1. (a) The first 10 basis learned from MNIST train data is shown as Figure 1.

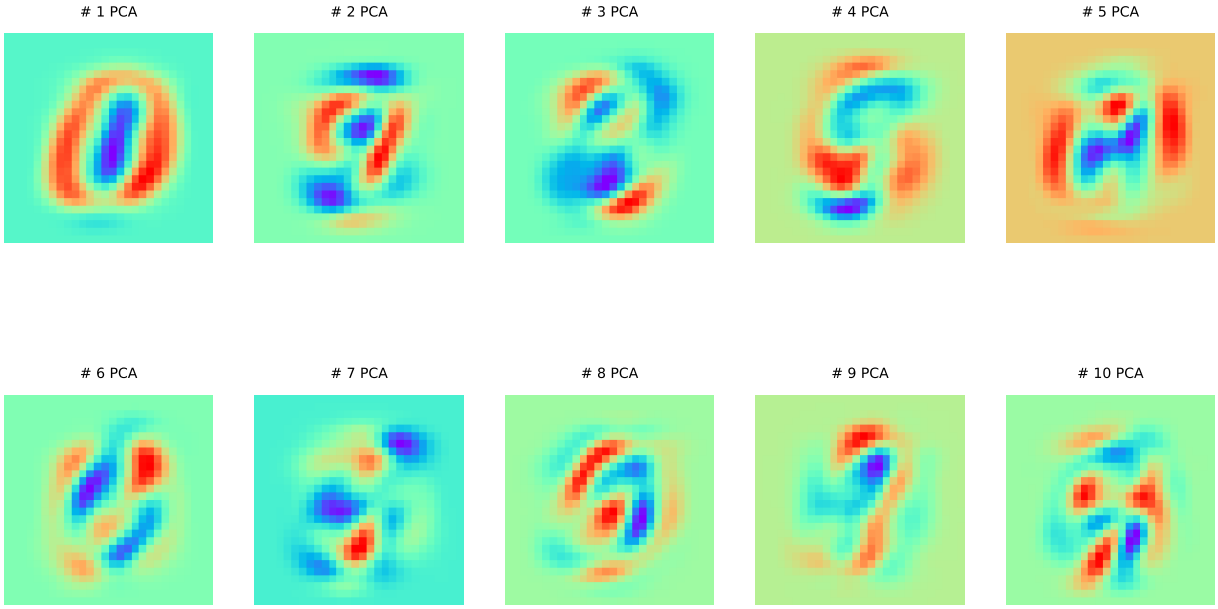

Figure 1: Top 10 Principle Components

(b) One example of the hand-written digit in MNIST that is correctly classified and one example that fails are shown in Figure 2

Figure 2: Left: Correctly Classified. Right: Incorrectly Classified

One example of the SVHN that is correctly classified and one example that fails are shown in Figure 3.

