ML ASSIGNMENT 4
Section A

Date : / /
Page No.

Section A

a) input $\longrightarrow$ $15 \times 15 \times 4$

Output layer width $\rightarrow$ $w_o = \left[\frac{w_i - w_k + 2p_w}{s_w}\right] + 1$

output height $\rightarrow$ $h_o = \left[\frac{h_i - h_k + 2p_h}{s_h}\right] + 1$

for 1st layer,

$$w_o = \left[\frac{15 - 5 + 2\times1}{1}\right] + 1 = 13$$

$$h_o = \left[\frac{15 - 5 + 2\times1}{1}\right] + 1 = 13$$

No. of output channels = 1

Volume of 1st layer o/p = $13 \times 13 \times 4$
=

for maxpooling layer $\rightarrow$

$$w_o = \left[\frac{w_i - w_p}{s}\right] + 1 = \left[\frac{13-3}{2}\right] + 1 = 6$$

$$h_o = \left[\frac{13-3}{2}\right] + 1 = 6$$

Volume of pooling layer = $6 \times 6 \times 1$

Date : / /
Page No.

for 3rd layer →

$$h_o = \left[\frac{6-5+2\times 2}{2} \right] + 1 = 3$$

$$w_o = \left[\frac{6-3+2\times 2}{2} \right] + 1 = 4$$

$$o/p \text{ image} = 3 \times 4 \times 1$$

b) Pooling can be used to reduce the dimension of the feature maps which as a result helps in improving the computation time of CNN model.
The model becomes more robust to variations in position of features in input, since pooling can help to extract the most useful features

(iii) Total No. of learnable parameters →

Conv. 2D layer → $5 \times 5 \times 4 = 100$

Max pooling → 0

2nd Conv. 2D layer → $5 \times 3 \times 4 = 60$

∴ Total parameters = $100 + 0 + 60 = 160$

Date : / /
Page No.

b) Cluster 1 mean = (3, 12)   cluster 2 mean = (8, 7)   cluster 3 mean = (2, 13)

$D((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1|$

	(3, 12)	(8, 7)	(2, 13)
(3, 12)	0	10	2
(3, 7)	5	5	7
(9, 6)	12	2	14
(6, 10)	5	5	7
(8, 7)	10	0	12
(7, 6)	10	2	12
(2, 13)	2	12	0

Cluster 1 ⟶ (3, 12) (3, 7) (6, 10)
Cluster 2 ⟶ (9, 6) (8, 7) (7, 6)
Cluster 3 ⟶ (2, 13)

Assumption → if Avg mean distance of a point is same from two or more centre points then I have assigned it randomly to one of the clusters.

New 3 cluster centers ⟶ (1 iteration)

$C_1 = \left(\frac{3+3+6}{3}, \frac{12+7+10}{3}\right)$ $C_2 = \left(\frac{9+8+7}{3}, \frac{6+7+6}{3}\right)$

$= 4, 9.667$ $= (8, 6.33)$

$C_3 = (2, 13)$

Date : / /
Page No.

	(4, 9.667)	(8, 6.333)	(2,13)
(3, 12)	3.333	10.667	2
(3, 7)	3.667	5.667	7
(9, 6)	8.667	1.333	14
(8, 7)	6.667	0.667	12
(7, 6)	6.667	1.333	12
(2, 13)	5.333	12.667	0
(6, 10)	2.333	5.667	7

Ans = Cluster 1 → (3, 7), (6, 10)
cluster 2 → (9, 6), (8, 7), (7, 6)
cluster 3 → (3, 12), (2, 13)

