Machine Learning
Assignment - 4

Ashwin Sheoran
2020288

Section-A

Ans 1)

a)

Ans1) Here we are given An image. 15×15×4, 4 is the Number of channels it is passed through given CNN.

Input $\longrightarrow$ Conv 2D

15×15

Kernel ~~5×5×4~~ 5×5×4×1

Padding = 1

Stride = 1

$\left[\left(\frac{17-5}{1}\right)+1\right] \times \left[\left(\frac{17-5}{1}\right)+1\right]$

13×13 Image.

Max Pooling.

Kernel 3×3

Stride 2.

$\left[\left(\frac{13-3}{2}\right)+1\right] \times \left[\left(\frac{13-3}{2}\right)+1\right]$

6×6 Image

Conv 2D

Kernel 5×3×4×1

Padding 2

Stride 2

$\left[\left(\frac{10-5}{2}\right)+1\right] \times \left[\left(\frac{10-3}{2}\right)+1\right]$

Output Image 3×4

Output Image is 3×4.

b) We require pooling to reduce the dimensions of the image to obtain the features that it consists of. Without pooling, it is difficult to handle the entire image of a big size.

c) Learnable parameters are the parameters that are learnt by the model while it trains, This number is given by
Kernel size * Number of kernels (given in lecture slides)
Learnable parameters = Kernel size * Number of kernels = 5x5x4x1 + 5x3x4x1 = 100 + 60 = 160
There are 160 Learnable parameters

Ans 2)
We have been given initial clusters A(3,12), B(8,7), C(2, 13)
We will 1st calculate the distance of points from all the 3 centres to check if the points are in correct clusters or in the wrong clusters

Points	Distance from A (3, 12)	Distance from B (8,7)	Distance from C (2,13)	Cluster in which point is
(3,12)	0	5	1	A
(3,7)	5	2.5	5.5	B
(9,6)	6	1	7	B
(6,10)	2.5	2.5	3.5	A
(8,7)	5	0	6	B
(7,6)	5	1	6	B
(2,13)	5	6	0	C

We have our cluster A = { (3,7) , (6,10) }
We have our clusters B = { (9,6) , (3,7) , (8,7) , (7,6) }
We have our clusters C = { (2,13) }
Now since we have points distributed into our clusters, we have to now find the new centres
New center of A = (3+6)/2 , (10+7)/2
New center of A = (4.5 , 8.5)
New center of B = (9+3+8+7) / 4 , (6+7+7+6) / 4
New center of B = (6.75 , 6.5)
New center of C = (2,13)

This was the 1st iteration, now we will do the next iteration, with our new cluster centres to check if you clusters are correct

2nd Iteration:-

Points	Distance from A (4.5, 8.5)	Distance from B (6.75,6.5)	Distance from C (2,13)	Cluster in which point is
(3,12)	2.5	4.6	1	C
(3,7)	1.5	2.1	5.5	A
(9,6)	3	1.3	7	B
(6,10)	1.5	2.1	3.5	A
(8,7)	2.5	0.8	6	B
(7,6)	2.5	0.3	6	B
(2,13)	3.5	5.6	0	C

We have our new clusters

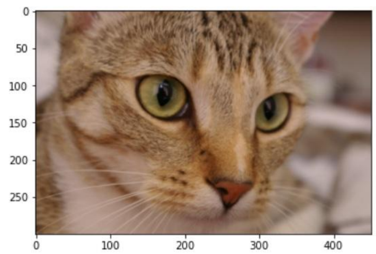
New clusters are
A = { (3,7) , (6,10) }
B = { (9,6) , (8,7) , (7,6) }
C = { (3,12) , (2,13) }

The cluster centres are
Centre A = (3 + 6) / 2 , (7 + 10) / 2 = (4.5 , 8.5)
Centre B = (9+8+7) / 3 , (6+7+6) / 3 = (8 , 6.3)
Centre C = (3+2) / 2 , (12+13)/2 = (2.5 , 12.5)
There 3 centres are
A = (4.5 , 8.5)
B = (8 , 6.3)
C = (2.5 , 12.5)

Section-B

Ans 1)
a)
Selected Image

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import skimage.data
img = skimage.data.chelsea()
# plt.imshow(img)
plt.imshow(img , cmap='gray')

data = img.reshape(1,img.shape[0], img.shape[1], img.shape[2])

np.random.seed(1)
```



Zero Padding Function

```
def padding(data):
    data_padded = np.pad(array = data, pad_width = ((0,0),(0 , 0), (0,0), (0,0)),
    mode = 'constant', constant_values = 0)
    return data_padded

