ML Assignment-4

Shobhit Raj (2022482)

1 Section A

1.1 Q1. a) a)

Given an input image with dimensions $M \times N$, P channels, and a kernel size $K \times K$, the feature map dimensions can be calculated using the following formula taught in class:

- **Width** of the output feature map:

$$\text{Width} = \left\lfloor \frac{M - K + 2Padding}{\text{stride}} + 1 \right\rfloor = M - K + 1$$

- **Height** of the output feature map:

$$\text{Height} = \left\lfloor \frac{N - K + 2Padding}{\text{stride}} + 1 \right\rfloor = N - K + 1$$

Since the stride is 1 and there's no padding, using 2D convulation, the output feature map will have dimensions :

$$\text{Output size} = (M - K + 1) \times (N - K + 1)$$

1.2 Q1. a) b)

To compute a single output pixel in the resulting feature map, we perform the following operations:

1. **Multiplications**: For each output pixel, the kernel performs element-wise multiplication with the corresponding patch of the input image. The kernel size is $K \times K$ and there are P channels in the input image. Therefore, the number of multiplications required for each output pixel is:

$$\text{Multiplications per output pixel} = K \times K \times P$$

2. **Additions**: After multiplying, the results are summed together to produce the output. For each output pixel, there are $K \times K \times P - 1$ additions (since one multiplication result is directly assigned to the output pixel and the rest are summed i.e. K^2P terms of multiplication are just added to each other).

$$\text{Additions per output pixel} = K \times K \times P - 1$$

Therefore, the total number of elementary operations (multiplications and additions) for each output pixel is:

$$\text{Total operations per output pixel} = \text{Multiplications} + \text{Additions} = (K \times K \times P) + (K \times K \times P - 1)$$

Simplifying, the total number of operations for a single output pixel is $= 2K^2P - 1$

1.3 Q1. a) c)

Now, consider the scenario where there are Q kernels (with $Q \geq 1$) of size $K \times K$, applied to an image of size $M \times N$ with P channels.

1. **Operations per output pixel**: For each output pixel, we have $K \times K \times P$ multiplications. After multiplying, the results are summed together to produce the output. For each output pixel, there are $K \times K \times P - 1$ additions (since one multiplication result is directly assigned to the output pixel and the rest are summed). Therefore, the total number of operations per output pixel is:

$$2K^2P - 1$$

When we take the Big-O notation, we remove constants, and it simplifies to:

$$\mathcal{O}(K^2P)$$

2. **Operations for the entire image**: The output feature map size is determined by the image size $M \times N$, the kernel size K, and the stride of 1 with no padding. The number of output pixels will be:

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

Hence, the total number of operations for one kernel is:

$$\mathcal{O}((M - K + 1)(N - K + 1) \times K^2P)$$

Since we have Q kernels, the total number of operations for all Q kernels is (time complexity of the forward pass) :

$$\mathcal{O}(Q \times (M - K + 1)(N - K + 1) \times K^2P)$$

3. When $\min(M, N) \gg K$: If $\min(M, N) \gg K$, the kernel size K becomes negligible compared to the image dimensions. In this case, the computational complexity simplifies to:

$$\mathcal{O}((M - K + 1)(N - K + 1) \times Q \times K^2P) \approx \mathcal{O}(M \times N \times Q \times K^2P)$$

This shows that when the kernel size is much smaller than the image dimensions, the complexity is dominated by the image size $M \times N$ and the number of kernels Q, with K and P having a reduced effect.

1.4 Q1. b)

Assignment Step: In this step, each data point x_i is assigned to the nearest centroid C_k based on the Euclidean distance:

$$\text{Cluster}(x_i) = \arg\min_k \|x_i - C_k\|^2$$

where x_i is the data point, C_k is the centroid of cluster k, and $\|x_i - C_k\|^2$ represents the squared Euclidean distance between the point and the centroid.

Update Step: After assigning the data points to clusters, the centroids are updated by calculating the mean of the points in each cluster:

$$C_k = \frac{1}{N_k} \sum_{x_i \in \text{Cluster}(C_k)} x_i$$

where N_k is the number of data points in cluster k, and C_k is the new centroid of the cluster.

The **Elbow Method** helps determine the optimal number of clusters by plotting the **within-cluster sum of squares (WCSS)** against the number of clusters. WCSS is calculated as:

$$\text{WCSS}(k) = \sum_{k=1}^{K} \sum_{x_i \in \text{Cluster}(C_k)} \|x_i - C_k\|^2$$

The "elbow" of the plot, where the rate of decrease slows down, indicates the optimal number of clusters. This is the value of k where adding more clusters does not significantly improve the WCSS.

