CS464 Introduction to Machine Learning

Fall 2022

Homework 2

Emre Can Şen-21902516

Q-1.1

First 10 principal components for red, green and blue are computed. Eigen values are computed.

Q-1.2

The found PCA's are reshaped to 64x64 matrix. Then RGB values of each PCA is stacked to create a 64x64x3 matrix and they are displayed here:

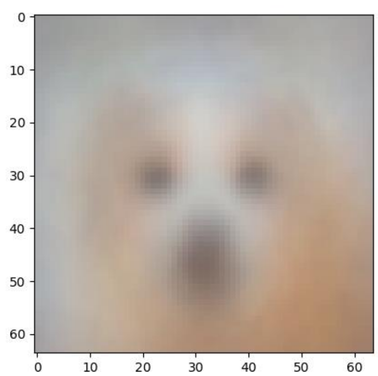


Figure 1- 1st Eigen Image

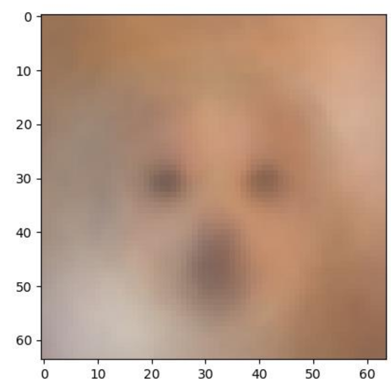
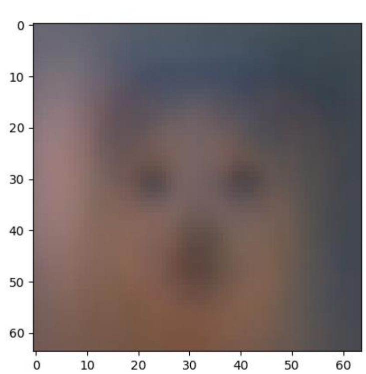


Figure 2- 2nd Eigen Image



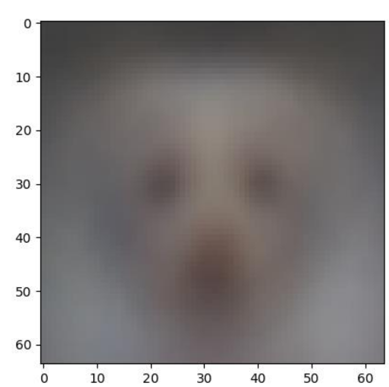
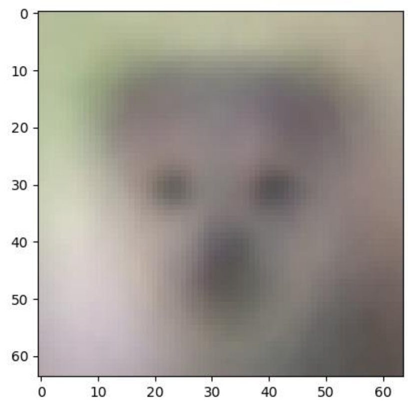


Figure 3- 3rd Eigen Image

Figure 4- 4th Eigen Image



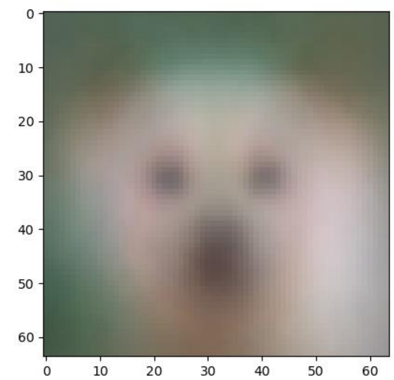
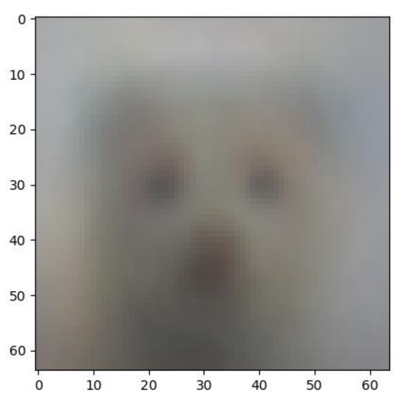