A_prev_pad = padding(data)  #adding the padding
```

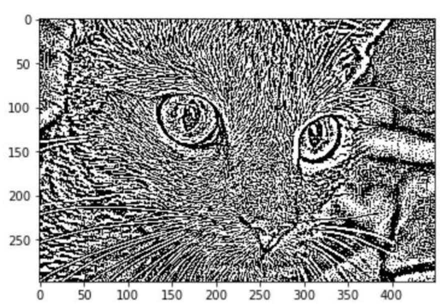
Here we are implementing Padding (Adding 2 zeroes to the edges of the image), this is done so that convolutions happen equally and the corners are not used fewer times than the middle parts of the image. The borders of the image are also given equal importance during convolution so extra layers of 0 are added so that the corner edges are pushed to the middle.

Forward Convolution

```
def conv_forward(A_org, fil,  stride):

  m, org_h, org_w, org_c = A_org.shape

  f, f, org_c, n_c = fil.shape

  out_h = int((org_h - f )/stride) + 1  ## Calculating the dimension of output image
  out_w = int((org_w - f )/stride) + 1

  out_img = np.ones((m, out_h, out_w, n_c))
  h_start=[0]*2
  h_end=[0]*2
  w_start=[0]*2
  w_end=[0]*2
  for i in range(out_h):
    for j in range(out_w):
      for k in range(n_c):
        h_start[0] = i * stride
        h_end[0] = f + h_start[0]
        w_start[0] = j * stride  ## Convulation process
        w_end[0] = f+ w_start[0]

        out_img[0,i,j,k] = np.sum( np.multiply(A_org[0, h_start[0]:h_end[0], w_start[0]:w_end[0], :] ,fil[:,:,:,k] ) )
  return out_img
```

```
filter = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]]).reshape((3,3,1,1)) ## The ken

## The center one is 8 rest are -1
conv_img = conv_forward(data, filter, 1)
plt.imshow(conv_img[0,:,:,0], cmap='gray',vmin=0, vmax=1)
```

```
<matplotlib.image.AxesImage at 0x7fd67af66040>
```



In convolution, we basically take an image and pass it through the filter(kernel); applying a kernel to an image helps us identify the features of the image.

Backward Convolution

```
def conv_backward(out_img, used_filter ,  rate , W , stride):
    filter  = np.zeroes(used_filter.shape)
    (m, out_h, out_w, out_c) = out_img.shape
    (f, f, n_C_prev, n_C) = W.shape

    in_h = ((out_h - 1)*stride) + f
    in_w = ((out_w - 1)*stride) + f
    in_h=[0]*2
    out_h=[0]*2
    in_w=[0]*2
    out_w=[0]*2
    org_img = np.zeros((m, in_h, in_w, n_C))

    for i in range(out_h):
        for j in range(out_w):
            for k in range(n_C):
                out_h[0] = i / stride
                out_w[0] = j / stride  ## back ward Convulation process
                in_h[0] = out_h[0] - f
                in_w = out_h[0] - f
    org_img[0,i,j,k] = np.sum( np.multiply(out_img[0, in_h[0]:out_h[0], in_w[0]:out_w[0], :] ,W[:,:,:,k] ) )
    return org_img
```

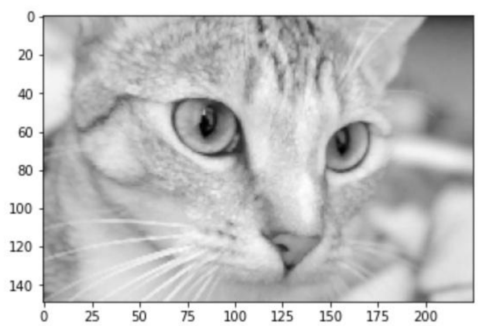
In Backward Convolution, we basically take a convoluted image and try to generate a non-convoluted image that it was originally before passing through the filter. We basically do an inverse of all the operations done during forward convolution to get the original image from the convoluted image using backward propagation.

Forward Pooling

