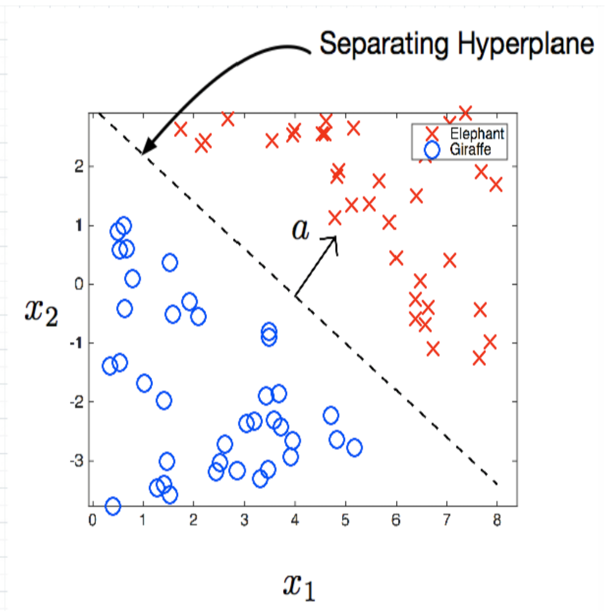

Figure 3: Hyperplane separates elephant and giraffe[3].

In the Figure 3, the elephant is defined as $a^T x - b > 0$, shows as red crosses in the figure; the giraffe is defined as $a^T x - b < 0$, shown as blue circles.
Now, let us to work on the LP formation to find the hyperplane we needed for classifier. Here are the assumptions we made for this example:

- Assume 2 features($x_1 \& x_2$)
- Assume m data points for training set
- Assume the number of elephants is q ($q \leqslant m$)
- Assume the number of giraffe is r ($r \leqslant m$)
- $u \& v$ are slack variables ($u_i, v_i > 0$)

Then we can write the problem into the following:

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{q} u_i + \sum_{i=1}^{r} v_i \\ \text{subject to} & a^T x^1 - b \geqslant 1 - u_1 \\ & a^T x^2 - b \geqslant 1 - u_2 \\ & \vdots \\ & a^T x^q - b \geqslant 1 - u_q \\ & a^T x^{q+1} - b \leqslant -(1 - v_{q+1}) \\ & a^T x^{q+2} - b \leqslant -(1 - v_{q+2}) \\ & \vdots \\ & a^T x^{q+r} - b \leqslant -(1 - v_{q+r}) \end{array} \tag{2}$$

We can simplify the LP formation by writing $a^T x - b$ in the format of $x^T a - b$ since it is a scalar, it turn s the LP into:

$$\begin{array}{ll} \text{minimize} & 1^T u + 1^T v \\ \text{subject to} & X_e a - b \geqslant 1 - u \\ & X_g a - b \leqslant -(1-v) \\ & u > 0 \\ & v > 0 \end{array} \tag{3}$$

where

$$X_e = \begin{bmatrix} x_1^1 & x_2^1 & \dots & x_n^1 \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^q & x_2^q & \dots & x_n^q \end{bmatrix} = \begin{bmatrix} (x^1)^T \\ (x^2)^T \\ \vdots \\ (x^q)^T \end{bmatrix} \tag{4}$$

and

$$X_g = \begin{bmatrix} (x^{q+1})^T \\ (x^{q+2})^T \\ \vdots \\ (x^{q+r})^T \end{bmatrix} \tag{5}$$

The analyze above has assumption that we do not have any outliers. However, in reality, many data we have will have some outliers. These outliers will affect the the accuracy of hyperplanes we calculated. In order to reduce the error introduced by outliers, we want to create a marginal area for each hyperplance. A marginal area can be defined as:

$$Margin = \frac{2}{\|a\|_2} \tag{6}$$

Minimize $\|a\|_2$ would maximize the marginal area. Figure 4 shows the marginal area for "The Zoo" example, with larger margin, given more robustness to outliers. Therefore, our final LP formation become:

$$\begin{array}{ll} \text{minimize} & \|a\|_2 + \gamma(1^T u + 1^T v) \\ \text{subject to} & X_e a - b \geqslant 1 - u \\ & X_g a - b \leqslant -(1-v) \\ & u > 0 \\ & v > 0 \end{array} \tag{7}$$

Here, γ is the weighting parameter to adjust the robustness to outliers. We need to try different γ value to see which one gives us the best result.