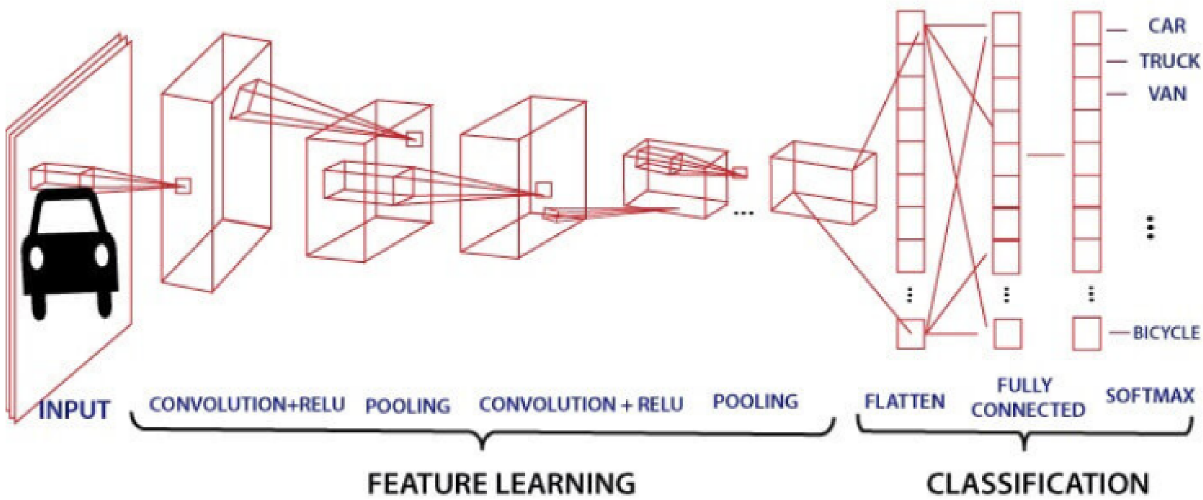
3 cluster centres After 2nd iteration →

$$C_1 = \left(\frac{3+6}{2}, \frac{7+10}{2}\right) = (45, 85)$$

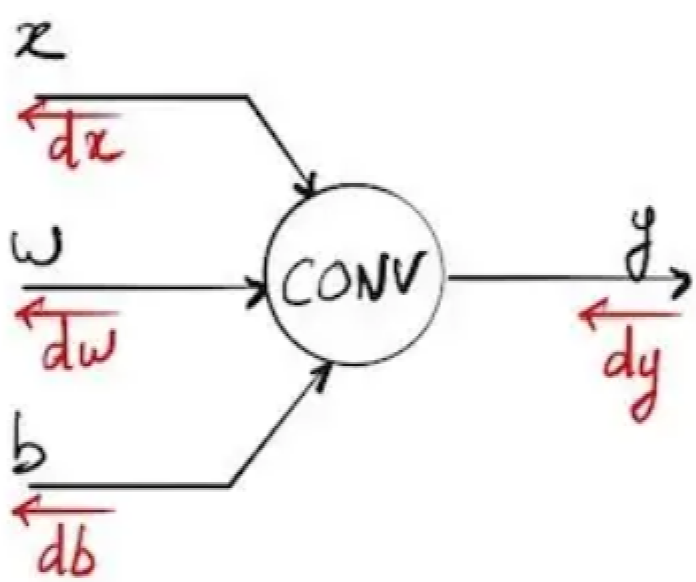
$$C_2 = \left(\frac{9+8+7}{3}, \frac{6+7+6}{3}\right) = (8, 6.333)$$

$$C_3 = \left(\frac{3+12}{2}, \frac{12+13}{2}\right) = (7.5, 12.5)$$

Section B
This is the basis fo cnn and how cnn works.For this particular assignment, we were expected to implement the convolution and the pooling layer.



Convolution Functions:



a) Convolution forward : This function is responsible for performing the convolution window operation on the entire numpy array part by part. In the above figure , the forward operation is carried out by this function in the end it stores the inputs and returns them. Most importantly it returns an output array with magnified last dimension.The input_i is used to extract each sample and do the computation on each. Input_ slice slices for each channel

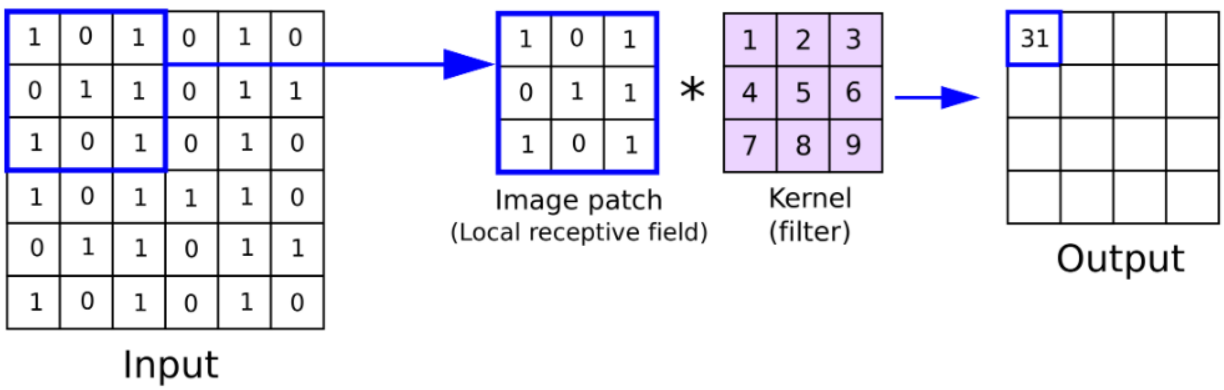
```
input = np.random.randn(8,3,3,2)
W = np.random.randn(2,2,2,6)
b = np.random.randn(1,1,1,6)
stride = 2
padding = 2
output,input,kernel,bias,stride,padding = convol_forward(input, W,b, 2,2)
print(output.shape)
d_output, d_kernel, d_bias = convol_backward(output,input,kernel,bias,stride,padding)
print(d_output.shape)
output,input,kernel,bias,stride,padding = convol_forward(d_output, d_kernel,d_bias, 2,2)
print(output.shape)
d_output, d_kernel, d_bias = convol_backward(output,input,kernel,bias,stride,padding)
print(d_output.shape)
```

```
input shape: (8, 3, 3, 2)
output shape: (8, 3, 3, 6)
after padding: (8, 7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
(8, 3, 3, 6)
(8, 3, 3, 2)
input shape: (8, 3, 3, 2)
output shape: (8, 3, 3, 6)
after padding: (8, 7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
i input (7, 7, 2)
(8, 3, 3, 6)
(8, 3, 3, 2)
```