```
def max_pool_forward(input, stride , filter_size) :
    m, h_prev, w_prev, c_prev = input.shape
    h_out = int(((h_prev - filter_size)/stride) + 1)
    w_out = int(((w_prev -filter_size)/stride) + 1)
    output = np.zeros((m, h_out, w_out, c_prev))

    h_start=[0]*2
    h_end=[0]*2
    w_start=[0]*2
    w_end=[0]*2
    for i in range(c_prev):
        for j in range(h_out):
            for k in range(w_out):
                w_start[0] = k * stride
                w_end[0] = w_start[0] + filter_size
                h_start[0] = j * stride
                h_end[0] = h_start[0] + filter_size
                output[0, j, k, i] = np.max(input[0,h_start[0]:h_end[0], w_start[0]:w_end[0], i])
    output.shape == (m, h_out, w_out, c_prev)
    return output
```

```
pooled_img =  max_pool_forward(data, 2 , 3) ## Stride = 2
plt.imshow(pooled_img[0,:,:,0], cmap = "gray")
print("The Dimension of the image before pooling are ", conv_img.shape)
print("The Dimension of the image after pooling are ", pooled_img.shape)
```

```
The Dimension of the image before pooling are  (1, 298, 449, 1)
The Dimension of the image after pooling are  (1, 149, 225, 3)
```



In max forward pooling, we try to make sure that we reduce the size of the image without altering any of the features of the original image; we do this by taking the image as a matrix and reducing its dimensions by taking maximum value from a group of pixels(elements). This reduces the dimensions, but the features are mostly unchanged.

Section-C

Ans 1)
a)

Preprocessing

```
def removecol(data):

    data = data.drop([ 'AHSCOL' , 'ARACE' , 'AREORGN', 'AUNMEM'  , 'AUNTYPE' , 'GRINREG' , 'GRINST' ,
     'PEFNTVTY' , 'PEMNTVTY' ,
     'PENATVTY' , 'PRCITSHP' , 'VETQVA'] , axis = 1 )

    return data
```

```
## preprocessing
df= df.replace('[?]', np.nan, regex=True)
df2= df2.replace('[?]', np.nan, regex=True)

limitPer = len(df) * .30
yourdf = df.dropna(thresh=limitPer, axis=1)

(df.fillna(df.mode().iloc[0]))
(df2.fillna(df2.mode().iloc[0]))
```

	AAGE	ACLSWKR	ADTIND	ADTOCC	AHGA	AHRSPAY	AMARITL	AMJIND	AMJOCC	ASEX	...	MIGMTR3	MIGMTR4	MIGSAME	MIGS
0	57	Self-employed-incorporated	11	2	High school graduate	0	Married-civilian spouse present	Manufacturing-durable goods	Executive admin and managerial	Male	...	Nonmover	Nonmover	Not in universe under 1 year old	Nc unive
1	44	Private	3	38	High school graduate	0	Married-civilian spouse present	Mining	Transportation and material moving	Male	...	Nonmover	Nonmover	Yes	Nc unive
2	54	Private	19	26	High school graduate	1550	Married-civilian spouse present	Manufacturing-nondurable goods	Adm support including clerical	Male	...	Nonmover	Nonmover	Yes	Nc unive
3	25	Private	33	2	9th grade	0	Never married	Retail trade	Executive admin and managerial	Male	...	Nonmover	Nonmover	Not in universe under 1 year old	Nc unive

After replacing ? with Nan, We have removed columns that have more than 30% missing data, We have removed the columns that can not be bucketed and hot encoded, and we have filled nan values with the modes.

Using bins in the data

