Mohammad Sufyan Azam,
2020312

Assignment 4

Section-A

A1) (a) ⓐ input image size = $15 \times 15 \times 4$

kernels = $h \times w \times I \times O$

Output after first conv layer - (kernel = $5 \times 5 \times 4 \times 1$)

$$size = \left(\frac{15 + 2\times1 - 5}{1} + 1\right) \times \left(\frac{15 + 2\times1 - 5}{1} + 1\right)$$

$= 13 \times 13$, and no of output channels = 1,

So, image size = $13 \times 13 \times 1$

Output after maxpooling layer - (kernel = 3×3)

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

$= 6 \times 6$

So, image size = $6 \times 6 \times 1$ (it doesn't affect ~~no~~ of channels)

Output image after second conv layer - (kernel = $5 \times 3 \times 4 \times 1$)

$$size = \left(\frac{6 + 2\times2 - 5}{2} + 1\right) \times \left(\frac{6 + 2\times2 - 3}{2} + 1\right)$$

$\approx 4 \times 4$ (rounding off)

Hence, output image size = $4 \times 4 \times 1$

Mohammad Sufyan Azam,
2020312

(b) Pooling helps to reduce the dimensionality of the image without loosing features, thus making it more computationally efficient. It also helps in making the model learn its features ~~rallo~~ thus introducing translational equivariance property to it, making it more robust. It also helps in ~~combat~~ solving the problem of vanishing gradient which may occur in a model if it has large no of neurons & layers.

(c) The total no of learnable parameters are -
First Conv layer: $5\times5\times4 = 100$ (kernel size)
A maxpooling layer doesn't have learnable parameters.
Second Conv layer: $5\times3\times4 = 60$
Thus, total parameters = 160 (without bias)

(b) No. If the k-means algorithm revisits a particular configuration then it ~~won't~~ will form a loop as it will visit the ~~steps~~ configurations after them too, thus making it run indefinitely. The k-means algo can't revisit a configuration before because of the nature of objective function defined on it. It states that minimize the intra-cluster distance while maximizing the inter-cluster distance which ~~hal~~ is essentially a hill-climbing algorithm, ~~the~~ Thus, it can't revisit a previous configuration which guarantees its convergence.

Mohammad Sufyan Azam,
2020312

(c) Linear filters can detect simple linear patterns like edges, lines, etc. Thus, it can be used to extract low-level features. Non-linear filters can identify more complex patterns too in the data, thus making it more usable to extract high-level features too. It increases the expressive power of CNNs.

Section B

Convolution:

All the necessary functions were implemented for it -

- **Forward Convolution Pass:** This involves sliding the kernel over the input image. At each position, an element-wise multiplication is performed between the kernel and the section of the image it currently covers, followed by summing up the results to get a single output pixel. This process is repeated for all positions that the kernel can slide to, resulting in a new image.
- **Backward Convolution Pass:** This was a part of the backpropagation process in training a Convolutional Neural Network model. This function calculates the gradient of the loss function with respect to the parameters of the convolution layer (the kernel values), and with respect to its input. This involves applying the chain rule to propagate the gradients backward through the network.
- **Windowing:** A small window of the input image is extracted for processing. This window is the section of the image that the kernel covers at a particular position. The size of the window is the same as the size of the kernel.
- **Zero-Padding:** This was achieved by adding zeros around the border of the input image.

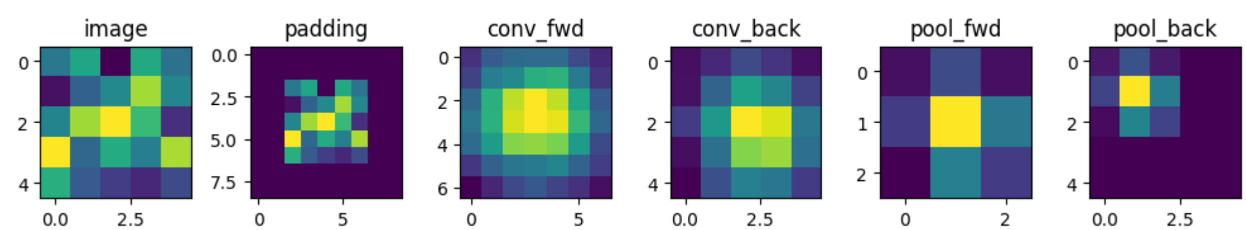
Mohammad Sufyan Azam,
2020312

Pooling:

All the necessary functions were implemented for it -

- **Forward Pooling Function:** This function applies a max pooling operation over the input image. It involved sliding a pooling kernel over the input, and for each position, taking the maximum value within the window was taken for creating the output image thus reducing image dimensionality.
- **Mask Creation:** This function helped in creating the mask during the forward pooling process to remember the location of the maximum value within each window. This mask was the same size as the window and has a value of 1 at the position of the maximum value and 0 elsewhere. This mask is used in the backward pooling function to distribute gradients back to the input.
- **Backward Pooling Function:** This function distributes the gradient from the output of the pooling layer back to its input, based on the masks created during the forward pooling function. The gradient is passed back only to the input value that was the maximum within its window. It is an essential part of the backpropagation process in training the CNN.
- **Value Distribution:** This is a step in the backward pooling function where the gradient is distributed back to the input. The way the gradient is distributed depends on the type of pooling operation. For max pooling, the gradient is passed back only to the position of the maximum value within each window (as indicated by the mask). For average pooling, the gradient is distributed evenly to all values within the window.