b) Convolution backward: This function is responsible for computing the gradients to reduce the loss. It changes the values in the input , kernel and bias array so that in the next iteration we get loss value to be lesser.
Basically we reach the input array again and then make changes to it accordingly in this function.The input_i is used to extract each sample and do the computation on each. Input_ slice slices for each channel

```
[74] np.random.seed(1)
     d_output, d_kernel, d_bias = convol_backward(output,input,kernel,bias,stride,padding)
     print("d_output",d_output.shape)
     print("d_kernel",d_kernel.shape)
     print("d_bias",d_bias.shape)

     d_output (8, 3, 3, 2)
     d_kernel (2, 2, 2, 6)
     d_bias (1, 1, 1, 6)
```

c) Convolution Window: This function is the core of the convolution forward function. Basically what i have done is that i will take a window that is of same shape as the kernel and then perform the convolution operation for that particular window.
I will have to do w1x1 + w2x2….wnxn + bias

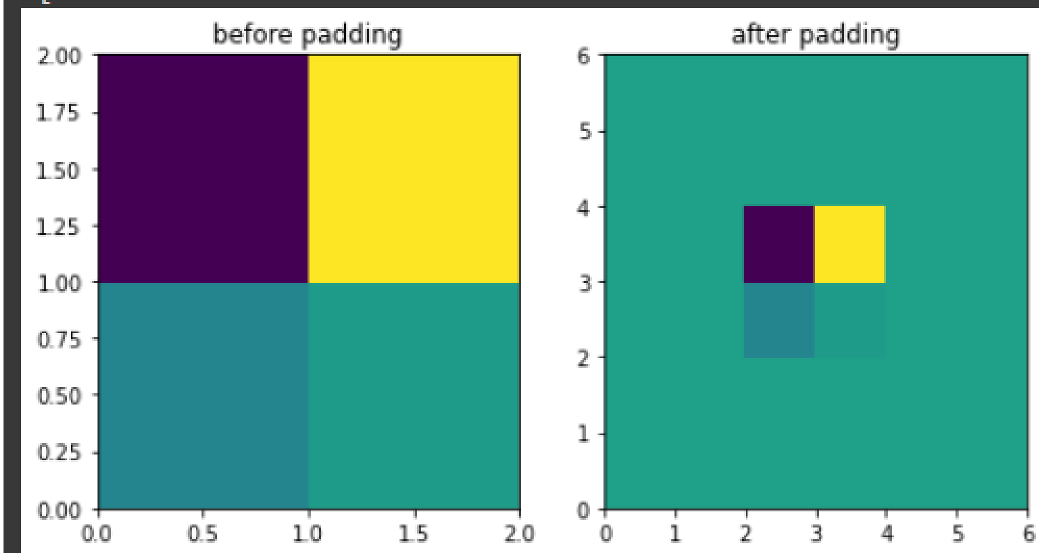



```
input window :  [[-0.37151912  3.16096597]
 [ 0.10995101 -1.91352322]]
kernel :  [[0.59982043 0.54938447]
 [1.38378103 0.14834924]]
convolution operation on given window of matrix = 1.3820192434310468
```

the z i get is 1.38 without adding the bias term. In case there is some other example where i am given any bias term for ex bias =1 , i simply add 1 to this returned value. i.e. 1.38+1 = 2.38.