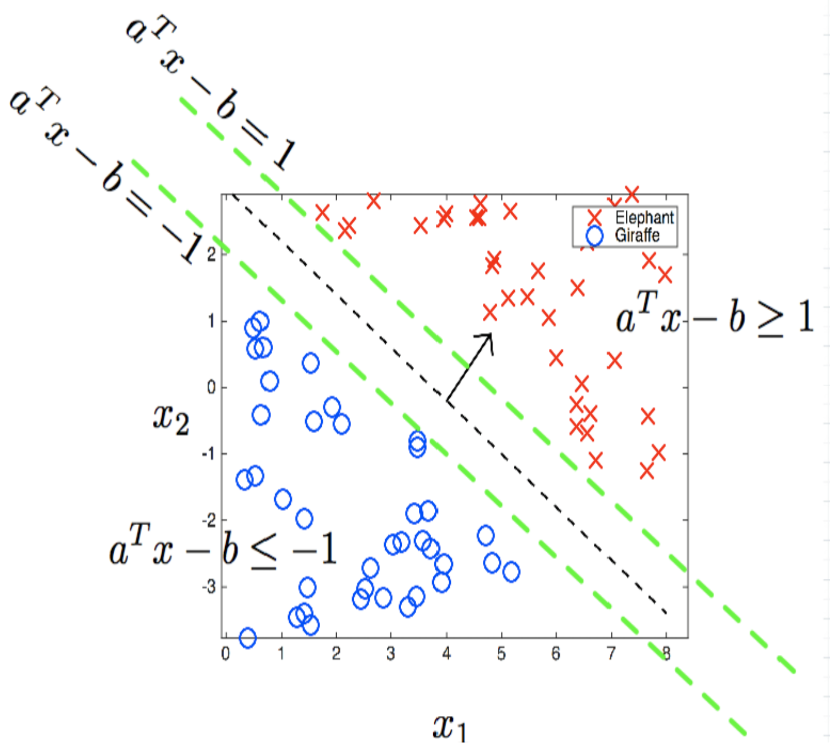

Figure 4: Example of "The Zoo" classification marginal area[**?**].

2.2 Hand-Written Character Recognition

From the previous section, we have introduced how to create a binary classifier like "The Zoo" example. Now, let us work on the hand-written character recognition problem. There are two options to create multiclass classifier:
Option 1: ONE vs. ALL classifiers (10 total)

- 0 vs. All the rest
- 1 vs. All the rest

  ⋮

- 9 vs. All the rest

Option 2: Pairwise classifiers (45 total)

- 0 vs. 1
- 1 vs. 2

  ⋮

- 0 vs. 9
- 1 vs. 2
- 1 vs. 3

⋮

- 8 vs. 9

In this report, we will introduction the pairwise option since it take less time for training and give us more accuracy result. From the previous section, we learned how to create binary classifier. With pairwise method, we basically only need to repeat the the same procedure for 45 times. The LP formation can be write as following:

$$\begin{array}{ll} \text{minimize} & \|a\|_2 + \gamma(1^T u + 1^T v) \\ \text{subject to} & X_i a - b \geqslant 1 - u \\ & X_j a - b \leqslant -(1 - v) \\ & u > 0 \\ & v > 0 \end{array} \tag{8}$$

where i and j represent the pair of character we want to work with. For example, if we want to create a classifier for 0 and 1, i would 1 and j would be 1 in this case.

3 Numerical Solution

3.1 Without Feature Modification

Since γ can be any number in the range of $0 < \gamma < \infty$. We will started different values for γ from log space, for example 1, 10, 100 etc. The following table shows overall accuracy with different γ values.

γ value vs. Accuracy	
γ value	Overall Accuracy
0.00005	0.9450
0.1	0.9173
0.5	0.9170
0.7	0.9172
10	0.9161
12	0.9160
200	0.9156
250	0.9163
300	0.9159
500	0.9138
800	0.9135
900	0.9130
3000	0.9102
5000	0.9100
8000	0.9099
10000	0.9095
50000	0.9075
100000	0.9066
1000000	0.9052

From the table above, we can see there is a trend where the overall accuracy reduces as the γ value increases. This indicates with small marginal area, the resulting hyperplanes give us the best result, also indicates the database have very limited outliers.

3.2 Feature Modifications

From previous sections, we were be able to achieve 94.5% accuracy for overall character recognition. This was achieved by taking all 784 pixels in $28x28$ image. There are two options we can change the feature for each character and try to improve the overall accuracy.
Option 1: Reduce unnecessary feature size There are several methods to reduce the image size in order to reduce the number of features.