```
df = binning(df)
df2 = binning(df2)
df
```

	Binned_AAGE	Binned_ADTIND	Binned_ADTOCC	Binned_AHRSPAY	Binned_CAPGAIN	Binned_CAPLOSS	Binned_DIVVAL	Binned_NOEMP	Binned_SEOTR
0	VERY HIGH	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
1	VERY HIGH	MEDIUM	VERY HIGH	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MEDIUM	MINIMAL
2	LESS	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
3	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
4	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
...	...	...	...	...	...	...	...	...	...
199518	VERY HIGH	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
199519	VERY HIGH	VERY HIGH	MEDIUM	MINIMAL	VERY HIGH	MINIMAL	VERY HIGH	MEDIUM	MINIMAL
199520	HIGH	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	VERY HIGH	VERY HIGH	MINIMAL
199521	LESS	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL	MINIMAL
199522	MEDIUM	VERY HIGH	VERY HIGH	MINIMAL	MINIMAL	MINIMAL	MINIMAL	VERY HIGH	MINIMAL

199523 rows × 12 columns

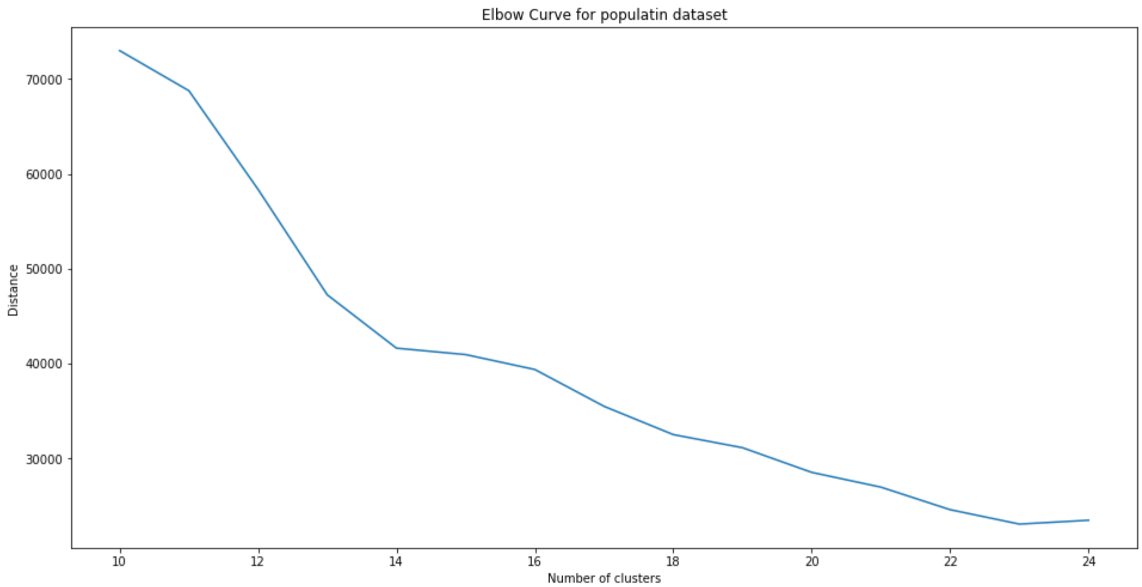
Imputation, Bucketization, and One-Hot Encoding

Bucketization of the data,

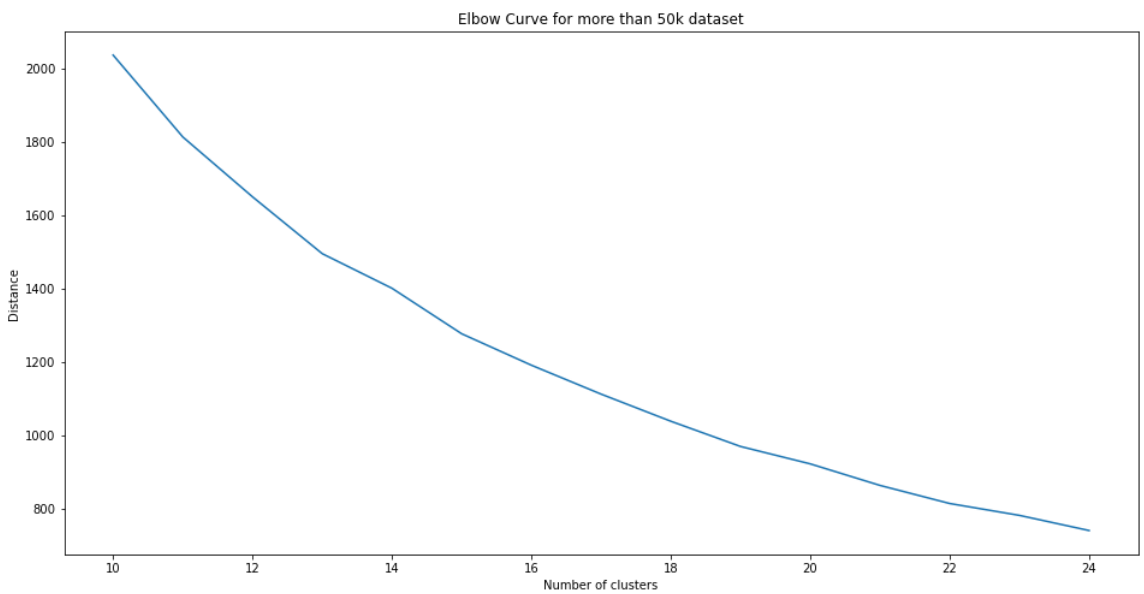