Randomly assigning centroids can sometimes lead to a global minimum, but not always. K-Means is sensitive to initial centroid placement, and as it will always lead to local minimum, it may result in suboptimal clustering, rather than global minimum.

2 Section B

We are required to perform KMeans clustering with k=2, meaning the dataset will be divided into two clusters. The initial centroids are provided as starting points for the clustering process :-

- Centroid 1: $u_1 = (3.0, 3.0)$
- Centroid 2: $u_2 = (2.0, 2.0)$

The dataset to be clustered consists of 25 data points in a 2-dimensional space.

The goal of this implementation is to group these points into two clusters based on the provided centroids and iteratively refine the centroids until convergence, using the Euclidean distance as the metric for determining cluster membership.

2.1 Q2. a) KMeans Clustering Algorithm Implementation

The algorithm followed four main steps :-

- **Initialization**: The centroids were set to the specified initial points.
- **Assignment**: For each data point, the Euclidean distance to each centroid was calculated, and the point was assigned to the nearest centroid, thereby forming clusters.
- **Update**: After assigning all points to clusters, the centroids were recalculated by taking the mean of all points in each cluster.
- **Convergence Check**: The algorithm continued to reassign points and update centroids until the change in centroids between iterations was below a set convergence threshold of 1e-4, or until a maximum of 100 iterations was reached. This threshold ensured the algorithm stopped once it achieved stability in centroid positions.

2.2 Q2. b) KMeans Clustering Results and Visualization

- **Initial Centroids:**
  - Centroid 1: $(3.0, 3.0)$
  - Centroid 2: $(2.0, 2.0)$
- **Initial Cluster Assignments:** Using the initial centroids, the algorithm assigned points to clusters, resulting in an initial distribution across the two clusters as follows :-
  - Clusters: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0]$
- **Convergence:** The algorithm reached convergence after 3 iterations.
- **Final Centroids:**
  - Centroid 1: $(5.8, 2.125)$
  - Centroid 2: $(4.2, -0.0556)$
- **Final Cluster Assignments:** With the refined centroids, the final clustering results show that data points were grouped more cohesively. The final assignment of each data point is as follows :-
  - Clusters: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0]$

- **Visualization:** To demonstrate the clustering process, we plotted the clusters at both the initial and final stages. In the initial plot, each point was assigned based on the initial centroids located at $(3.0, 3.0)$ and $(2.0, 2.0)$. In the final plot, points were assigned based on the converged centroids at $(5.8, 2.125)$ and $(4.2, -0.0556)$, which provided a clearer separation of the data points into two distinct clusters. The plots are shown below :-