- cut of the border pixels since they don't contain any useful information
- resize the image to smaller size

Option 2: Increase feature size

- adding image features after high-pass and low-pass filter

3.2.1 Boarder Cutting

First let us try to cut out the border pixels since they don't contain any useful information. This should help us to reduce the time needed for training. After some image testing with database. Without losing information on the character, the smallest size we can have is $24x24$. Figure 5 shows the image comparison before and after the boarder cutting.

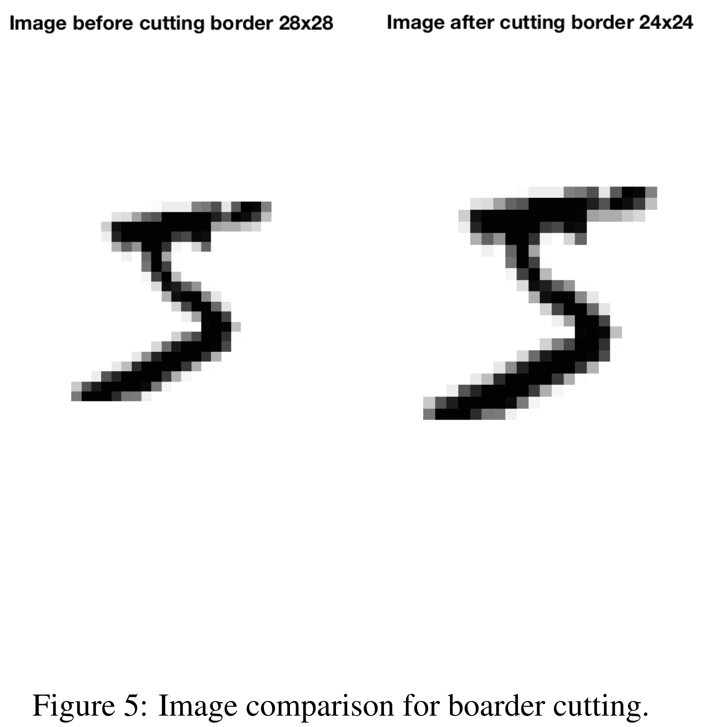
Image before cutting border 28x28
Image after cutting border 24x24

Figure 5: Image comparison for boarder cutting.

Performance Comparison for Boarder Cutting		
With Boarder Cutting	Overall Accuracy	Time
No	94.50%	5989.60 s
Yes	91.82%	7504.71 s

From the table above, we can see that after cutting the boarder pixels, we didn't get any improvement on overall accuracy. The time for less feature is longer than the original training, it was due to running the training process on different computers. Theoretically, we expect less training time for smaller feature size.

3.2.2 Image Resize

With boarder cutting, we are limit to reduce the image size to $24x24$. Now, we can use image resize method, which allow us to reduce the image to even smaller size but maintain the shape. Figure 6 shows the comparison between the original image and resized image. Notice that we have reduced the image size to $20x20$ pixels.

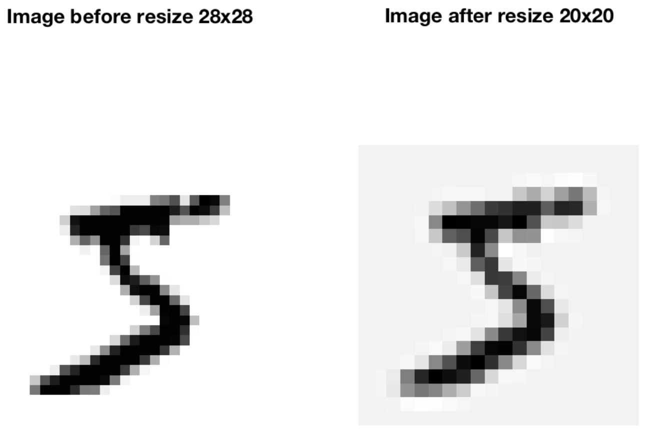

Figure 6: Image comparison for image resize.

Performance Comparison for Resizing		
With Resizing	Overall Accuracy	Time
No	94.50%	5989.60 s
Yes	88.90%	4733.50 s

From the results table as shown above, with less feature to calculate. The training time has decreased 20.97%. However, it also reduces the overall accuracy from 94.50% to 88.90%. This may due to the information lost during the resizing processing. The resizing method we used here is 'bicubic', which the output pixel value is a weighted average of nearest $4x4$ neighborhood.

