Machine Learning(CSE-343)
Assignment - 4

Section A - Theoretical

Part (a): Consider the forward pass of a convolutional layer in a neural network architecture, where the input is an image of dimensions M × N (where min(M, N) ≥ 1) with P channels (P ≥ 1) and a single kernel of size K × K (1 ≤ K ≤ min(M, N)).

a. **Given a stride of 1 and no padding, determine the dimensions of the resulting feature map.**
   Solution:

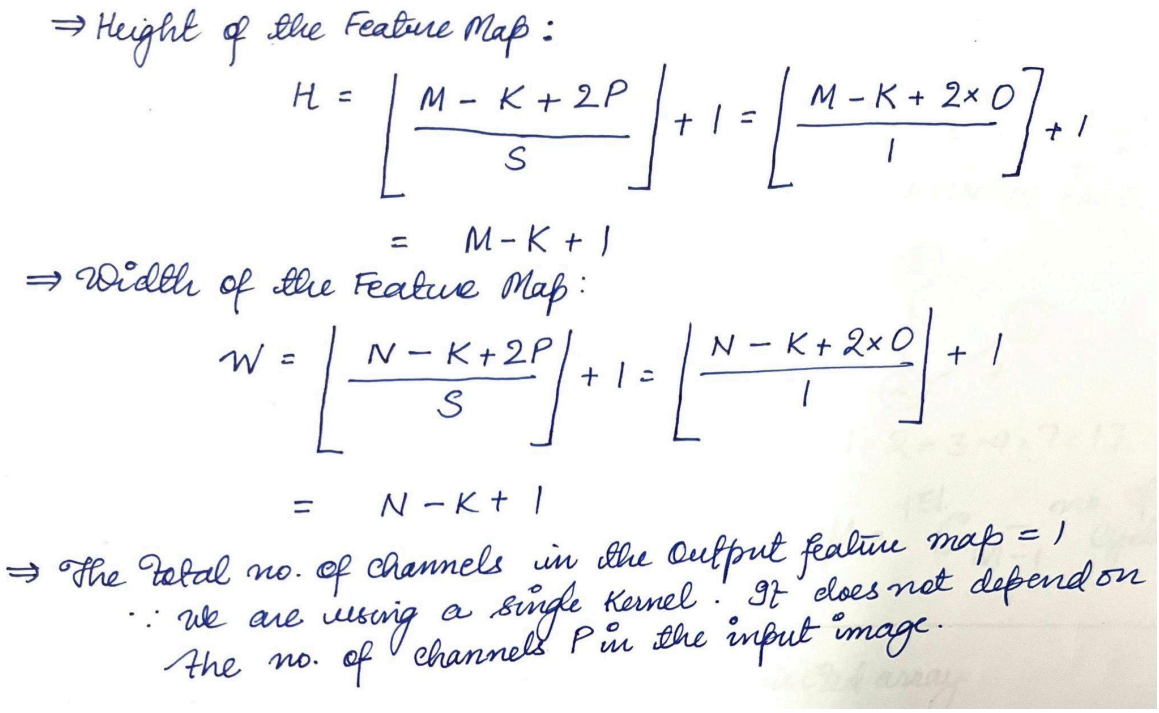
   - Dimension of the Input image: M x N where min(M, N) ≥ 1
   - Dimension of the filter: K x K where 1 ≤ K ≤ min(M, N)
   - Padding: 0
   - Stride: 1

   After performing a convolution operation on the input image using a kernel or filter, the dimension of the feature map (or the output) for each spatial dimension (height or width) is given by:

$$O = \left\lfloor \frac{(I - K + 2P)}{S} \right\rfloor + 1$$

   Where:
   - I : Dimension of the input image (height or width).
   - P: Amount of padding applied on all sides of the input.
   - S: Stride - Amount by which each filter will move across the image.
   - K: Size of the Kernel(Height or Width).

   The dimensions of the feature map are given below:



Thus, the dimensions of the Outer Feauter Map: H x W x 1= **(M - K + 1) x (N - K + 1) x 1**

b. **Compute the number of elementary operations (multiplications and additions) required to compute a single output pixel in the resulting feature map.**
Solution:

These are the operations involved in the convolution of the input image:

- **Multiplication**: Each element of the kernel is multiplied by the corresponding element in the selected window of the input image. This operation is performed element-wise.

Total Multiplications = (Kernel Dimensions) × (Total Sliding Windows) × (Number of Channels)

- **Addition**: The resulting element-wise products from the multiplication step are summed up to produce a single value, which forms one entry in the feature map.

Total Additions = Total Number of Multiplications - 1

It is because we start with the first multiplication result and add the rest.

Kernel Dimensions = **K x K**
Total Sliding Windows i.e. total number of times Kernel slide over the patches(window) of the input image = Total number of entries of the Feature Map
= (M - K + 1) x (N - K + 1) x 1
Total Number of channels in the input image = P

$\Rightarrow$ Total no. of Multiplications $= (K \times K \times P) \times (M-K+1) \times (N-K+1)$

$\Rightarrow$ Total no. of Additions = Total Multiplications $- 1$

$= (K \times K \times P) \times (M-K+1) \times (N-K+1) - 1$

$\Rightarrow$ Thus, the total no. of elementary operations is :—

$=$ Total no. of Multiplications + Total no. of Additions

$= (K \times K \times P) \times (M-K+1) \times (N-K+1) + (K \times K \times P) \times (M-K+1) \times (N-K+1) - 1$

$= (M-K+1) \times (N-K+1) \times [(K \times K \times P) + (K \times K \times P - 1)]$

$= (M-K+1) \times (N-K+1) \times [K^2P + K^2P - 1]$

$= (M-K+1) \times (N-K+1) \times (2K^2P - 1)$

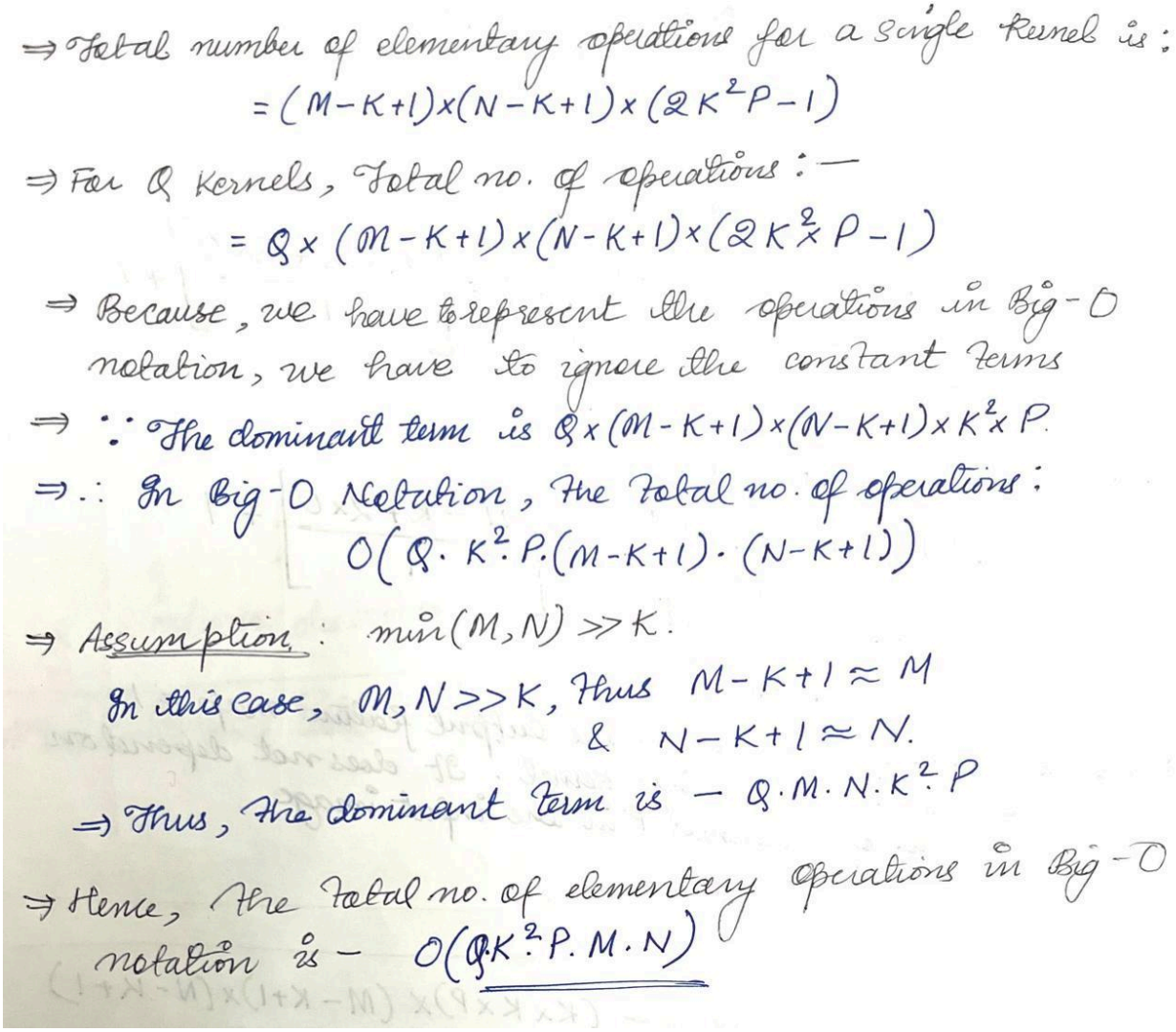
The above is the number of elementary operations for computing every entry in the Feature Map.