Figure 5- 5th Eigen Image

Figure 6- 6th Eigen Image



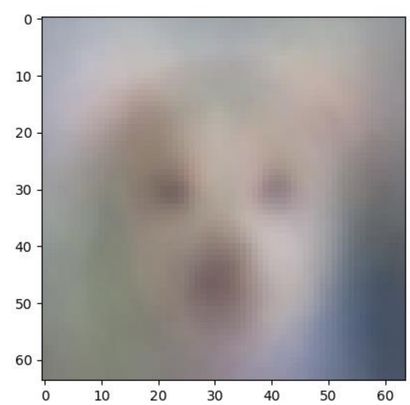


Figure 7 – 7th Eigen Image

Figure 8- 8th Eigen Image

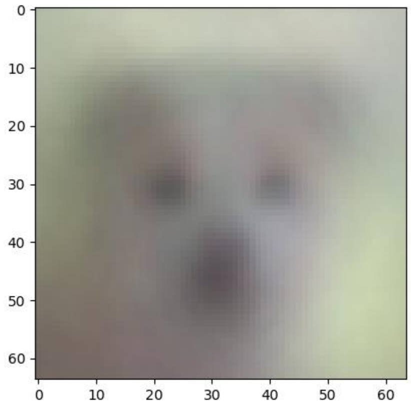


Figure 9 – 9th Eigen Image

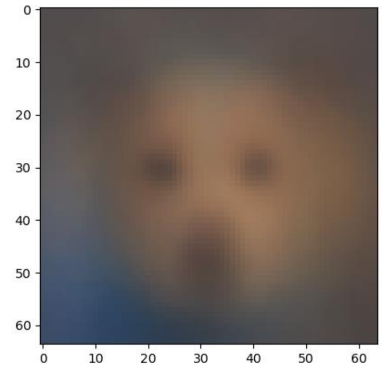


Figure 10- 10th Eigen Image

These are the most important dimensions that are available on the whole set, projected to first 10 images.

Q-1.3

In order to reconstruct the dog images, it is sufficient to dot product the PCA components with RGB values and then reformat them to fit 64x64 pixel size and draw the image. This dot product results in an array that contains the reconstructions of all images, so the first of the 5239 data was taken to figure. The more principal components used, the more detailed the image is.

