CSCE 633 – Machine Learning

Homework 03

Amiya Ranjan Panda

UIN – 727006179

Q1. a

$F = [1, 2, 1, 3, 2, 3, 1, 2, 3, 8, 7, 8, 9, 9, 7, 8]$

$W = [1, 1, 1]$

Avoiding boundary Case, we can do, effective Calculation from

$F[1:16]$

i.e. first and last element will not be Considered.

for each step, we have to add the adjecent elements to the target element and update the output vector.

$(f * w) =$

$[4, 6, 6, 8, 6, 6, 6, 13, 18, 23, 24, 26, 25, 24]$

Q1.b.

Similarly,

$F = [1, 2, 1, 3, 2, 3, 1, 2, 3, 8, 7, 8, 9, 9, 7, 8]$

$W = [1\ 0\ -1]$

This filter slides over the vector and Calculate the difference between left and right pixels to the target element.

$(f * w) =$

$[0, -1, -1, 0, 1, 1, -2, -6, -4, 0, -2, -1, 2, 1]$

Q1.b(i). Implementation of convolution

Herein, I have performed convolution operation on the target matrix which is **zero padded prior to it**. In my implementation, I have used four nested for loops which can be avoided by using elementwise matrix multiplication operation provided by various python libraries. The input of the function is the target matrix and the filter (whose size is configurable).

Q1. b(ii). The target input is

```
array([[164, 188, 164, 161, 195],
       [178, 201, 197, 150, 137],
       [174, 168, 181, 190, 184],
       [131, 179, 176, 185, 198],
       [ 92, 185, 179, 133, 167]])
```

Kernel 1 :

```
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
```

Output 1:

```
array([[ 731., 1092., 1061., 1004.,  643.],
       [1073., 1615., 1600., 1559., 1017.],
       [1031., 1585., 1627., 1598., 1044.],
       [ 929., 1465., 1576., 1593., 1057.],
       [ 587.,  942., 1037., 1038.,  683.]])
```

Kernel 2:

```
array([[-1, -2, -1],
       [ 0,  0,  0],
       [-1, -2, -1]])
```

Output 2:

```
array([[ -557.,  -777.,  -745.,  -634.,  -424.],
       [-1032., -1395., -1397., -1426., -1109.],
       [ -998., -1442., -1461., -1378., -1005.],
       [ -885., -1332., -1396., -1357., -1025.],
       [ -441.,  -665.,  -716.,  -744.,  -581.]])
```

Kernel 3:

```
array([[-1, -1, -1],
       [-1,  9, -1],
       [-1, -1, -1]])
```

Output 3:

```
array([[ 909.,  788.,  579.,  606., 1307.],
       [ 707.,  395.,  370.,  -59.,  353.],
       [ 709.,   95.,  183.,  302.,  796.],
       [ 381.,  325.,  184.,  257.,  923.],
       [ 333.,  908.,  753.,  292.,  987.]])
```

Different kernels are used for specific operations. For example, gaussian kernel is used for smoothing the matrix, average operation is performed over the matrix by applying kernel similar to the first kernel etc.

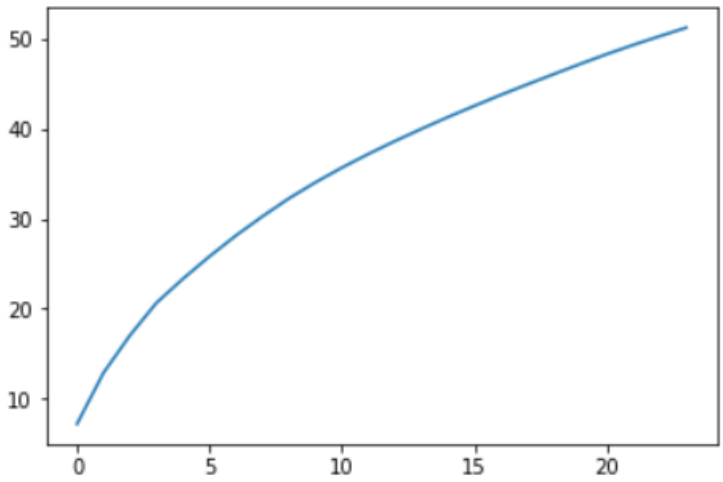
Q2.a. PCA Implementation

First the input is resized to **160 X 160** images to avoid "memory error" which occur during SVD calculation due to inability of the system.

