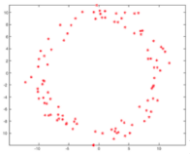
Xiaodong Huo Deep learning Hw1

1. Part 1
    1. Hand crafted feature are the properties derived from the image itself that are manually generated. For example, Gabor feature; HOG feature and LBP features.\
    2. Learned feature are automatically obtained from machine learning algorithm. For example: dictionary learning; Visual bag of words; K-means clustering, Gaussian clustering
    3. Learned features are effective at automatically solving a specific task. Learned feature extraction and classification outperform the hand-crafted feature extractions. However, we can't control the features extracted by the learned features, so that sometimes we produce good features but sometimes we do not. The hand-crafted features are always guaranteed to have the result we desired when we use the features to a specific use.
    4. PCA: The first component is to maximize the variance to have the point as spread as possible. The second component is $\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i (t_1)_{(i)}^2 \right\} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i (\mathbf{x}_{(i)} \cdot \mathbf{w})^2 \right\}$; and minimize residual; $1 - \sum_{i=1}^{k} \lambda_i / \sum_{k=1}^{n} \lambda_k$
    5. The max number of features LDA give you is 1 less than the number of classes; Feature number = #Classes -1
    6. LDA limitations: LDA performs bad on non-linear problems; LDA is unsupervised, where sometimes weak supervision is desirable; LDA is sensitive to overfit and LDA model is difficult to validate.
    7. 1. Limitations of PCA; PCA assumes the distribution is a Gaussian, works bad on non-Gaussian data; for example,  2. PCA is not optimal for discrimination.
    8. 1. Chi-Square distance matric; $D_{\chi^2} = \sum_i \frac{(h_1(i)-h_2(i))^2}{h_1(i)+h_2(i)}$ 2. Histogram Intersection; $D_{\cap} = 1 - \frac{\sum_i (min(h_1(i),h_2(i))}{min(|h_1(i)|,|h_2(i)|)}$ 3. L=2 or Euclidean Distance $D_{L2} = \sqrt{\sum_i (h_1(i) - h_2(i))^2}$
    9. $l_0 - norm$ captures the non-zero elements, thus, the minimizing the $l_0 - norm$ gives us the sparest solution: $(\ell^0): \quad \hat{\boldsymbol{x}}_0 = \arg\min \|\boldsymbol{x}\|_0 \quad \text{subject to} \quad A\boldsymbol{x} = \boldsymbol{y},$
    10. The problem of finding the sparest solution of $\hat{\boldsymbol{x}}_0 = \boldsymbol{x}$, which is an underdetermined system of linear equations is NP-hard and difficult to approximate. The $l_1 - norm$ on the other hand, can approximate $l_0 - norm$ if the solution x0 is sparse enough. The problem, $(\ell^1): \quad \hat{\boldsymbol{x}}_1 = \arg\min \|\boldsymbol{x}\|_1 \quad \text{subject to} \quad A\boldsymbol{x} = \boldsymbol{y}.$ can be solved using standard linear programming. In very sparse solution case, can be easily done by other techniques, e.g. Homotopy algorithms.
    11. Computationally expensive to calculate Euclidean distance repetitively; limitation of

types of data variables that we can use e.g.(categorical labels); The mean is greatly affected by outliers.