In order to compute the number of elementary operations (multiplications and additions) required to compute a single output pixel in the resulting feature map
= $2K^2.P - 1$

c. **Now, consider the scenario where there are Q kernels (Q ≥1) of size K ×K. Derive the computational time complexity of the forward pass for the entire image in Big-O notation as a function of the relevant dimensions. Additionally, provide another Big-O notation assuming min(M, N) ≫ K.**
Solution:

When we use “Q” Kernels each of dimension K x K, then each Kernel or Filter is going to generate a feature map whose dimension is given by:
= (M - K + 1) x (N - K + 1) x 1

Each Convolution operation with a Kernel involves the number of computations, computed in the previous step, which is going to be repeated for Q Kernels.



Because, min(M,N) >> K, we will have K^2 grows slower than $M \cdot N$, due to min(M,N) >> K. Hence, we consider $M \cdot N$ as the dominant product involving M and N.

Hence, the final Big-O notation for the total number of operations is: ***O(Q · P · M · N)***

Part (b): Explain the Assignment Step and Update Step in the K-Means algorithm. Discuss any one method that helps in determining the optimal number of clusters. Can we randomly assign cluster centroids and arrive at global minima?

Solution:

K-Means is a type of clustering algorithm that is used to partition the data in to K distinct clusters or groups.

K-Means is applied on unsupervised data (data with no labels) in which we group or cluster the data points such that:

- There is **high intra-cluster similarity** i.e. data points within the same cluster are more similar or related to each other.
- There is **low inter-cluster similarity** i.e. data points in different clusters should be more dissimilar to each other.

There are two major steps in K-Means algorithm:

- Assignment
- Update

Assignment Step:
This step includes deciding the class memberships of the given finite number of sample points by assigning them to cluster centroid based on the distance between point and the centroid.

⇒ For the Unsupervised dataset X containing the data points $x_1, x_2, x_3 \ldots, x_N$, where each $x_i \in \mathbb{R}^m$.

⇒ First phase of the Assignment Step is Calculating the Distance.

⇒ Distance Calculation:

→ For each data point x_i, we compute its euclidean distance to every cluster centroid C_K, where $C_K \in \mathbb{R}^m$ for $K = 1, 2, \ldots, K$.

⇒ The distance is calculated b/w point x_i & cluster centroid C_K is calculated as —

$$d(x_i, C_K) = \sqrt{\sum_{j=1}^{m} (x_{ij} - C_{Kj})^2}$$

→ Here, x_{ij} — j^{th}-feature of data point x_i.

C_{Kj} — j^{th} feature of the centroid C_K.

⇒ Second phase of Assignment Step is Assigning the cluster to a datapoint.

⇒ Cluster Assignment:

→ Now, we'll assign each data point x_i to the cluster K whose centroid is closest:

$$\text{Cluster}(x_i) = \arg\min_K d(x_i, C_K)$$

→ This results in K clusters, with each data point assigned to exactly one cluster.

Randomly assigning the cluster centroid does not guarantee that we will arrive at global minima.

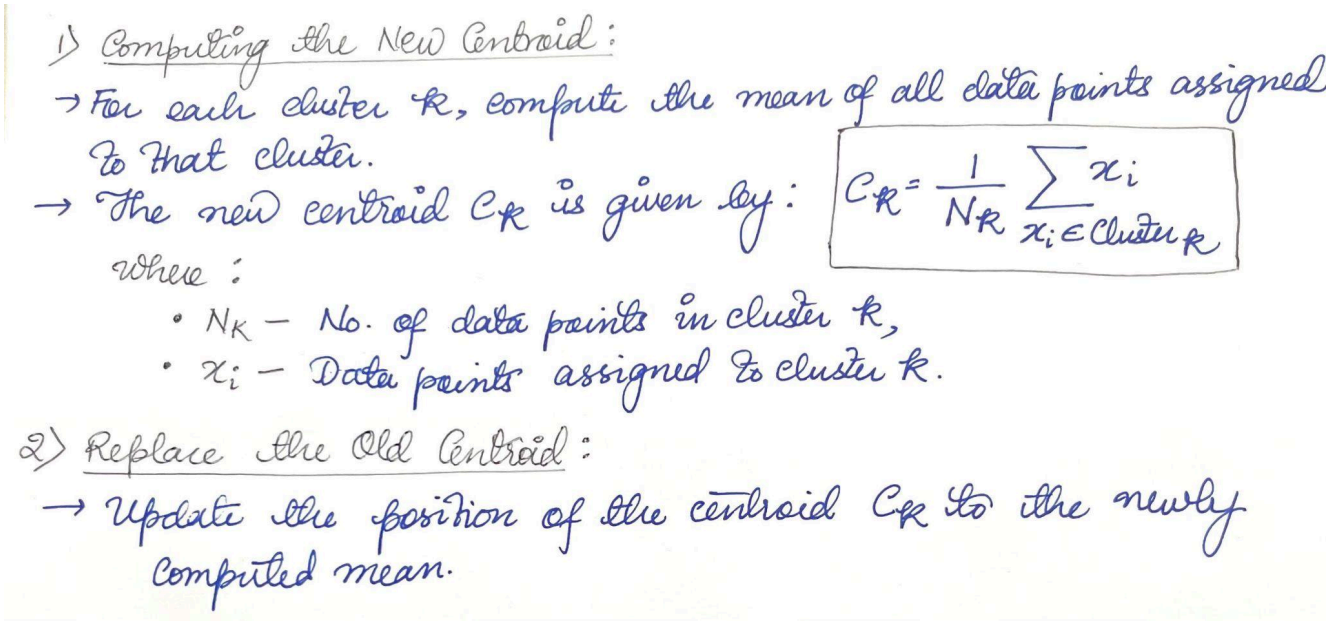
This is because the centroids can be selected in such a way that they are closer to each other.

So, the cluster formed with this type of randomly initialised centroids don't represent the intrinsic nature or properties of the data points, thus, it can lead to suboptimal clustering.

Update Step:

In this step we recalculate or readjust the centroid of each cluster by taking the mean of all the data points lying in that particular cluster.

There are some phases in the Update step:



One method that we use to find optimal number of cluster is Elbow Method. It identifies the optimal number of clusters by evaluating how Within-Cluster Sum of Squares(WCSS) changes against increasing Values of K - the number of clusters.

→ Steps in the Elbow Method :

1) We train and fit the K-Means algorithm to the dataset for values of K. Note, in this we have to define the maximum no. of cluster we need to take.
→ So, K can take values such as: 1, 2, 3, . . . , K.

2) Calculating WCSS :
→ For each value of K, we calculate the total within-cluster sum of squares (WCSS):

$$WCSS = \sum_{K=1}^{K} \sum_{x_i \in C_K} \| x_i - C_K \|^2$$

Where :
- x_i – Data point in cluster C_K,
- C_K – Centroid of cluster C_K,
- $\| x_i - C_K \|^2$ – Squared Euclidean distance between the point & its cluster centroid.

3) After computing WCSS for different values of K, we plot a graph of WCSS vs K, in which x-axis represents the no. of clusters K & the y-axis represent the corresponding WCSS.

4) Identifying the Elbow Point :
→ In the graph, we look for "elbow" point – the point where the rate of decrease in WCSS slows down significantly, indicating diminishing returns.
→ The optimal K is at this elbow point by increasing K beyond this does not significantly improve the clustering quality.

⇒ This Algorithm works because :
- For smaller values of K – WCSS ~~represent~~ decreases rapidly because the clusters become tighter as the no. of centroids increases.
- For points beyond a certain K – Adding more clusters results in minimal improvement, as each data point is already near a centroid.

Random assignment of the cluster centroids in the K-Means algorithm does not guarantee arriving at a global minimum, but it may lead to a local minimum.

→ The main aim of K-Means algorithm is to minimize the Within-cluster Sum of Squares (WCSS), which is the sum of squared distances between each data point & its assigned centroid:

$$WCSS = \sum_{i=1}^{K} \sum_{x_i \in C_K} \|x_i - C_K\|^2$$

→ This optimization problem is non-convex, that means there can be multiple local minima. As for convex optimization problem, there exists a global minima.

⇒ The outcome of the problem depends heavily on the initial placement of centroids.

⇒ Thus, poor initialization can lead to suboptimal cluster assignments.

⇒ When centroids are initialized randomly, they might:

- Be too close to each other, that may lead suboptimal clustering.
- Start in a region where there are few or no data points.
- Miss dense regions of data entirely.

⇒ This can cause the algorithm to converge to a local minimum of J, which is not necessarily the global presence.

Random Assignment can work if initalised cluster centroids happens to be close to the true position of the cluster centroids. However, this is highly unlikely, especially for large and complex datasets.

We can use the methods like K-Means++ to deal with the problem of Random Initialisation Trap, which ensures that centroids are initialised far apart, so that we have better chances of clustering as it reduces the chances of poor clustering.