To further proceed, first each row of **the input matrix is subtracted from the column wise me**an values i.e. **"mean face"** is subtracted from each of the image. Then the data is zero **standard normalized** to make the data centric. Then we calculate the principal component matrix and the strength(eigenvalue) matrix, I used the **"singular value decomposition"** technique. Then to calculate the cumulative energy we used the eigenvalues obtained from the SVD. Below given is the **cumulative energy per k up to top 50 principal components** and the graph.

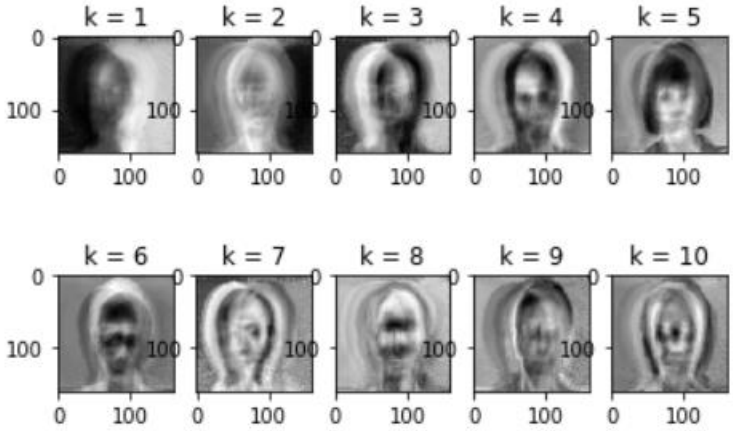
```
[7.205504641304216,
12.890077099495834,
17.05567848195133,
20.667610999576937,
23.337618467081768,
25.82177174471772,
28.142851416293478,
30.266234932431647,
32.265712319487406,
34.04755871842225,
35.684913902576604,
37.199010616010916,
38.63139539259496,
39.98686867773926,
41.298611380796,
42.544621176288224,
43.768503878574535,
44.93204870041838,
46.07330482958034,
47.20706219211452,
48.296483773715956,
49.31818688647496,
50.28782974468123,
51.23770538497835,
52.17379299021355,
53.08504906384827,
```

```
53.98523184690427,
54.86412864061499,
55.733662661292094,
56.542634946659135,
57.314529701133075,
58.080992939047846,
58.834197666162744,
59.57628046346096,
60.29381689042307,
61.00153217544369,
61.69903153737003,
62.365879671332394,
63.02137572548059,
63.66197795934706,
64.29594201945211,
64.91338173121878,
65.5186391513868,
66.12086402696914,
66.7163476110637,
67.29207877489,
67.86079694175939,
68.41689145722137,
68.968424971381]
```