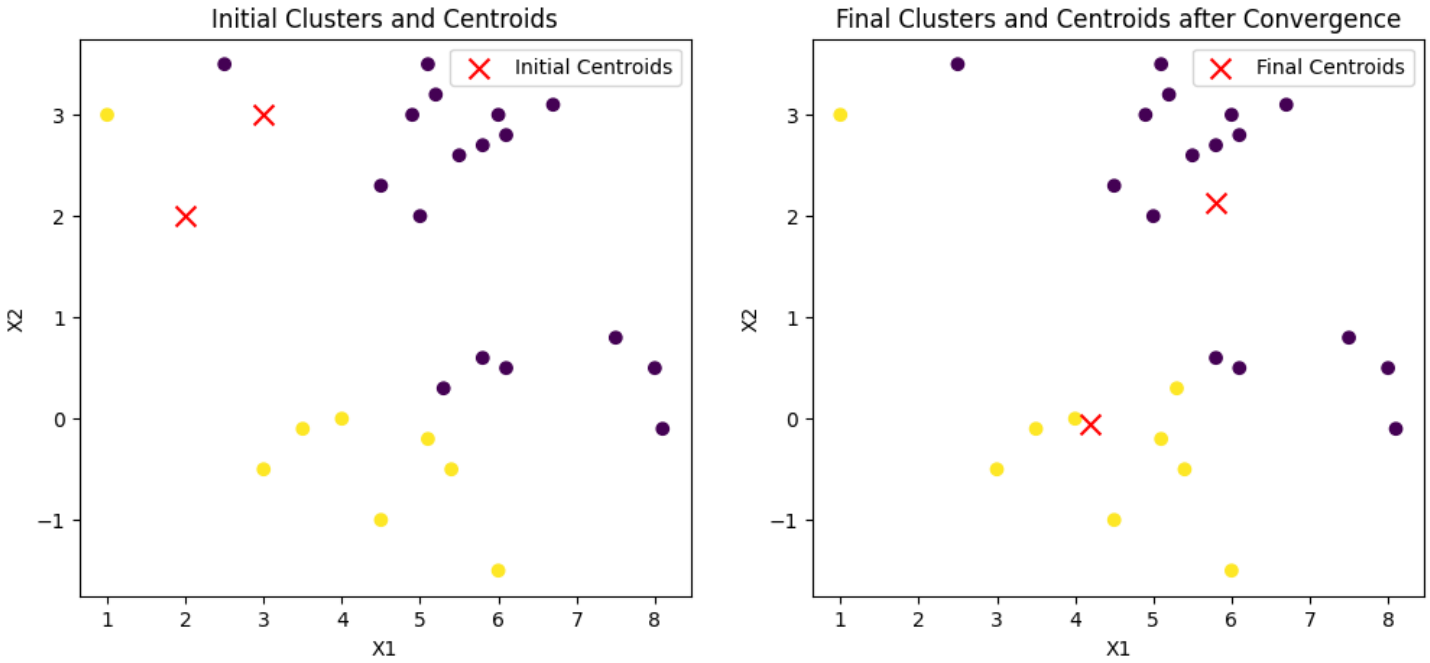


Figure 1: Visualization of clusters and centroids

2.3 Q2. c) KMeans with Random Initialization

I randomly initialized the centroids from the data points. The results obtained from this initialization are given below :-

- **Random Centroids:**
  - Centroid 1: $(4.9, 3.0)$
  - Centroid 2: $(4.5, 2.3)$
- **Initial Cluster Assignments:** Using the random centroids, the algorithm assigned points to clusters, resulting in an initial distribution across the two clusters as follows :-
  - Clusters: $[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$
- **Convergence:** The algorithm reached convergence after 3 iterations.
- **Final Centroids:**
  - Centroid 1: $(4.8583, 2.8917)$
  - Centroid 2: $(5.5615, -0.0923)$
- **Final Cluster Assignments:** With the refined centroids, the final clustering results show that data points were grouped more cohesively. The final assignment of each data point is as follows :-
  - Clusters: $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