Section B - Scratch Implementation

Given:

- Number of clusters(k) = 2.
- Initial Centroids: u_1 = (3.0, 3.0), u_2 = (2.0, 2.0).
- The matrix X consists of the following data points:

$$\mathbf{X} = \begin{pmatrix} 5.1 & 3.5 \\ 4.9 & 3.0 \\ 5.8 & 2.7 \\ 6.0 & 3.0 \\ 6.7 & 3.1 \\ 4.5 & 2.3 \\ 6.1 & 2.8 \\ 5.2 & 3.2 \\ 5.5 & 2.6 \\ 5.0 & 2.0 \\ 8.0 & 0.5 \\ 7.5 & 0.8 \\ 8.1 & -0.1 \\ 2.5 & 3.5 \\ 1.0 & 3.0 \\ 4.5 & -1.0 \\ 3.0 & -0.5 \\ 5.1 & -0.2 \\ 6.0 & -1.5 \\ 3.5 & -0.1 \\ 4.0 & 0.0 \\ 6.1 & 0.5 \\ 5.4 & -0.5 \\ 5.3 & 0.3 \\ 5.8 & 0.6 \end{pmatrix}$$

Solution (a):

In this part, I have implemented a class named **KMeans** which has the following attributes:

- **centroid_init**: It is the variable that store the initial centroids(if available).
- **k**: It represents the number of clusters.
- **max_iters**: It represents the maximum number of iterations, which is set to 100.
- **convergence_threshold**: It represents the threshold below which convergence will be achieved.
- **centroids**: It stores centroids in the K-Means clustering model.
- **clusters**: It stores the clusters present in the Model.

Class **KMeans** performs the following functions:

- **Initialisation**: It initalises the K-Means clustering model with the given centroids if specified.
- **Assignment**: It will assign cluster to each data point.
- **Update**: It update the centroid based on the assignment of the data points.
- **Check Convergence**: It checks whether convergence is achieved in the model or not. It will going to terminate the algorithm if converence is achieved before maximum number of iterations.

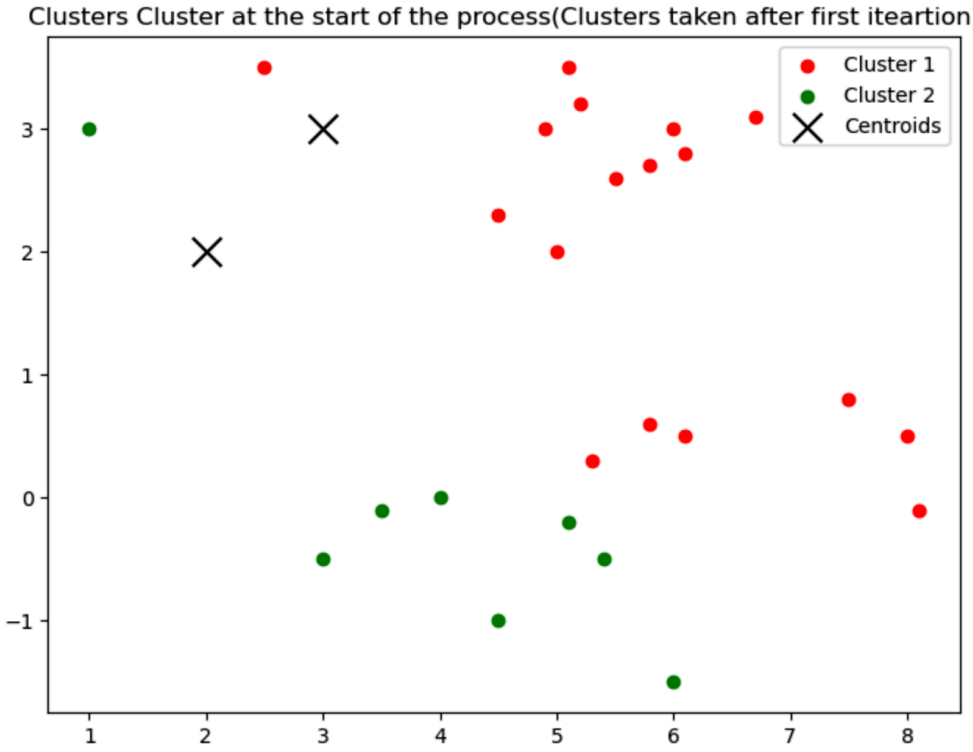
(b) Find the values of final centroids after the algorithm converges. Plot the two clusters at the start of the process and at the end.

Solution (b):

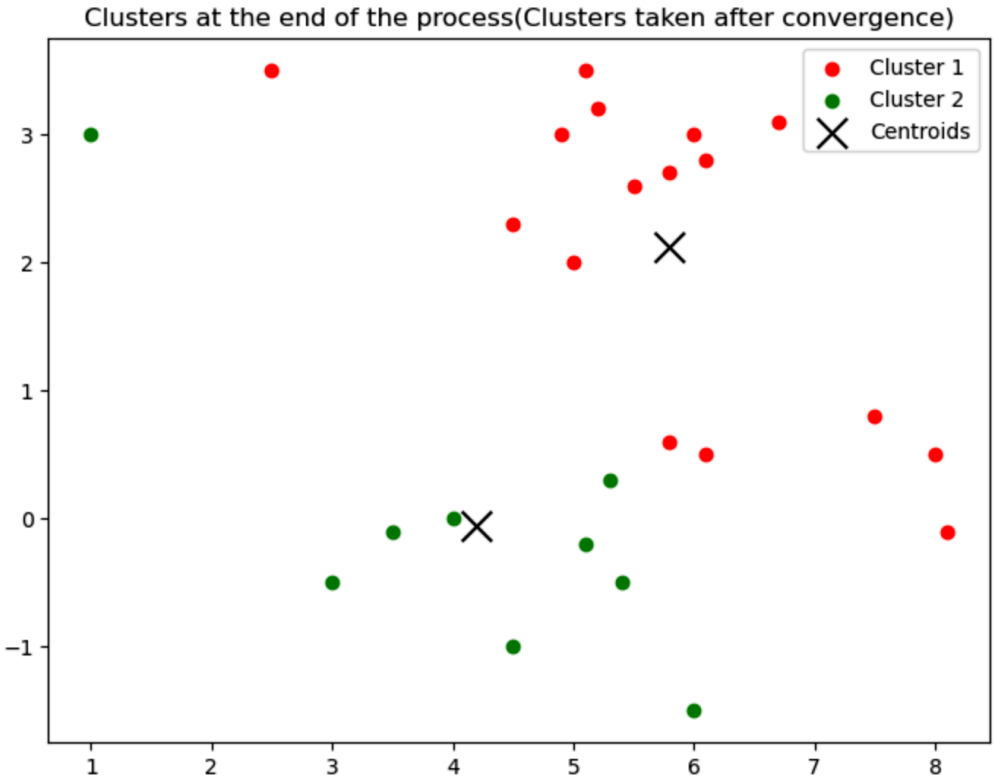
After the convergence of the K-means model, these are the values of the final centroids:

```
Final Centroids :
[[ 5.8          2.125     ]
 [ 4.2         -0.05555556]]
```

The following are the graphs of the k-clusters where k = 2 at the starting and the ending of the process:



Clusters at the starting of the process with centroids at (2.0, 2.0) and (3.0, 3.0)



Clusters at the starting of the process with final centroids placed at (5.8, 2.125) and (4.2, -0.056)

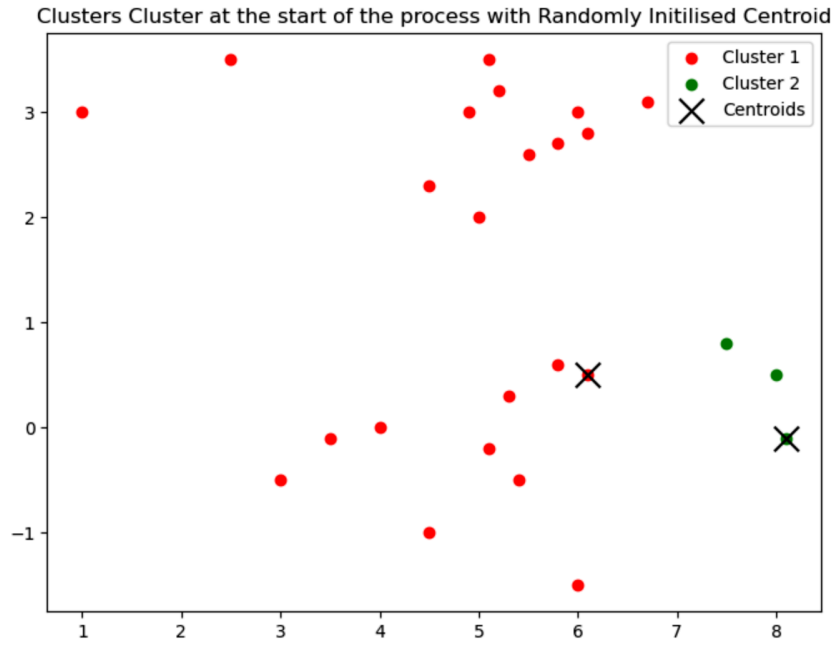
(c) . Compare the results using the provided initial centroids versus using random initialization of centroids.