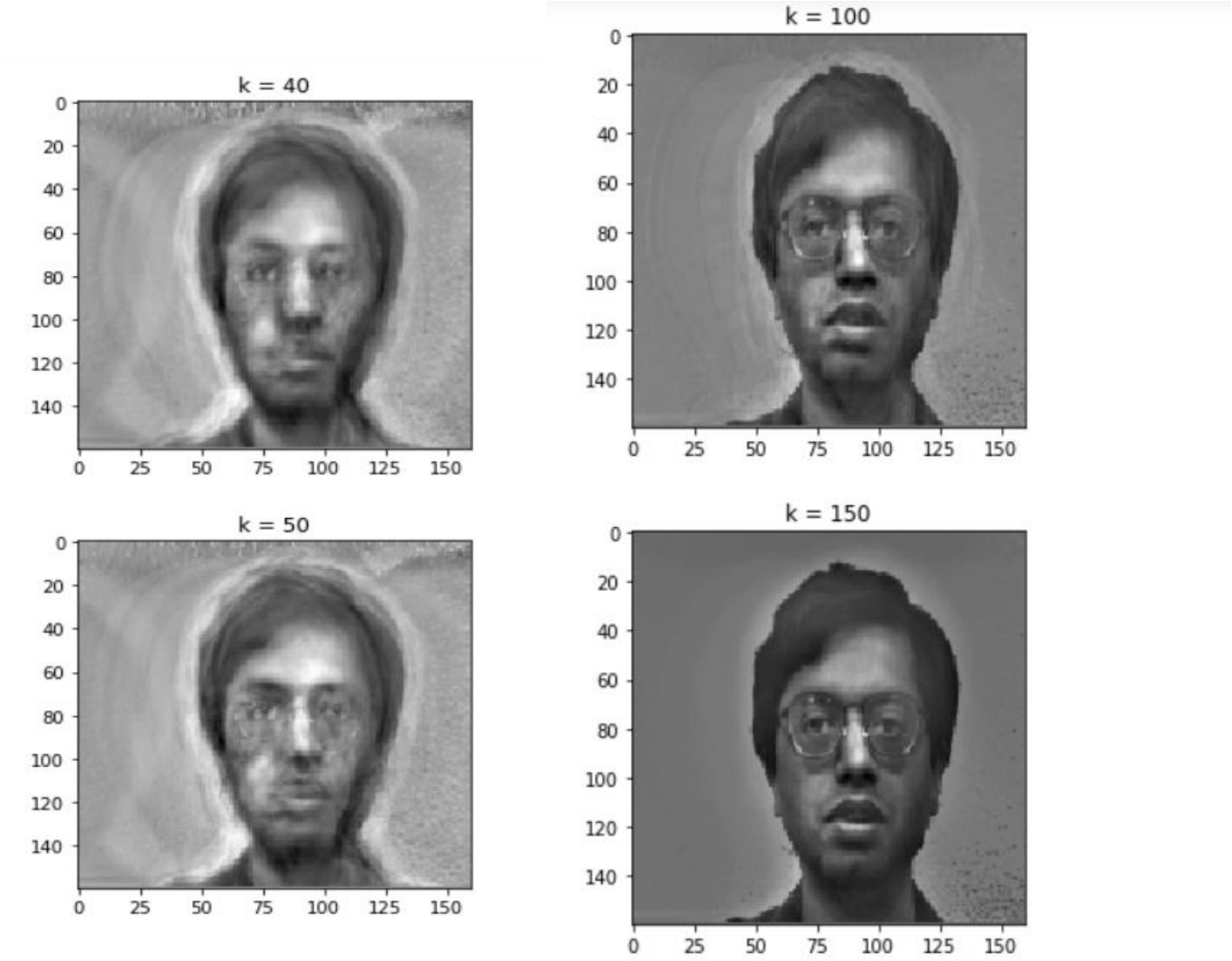
Q2.b. From the above, we can observe that top **23 principal components** of consists of 50 percent of energy.

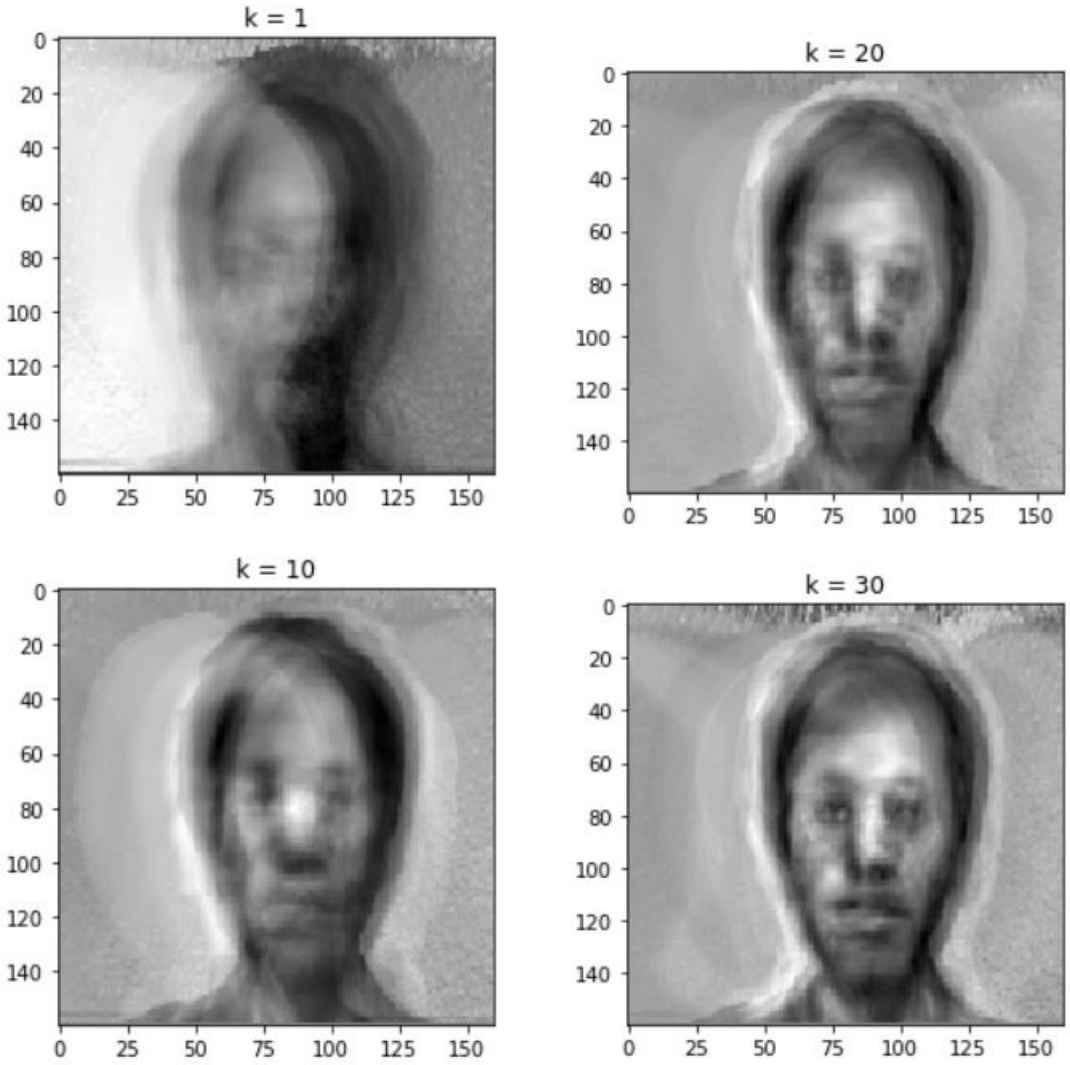
Q2.C. Below given is the top 10 eigen faces.