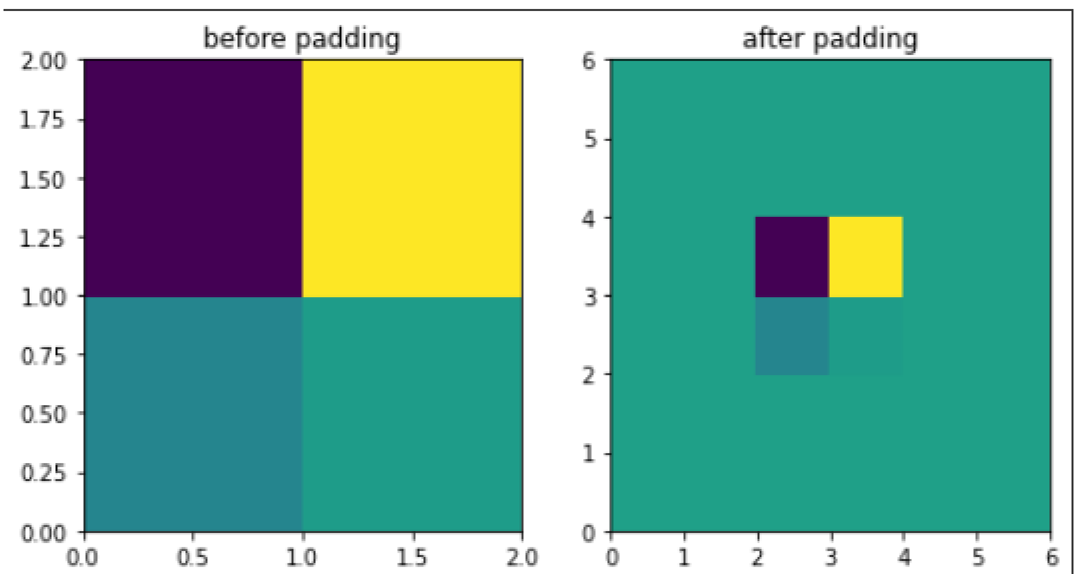
d) Zero padding: This function takes as input the greyscale or rgb image pixel array and a padding number and then returns an array that consists of zeros around the pixel array. for ex.
I gave as input a greyscale image array of size 2,2 and now the size after padding =2 becomes 6,6

```
input shape = (2, 2)
padded input shape = (6, 6)
[[ 0.          0.          0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.        ]
 [ 0.          0.         -0.41675785 -0.05626683  0.          0.        ]
 [ 0.          0.         -2.1361961   1.64027081  0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.          0.          0.        ]]
```



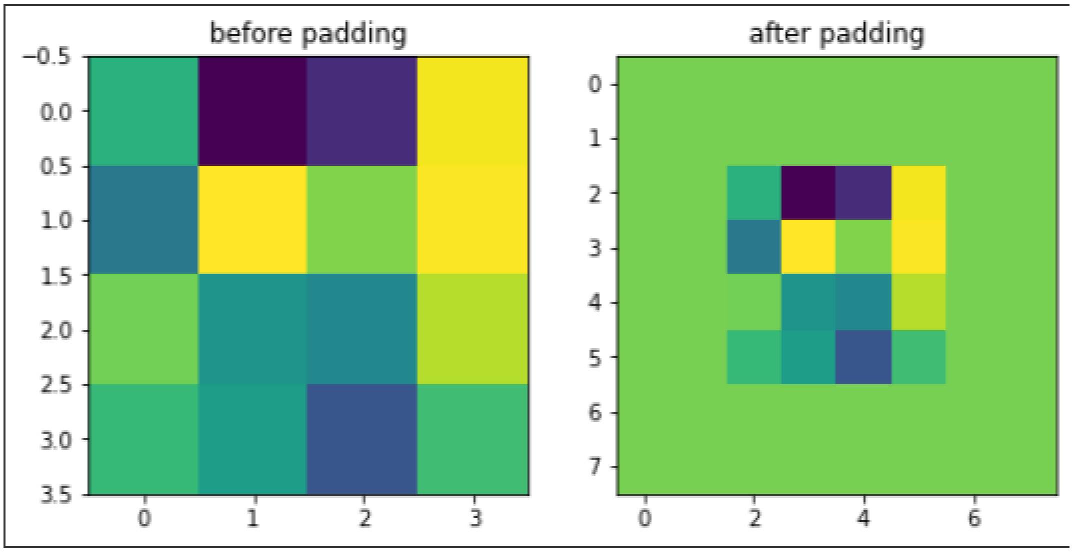
if the input shape is (2,2,1),

```
padded input shape = (6, 6, 1)
```



and in case in input we give 4 dimensions , which we will be normally doing , then the first dimension basically tells us the number of dimensions and rest 3 are the length breadth height as in the above example.

```
input shape = (10, 4, 4, 2)
padded input shape = (10, 8, 8, 2)
```