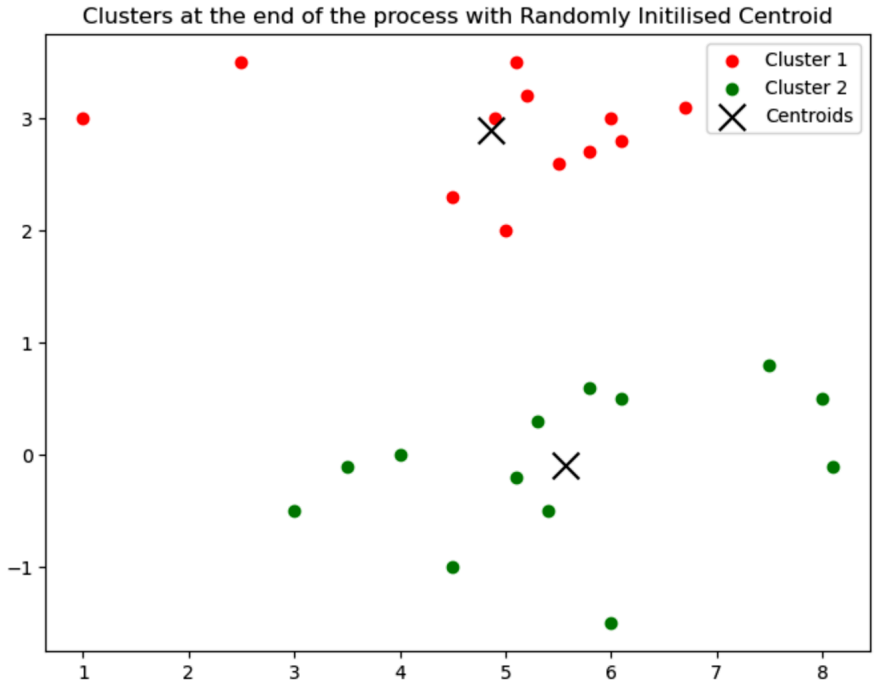
Solution:

These are the final value of centroids observed after training the model along with the initially randomly initialised centroids:

```
Initialised Centroid:
[[ 6.1  0.5]
 [ 8.1 -0.1]]
Final Centroids:
[[ 4.85833333  2.89166667]
 [ 5.56153846 -0.09230769]]
```

The following curves represents the clusters at the start and end of the process when centroids are intialised randomly:





Comparison:

- **Initial Centroids:**
  The centroids : u1 = (2.0,2.0), u2 = (3.0,3.0) are more closer to the final value of centroids as compared when centroids are randomly initalised:

u1 = (6.1,0.5) , u2=(8.1,−0.1). These centroids are further spread out which emphasizes different regions of the data at the start.

- **Final Centroids:**
  - **For the case of Provided initial centroids:**
    The final centroids are much shifted from the initial points, which suggests that the model has to adjust in an extensive manner in order to converge. This also tells us that the provided centroids are suboptimal for capturing the natural clusters in the data.

  - **For the Randomly Initialised Case:**
    The final centroids are more closer to the initial values defined, which suggests that random initialization was closer to the true cluster centers.

- **Deviation from Initial Centroids:**
  - **For the case of Provided initial centroids:**
    There is high deviation from the initial centroid to the final centroid, which suggests that the initial centroid points provided are less sufficient and effective for the Clustering. This could have lead to slower convergence or suboptimal clustering.

  - **For the Randomly Initialised Case:**
    Smaller deviations were observed, indicating better initial alignment with the data. This suggests faster convergence for the clustering process.

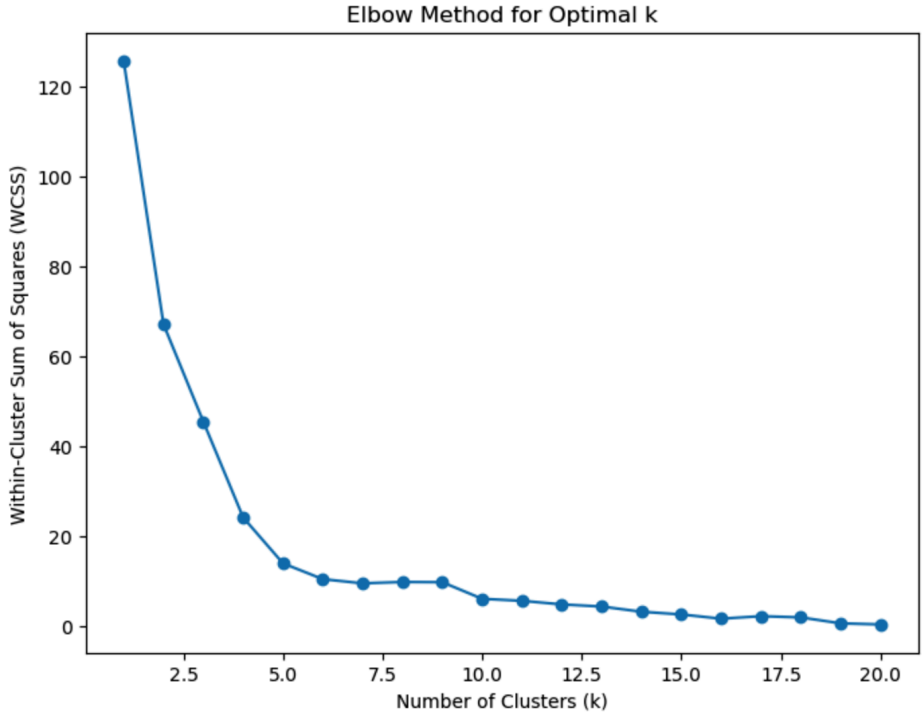
- **Cluster Characteristics:**
  - **For the case of Provided initial centroids:**
    The final cluster centers are not present at the center region of the respective clusters. Also, the cluster is more compact and it might miss the data in the peripheral regions.

  - **For the Randomly Initialised Case:**
    The final centroids appear more spread out, capturing clusters across a broader range of the feature space.

(d). Determine the optimal number of clusters, M, using the Elbow method. Plot the Within-Cluster Sum of Squares (WCSS) against different values of k to find the elbow point. Randomly initialize M centroids, perform clustering and plot the resulting clusters
Solution:

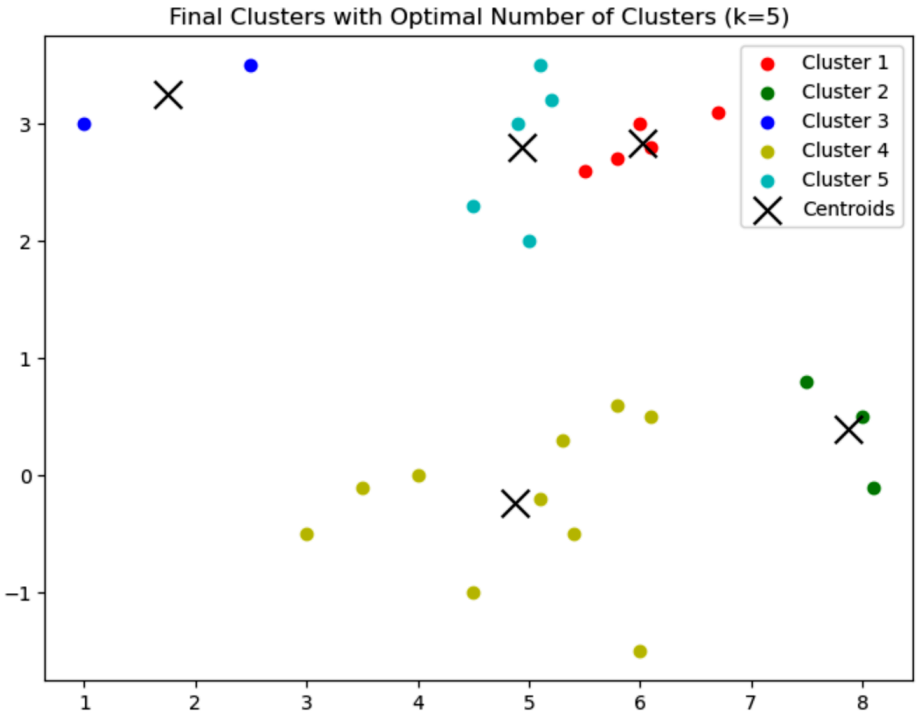
Using the Elbow method, the optimal number of clusters that we observed is **5**. After K = 5, the decrease in the Within-Cluster Sum of Squares(WCSS) is slower and it shows less variation.

Plot of WCSS vs K, where maximum K value can be 20:




```
Optimal number of clusters (Elbow Method): 5
```

We take K = 5 and performing clustering on the given dataset by initialising the centroids randomly.
The plot for clusters with K = 5 is given below:



Section C - Algorithm implementation using packages

For this question, you are expected to work with the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60,000 32x32 RGB images of 10 classes, with 6,000 images per class. You are expected to work with 3 classes from the available classes as per your choice. Hence, you should have roughly 18,000 images in your training curated dataset, with 15,000 images in the train and 3,000 in the test dataset, respectively.