Q2.d. Reconstruction based on top k principal components.

From the below given images, we can infer that top 150 principal components are sufficient to reconstruct a visually good quality image.





Q2.e. Face recognition using SVM model

I have used the algorithm using **SVM "linear" kernel**.

First the input is resized to **80 X 80** images to avoid **"memory error" which occur during SVD** calculation due to inability of the system.

As a part of **5 fold cross validation**, I have iterated the model between various values of C and K(number of principal components) and below given is the output.

```
[['c = 0.0009765625', 'k = 30', 'accuracy = 0.16666666666666666'],
 ['c = 0.0009765625', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 0.0009765625', 'k = 90', 'accuracy = 0.20833333333333334'],
```

```
['c = 0.00390625', 'k = 30', 'accuracy = 0.16666666666666666'],
 ['c = 0.00390625', 'k = 60', 'accuracy = 0.20833333333333334'],
 ['c = 0.00390625', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 0.015625', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 0.015625', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 0.015625', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 0.0625', 'k = 30', 'accuracy = 0.16666666666666666'],
 ['c = 0.0625', 'k = 60', 'accuracy = 0.20833333333333334'],
 ['c = 0.0625', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 0.25', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 0.25', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 0.25', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 1', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 1', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 1', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 4', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 4', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 4', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 16', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 16', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 16', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 64', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 64', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 64', 'k = 90', 'accuracy = 0.20833333333333334'],
['c = 256', 'k = 30', 'accuracy = 0.16666666666666666'],
['c = 256', 'k = 60', 'accuracy = 0.20833333333333334'],
['c = 256', 'k = 90', 'accuracy = 0.20833333333333334']]
```

From the output of the cross-validation, we can infer that the model gives the best output at C = 256 and K = 90.

So, I trained the model using the same parameters and the accuracy comes out to be 44.44444444444444%.

Q2.f. Face recognition using CNN

I have used KERAS library for this implementation.

First the input is resized to **160 X 160** images to avoid “memory error” which occur during SVD calculation due to inability of the system.

The input of the model is **not the output of PCA** or the reduced feature space.