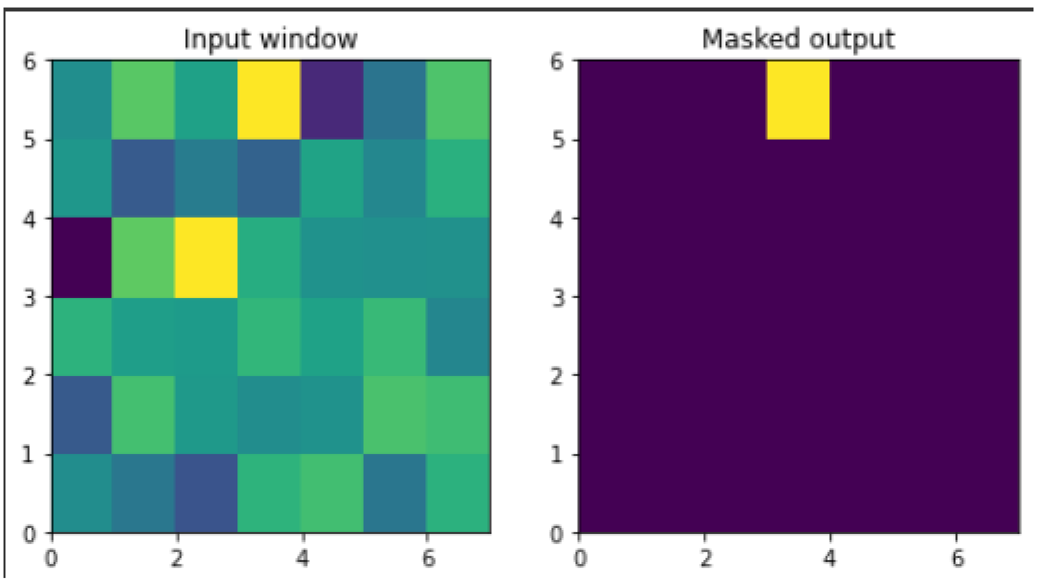
Pooling Layer functions:

a) Creating mask:

This function is used to carry out max pooling efficiently. It takes as input a window from the main input and then returns an output of the same size. This output window has zeroes at all places but a 1 at the index where the maximum element of the input window was present.

For ex. -

```
input : [[ 0.12182127  1.12948391  1.19891788  0.18515642 -0.37528495 -0.63873041
   0.42349435]
 [ 0.07734007 -0.34385368  0.04359686 -0.62000084  0.69803203 -0.44712856
   1.2245077 ]
 [ 0.40349164  0.59357852 -1.09491185  0.16938243  0.74055645 -0.9537006
  -0.26621851]
 [ 0.03261455 -1.37311732  0.31515939  0.84616065 -0.85951594  0.35054598
  -1.31228341]
 [-0.03869551 -1.61577235  1.12141771  0.40890054 -0.02461696 -0.77516162
   1.27375593]
 [ 1.96710175 -1.85798186  1.23616403  1.62765075  0.3380117  -1.19926803
   0.86334532]]
[[5 0]]
mask :  [[0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [1 0 0 0 0 0 0]]
```



b) Distribute values - This function is responsible for finding the average of the window in simple words. It takes the input window as a parameter and then return the average value from that window and then the new window where all values are average.

```
def distribute_value(input_window):
    average = np.sum(input_window) / (input_window.shape[0] * input_window.shape[1])
    output = np.ones(input_window.shape) * average
    return output,average
```

```
input_window = np.random.randn(2,2)
print("input :", input_window)
input_dv = distribute_value(input_window)
print("dv : ", input_dv)
```

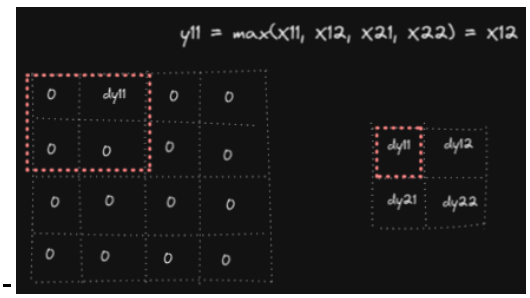
```
input : [[-0.74715829  1.6924546 ]
 [ 0.05080775 -0.63699565]]
dv :  (array([[0.0897771, 0.0897771],
       [0.0897771, 0.0897771]]), 0.08977710387089613)
```