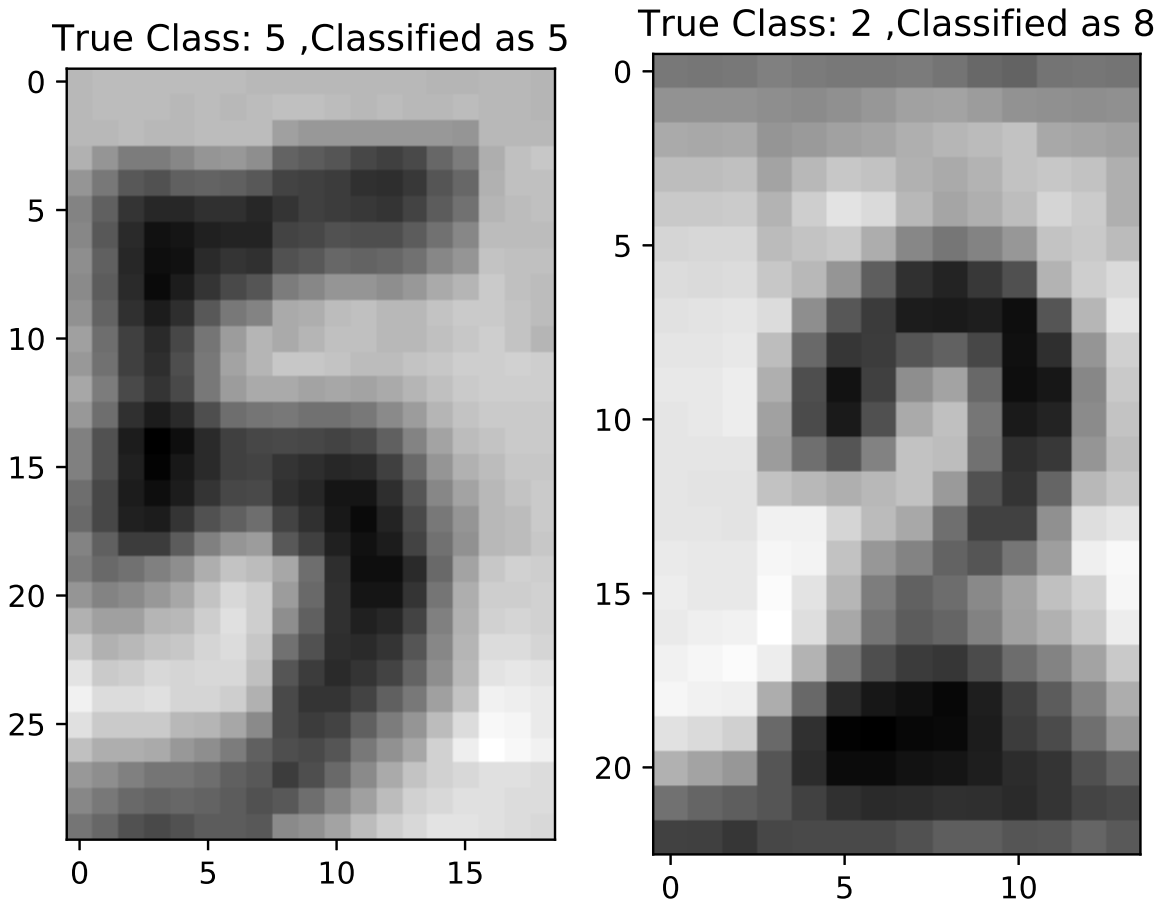

Figure 3: Left: Correctly Classified. Right: Incorrectly Classified

(c) First, we do an analysis based on a subset of the training data. We select first 300 images from SVHN dataset, and retrieve 576 letters to be classified. By finding 3 nearest neighbors with top 25 PCA, we correctly classify 93 images. Thus, the error rate is 83.85%.
We also classify all 10000 test images from MNIST dataset. By finding 3 nearest neighbors with top 25 PCA, we correctly classify 9731 images. Thus, the error rate is 2.69%.
We found that even if we have good performance on MNIST data set, the classification performance is poor on SVHN dataset. First of all, letters in SVHN dataset has lower resolution than MNIST dataset. After up-sampling to match 28×28 template in MNIST, the test image is really blurred. Secondly, images in MNIST dataset have black background and white letter. However, in the SVHN dataset, color of both letter and background varies a lot. This may generate a lot of noise for classification.