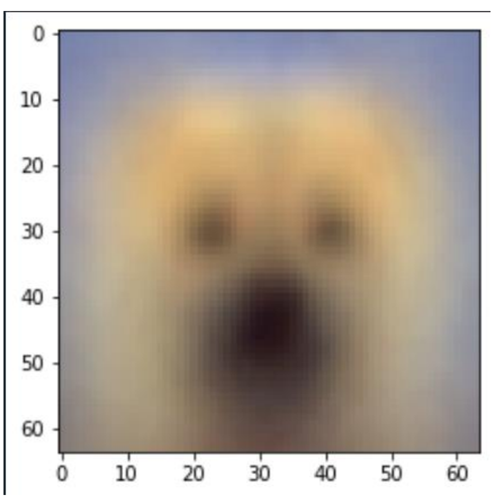


Figure 11- K=1 Image Reconstruction

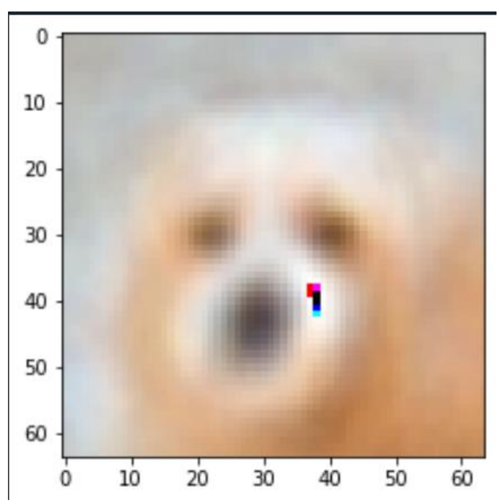


Figure 12- K=50 Image Reconstruction

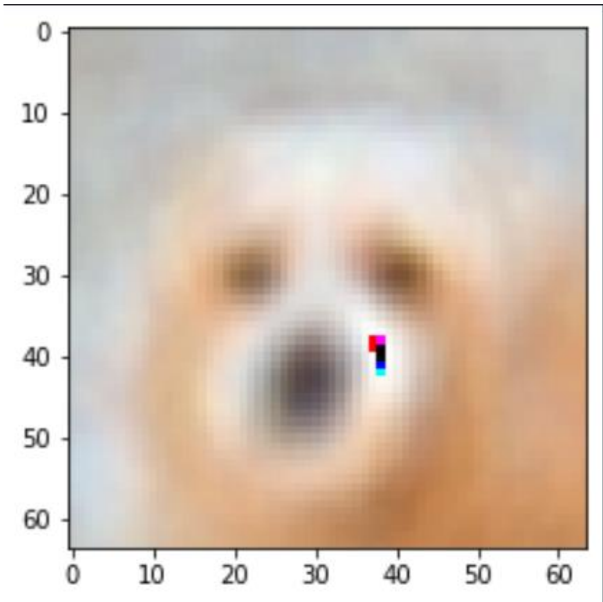


Figure 13- K=250 Image Reconstruction

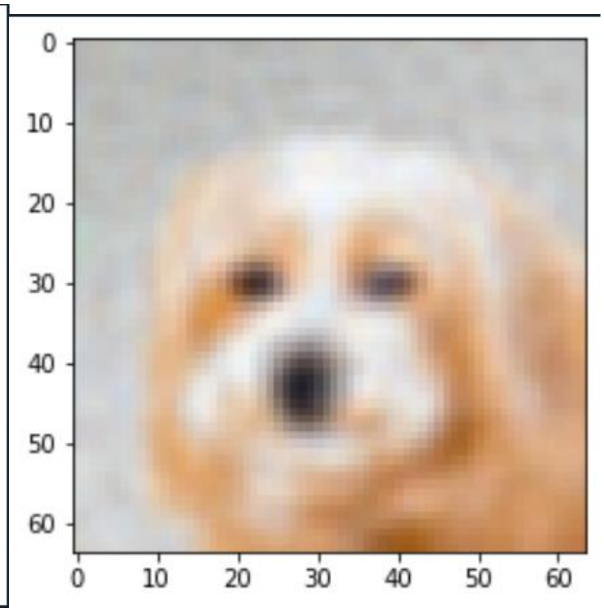


Figure 14- K=500 Image Reconstruction

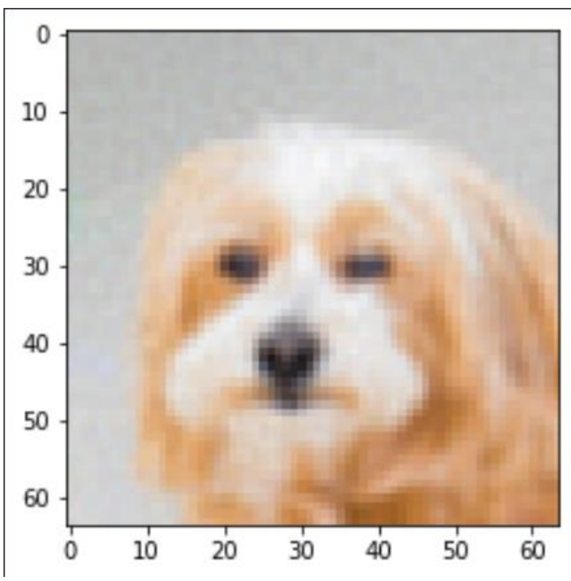


Figure 15- K=1000 Image Reconstruction

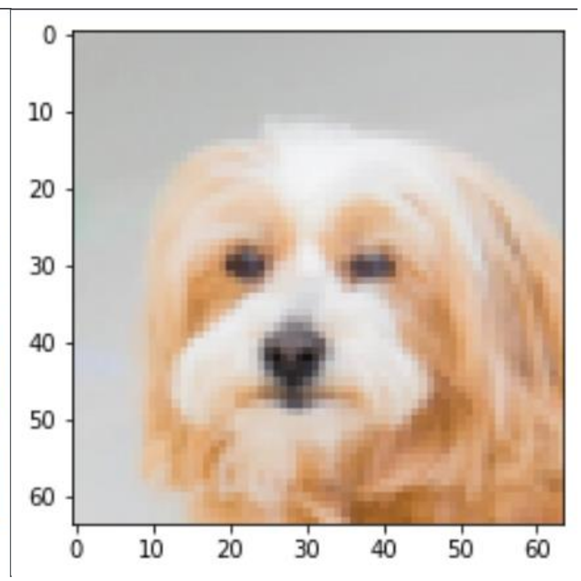


Figure 16- K=4096 Image Reconstruction

Q.2.1