Solution - 1:

This part includes data preparation in which:

- I first loaded the CIFAR-10 dataset using PyTorch.
- After that I split the dataset into training and testing sets.
  The size of these datasets are provided below:

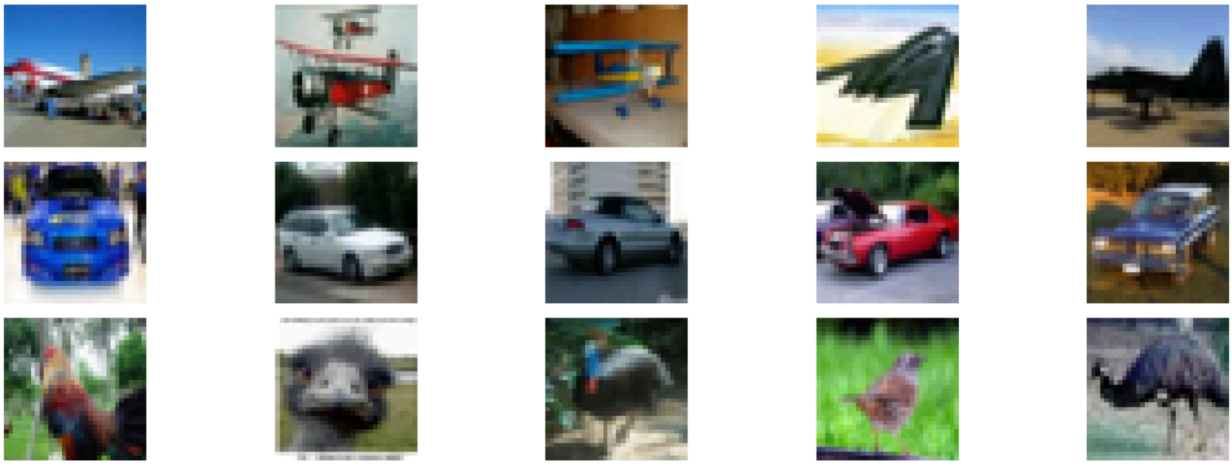
```
Training Dataset size:  15000
Training Dataset size:  3000
```

- After that I selected 3 classes with labels 0, 1 and 2 from the total of 10 classes. These classes consist of images of Planes, Birds and Cars.

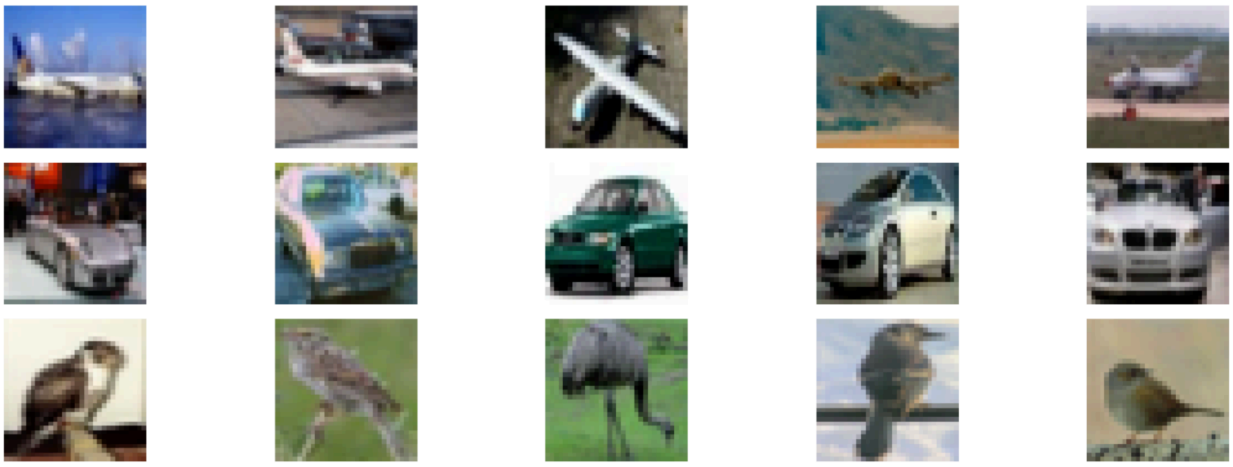
- Then, I performed stratified split of the training dataset in the ratio of 0.8:0.2 to create new training dataset and validation dataset.
- After that I created custom class known as “Dataset” which performs three functions:
    - Initialise the dataset.
    - Get Items (Image, label) from the dataset based on the index passed as an argument.
    - Provide length of the dataset.
- Then, I created separate loaders for each training, validation and testing dataset.

Solution - 2:

The following are the 5-images vizualised from each class from the training dataset:



The following are the 5-images vizualised from each class from the Validation dataset:



Solution - 3:

For this part, I created a class named CNN which consists of the following functions:

- **Initialise**: This initialises the CNN object with the use of constructor, which includes:
    - Two convolution layers.
    - Two max-pooling layers.
    - MLP(Multi-Layer Perceptron), that will be used for Classification.
    - ReLU activation after each layer except the classification head.
- **Get Feature Size**: It basically returns the total number of elements in a particular feature map.
- **Forward**: It performs the forward pass for an image, which involves passing the image through convolution layers and then pooling layers.

There are two convolutional layers present in the model, with:

- **First Convolution Layer**: This layer performs the convolution operation on the input image with a Kernel or Filter which has the following configurations:
    - Dimension of Kernel: 5 x 5
    - Channels in Kernel: 16
    - Padding: 1
    - Stride: 1
- **Second Convolution Layer**: This layer performs the convolution operation on the feature map after first convolution and pooling operation with a Kernel or Filter which has the following configurations:
    - Dimension of Kernel: 3 x 3
    - Channels in Kernel: 32
    - Padding: 0
    - Stride: 2

There are two max pooling layers followed by each convolution layer:

- **First Pooling Layer**: This max poolng layer performs pooling operation on the feature map produced after first convolution operation on the input image with a Kernel which has the following configurations:
    - Dimension of Kernel: 3 x 3
    - Stride: 2
- **Second Pooling Layer**: This max pooling layer performs pooling operation on the feature map produced after second convolution operation on the feature with a Kernel which has the following configurations:
    - Dimension of Kernel: 3 x 3
    - Stride: 3

After getting the final feature map before the fully connected layer, this output feature map is flattened to form a 1D-vector.

Then, in the Fully connected layer a MLP is implemented which has two fully connected layers in which:

- **First Fully Connected Layer**: First Layer of the MLP consist of 16 neurons, the output of this neuron is passed through the ReLU activation function.
- **Second Fully Connected Layer**: the second layer of the MLP acts the classification head, which is used to finally classify the image based on the three categories of classes chosen earlier.

Solution 4:

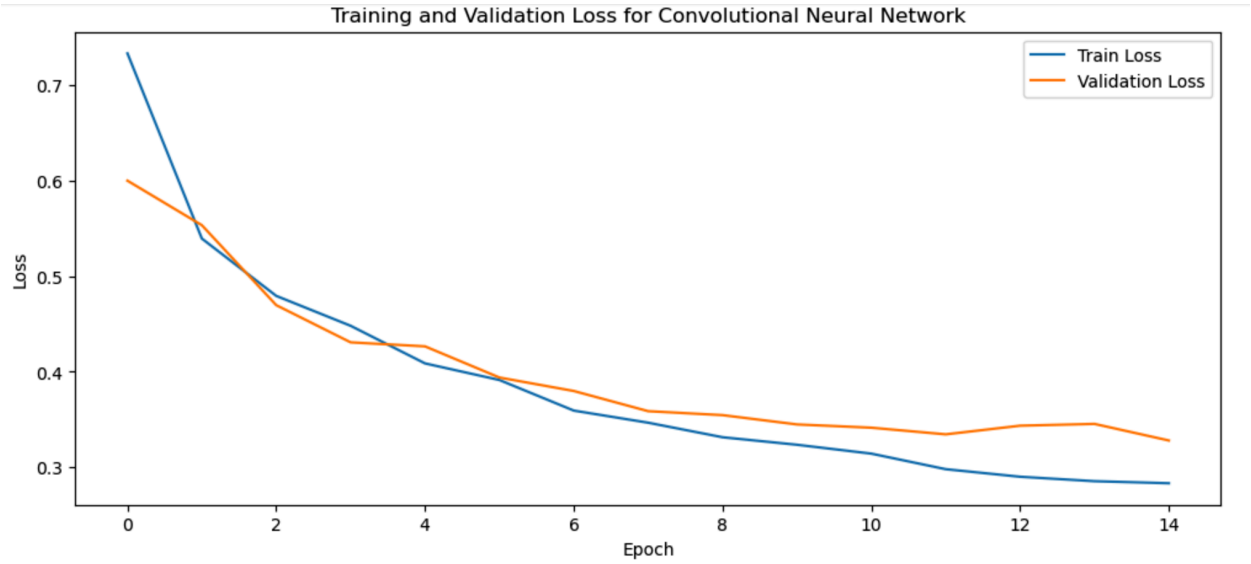
We train the previously created Convolution Neural Network Model(CNN) using the cross-entropy as the loss function and **Adam** as the optimizer.
This CNN model is trained for 15 epochs.
Following are the result that we obtained after training our CNN model on each epoch. The following results contain Training Loss, Validation Loss, Training Accuracy and Validation Accuracy:

```
Epoch Number [1/15]: Train Loss: 0.7330, Train Accuracy: 0.6846, Validation Loss: 0.5999, Validation Accuracy: 0.7530
Epoch Number [2/15]: Train Loss: 0.5394, Train Accuracy: 0.7833, Validation Loss: 0.5533, Validation Accuracy: 0.7820
Epoch Number [3/15]: Train Loss: 0.4794, Train Accuracy: 0.8122, Validation Loss: 0.4697, Validation Accuracy: 0.8147
Epoch Number [4/15]: Train Loss: 0.4481, Train Accuracy: 0.8238, Validation Loss: 0.4308, Validation Accuracy: 0.8360
Epoch Number [5/15]: Train Loss: 0.4088, Train Accuracy: 0.8393, Validation Loss: 0.4266, Validation Accuracy: 0.8280
Epoch Number [6/15]: Train Loss: 0.3914, Train Accuracy: 0.8483, Validation Loss: 0.3940, Validation Accuracy: 0.8470
Epoch Number [7/15]: Train Loss: 0.3594, Train Accuracy: 0.8608, Validation Loss: 0.3799, Validation Accuracy: 0.8487
Epoch Number [8/15]: Train Loss: 0.3467, Train Accuracy: 0.8651, Validation Loss: 0.3588, Validation Accuracy: 0.8590
Epoch Number [9/15]: Train Loss: 0.3315, Train Accuracy: 0.8728, Validation Loss: 0.3546, Validation Accuracy: 0.8620
Epoch Number [10/15]: Train Loss: 0.3237, Train Accuracy: 0.8766, Validation Loss: 0.3449, Validation Accuracy: 0.8657
Epoch Number [11/15]: Train Loss: 0.3144, Train Accuracy: 0.8771, Validation Loss: 0.3415, Validation Accuracy: 0.8680
Epoch Number [12/15]: Train Loss: 0.2981, Train Accuracy: 0.8828, Validation Loss: 0.3345, Validation Accuracy: 0.8663
Epoch Number [13/15]: Train Loss: 0.2902, Train Accuracy: 0.8875, Validation Loss: 0.3436, Validation Accuracy: 0.8683
Epoch Number [14/15]: Train Loss: 0.2856, Train Accuracy: 0.8902, Validation Loss: 0.3455, Validation Accuracy: 0.8650
Epoch Number [15/15]: Train Loss: 0.2834, Train Accuracy: 0.8904, Validation Loss: 0.3281, Validation Accuracy: 0.8697
```