c) Pooling forward - I first used a of statement to check for which type of pooling to perform. Later in case of max pooling I use the np.max function and for average pooling I use the np.average.first I compute the dimensions of the output . I compute the height and width of the layer using this formula the last dimension for channels remains the same. The input_i is used to extract each sample and do the computation on each. Input_ slice slices for each channel

```
o_dim0 = input.shape[0]
o_dim1 = int((input.shape[1] - f_s)/stride + 1)
o_dim2 = int((input.shape[2] - f_s)/stride + 1)
o_dim3 = input.shape[3]
output = np.zeros([o_dim0, o_dim1, o_dim2, o_dim3])
```

```
input = np.random.randn(8,4,4,2)
output,input, f_s, stride,pool_type = pool_forward(input,2,2,2)
print("Average pooling done")
print("output -", output)
```

```
input shape: (8, 4, 4, 2)
output shape: (8, 2, 2, 2)
Average pooling done
output - [[[[ 0.82736157  0.27962732]
   [ 0.03052259  0.68015756]]

  [[-0.23993969  0.56976662]
   [-0.20677016 -0.29956769]]]


 [[[ 0.38852168  0.56195571]
   [-0.26247947 -0.15615902]]
```

d) Pooling backward - 

This function is similar to the convolution backward but the inner computation is a little different. In case of max pool we use the masking function and for average the distribute value function is used. In both cases we get the output as same dimensions as the input of forward pooling. The input_i is used to extract each sample and do the computation on each. Input_ slice slices for each channel

```
input = np.random.randn(7,7,3,2)
output,input, f_s, stride,pool_type = pool_forward(input,2,1,1)
print("max pooling done")
#print("output -", output)
d_output = np.gradient(output)
d_output = np.array(d_output)
d_output = np.resize(d_output,output.shape)
print(d_output.shape)
print(output.shape)
print(input.shape)
pb_out = dA_prev = pool_backward(d_output, input, f_s, stride, pool_type)
print(pb_out.shape)
```

```
input shape: (7, 7, 3, 2)
output shape: (7, 6, 2, 2)
max pooling done
(7, 6, 2, 2)
(7, 6, 2, 2)
(7, 7, 3, 2)
[[0 0]]
[[1 0]]
[[1 1]]
[[1 0]]
[[1 1]]
[[0 0]]
```

Section C

Section C

a) dataset loading done
b) replaced the ? to nan values using the np.nan . later to confirm the nan value counts.The below 4 features contain more than 40% nan values. These features were hence dropped.

```
30    PARENT    5

[7] df = df.replace(" ?",np.nan)
    df2 = df2.replace(" ?",np.nan)

df
```

```
0] df.isnull().sum()
   df2.isnull().sum()
```

```
AAGE           0
ACLSWKR        0
ADTIND         0
ADTOCC         0
AHGA           0
AHRSPAY        0
AHSCOL         0
AMARITL        0
AMJIND         0
AMJOCC         0
ARACE          0
AREORGN        0
ASEX           0
AUNMEM         0
AUNTYPE        0
AWKSTAT        0
CAPGAIN        0
CAPLOSS        0
DIVVAL         0
FILESTAT       0
GRINREG        0
GRINST        14
HHDFMX         0
HHDREL         0
MIGMTR1     1906
MIGMTR3     1906
MIGMTR4     1906
MIGSAME        0
MIGSUN      1906
NOEMP          0
PARENT         0
PEFNTVTY     162
PEMNTVTY     134
PENATVTY      93
PRCITSHP       0
SEOTR          0
VETQVA         0
VETYN          0
WKSWORK        0
YEAR           0
dtype: int64
```

```
# #59856
df = df.drop(['MIGMTR1','MIGMTR3','MIGMTR4','MIGSUN'],axis=1)
df2 = df2.drop(['MIGMTR1','MIGMTR3','MIGMTR4','MIGSUN'],axis=1)
```

c) From the dataset description I checked for the numerical values and further performed the bins operation on them.
I separated them into 4 different labels each.
I computed the mode of each column and then replaced the nan values with the mode

```
[14] num_df = df3[df3['Type'] == 'numerical']
     num_df
```