The model has **3 convolution layers followed by maxpool (2)**. Then the output is **flattened for the Fully Connected Layer and dropout** is introduced to avoid overfitting. The activation function of **FCC is “softmax**”. Loss function is taken as “**categorical_crossentropy**” and “**adam**” optimizer is used with **learning rate 0.001**.I have performed the model learning and testing for different values of “**activation function”, “dropout”, “filter size” and “stride**” for **8 epochs**. Below given is the output:

```
[['act = relu',
  'drp = 0.2',
  'fil = 3',
  'strd = 1',
  'accuracy = 0.8888888902134365'],
 ['act = relu',
  'drp = 0.2',
  'fil = 3',
  'strd = 2',
  'accuracy = 0.6000000079472859'],
 ['act = relu',
  'drp = 0.2',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.8888888915379842'],
 ['act = relu',
  'drp = 0.2',
  'fil = 5',
  'strd = 2',
  'accuracy = 0.6000000052981906'],
 ['act = relu',
  'drp = 0.3',
  'fil = 3',
  'strd = 1',
  'accuracy = 0.8444444484180874'],
 ['act = relu',
  'drp = 0.3',
  'fil = 3',
  'strd = 2',
  'accuracy = 0.466666673289405'],
 ['act = relu',
  'drp = 0.3',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.8888888915379842'],
 ['act = relu',
  'drp = 0.3',
  'fil = 5',
  'strd = 2',
  'accuracy = 0.666666669315762'],
 ['act = tanh',
  'drp = 0.2',
  'fil = 3',
  'strd = 1',
  'accuracy = 0.8444444457689921'],
 ['act = tanh',
  'drp = 0.2',
  'fil = 3',
  'strd = 2',
  'accuracy = 0.7111111177338494'],
 ['act = tanh',
  'drp = 0.2',
  'fil = 5',
  'strd = 1',
```

```
  'accuracy = 0.8222222235467699'],
 ['act = tanh',
  'drp = 0.2',
  'fil = 5',
  'strd = 2',
  'accuracy = 0.8666666666666667'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 3',
  'strd = 1',
  'accuracy = 0.866666669315762'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 3',
  'strd = 2',
  'accuracy = 0.5555555621782938'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.866666669315762'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 5',
  'strd = 2',
  'accuracy = 0.9111111124356588']]
```

As observed the best output is achieved as 91% activation = “tanh”, dropout = “0.3”, filter = 5 and stride = 2.

Q2.g. Data Augmentation

After data augmentation, again the model is run for different values of “**activation function”, “dropout”, “filter size” and “stride**”. Below given is the output:

```
[['act = relu',
  'drp = 0.2',
  'fil = 3',
  'strd = 1',
  'accuracy = 0.8222222261958652'],
 ['act = relu',
  'drp = 0.2',
  'fil = 3',
  'strd = 2',
  'accuracy = 0.48888889683617487'],
 ['act = relu',
  'drp = 0.2',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.9111111124356588'],
```

```
['act = relu',
 'drp = 0.2',
 'fil = 5',
 'strd = 2',
 'accuracy = 0.6888888915379842'],
['act = relu',
 'drp = 0.3',
 'fil = 3',
 'strd = 1',
 'accuracy = 0.8888888915379842'],
['act = relu',
 'drp = 0.3',
 'fil = 3',
 'strd = 2',
 'accuracy = 0.40000000794728596'],
['act = relu',
 'drp = 0.3',
 'fil = 5',
 'strd = 1',
 'accuracy = 0.9333333333333333'],
['act = relu',
 'drp = 0.3',
 'fil = 5',
 'strd = 2',
 'accuracy = 0.6000000066227383'],
['act = tanh',
 'drp = 0.2',
 'fil = 3',
 'strd = 1',
 'accuracy = 0.8666666679912143'],
['act = tanh',
 'drp = 0.2',
 'fil = 3',
 'strd = 2',
 'accuracy = 0.666666669315762'],
['act = tanh',
 'drp = 0.2',
 'fil = 5',
 'strd = 1',
 'accuracy = 0.8222222248713176'],
['act = tanh',
 'drp = 0.2',
 'fil = 5',
 'strd = 2',
 'accuracy = 0.8888888902134365'],
['act = tanh',
 'drp = 0.3',
 'fil = 3',
 'strd = 1',
 'accuracy = 0.8000000052981906'],
['act = tanh',
 'drp = 0.3',
 'fil = 3',
 'strd = 2',
```

```
  'accuracy = 0.5777777830759684'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.8444444470935397'],
 ['act = tanh',
  'drp = 0.3',
  'fil = 5',
  'strd = 2',
  'accuracy = 0.8444444444444444']]
```

As we can observe from the above, the best accuracy achieved for the model after data augme ntation is 93% which is 2% more than the previous one. But, it is achieved for different paramet er set.

```
['act = relu',
  'drp = 0.3',
  'fil = 5',
  'strd = 1',
  'accuracy = 0.9333333333333333']
```

We can observe that CNN is better than a linear SVM.

Data augmentation helps in achieving high accuracy for small dataset.

We can further optimize our CNN model by changing no. of layers, learning rate, optimizer, no. of epochs etc.