The model is saved in the .pth file named as cnn_model.pth.

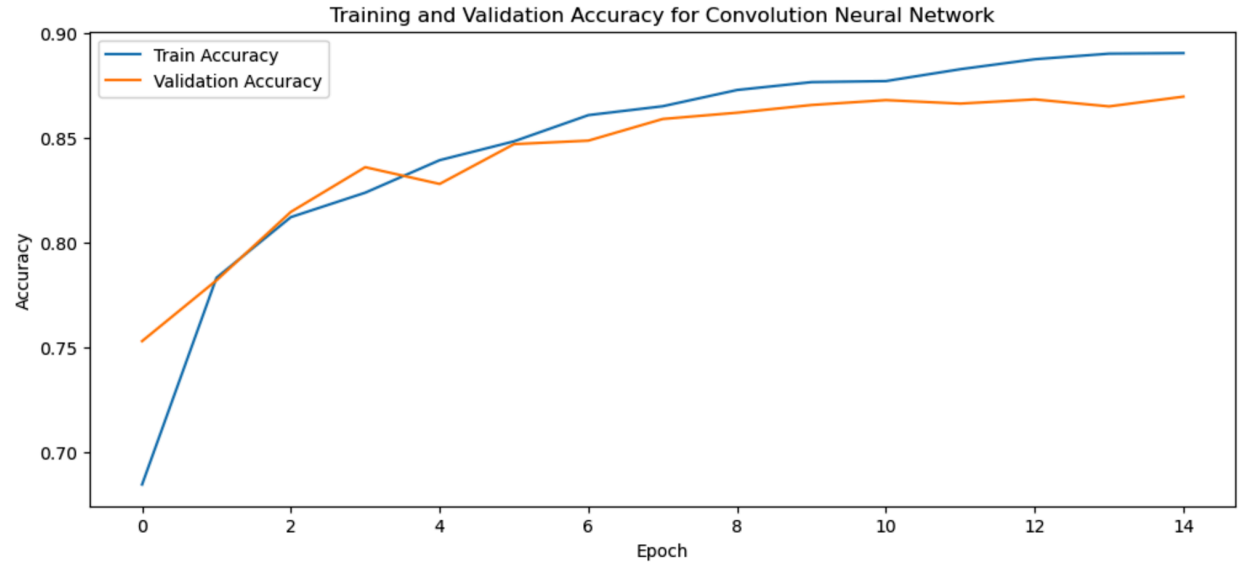
Solution 5:

The Plot for Training and Validation Loss vs Epochs for the CNN model is provided below:



Loss vs Epochs for Training and Validation Dataset

The Plot for Training and Validation Accuracy vs Epochs for the CNN model is provided below:



Accuracy vs Epochs for Training and Validation Dataset

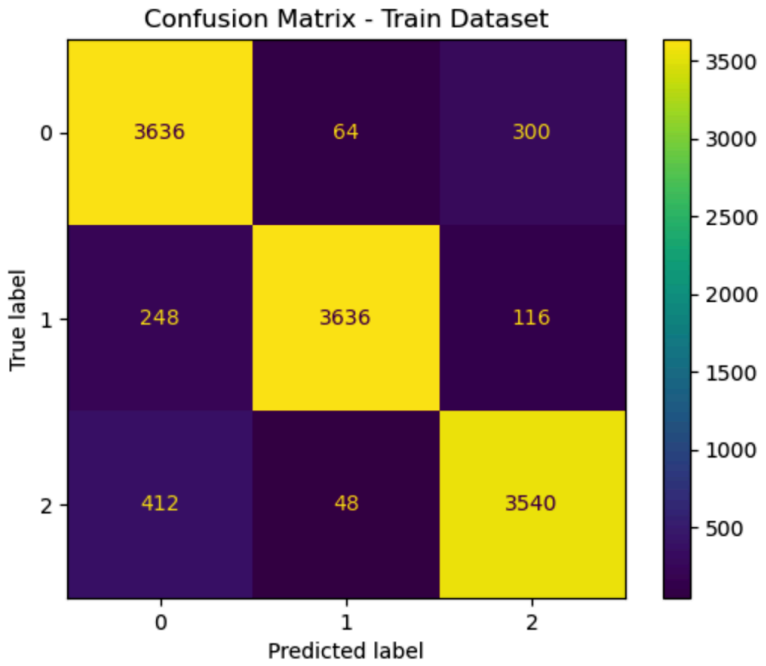
Plot Analysis:

- In CNN model, the loss decreases in a steady manner as the number of epochs increases. By the final epoch, it achieves a **training loss of 0.2834,** and the loss appears to be constant after that.
- The Accuracy also show improvement as it is gradually increasing with the number of epochs. By the final epoch, it achieves a **training accuracy of 89.04%**, indicating convergence and stable learning.

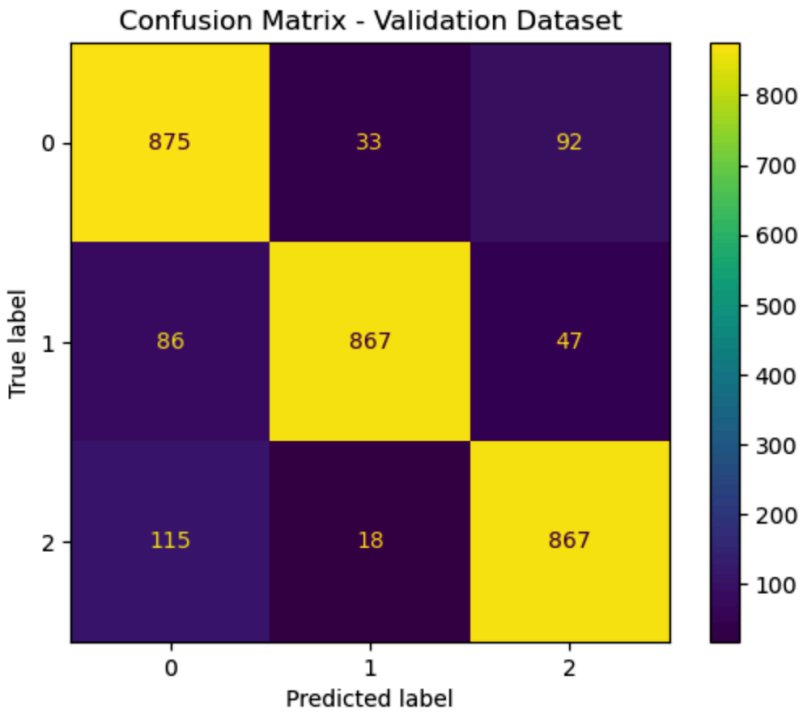
The Accuracy and F1-score for the Test Dataset by CNN model is provided below:

```
Test Accuracy: 0.8767
Test F1-Score: 0.8775
```