	Column Code	Distict Values	Column	Type
0	AAGE	91	age	numerical
5	AHRSPAY	1240	wage per hour	numerical
16	CAPGAIN	132	capital gains	numerical
17	CAPLOSS	113	capital losses	numerical
18	DIVVAL	1478	divdends from stocks	numerical
38	WKSWORK	53	weeks worked in year	numerical

```
[15] df['AAGE'] = pd.cut(df['AAGE'],4,labels=['children','youth','adults','seniors'])
     df2['AAGE'] = pd.cut(df2['AAGE'],4,labels=['children','youth','adults','seniors'])
```

```
[16] df['AAGE'].value_counts().sort_index()
     df2['AAGE'].value_counts().sort_index()
```

```
children     795
youth       2038
adults       637
seniors      110
Name: AAGE, dtype: int64
```

```
print(df['AHRSPAY'].unique())
print(df['AHRSPAY'].max())
print(df2['AHRSPAY'].unique())
print(df2['AHRSPAY'].max())
```

```
[   0 1200  876 ... 3156 2188 1092]
9999
[   0 1550 2900 2100 1250 3259 1800 1480  500 2000 1200 1500  400 1600
  620 3300  585 2045 2751  897 2500  800 1700  885 1850  700 2200  750
  600 2300 1925 1300  625 2250 2291 4807  445 2552 2700 1260 1880 1100
 2400 1076 1400 2512  715  850 1385 2350 1900  963 3000 2739 1535 1844
 1910  951 3700 1000  450 1950 2450 1721 1215 1693  550 2532 1799  790
 2326 2010 2915 8000  525 3200 2155 1677 2613  840 1175 2256 2116 1758
 3750]
8000
```

```
df['AHRSPAY'] = pd.cut(df['AHRSPAY'], bins=[0, 500, 1500, 3000, 10000])
df['AHRSPAY'].value_counts().sort_index()
df2['AHRSPAY'] = pd.cut(df2['AHRSPAY'], bins=[0, 500, 1500, 3000, 10000])
df2['AHRSPAY'].value_counts().sort_index()
```

```
(0, 500]          7
(500, 1500]      47
(1500, 3000]     86
```

```
[28] for i in df.columns[df.isna().any()]:
       mode=df[i].mode()[0]
       df[i].fillna(mode,inplace=True)
       df2[i].fillna(mode,inplace=True)
```

```
[29] print(df.isna().sum())
     print(df2.isna().sum())

     CAPGAIN     0
     CAPLOSS     0
     DIVVAL      0
     FILESTAT    0
     GRINREG     0
     GRINST      0
     HHDFMX      0
     HHDREL      0
     MIGSAME     0
     NOEMP       0
     PARENT      0
     PEFNTVTY    0
     PEMNTVTY    0
     PENATVTY    0
     PRCITSHP    0
     SEOTR       0
     VETQVA      0
     VETYN       0
     WKSWORK     0
     YEAR        0
     dtype: int64
```

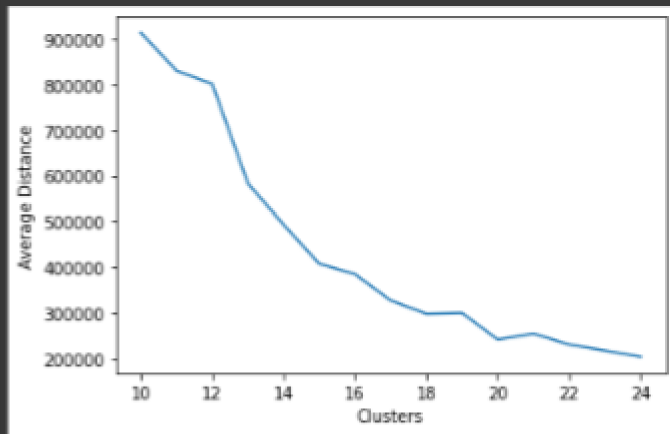
I performed one hot encoding and then on the obtained df performed Pca
Using Pca the number of features were reduced to 35.

```
     dtype: int64

[30] df = pd.get_dummies(df)
     df_encode = df.to_numpy()
     print(df.shape)

     (199523, 389)

[31] df2 = pd.get_dummies(df2)
     df2_encode = df2.to_numpy()
```

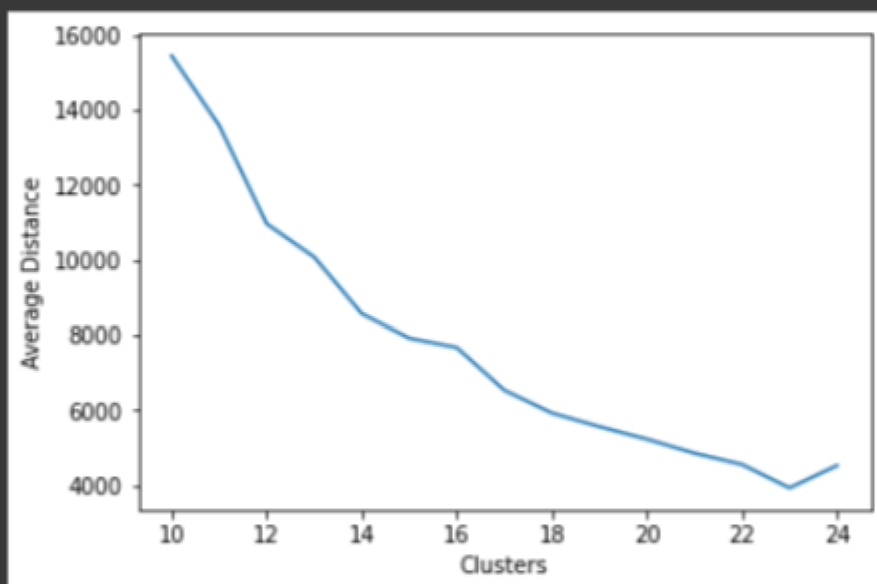
For the general population dataset the optimal k value is 21 as post that the value is decreasing almost linearly
For the more than 50k dataset, the optimal k value is 16 and the reason remains the same.