3.2.3 Image Filters

Now, lets play with the problem where we are going to use low-pass and high-pass filter to create more features for each image. Low-pass image is simply a Gaussian filter and high-pass filter can be achieved by using the original image minus the low-pass filtered image. Figure 7 shows the comparison between original image, low-pass filtered image and high-pass filtered image. Now, we have increased the feature size from 784 to $784x3 = 2352$.

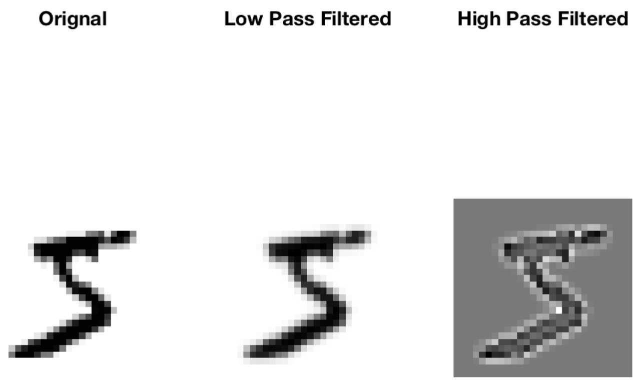

Figure 7: Image comparison for low/high pass filters.

Performance Comparison for Image Filters		
With Filters	Overall Accuracy	Time
No	94.50%	5989.60 s
Yes	92.33%	81529 s

The table above shows the overall accuracy and time used for training with and without the image filters. As we can see, with larger feature size, it takes significant more time for training. In our case, we expand the feature by three time, but the time required for training increased 12.6 time. Theoretically, we are expecting higher accuracy with larger feature size. However, our testing result was not satisfying. It may due to the σ value we used for Gaussian filters. We did not try other σ values since the training processing is extremely time consuming.

4 Conclusion

In conclusion, we presented the using Linear Programming. We went through the process of analyzing the program, solving a small but similar "The Zoo" problem. Then we expand the binary classification into multiclass classification by creating multiple classifiers. As the numerical result, the best we can get is 94.5%, with the $\gamma = 0.00005$. Next, we compared the performance result with increased/decrease feature size data. Unfortunately, we didn't get any improvement on it. For future consideration, we recommend to increasing the feature size with more information in it. For example, change the σ value for Gaussian filter and other image processing techniques such as image sharpening or edge detection etc.

References

[1] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[2] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[3] Shalom Ruben. Lecture notes optimal design, April 2018.