12. Nearest neighbor algorithm finds the closest train image. The distance is calculated by the sum of pixel difference between test image and training image. Where K-NN finds the k closest matches which are k closest training examples. For classification output, the output is the class with the most voting of its neighbors.
13. Visual bag of words features extraction: First, we perform texture recognition which is characterized by repetition of basic elements and textons to extract features. We then learned these features as "words". A new image can them be described as "words". The images can be classified by the biggest frequencies that a "visual words" appears.
14. We first partition the data set into k bins of equal sizes; For example, we can N data points and k bins. Thus, in each bin, we have N/k data points. In k-fold cross validation, we run k separate learning experiments. We then pick one of these bins as a testing bin and the remaining (k-1) bins as a training bin. We train the ML algorithm, then we test the performance on the test set. We run this k-fold cross validation k times and then we average the performance for the ten different hold out sets. So this taking longer but more accurate since we used all of our data for training and testing.
15. Dictionary learning has two steps, Sparse coding and updates the Dictionary. The learning algorithm alternates between inference and dictionary learning. We first find the sparse codes $h(x^t)$for all x^t in the training set. We then updates the dictionary: $\mathbf{A} \Leftarrow \sum_{t=1}^{T} \mathbf{x}^{(t)}\, \mathbf{h}(\mathbf{x}^{(t)})^\top$ $\mathbf{B} \Leftarrow \sum_{t=1}^{T} \mathbf{h}(\mathbf{x}^{(t)})\, \mathbf{h}(\mathbf{x}^{(t)})^\top$ The sparse coding object function is to minimize the dictionary and minimize the sparse representation of our training examples. $\min_{\mathbf{D}} \frac{1}{T} \sum_{t=1}^{T} \min_{\mathbf{h}^{(t)}} \frac{1}{2} \overbrace{||\mathbf{x}^{(t)} - \underbrace{\mathbf{D}\, \mathbf{h}^{(t)}}||_2^2}^{\searrow} + \underbrace{\lambda}_{\swarrow} \overbrace{||\mathbf{h}^{(t)}||_1}^{\swarrow}$; Therefore, Sparse coding is one step in sparse dictionary learning which finds a sparse representation of the input data.

Part2

My code has two parts: the 1st part:

#this code is the NN algorithm developed without using library; the output of this file is the count and correct number of matching.

#need to select the m (number of training image) first which is on line 13

The second part is:

#this code is the NN algorithm developed with library; for question2, run function:k23410(); print_img() and error(); for question3

run question3(); for question 4, run question4hog() and uncommon line 50, question4lbp() and uncommon line51 for 2.2 run cross(p,k);

And a bokeh plotting code;

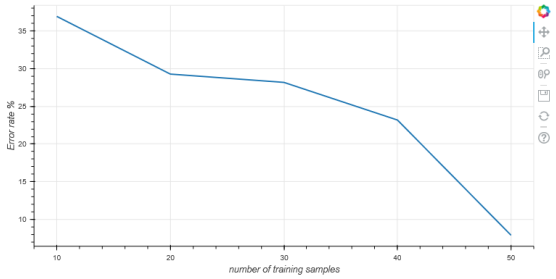


Figure 1: question1. NN

1. As we can see from this graph, as the number of training samples increase, we have a decreasing error rate. The code I developed for this part it's not very efficient, the total runtime was 1hour. When we have more training samples, the error rate decreases. This graph is plotted by bokeh
2. In the second question, I run the code for each k= [2,3,5,10] and the m = 10, 20, 30, 40, 50 []#(number of training image) iteratively. The data is exported from pandas to excel and plotted in excel. As we can see. For the same k, we have a decreasing error rate as m increases. For the same m, we have decreasing error rate as k increases too. However, that's not always the case. When k = 10, the error rate increases again. Thus, error rate does not always decreasing as k increases.