d) For k median clustering I used the pyclustering library and later plotted these graphs for k values from 10 to 24

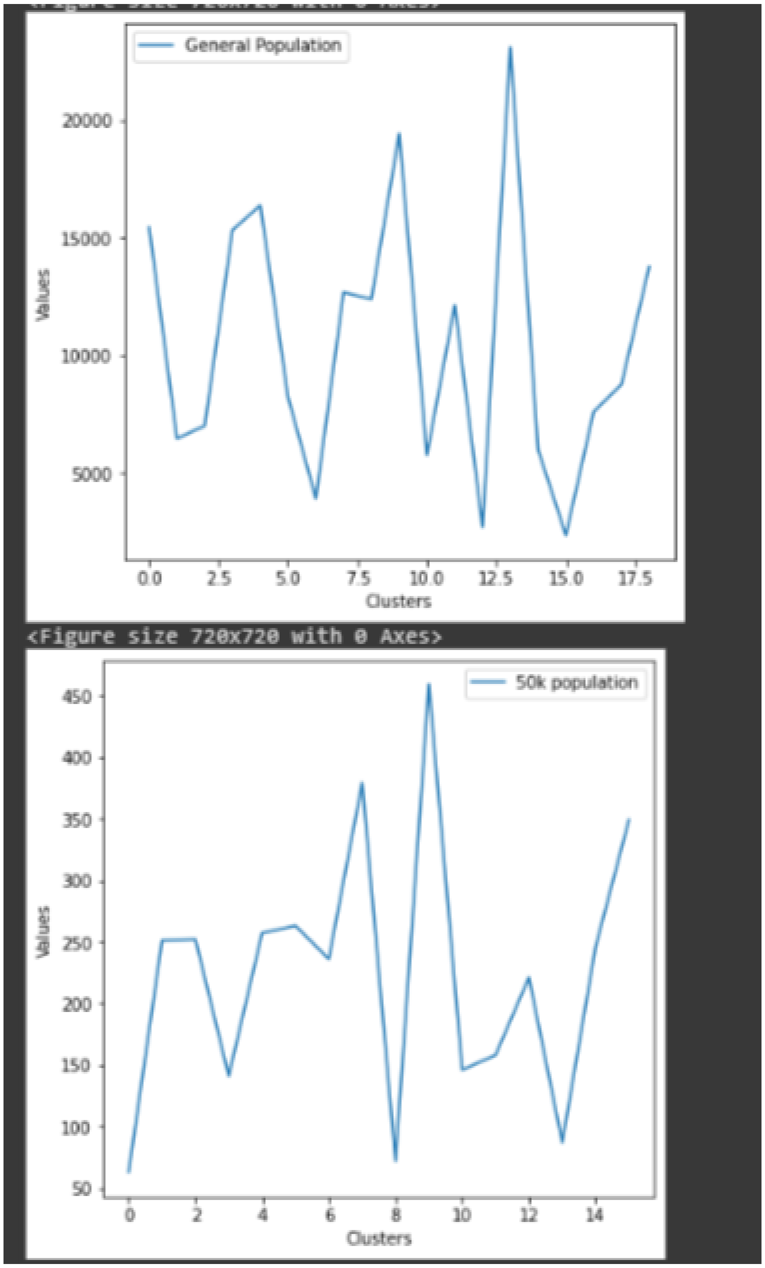
```
plt.show()
```




```
plt.show()
```




```
45] samples2 = pca2_df[0:16]
```



e) I have simultaneously performed all the computations on the more than 50k dataset too hence done above

f) Overrepresentation is the representation of a group of data points that differ substantially higher from the representation of other data points.The cluster number 1,3,5,8,9,13,18 are over represented in the general population dataset while the cluster 2,4,6,10,11,12 14,15 are over represented in the 50k dataset than in general population dataset.Post this I performed the inverse Pca on the datasets for these particular overrepresented features.