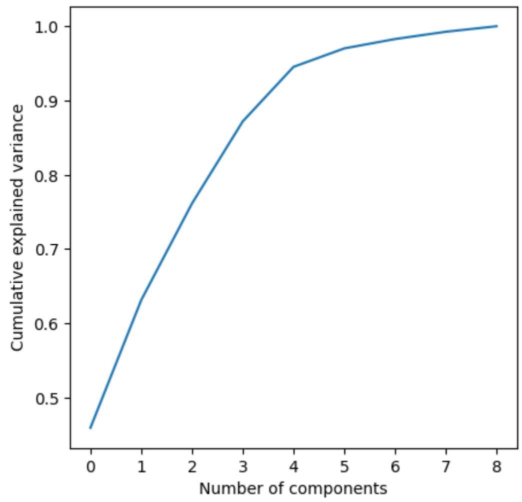
A random image and kernel was created of specified size. The result of each step was observed as follows -



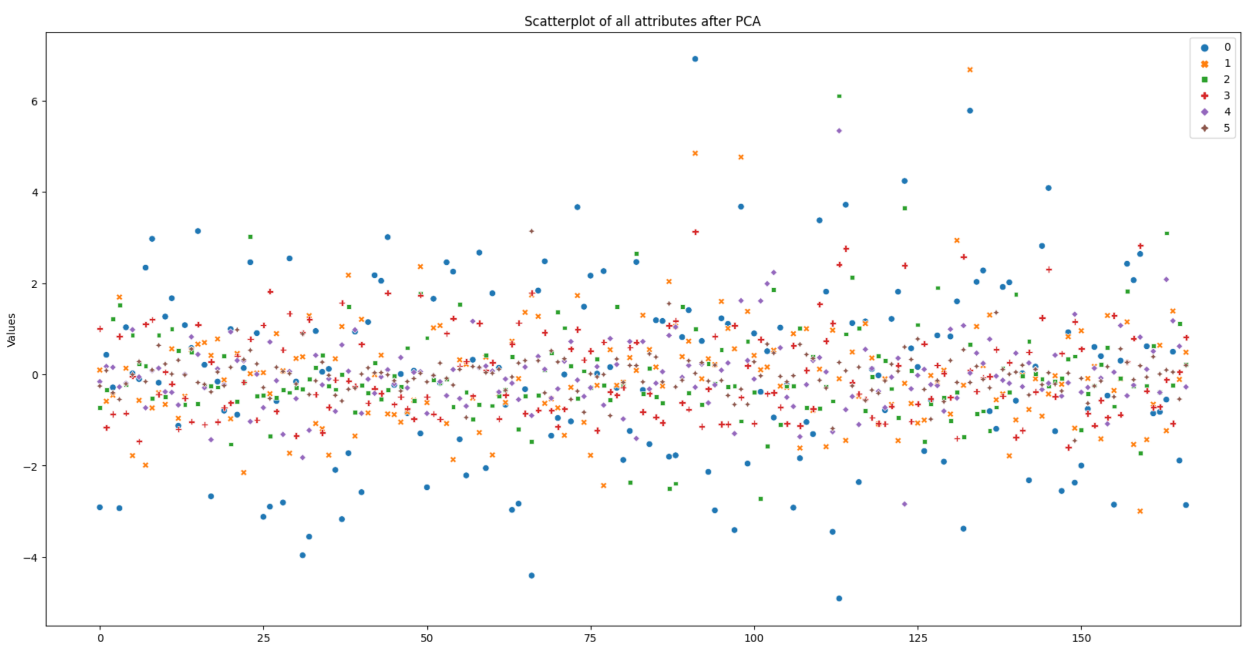
Mohammad Sufyan Azam,
2020312

Section C

To standardize the data StandardScaler() was used.
After applying PCA for varying number of components, we got the following graph -