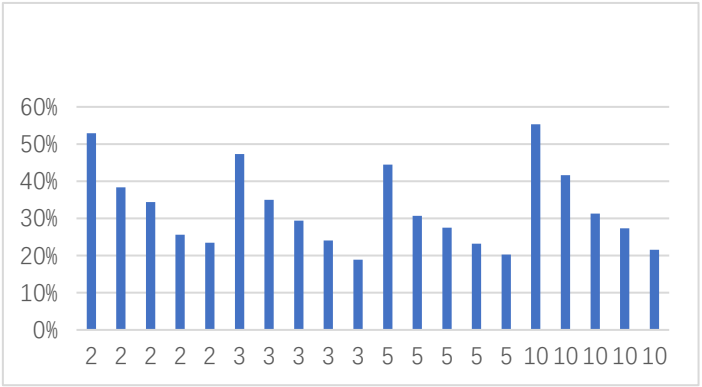


Figure 2: question2. KNN

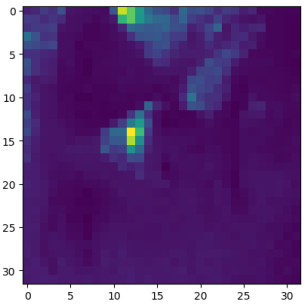


Figure3 the test Image

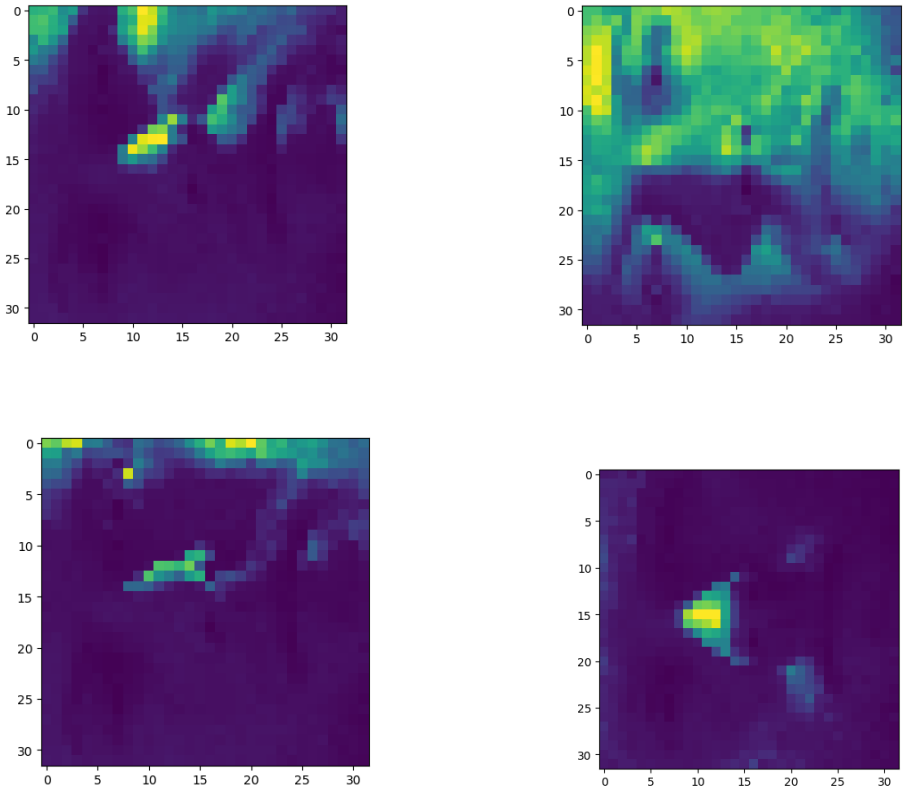


Figure4, mistaken images for the test Image

3. As we can see, the distance metric from p = 1, 3, 5, 10 is maximized when p= 3;

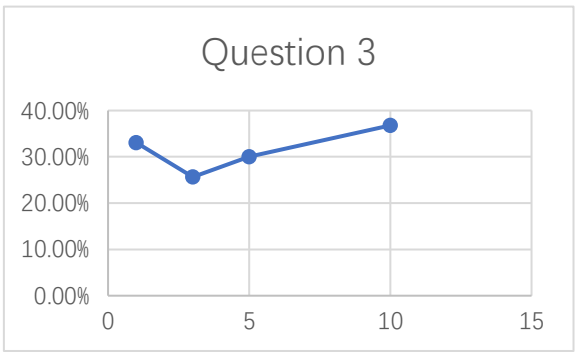


Figure 5 , different p and error rate

4. The error rate for Hog and LBP are:

p	hog
1	41.08
2	42.95

and for LBP is

p	lbp
1	4
2	5

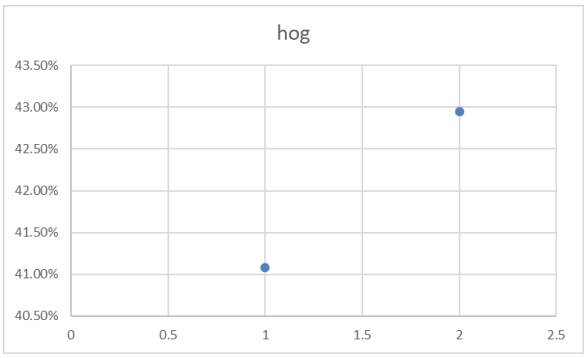


Figure 6 HOG error rate

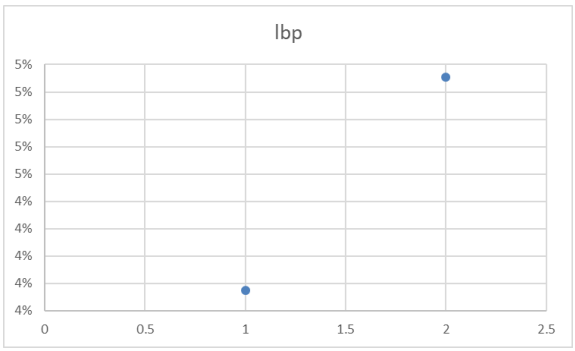


Figure 7 LBP error rate

5. The lowest error rate I achieved is 4% from p=1 lbp;

2.2 Validation set

From all the k,p combinations, I find the k= 2 and p = 5 has the highest score. Thus, we should maximize as k= 2 and p = 5.