2. The Cathedral in the question is Orvieto Cathedral (Duomo di Orvieto) in Orvieto, Terni, Italy.
   Because the photo given in the question is a typical Gothic architecture, we first obtain a list of 403 historical Cathedrals, Churches and Basilicas in Italy. Then we use a Flicker Crawler to search images of facades of each potential names. We omit cathedrals with very few images results, by assuming those cathedrals are less visited. After that process, we have 30 cathedrals that are tagged more than 200 times over the last 10 years on Flicker.
   Since there are many outliers in the search result, we choose the top 50 relevant images to find matches with SIFT features. We first obtain number of match features, whose error is less than a threshold. Then we look at the image with the most match.

```
Find match from photos of Bologna_Cathedral
Max match is 23
Find match from photos of Genoa_Cathedral
Max match is 13
Find match from photos of Pisa_Cathedral
Max match is 5
Find match from photos of Orvieto_Cathedral
Max match is 88
Find match from photos of Muro_Cathedral
Max match is 3
Find match from photos of Messina_Cathedral
Max match is 9
Find match from photos of Lucca_Cathedral
Max match is 5
Find match from photos of Basilica_of_st_Nicholas
Max match is 9
Find match from photos of Basilica_of_Santa_Maria_Maggiore
Max match is 6
Find match from photos of Amalfi_Cathedral
Max match is 4
Find match from photos of Basilica_of_st_Andrew
Max match is 5
Find match from photos of Parma_Cathedral
Max match is 6
Find match from photos of Assisi_Cathedral
Max match is 3
Find match from photos of st_John_Lateran
Max match is 4
Find match from photos of Florence_Cathedral
Max match is 9
Find match from photos of Porto_Cathedral
Max match is 4
Find match from photos of Verona_Cathedral
Max match is 3
Find match from photos of Basilica_of_San_Nicola
Max match is 4
Find match from photos of Siena_Cathedral
Max match is 5
Find match from photos of Church_of_San_Pietro
Max match is 4
Find match from photos of Palermo_Cathedral
Max match is 4
Find match from photos of Scala_Cathedral
Max match is 6
Find match from photos of Como_Cathedral
Max match is 6
Find match from photos of Bari_Cathedral
Max match is 9
Find match from photos of Naples_Cathedral
Max match is 3
Find match from photos of Milan_Cathedral
Max match is 3
Find match from photos of Ferrara_Cathedral
Max match is 6
Find match from photos of st_Mark_Basilica
Max match is 3
Find match from photos of Iglesias_Cathedral
Max match is 5
Find match from photos of Latina_Cathedral
Max match is 7
Classified match is Orvieto_Cathedral
(EECS504) zixu@zixuzhang-Ubuntu:~/Extra_Disk/GoogleDrive/UMich Academic/Fall 2018/EECS504/HW3/HW_3/Q2$
```

Figure 4: Sample Output

Appendix

Code for PCA

```
def Get_PCA(train_data, num_basis):
'''
get first num_basis of principle components in train_data
Args:
train_data: input data
num_basis: take first num_basis principle basis
'''
    num_data, dim_data_h, dim_data_w=train_data.shape
    train_data_3D = train_data.astype(np.float)
    data_class_2DMat = np.reshape(train_data_3D,(num_data,
       dim_data_h*dim_data_w))
    '''
    make data to be centered at 0
    '''
    data_mean = np.mean(data_class_2DMat,axis=0)
    train_data_shifted = data_class_2DMat-data_mean
    '''
    Use svd to get PCA of the data
    '''
    U,S,Vh = np.linalg.svd(train_data_shifted,full_matrices=False)
    '''
    Unit vector of principle direction
    '''
    PCA_dir = Vh.T
    '''
    US is the score of each data
    '''
    PCA_socre = np.matmul(U,np.diag(S))
    '''
    extract first num_basis basis.
    '''
    PCA_dir_out = PCA_dir[:,:num_basis]
    PCA_socre_out = PCA_socre[:,:num_basis]
    return PCA_dir_out, PCA_socre_out, data_mean
```

Code for KNN

```
def kNN(img2classify, train_PCA, train_PCA_socre,
     train_mean, train_label, num_neighbor):
'''
k-nearest neighboor
Args:
img2classify: [28*28] data to be label
'''

    img2data = np.reshape(img2classify, [1,train_PCA.shape[0]])
    img2data_shifted = img2data-train_mean
    dataScore = np.matmul(img2data_shifted, train_PCA)
    '''
    #get Eculidian distance between data and all train sample
    '''
    diff  = train_PCA_socre - dataScore
    Ecu_dis = np.linalg.norm(diff, axis=1)
    sort_dis_idx = np.argsort(Ecu_dis)
    '''
    #build historgram of k nearest neighbor
    '''
    k_hist = np.zeros(10)
    for i in range(num_neighbor):
    cur_label = train_label[sort_dis_idx[i]]
    k_hist[cur_label]+=1
    data_label = np.argmax(k_hist)
    return data_label
```