5 Appendix

5.1 MATLAB Code

```
% MCEN 5125
% Project 2
% Hand-Written Character Recognition
% Hanwen Zhao
% MEID: 650-703
tic
% load training data
load('mnist.mat')
% uncomment for less features(boarder cutting)
%images = images_play1;
% uncomment for less features(resizing)
%images = images_play3;
% uncomment for more features
%images = images_play2;
% allocate memory to store numbers of image
numbers = zeros(1,10);
% calculate numbers of images for each "number"
 for i = 1:60000
     numbers(labels(i)+1) =  numbers(labels(i)+1) + 1;
 end
% allocate memoery
[~,n] = size(images);
for i = 1:10
    x{1,i} = zeros(numbers(i), n);
end
% store all features into cell array
cindex = ones(1,10);
for i = 1:60000
     x{1,labels(i)+1}(cindex(labels(i)+1),:) = images(i,:);
     cindex(labels(i)+1) = cindex(labels(i)+1) + 1;
end
% Weighting paramater
gamma = 0.00005;
% initilize cell arrays for A B
A = cell(9,10);
B = cell(9,10);
% setup cvx to solve LP problems
% loop through all 45 classifier
for i = 1:9
     for j = i+1:10
         cvx_begin
             variables a(n,1) b u(numbers(i),1) v(numbers(j),1);
```

```
            minimize(norm(a)+gamma*(ones(1,numbers(i))*u+ones(1,
                numbers(j))*v));
            subject to
            x{1,i}*a - b*ones(numbers(i),1) >= ones(numbers(i),1)
                 - u;
            x{1,j}*a - b*ones(numbers(j),1) <= -ones(numbers(j)
                ,1) - v;
            u > 0;
            v > 0;
        cvx_end
        % save A and B in upper triangle format
        A{i,j} = a;
        B{i,j} = b;
    end
end
toc
```

```
% MCEN 5125
% Project 2
% Hand-Written Character Recognition
% Hanwen Zhao
% MEID: 650-703

% load data
%load('0.00005.mat')
% create container for classified label
classifiedLabels = zeros(10000,1);
% uncomment for boarder cutting
%images_test = images_test_play1;
%A = Ap1; B = Bp1;
% uncomment for filters
images_test = images_test_play2;
A = Ap2; B = Bp2;
% uncomment for resizing
%images_test = images_test_play3;
%A = Ap3; B = Bp3;
% use classifer to identify the label
for i = 1:10000
    p = 1; q = 10;
    for n = 1:9
        if double(images_test(i,:)) * A{p,q} - B{p,q} >= 0
            q = q - 1;
            if n == 9
                classifiedLabels(i) = p - 1;
            end
        else
```

```
            p = p + 1;
            if n == 9
                classifiedLabels(i) = q - 1;
            end
        end
    end
end

% allocate memory to store numbers of image
test_numbers = zeros(1,10);
% calculate numbers of each "number"
 for i = 1:10000
     test_numbers(labels_test(i)+1) =  test_numbers(labels_test(i)
        +1) + 1;
 end
 % allocate memory for accuracy
accuracy = zeros(1,10);
% calcuate accuracy
for i = 1:10000
    if labels_test(i) == classifiedLabels(i)
        accuracy(labels(i)+1) = accuracy(labels(i)+1) + 1;
    end
end
fprintf('The accuracy for each number is ')
accuracy = (accuracy./test_numbers)'
fprintf('The total accuracy is % 4.6f.\n', mean(accuracy))
```

```
% MCEN 5125
% Project 2
% Hand-Written Character Recognition
% Hanwen Zhao
% MEID: 650-703

% load training data
load('mnist.mat')
% allocate memory
images_play1 = int32(zeros(60000, 576));
images_play2 = int32(zeros(60000, 784*3));
images_play3 = int32(zeros(60000, 400));
images_test_play1 = int32(zeros(10000, 576));
images_test_play2 = int32(zeros(10000, 784*3));
images_test_play3 = int32(zeros(10000, 400));
% reduce images and images_test size to 24*24
for i = 1:60000
    temp = reshape(images(i,:),28,28);
    temp = temp(3:end-2,3:end-2);
    temp = reshape(temp,1,24*24);
    images_play1(i,:) = temp;
end
for i = 1:10000
    temp = reshape(images_test(i,:),28,28);
    temp = temp(3:end-2,3:end-2);
    temp = reshape(temp,1,24*24);
    images_test_play1(i,:) = temp;
end
% use image resize to reduce the size to 20*20
% reduce images and images_test size to 24*24
for i = 1:60000
    temp = reshape(images(i,:),28,28);
    temp = imresize(temp,0.7);
    temp = reshape(temp,1,20*20);
    images_play3(i,:) = temp;
end
for i = 1:10000
    temp = reshape(images_test(i,:),28,28);
    temp = imresize(temp,0.7);
    temp = reshape(temp,1,20*20);
    images_test_play3(i,:) = temp;
end
% increase image and image_test size by applying low pass and high pass
% filter
for i = 1:60000
    temp = reshape(images(i,:),28,28);
    % Low-Pass filter(Gaussian Filter)
    tempLow = imgaussfilt(temp);
    % High-Pass filter can be achieved by originalImage - lowPassImage
    tempHigh = temp - tempLow;
    % reshape to original size
    temp = reshape(temp,1,784);
    tempHigh = reshape(tempHigh,1,784);
    tempLow = reshape(tempLow,1,784);
    tempHigh = reshape(tempHigh,1,784);
    % concatenate features
    tempF = [temp,tempLow,tempHigh];
    images_play2(i,:) = tempF;
end
for i = 1:10000
    temp = reshape(images_test(i,:),28,28);
    % Low-Pass filter(Gaussian Filter)
    tempLow = imgaussfilt(temp);
    % High-Pass filter can be achieved by originalImage -
       lowPassImage
    tempHigh = temp - tempLow;
    % reshape to original size
    temp = reshape(temp,1,784);
    tempLow = reshape(tempLow,1,784);
    tempHigh = reshape(tempHigh,1,784);
    % concatenate features
    tempF = [temp,tempLow,tempHigh];
    images_test_play2(i,:) = tempF;
end
```