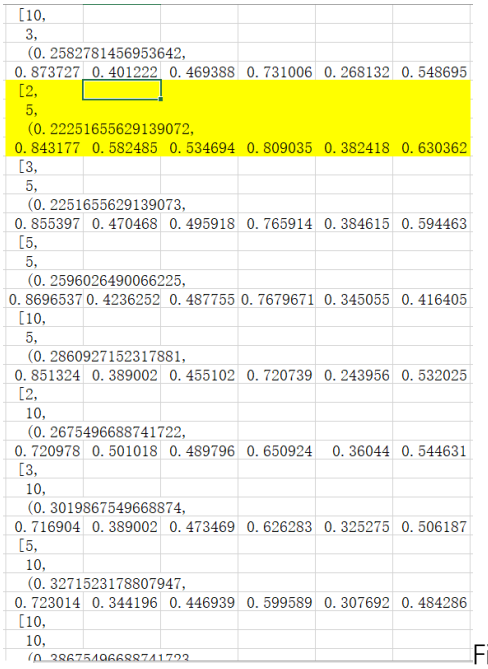

[10,
3,
(0. 2582781456953642,
0. 873727 0. 401222 0. 469388 0. 731006 0. 268132 0. 548695
[2,
5,
(0. 22251655629139072,
0. 843177 0. 582485 0. 534694 0. 809035 0. 382418 0. 630362
[3,
5,
(0. 2251655629139073,
0. 855397 0. 470468 0. 495918 0. 765914 0. 384615 0. 594463
[5,
5,
(0. 2596026490066225,
0. 8696537 0. 4236252 0. 487755 0. 7679671 0. 345055 0. 416405
[10,
5,
(0. 2860927152317881,
0. 851324 0. 389002 0. 455102 0. 720739 0. 243956 0. 532025
[2,
10,
(0. 2675496688741722,
0. 720978 0. 501018 0. 489796 0. 650924 0. 36044 0. 544631
[3,
10,
(0. 3019867549668874,
0. 716904 0. 389002 0. 473469 0. 626283 0. 325275 0. 506187
[5,
10,
(0. 3271523178807947,
0. 723014 0. 344196 0. 446939 0. 599589 0. 307692 0. 484286
[10,
10,
(0. 3867549668741723

Figure 8 validation test.