For this part, batch-gradient algorithm was implemented with Gaussian weights and also learning rate hyper parameter was added.

Cost after iteration 0: 0.818306

Cost after iteration 10: 0.807759

Cost after iteration 20: 0.798251

Cost after iteration 30: 0.789705

Cost after iteration 40: 0.782042

Cost after iteration 50: 0.775187

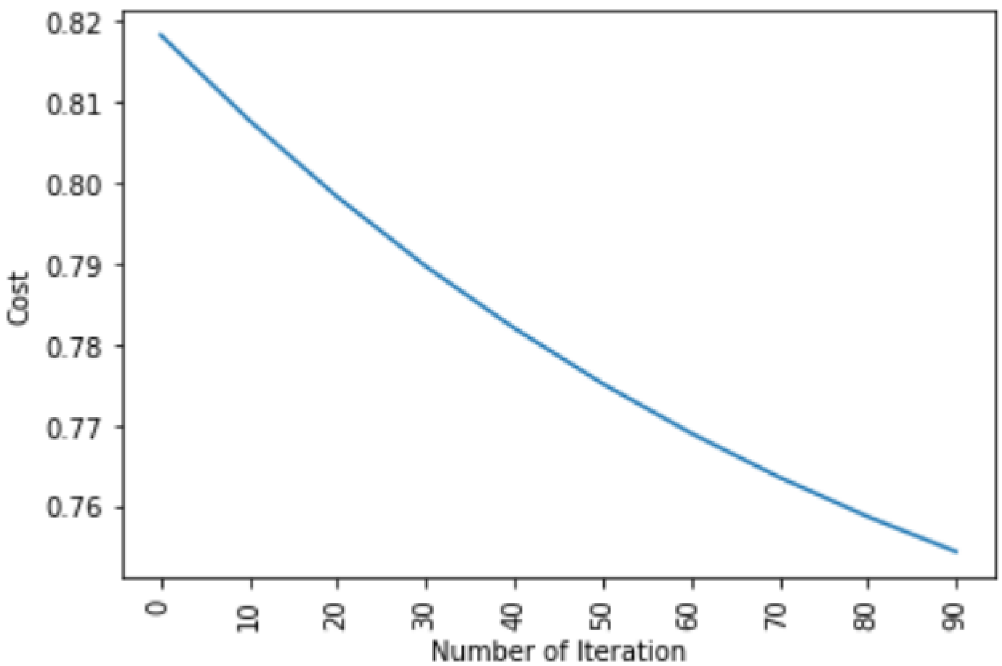
Cost after iteration 60: 0.769065

Cost after iteration 70: 0.763605

Cost after iteration 80: 0.758739

Cost after iteration 90: 0.754404

Test acc.: 62.96666666666667 %



Q2.2

Q2.3

Learning rate hyper parameter was implemented.