- **Visualization:**

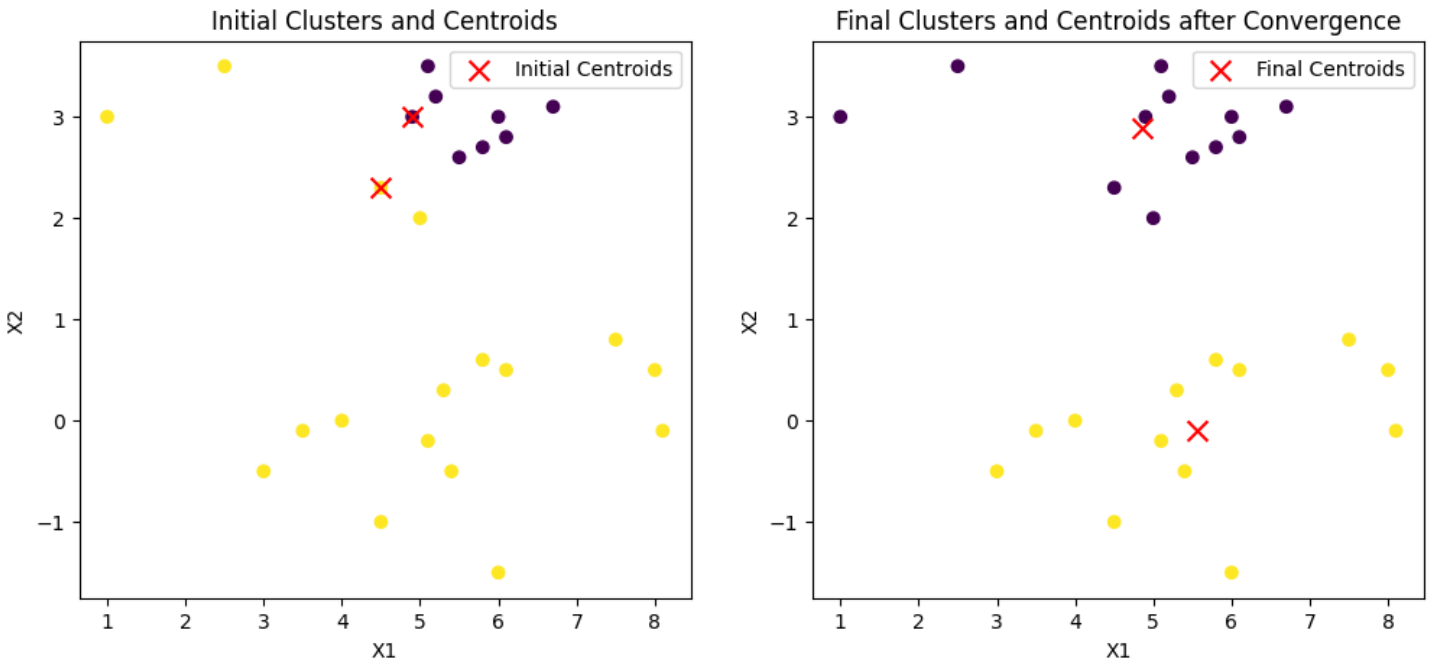


Figure 2: Visualization of clusters and centroids with random initialization

Comparing the results from provided initialization & random initialization shows that both methods achieved convergence of clusters, but the cluster boundaries and assignments slightly differ due to the difference in initial centroids.

I used the KMeans loss function (WCSS) to evaluate the clustering quality for both the initializations. The **initial loss** for the provided centroids was significantly higher (301.75) compared to the random centroids (152.01), indicating that the random centroids were initially closer to the data distribution. After convergence, the **final loss** for the provided centroids was 83.67, while the random initialization achieved a lower final loss of 67.16. This suggests that random initialization resulted in tighter, more cohesive clusters in this case, showing different initialization can lead to different local minima for KMeans.

These results demonstrate that clustering quality in KMeans strongly depends on the initial assignment of centroids, as it directly impacts the algorithm's convergence and final cluster structure.

2.4 Q2. d) Elbow Method to determine Optimal Clustering

The elbow method was applied to determine the optimal number of clusters M. The Within Cluster Sum of Squares (WCSS) was calculated for values of k ranging from 1 to 10. The WCSS decreased significantly as k increased but showed a diminishing return at $\mathbf{k = 5}$, indicating the "elbow point."

So, I chose $\mathbf{M = 5}$ from visual inspection. The plot of the Elbow Method is given below :-

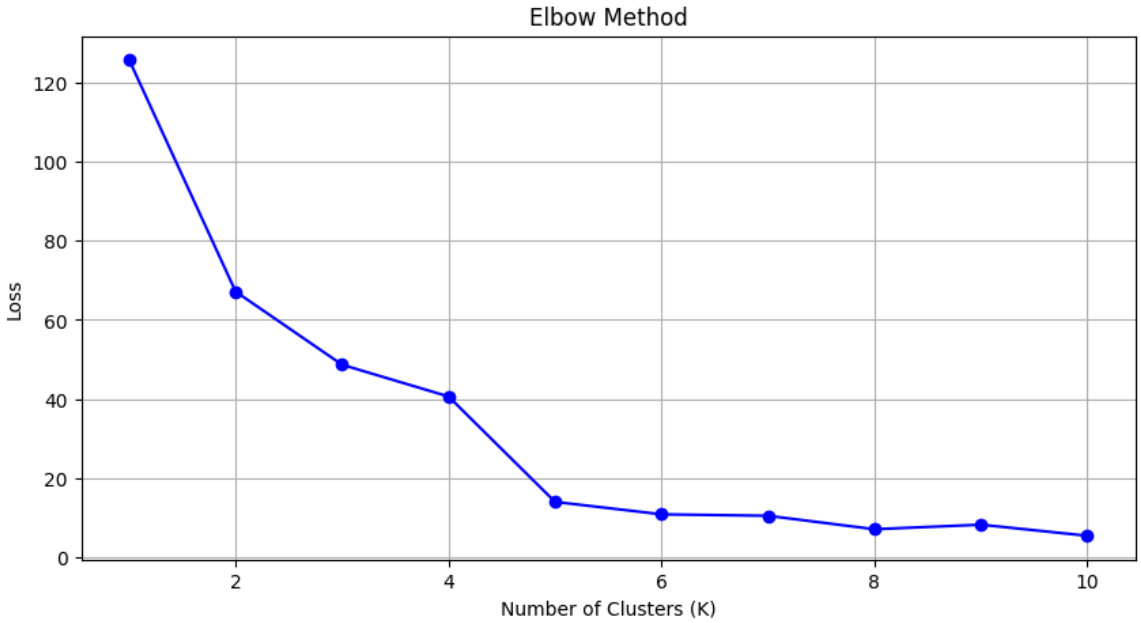


Figure 3: Elbow Method: Determining Optimal Cluster Count

Clustering was performed with $\mathbf{M = 5}$ clusters using randomly initialized centroids. The resulting clusters were well-separated, with the final centroids accurately representing the data distribution. This demonstrates the effectiveness of the elbow method in identifying the appropriate number of clusters for KMeans. The resulting clusters plot is given below :-