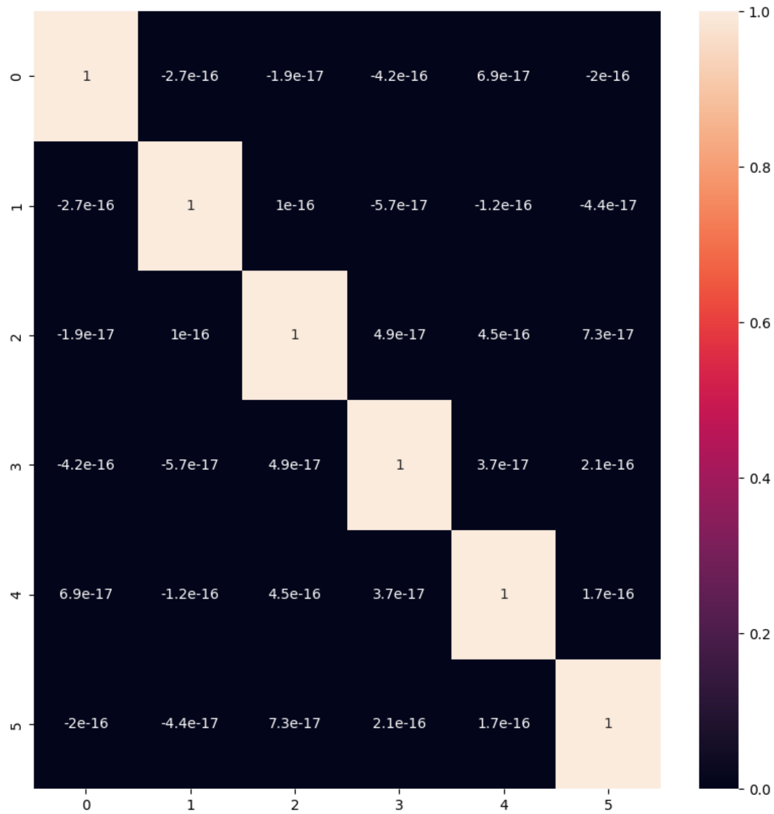


Thus, from here, we could determine that the number of components for optimal PCA would be 6. This was also cross verified using PCA with 'mle' as the number of components, and the results matched with it.
Scatterplot after applying PCA for optimal n_components -

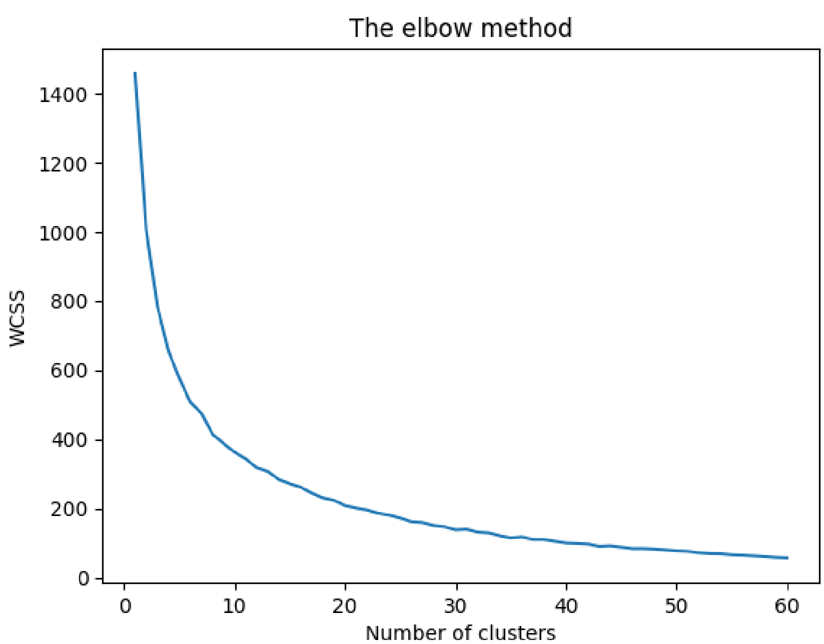


Mohammad Sufyan Azam,
2020312

Heatmap for the dataset after applying optimal n_components in PCA -



K-Means Clustering Algorithm -



Mohammad Sufyan Azam,
2020312

SIlhouette Method -

```
Optimal number of clusters with threshold 0.001: 3
0.29637364602520705
```

The second line denotes the score.

```
[0 2 2 0 2 2 2 1 1 2 2 1 2 2 2 1 2 0 2 2 2 0 2 1 2 0 0 2 0 1 2 0 0 2 2 2 0
 0 0 2 0 2 1 1 1 2 2 2 2 0 0 2 2 1 1 0 0 2 1 0 1 2 2 0 0 2 0 2 1 2 2 2 0 1
 1 1 2 1 2 2 0 0 1 2 0 2 2 0 0 2 2 1 2 0 0 2 2 0 1 0 2 2 2 2 2 2 0 2 0 2 1
 1 0 0 1 2 0 2 2 2 2 2 1 1 2 2 0 2 2 0 2 2 0 1 1 1 2 0 1 1 2 2 0 2 1 1 2 0
 2 0 0 2 2 2 2 0 2 1 1 1 2 2 2 2 2 0 0]
[[-2.43461995  0.41127566 -0.09616666  0.69171004 -0.14145844 -0.04411636]
 [ 2.76977273 -0.21301421  0.06888632  0.86680107 -0.17822588 -0.02626124]
 [ 0.17518237 -0.13882672  0.02428483 -0.75851441  0.15553189  0.03593897]]
786.7321895964274
11
```

The first array denotes the labeling of each data point. The second array denotes the centroids of each data point. The third line denotes the wcss score from kmeans.inertia_ and the fourth line denotes the number of iterations.