Q2.4

Appendix

PART-1

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  4 15:31:41 2022

@author: Emre
"""

import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from PIL import Image

def open_directory(path, size):
    im_l =[]
    im_flattened = []
    for image in Path(path).glob('*'):
        img=Image.open(image)
        img_array=np.array(img.resize((size,size), Image.BILINEAR))
        im_l.append(img_array)
        img_flat = img_array.flatten().reshape((size*size),3)
        im_flattened.append(img_flat)
    return im_l, im_flattened



def PCA(array, num_comp):
```

```
    covariance = np.cov(array)

    eig_val, eig_vec = np.linalg.eigh(covariance)
    sort_eig  = np.argsort(-eig_val)
    eig_val = eig_val[sort_eig]
    eig_vec = eig_vec[:, sort_eig]

    Projection = eig_vec[:, range(num_comp)]
    Z = Projection @ Projection.T
    return Z

file_directory = 'C:/Users/Emre/.spyder-py3/afhq_dog'

im_l, im_flattened = open_directory(file_directory, 64)

im_rgb = np.asarray(im_flattened)

im_red = im_rgb[:,:,0]
im_green = im_rgb[:,:,1]
im_blue = im_rgb[:,:,2]

# for k in (1000):
color_list = []
PVEs = []
for color in (im_red, im_green, im_blue):
    a = PCA(color.T, 10)
    color_list.append(a)

Red = im_red @ color_list[0]
Green = im_green @ color_list[1]
```

```
Blue = im_blue @ color_list[2]

pca_out = np.array([Red.T, Green.T, Blue.T])
pca_out -= pca_out.min()
pca_out /= pca_out.ptp()
pca_out_Transpose = pca_out.T
pca_out_final = pca_out_Transpose[:10,:,:].reshape(10,64,64,3)
# pca_out_final = pca_out_Transpose[0,:,:].reshape(64,64,3)

for i in range(10):
    plt.figure(i+1)
    plt.imshow(Image.fromarray((pca_out_final[i,:,:,:]*255).astype(np.uint8)))
    plt.show()