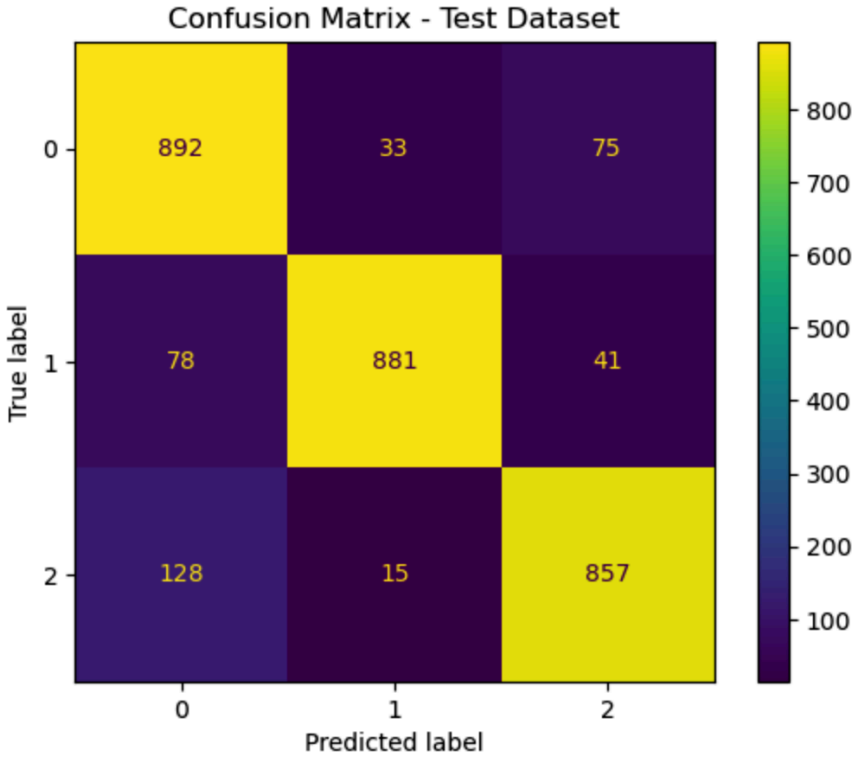
The Confusion Matrix for the Training Dataset is:



The Confusion Matrix for the Validation Dataset is:



The Confusion Matrix for the Test Dataset is:



Solution 6:

Initial Steps:

- First, I created a class named MLP which is used to create objects which represents an MLP model.
- In this class, we intialise each MLP model with two layers:
    - First Layer: This Layer of MLP consists of 64 neurons and the output of this neuron is then passed through the activatioin function known as ReLU(Rectified Linear Unit).
    - Second Layer: This layer of the MLP model acts as a classification head and it classifies each image based and its label.

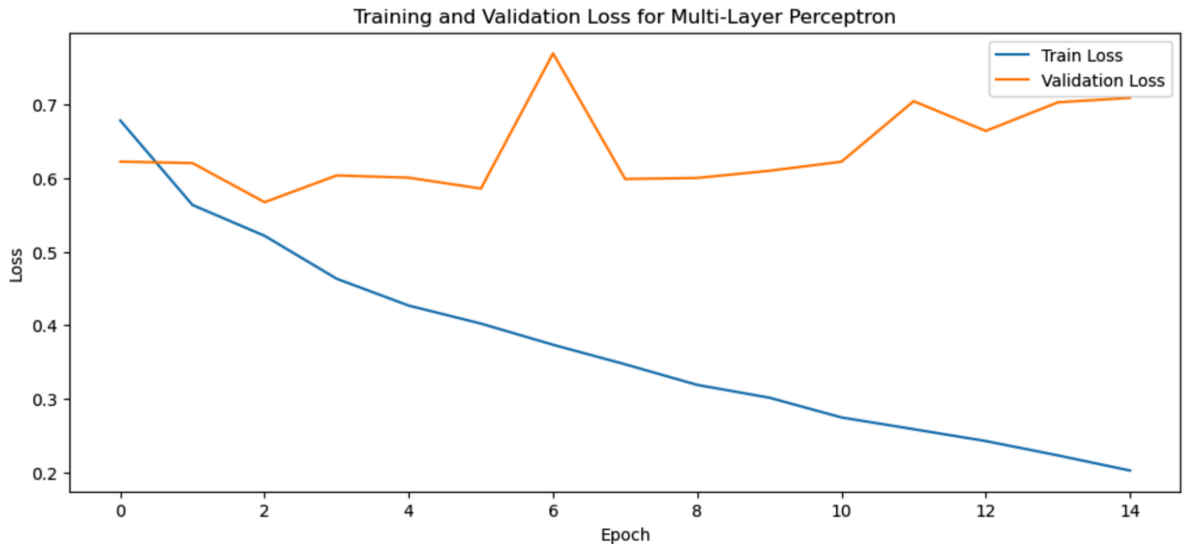
We train the previously created Multi-Layer Perceptron Model(MLP) using the cross-entropy as the loss function and **Adam** as the optimizer.
This CNN model is trained for 15 epochs.
Following are the result that we obtained after training our MLP model on each epoch. The following results contain Training Loss, Validation Loss, Training Accuracy and Validation Accuracy:

```
Epoch Number: [1/15]: Training Loss: 0.6779, Train Accuracy: 0.7266, Validation Loss: 0.6221, Validation Accuracy: 0.7577
Epoch Number: [2/15]: Training Loss: 0.5633, Train Accuracy: 0.7784, Validation Loss: 0.6201, Validation Accuracy: 0.7507
Epoch Number: [3/15]: Training Loss: 0.5215, Train Accuracy: 0.7971, Validation Loss: 0.5671, Validation Accuracy: 0.7800
Epoch Number: [4/15]: Training Loss: 0.4632, Train Accuracy: 0.8227, Validation Loss: 0.6033, Validation Accuracy: 0.7727
Epoch Number: [5/15]: Training Loss: 0.4267, Train Accuracy: 0.8330, Validation Loss: 0.6003, Validation Accuracy: 0.7707
Epoch Number: [6/15]: Training Loss: 0.4023, Train Accuracy: 0.8468, Validation Loss: 0.5856, Validation Accuracy: 0.7757
Epoch Number: [7/15]: Training Loss: 0.3736, Train Accuracy: 0.8602, Validation Loss: 0.7689, Validation Accuracy: 0.7210
Epoch Number: [8/15]: Training Loss: 0.3471, Train Accuracy: 0.8732, Validation Loss: 0.5985, Validation Accuracy: 0.7757
Epoch Number: [9/15]: Training Loss: 0.3191, Train Accuracy: 0.8824, Validation Loss: 0.6000, Validation Accuracy: 0.7780
Epoch Number: [10/15]: Training Loss: 0.3018, Train Accuracy: 0.8887, Validation Loss: 0.6097, Validation Accuracy: 0.7827
Epoch Number: [11/15]: Training Loss: 0.2750, Train Accuracy: 0.9003, Validation Loss: 0.6220, Validation Accuracy: 0.7880
Epoch Number: [12/15]: Training Loss: 0.2590, Train Accuracy: 0.9049, Validation Loss: 0.7041, Validation Accuracy: 0.7770
Epoch Number: [13/15]: Training Loss: 0.2430, Train Accuracy: 0.9109, Validation Loss: 0.6638, Validation Accuracy: 0.7823
Epoch Number: [14/15]: Training Loss: 0.2235, Train Accuracy: 0.9214, Validation Loss: 0.7025, Validation Accuracy: 0.7810
Epoch Number: [15/15]: Training Loss: 0.2029, Train Accuracy: 0.9307, Validation Loss: 0.7087, Validation Accuracy: 0.7783
```

The model is saved in the .pth file named as mlp_model.pth.

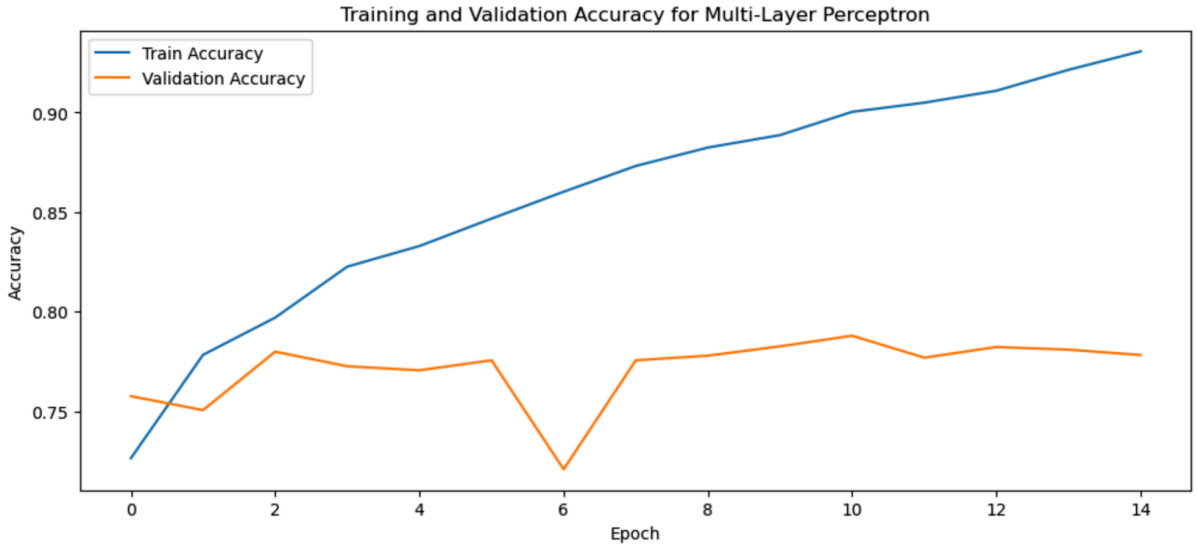
Solution 7:

The Plot for Training and Validation Loss vs Epochs for the MLP model is provided below:



Loss vs Epochs for Training and Validation Dataset

The Plot for Training and Validation Loss vs Epochs for the MLP model is provided below:

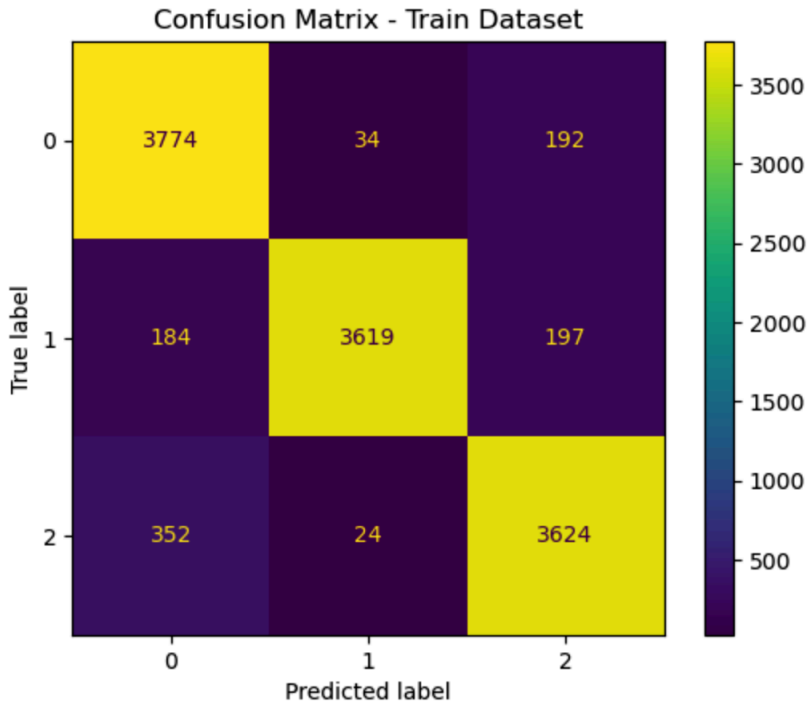


Accuracy vs Epochs for Training and Validation Dataset

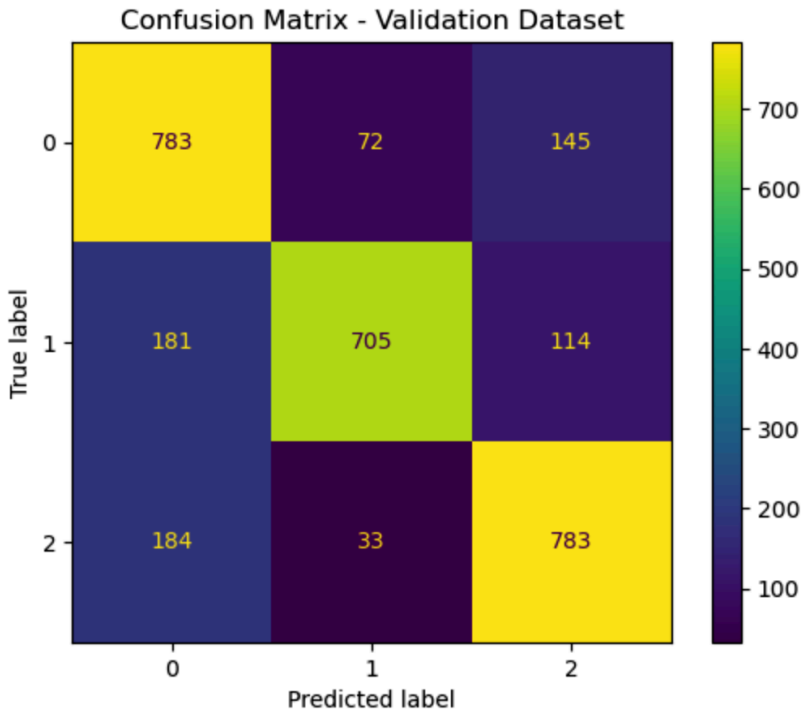
The Accuracy and F1-score for the Test Dataset by MLP model is provided below:

```
Test Accuracy: 0.8147
Test F1-Score: 0.8148
```

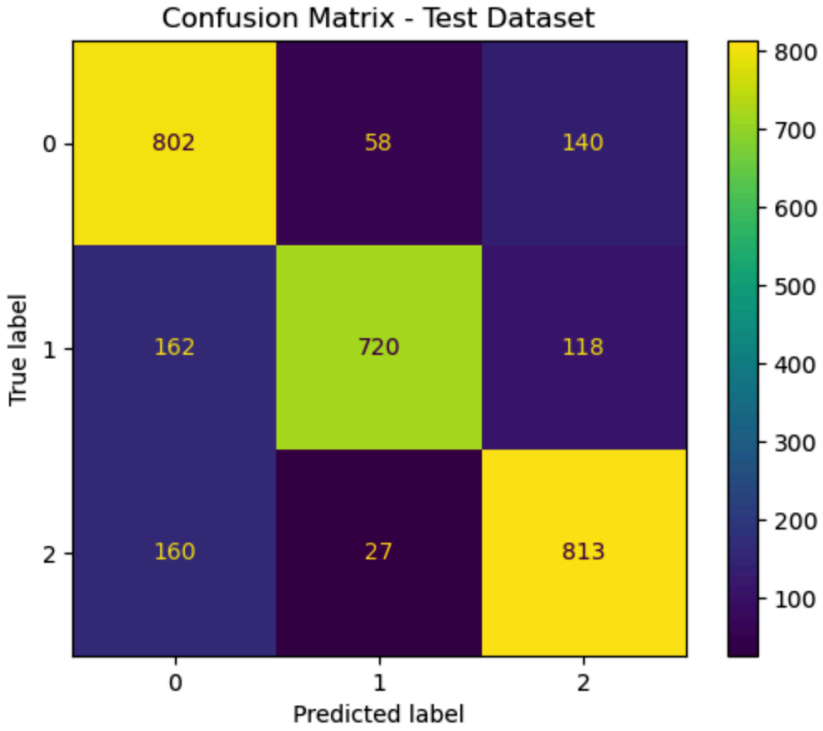
The Confusion Matrix for the Training Dataset is:



The Confusion Matrix for the Validation Dataset is:



The Confusion Matrix for the Test Dataset is:



Training Loss and Accuracy Progression

- **CNN model:**
  - In CNN model, the loss decreases in a steady manner as the number of epochs increases. By the final epoch, it achieves a **training loss of 0.2834,** and the loss appears to be constant after that.
  - The Accuracy also show improvement as it is gradually increasing with the number of epochs. By the final epoch, it achieves a **training accuracy of 89.04%**, indicating convergence and stable learning.
- **MLP model:**
  - In MLP model, the loss function decreases gradually with the rise in the number of epochs, reaching the lowest training loss of about **0.2029.**
  - Similarly, accuracy also increase gradually with the increase in the number of epochs, reaching the highest accuracy of **93.07%** at the end of the epochs i.e. at 15th epoch.

- **Analysis and Comparison:**
  - Both CNN and MLP model show improvement as number of epochs increases, however, the MLP model shows faster initial gains reaching higher training earlier in epochs than the CNN model.
  - From the observed plots of Training Loss and Accuracy vs Epochs for CNN and MLP model, it can be said that MLP seems to fit the training

data more strongly, possibly indicating a higher capacity for overfitting compared to the CNN.

Validation Loss and Accuracy Progression

- **CNN Model:**
  - Validation Loss decreases while Validation Accuracy steadily show improvements over epochs. The lowest Valdiation Accuracy at the 15th epochs is **86.97%**.
  - There is a consistency between training and validation metrics suggests that the CNN is generalizing well.
- **MLP Model:**
  - Validation loss is higher and fluctuates more compared to the CNN, even increasing in later epochs.
  - Validation accuracy plateaus around **77.83%** by the final epoch.
  - Also, there is a huge difference between the Training and Validation Loss and Accuracy, which increases with the increase in epochs.
- **Analysis and Comparison:**
  - From the data, it is clear that CNN generalizes better to unseen validation data as compared to MLP which struggles with overfitting.

Test Set Performance

- **CNN Model:**
  - **Test Accuracy:** 87.67%
  - **Test F1-Score:** 0.8775
- **MLP Model:**
  - **Test Accuracy:** 81.47%
  - **Test F1-Score:** 0.8148
- After evaluating both the Models on the Test dataset, it can be said that CNN model outperforms the MLP mode in terms of its performance as it has higher accuracy and F1-Score on the test set. It suggests that CNN is more capable in genearlizing well on the unseen data and better at handling misclassification cases.

Confusion Matrix:

- After analysing the confusion matrix for the CNN and MLP models on the training, testing and validation dataset, we observed that CNN model performs better across all classes, by generating less misclassifications as compared to the MLP model.
- In CNN model, misclassifications are more focused, with most errors occurring in adjacent classes.

- While in MLP model, Misclassifications are broader, especially in class 1 and class 0, indicating poorer generalization for these classes.