Reference: code for problem 1

```
############################Initialization
############################Question 1
#this code is the NN algorithm developed without using library; the output of this file is the count and correct number of matching;
#need to select the m (number of training image) first which is on line 13
from scipy.io import loadmat
import time
import collections
import numpy as np
from numpy.linalg import norm
import random
import pandas
import cv2
lst_m = [10, 20, 30, 40, 50]    # number of training image
m= lst_m[0]    # initialization of the nummber of training set

start = time.time()


import math
loadtemp = loadmat('YaleB_32x32.mat')
gnd = loadtemp["gnd"]    # 2414 X 1
fea = loadtemp["fea"]    # (2414, 1024)


fea_copy2 = np.copy(fea)


n_training = []
n_testing = []
unique, counts = np.unique(gnd, return_counts=True)
D = dict(zip(unique, counts))
for i in range(len(counts)):
    n_training.append(m)    # random.choice(m)
for i in range(len(counts)):
    diff_temp = int(D[i+1]) - int(n_training[i])
    n_testing.append(diff_temp)
# select index in training
############################################################
def list_gen(n):
    lst = []
    for i in range(int(n)):
        lst.append(i + 1)
    return lst
def rand_index():
    for Dic in D:
        card = list_gen(D[Dic])
        lst_training = []
        lst_testing = []
        for item in n_training:
            # temp1 = abs(int(Dic) - int(item))+1
            random.shuffle(card)
            lst_training.append(card[:int(item)])
            lst_testing.append(card[int(item):])
```

```
                break
            conn.append(lst_training)
            conn1.append(lst_testing)
        index_training = pandas.DataFrame(conn)      # index +1
        index_testing = pandas.DataFrame(conn1)    # index +1
        return conn, conn1, index_testing , index_training
#######training set
def segmen():
        x = []
        y = []
        m=0
        n=0
        for Dic in D:
            x.append(m)
            m += D[Dic]
            n = m
            y.append(n)
        return x, y

def training_set():
        def temp(x,y):
            lst_training = []
            lst_training0 = []
            for Dic in D:
                lst = fea[x: y]
                for i in range(38):
                    for j in range(n_training[i]):    # length 10
                        try:
                            lst_training.append(lst[conn[i][0][j]])    # fga
                        except:
                            pass
                    break
                lst_training0.append(lst_training)
                break
            return lst_training0

        def temptest(x,y):
            lst_testing = []
            lst_testing0 = []
            for Dic in D:
                lst = fea[x: y]
                for i in range(38):
                    for j in range(n_testing[i]):    # length 10
                        try:
                            lst_testing.append(lst[conn1[i][0][j]])    # fga
                        except:
                            pass
                    break
                lst_testing0.append(lst_testing)
                break
            return lst_testing0

        lst1 = []
        lst2 = []

        for i in range(38):
            x, y = segmen()
            lst_training = temp(x[i],y[i])
```

```
            lst_testing = temptest(x[i],y[i])
            lst1 += lst_training
            lst2 += lst_testing

    return lst1, lst2

def df_training(index_training, index_testing):
    pass
def index():
    pass

def l2_norm(x,y):
    lst = []
    n = 0
    for i in range(0,len(x)):
        lst.append(abs(int(x[i])-int(y[i])))
    n = norm(np.array(lst))
    return n
def k_means(x,training_set, kn = 1): # for one data
    lst = []
    ni = []
    voting =0
    for i in range(len(training_set)):
        for j in range(len(training_set[i])):
            k = l2_norm(training_set[i][j],x)
            lst.append([k,i,j])
    y = np.array(lst)
    for i in range(int(kn)):
        x = y[np.argmin(y, axis=0)[0], 1]
        ni.append(x)
        index = np.argmin(y, axis=0)[0]
        y = np.delete(y, index, axis=0)
    return ni

def k_means_testing(testing_set):
    count = 0
    correct = 0
    for i in range(len(testing_set)):
        for j in range(len(testing_set[i])):
            x = testing_set[i][j]
            if k_means(x,training_set)[0] == i:
                correct += 1
            count +=1
        break   ###
    return count, correct

global conn
conn = []
global conn1
conn1 = []
conn, conn1, index_testing, index_training = rand_index()
training_set, testing_set = training_set()
count, correct = k_means_testing(testing_set)
print(count)
print(correct)
end = time.time()
print("time",end - start)
```

```
# for i in range(38):
#         training.append(conn[i])
#fea[i]

# def main():
# df_training.to_excel("test.xlsx")
# df_testing = pandas.DataFrame(lst_testing)
```