# img1 =(pca_out_final*255).astype(np.uint8)
# plt.imshow(img1)

# img1 = im_l[0] @ (first * 255).astype(np.uint8)
# plt.imshow(img1)
```

PART-2

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec  5 03:25:51 2022

@author: Emre
"""

import numpy as np
import pandas as pd
```

```
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split


data = pd.read_csv("dataset.csv")


# y_sk = data.label.values

# x_data_sk = data.drop(["label"],axis=1)

# x_sk= (x_data_sk - np.min(x_data_sk)) / (np.max(x_data_sk) - np.min(x_data_sk)).values


# x_train_sk, x_test_sk, y_train_sk, y_test_sk = train_test_split(x_sk,y_sk,test_size = 0.2, random_state =
42)


# x_train_sk = x_train_sk.T

# x_test_sk = x_test_sk.T

# y_train_sk = y_train_sk.T

# y_test_sk = y_test_sk.T


x_data = data.drop(["label"],axis=1)

y = data.label.values

x = (x_data - np.min(x_data)) / (np.max(x_data) - np.min(x_data)).values

x['label']=y.tolist()

data=x


data = data.sample(frac = 1, random_state=42)


train_size = 0.7

valid_size=0.2


train_index = int(len(data)*train_size)
```

```
data_train = data[0:train_index]
data_rem = data[train_index:]

valid_index = int(len(data)*valid_size)

data_valid = data[train_index:train_index+valid_index]
data_test = data[train_index+valid_index:]

x_train_wosk, y_train_wosk = data_train.drop(columns='label').copy(), data_train['label'].copy(),
x_valid_wosk, y_valid_wosk = data_valid.drop(columns='label').copy(), data_valid['label'].copy()
x_test_wosk, y_test_wosk = data_test.drop(columns='label').copy(), data_test['label'].copy()

x_train_old=x_train_wosk
y_train_old=y_train_wosk

x_train_wosk=pd.concat([x_train_wosk, x_valid_wosk])
x_train_wosk=x_train_wosk.T
x_test_wosk = x_test_wosk.T
y_train_wosk=pd.concat([y_train_wosk, y_valid_wosk])
y_train_wosk=y_train_wosk.T
y_train_wosk=y_train_wosk.to_numpy(dtype="int64")
y_test_wosk = y_test_wosk.T
y_test_wosk=y_test_wosk.to_numpy(dtype="int64")

def sigmoid(z):

    y_hold = 1 / (1+np.exp(-z))

    return y_hold
```

```
def weights_and_bias(dimension):

    w = np.random.normal(loc=0.0, scale=1.0, size=(dimension, 1))
    b = 0.0
    return w,b

def full_batch(w,bias,x_train,y_train):

    z = np.dot(w.T,x_train) + bias
    y_hold = sigmoid(z)
    loss = (-y_train)*np.log(y_hold) - ((1-y_train))*np.log(1-y_hold)
    cost = (np.sum(loss)) / x_train.shape[1]

    #backward propogation
    weight = (np.dot(x_train,((y_hold-y_train).T)))/x_train.shape[1]
    derivative_bias = np.sum(y_hold-y_train)/x_train.shape[1]
    gradients = {"weight": weight,"derivative_bias": derivative_bias}
    return cost,gradients

def predict(w,bias,x_test):
    # x_test is a input for forward propagation
    z = sigmoid(np.dot(w.T,x_test)+bias)
    Y_prediction = np.zeros((1,x_test.shape[1]))
    # if z is bigger than 0.5, our prediction is one means has diabete (y_hold=1),
    # if z is smaller than 0.5, our prediction is zero means does not have diabete (y_hold=0),
    for i in range(z.shape[1]):
        if z[0,i]<= 0.5:
            Y_prediction[0,i] = 0
        else:
            Y_prediction[0,i] = 1
```

```
    return Y_prediction


def update(w, bias, x_train, y_train, learning_rate,number_of_iterarion):
    cost_l = []
    cost_l_two = []
    index = []
    # updating(learning) parameters is number_of_iterarion times
    for i in range(number_of_iterarion):
        # make forward and backward propagation and find cost and gradients
        cost,gradients = full_batch(w,bias,x_train,y_train)
        cost_l.append(cost)
        # lets update
        w = w - learning_rate * gradients["weight"]
        bias = bias - learning_rate * gradients["derivative_bias"]
        if i % 10 == 0:
            cost_l_two.append(cost)
            index.append(i)
            print ("Cost after iteration %i: %f" %(i, cost)) #if section defined to print our cost values in every
10 iteration. We do not need to do that. It's optional.
    # we update(learn) parameters weights and bias
    parameters = {"weight": w,"bias": bias}
    plt.plot(index,cost_l_two)
    plt.xticks(index,rotation='vertical')
    plt.xlabel("Number of Iteration")
    plt.ylabel("Cost")
    plt.show()
    return parameters, gradients, cost_l
```

```
def logistic_regression(x_train, y_train, x_test, y_test, learning_rate ,  num_iterations):
    # initialize
    dimension =  x_train.shape[0]
    w,bias = weights_and_bias(dimension)

    parameters, gradients, cost_l = update(w, bias, x_train, y_train, learning_rate,num_iterations)

    y_prediction_test = predict(parameters["weight"],parameters["bias"],x_test)


    # Print train/test Errors

    print("Test acc.: {} %".format(100 - np.mean(np.abs(y_prediction_test - y_test)) * 100))



logistic_regression(x_train_wosk, y_train_wosk, x_test_wosk, y_test_wosk,learning_rate = 0.01,
num_iterations = 100)
```