Code for SVHN Test

```
def test_SVHN(train_PCA, train_mean, train_PCA_socre, train_label,
    test_box, test_data_path, num_neighbor, num_to_test):
    print("\n****** Start testing SVHN dataset with "+str(
       num_neighbor)+" nearest neighbors ******")
    Img_data_all = test_box.getAllDigitStructure_ByDigit()
    print("****** Loaded "+str(len(Img_data_all))+" images from
       SVHN dataset ******")
    #initialize some values for analysis
    num_total_test = 0
    num_false_class = 0
    test_histogram =[]
    test_false_histogram=[]
    if num_to_test == -1:
        num_to_test = len(Img_data_all)
```

```
    for cur_img_data in Img_data_all[:num_to_test]:
        cur_img_gray = etai.read(test_data_path+"/"+cur_img_data['
           filename'], flag=cv2.IMREAD_GRAYSCALE)
        height, width = cur_img_gray.shape
        cur_num_bbox = len(cur_img_data['boxes'])
        num_total_test+=cur_num_bbox
        for j in range(cur_num_bbox):
            #check bounding box does not lay outside of image
            cur_box = cur_img_data['boxes'][j]
            x_idx_1 = int(max(cur_box['left'],0))
            x_idx_2 = int(min(cur_box['left']+cur_box['width'],
               width))
            y_idx_1 = int(max(cur_box['top'],0))
            y_idx_2 = int(min(cur_box['top']+cur_box['height'],
               height))
            '''
            extract bounded image and resize to 28*28
            '''
            box_img = cur_img_gray[y_idx_1:y_idx_2, x_idx_1:
               x_idx_2]
            box_img_resize = resize_SVHN_img(box_img)
            '''
            test with knn
            '''
            box_class = kNN(box_img_resize, train_PCA,
               train_PCA_socre, train_mean, train_label,
               num_neighbor)
            if box_class == 0:
                box_class = 10
            '''
            save histogram
            '''
            cur_box_hist = cur_box
            cur_box_hist['filename'] = cur_img_data['filename']
            cur_box_hist['classify'] = box_class
            test_histogram.append(cur_box_hist)
            if box_class!=cur_box['label']:
                num_false_class+=1
                test_false_histogram.append(cur_box_hist)

    Error_rate = num_false_class/num_total_test
    print("****** Finish testing SVHN dataset. Totoal "+str(
       num_total_test)+" images tested. "+
    str(num_total_test-num_false_class)+" images are labeled
       correctly. Error rate = "+str(Error_rate*100)+"% ******")
    return Error_rate, test_histogram, test_false_histogram
```

```
50
51 def resize_SVHN_img(input_img):
52     bg_color = input_img[-1,-1]
53     output_img = etai.resize(input_img, width=28, height=28,
          interpolation=cv2.INTER_AREA)
54     if bg_color>100:
55         output_img = 255 - output_img
56     return output_img
```

Code for MNIST Test

```
1 def test_MNIST(train_PCA, train_mean, train_PCA_socre, train_label
     , test_img, test_label, num_neighbor, num_to_test):
2     print("\n****** Start testing MNIST dataset with "+str(
          num_neighbor)+" nearest neighbors ******")
3     num_test_data = test_img.shape[0]
4     test_img_3D = test_img.astype(np.float)
5     test_output_class = np.zeros(num_test_data)
6     test_output_TF = np.ones(num_test_data)
7     num_false_class = 0
8     if num_to_test==-1:
9         num_to_test = num_test_data
10    '''test all images'''
11    for i in range(num_to_test):
12        cur_class = kNN(test_img_3D[i,:,:], train_PCA,
              train_PCA_socre, train_mean, train_label, num_neighbor)
13        test_output_class[i] = cur_class
14        if cur_class != test_label[i]:
15            test_output_TF[i]=0
16            num_false_class+=1
17    Error_rate = num_false_class/num_to_test
18    print("****** Finish testing MNIST dataset. Totoal "+str(
          num_to_test)+" images tested. "+
19    str(num_to_test-num_false_class)+" images are labeled
          correctly. Error rate = "+str(Error_rate*100)+"% ******")
20    return Error_rate, test_output_class, test_output_TF
```