Reference code for part 2

```
################################################################################
##############################Initialization
############################Question 2,3,4,5.    2.2
#this code is the NN algorithm developed with library; for question2, run    function:k23410(); print_img() and error(); for question3
# run question3(); for question 4, run question4hog() and uncommon line 50, question4lbp() and uncommon line51 for 2.2 run cross(p,k);
from scipy.io import loadmat
import numpy as np
import pandas
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import datasets
from sklearn.neighbors import KNeighborsClassifier
from skimage import feature
import matplotlib.pyplot as plt
from skimage.feature import hog
from skimage.feature import local_binary_pattern
from sklearn.model_selection import cross_val_score
from sklearn import metrics
from bokeh.plotting import figure, show,output_file

loadtemp = loadmat('YaleB_32x32.mat')
gnd = loadtemp["gnd"]    # 2414 X 1
fea = loadtemp["fea"]    # (2414, 1024)

def hog():
    fea_hog = []
    for i in range(len(fea)):
        img = fea[i].reshape((32, 32))
        hog_image = hog(img, orientations=8, cells_per_block=(1,1))
        fea_hog.append((hog_image))
    fea_hog = np.array(fea_hog)
    datasethog = pandas.DataFrame(fea_hog)
    return datasethog

def lbp():
    fea_lbp = []
    for i in range(len(fea)):
        img = fea[i].reshape((32, 32))
        lbp_img = local_binary_pattern(img, P =4, R = 4 )
```

```
            fea_lbp.append((lbp_img))
        fea_lbp = np.array(fea_lbp)
        kk = np.array(fea_lbp)
        fea_lbp = kk.reshape(2414,1024)
        datasetlbp = pandas.DataFrame(fea_lbp)
        return datasetlbp
dataset = pandas.DataFrame(fea)
#hog
index = pandas.DataFrame(gnd)
y = index
X = dataset
# X = hog()
# X= lbp()
Splitm = [(64-10)/64, (64-20)/64, (64-30)/64, (64-40)/64, (64-50)/64]
m = [10, 20, 30, 40, 50]
k = [2,3,5,10]
p = [ 1, 2, 3, 5, 10]

def cross_validation(k=1,p=2):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(20/64), random_state=0)
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_train = scaler.transform(X_train)
        X_test = scaler.transform(X_test)
        classifier = KNeighborsClassifier(n_neighbors=k,p = p)
        classifier.fit(X_train, y_train.values.ravel())
        scores = cross_val_score(classifier, X, y.values.ravel(), cv=5)    # score array
        print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))   #The mean score and the 95%
confidence interval
        y_pred = classifier.predict(X_test)
        count = 0
        correct = 0
        error_locations = []
        real_locations = []
        for i in range(len(y_pred)):
            try:
                if abs((int(y_test.iloc[i,0]) - int(y_pred[i]))) <= 6 :
                    correct+=1
                else:
                    error_locations.append(y_pred[i])
                    real_locations = y_test.iloc[i,0]
                count += 1
            except:
                pass
        rate = (count-correct)/count
        return rate , scores

def main(n,k=1,p=2):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=n, random_state=None)
        scaler = StandardScaler()
        scaler.fit(X_train)
        X_train = scaler.transform(X_train)
        X_test = scaler.transform(X_test)
        classifier = KNeighborsClassifier(n_neighbors=k,p = p)
        classifier.fit(X_train, y_train.values.ravel())
        y_pred = classifier.predict(X_test)
        count = 0
        correct = 0
```

```
    error_locations = []