For example, some buckets of the 1st dataset.

EOTR	VETYN	...	CapitalLoss_Medium	CapitalLoss_High	Dividend_Minimal	Dividend_Less	Dividend_Medium	Dividend_High	Weeks_Minimal	Weeks_Less	Weeks_
0	2	...	0	0	0	0	0	0	0	0	
0	2	...	0	0	0	0	0	0	0	0	
0	2	...	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	2	...	0	0	0	0	0	0	0	0	
0	2	...	0	0	1	0	0	0	0	0	
0	2	...	0	0	1	0	0	0	0	0	
0	2	...	0	0	0	0	0	0	0	0	
0	2	...	0	0	0	0	0	0	0	0	

We have used PCA to reduce the data dimensions to 20.



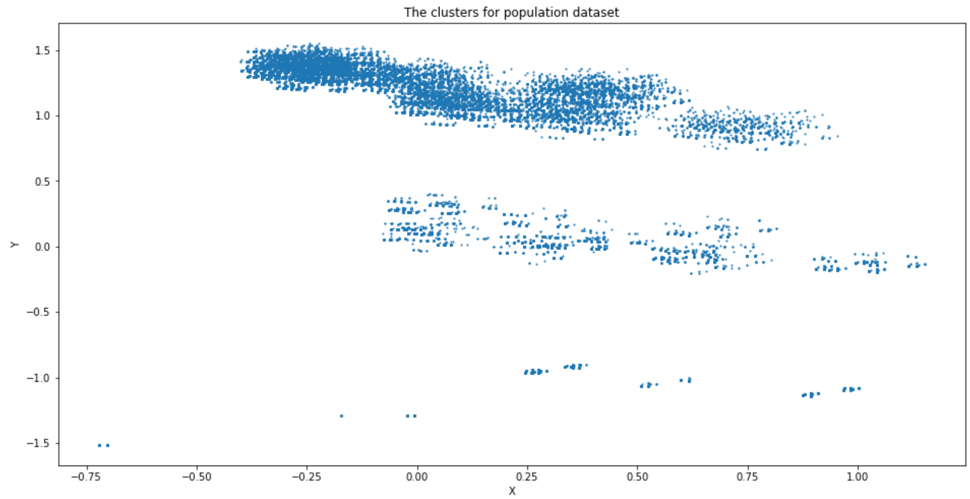
The best value of k that I think is 14, as we can see a clear change in slope (elbow)

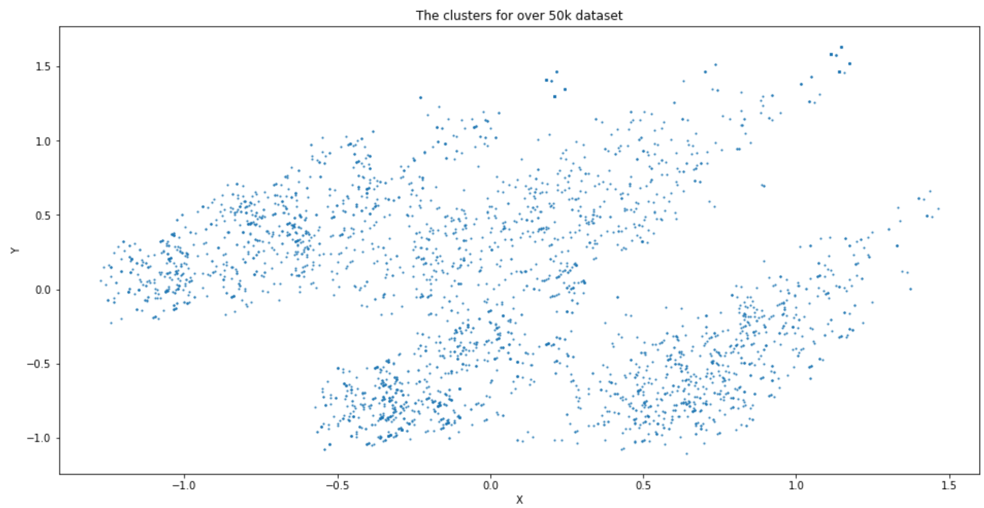


The best value of k that I think is 15, as we can see a clear change in slope.

ScatterPlots

For these scatterplots, we reduced the dimensions to 2: x and y





Comparisons

Visually looking, we can see that for some clusters, we can see that the population dataset has more points per cluster than the over 50k dataset; this is because 50k is smaller than population dataset.

The over 50k dataset has fewer points and is more spread out and thus has more clusters than the population dataset.