    real_locations = []
    for i in range(len(y_pred)):
        try:
            if abs((int(y_test.iloc[i,0]) - int(y_pred[i]))) <= 6 :
                correct+=1
            else:
                error_locations.append(y_pred[i])
                real_locations = y_test.iloc[i,0]
            count += 1
        except:
            pass
    rate = (count-correct)/count
    return n,rate ,error_locations , real_locations

#general Question2

def k23410():   #Question2
    lst = []
    for j in range(len(k)):
        for i in range(len(Splitm)):
            n,rate ,error_locations , real_locations = main(Splitm[i],k[j])
            lst.append((k[j], m[i], rate))
    df_out = pandas.DataFrame(lst)
    df_out.to_excel("question2.xlsx")
    return df_out

#images,wrong samples
#################
# lst = []
def error(): #Question2
    n,rate ,error_locations , real_locations = main(Splitm[2] , 10)
    real_img = fea[real_locations].reshape((32, 32))
    false_img1 = fea[error_locations[4]].reshape((32, 32))
    imgplot = plt.imshow(false_img1)
    plt.show()

def print_img():
    n,rate ,error_locations , real_locations = main(Splitm[2] , 10)
    real_img = fea[real_locations].reshape((32, 32))
    false_img1 = fea[error_locations[4]].reshape((32, 32))
    imgplot = plt.imshow(false_img1)
    plt.show()

def k1():
    lst1= []
    for i in range(len(Splitm)):
        n,rate ,error_locations , real_locations = main(Splitm[i])
        lst1.append(( m[i], rate))
    df1_out = pandas.DataFrame(lst1)
    df1_out.to_excel("question111.xlsx")
    return df1_out
# add a line renderer
def question3():
    lst = []
    for i in range(len(p)):
        n, rate, error_locations, real_locations = main(Splitm[2], k[2], p[i])
        lst.append((p[i], rate))
```

```
    df1_out = pandas.DataFrame(lst)
    return df1_out

def question4lbp():
    lst = []
    X = lbp()
    for i in range(2):
        n, rate, error_locations, real_locations = main(Splitm[2], k[2], p[i])
        lst.append((p[i], rate))
    df1_out = pandas.DataFrame(lst)
    return df1_out
def question4hog():
    lst = []
    X = hog()
    for i in range(2):
        n, rate, error_locations, real_locations = main(Splitm[2], k[2], p[i])
        lst.append((p[i], rate))
    df1_out = pandas.DataFrame(lst)
    return df1_out

def cross(p,k):    #question 2.2 cross validation
    score = []
    for i in p:
        for j in k:
            x = cross_validation(j, i)
            score.append([j,i,x])
    return score

################################################################################
code for plotting:

from bokeh.plotting import figure, show,output_file
import pandas as pd
df = pd.read_csv("NN1.csv")
df1 = df.copy()
df1.loc[4]  =  [df.iloc[1,0]-df.iloc[2,0],df.iloc[1,1]-df.iloc[2,1],df.iloc[1,2]-df.iloc[2,2],  df.iloc[1,3]-df.iloc[2,3],
df.iloc[1,4]-df.iloc[2,4]]
df1.loc[5] =
[df1.iloc[4,0]/df1.iloc[1,0]*100,df1.iloc[4,1]/df1.iloc[1,1]*100,df1.iloc[4,2]/df1.iloc[1,2]*100,df1.iloc[4,3]/df1.iloc
[1,3]*100,df1.iloc[4,4]/df1.iloc[1,4]*100]

# df1.loc[5] =
[df.iloc[4,0]/df.iloc[1,0]*100,df.iloc[4,1]/df.iloc[1,1]*100,df.iloc[4,2]/df.iloc[1,2]*100,df.iloc[4,3]/df.iloc[1,3]*100,
df.iloc[4,4]/df.iloc[1,4]*100]# adding a row
# df1.index = df1.index + 1    # shifting index
# df1 = df1.sort_index()

output_file("line.html", title="question 1 k=1, error rate vs number of training sets")

p = figure(plot_width=800, plot_height=400)
p.xaxis.axis_label = "number of training samples"
p.yaxis.axis_label = "Error rate %"
# add a line renderer
p.line(df1.iloc[0], df1.iloc[5], line_width=2)

show(p)
```