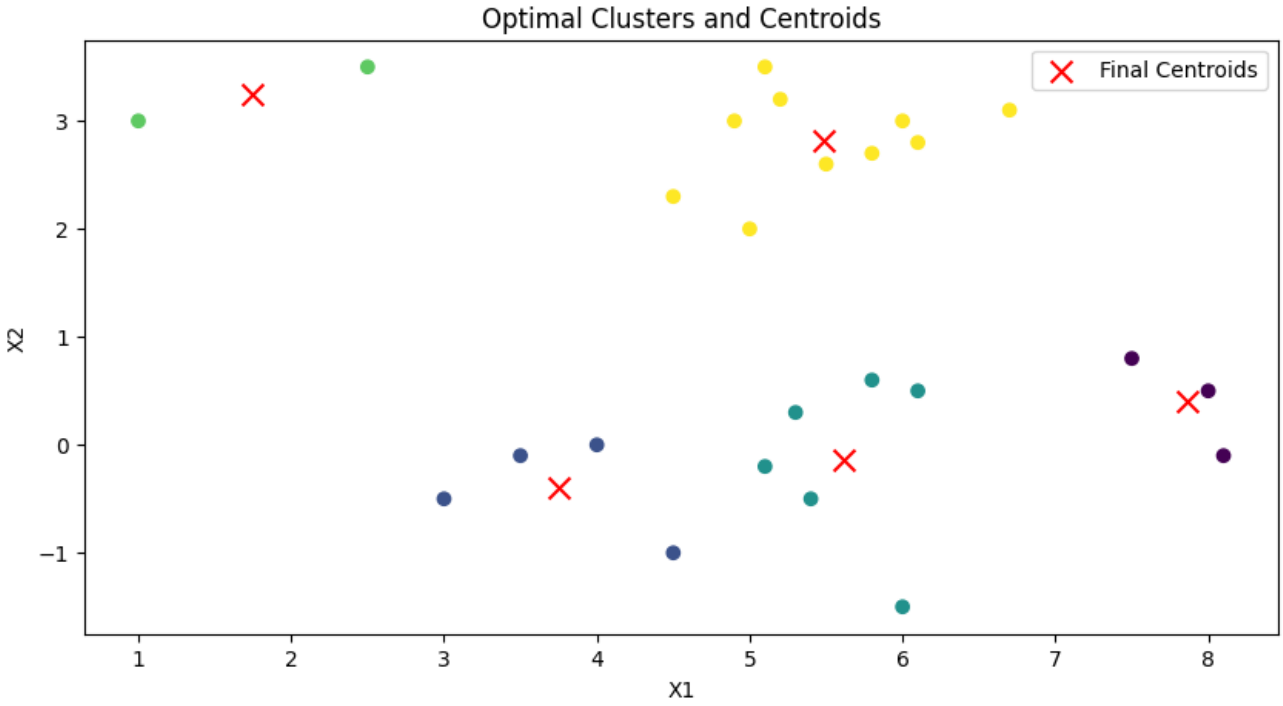


Figure 4: Optimal Clustering Results with M=5 Clusters

Clustering Results for M = 5:

- **Initial Centroids:**
  Centroid 1: $(5.8, 0.6)$, Centroid 2: $(4.5, -1.0)$, Centroid 3: $(5.1, -0.2)$, Centroid 4: $(4.5, 2.3)$, Centroid 5: $(6.7, 3.1)$
- **Initial Cluster Assignments:** Using the random centroids, the algorithm assigned points to clusters, resulting in an initial distribution across the two clusters as follows :-
  - Clusters: $[3, 3, 4, 4, 4, 3, 4, 3, 3, 3, 0, 0, 0, 3, 3, 1, 1, 2, 1, 1, 2, 0, 2, 2, 0]$
- **Convergence:** The algorithm reached convergence after 5 iterations.
- **Final Centroids:**
  Centroid 1: $(7.87, 0.4)$, Centroid 2: $(3.75, -0.4)$, Centroid 3: $(5.62, -0.13)$, Centroid 4: $(1.75, 3.25)$, Centroid 5: $(5.48, 2.82)$
- **Final Cluster Assignments:** With the refined centroids, the final clustering results show that data points were grouped more cohesively. The final assignment of each data point is as follows :-
  - Clusters: $[4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0, 0, 3, 3, 1, 1, 2, 2, 1, 1, 2, 2, 2, 2]$
- **Visualization:**

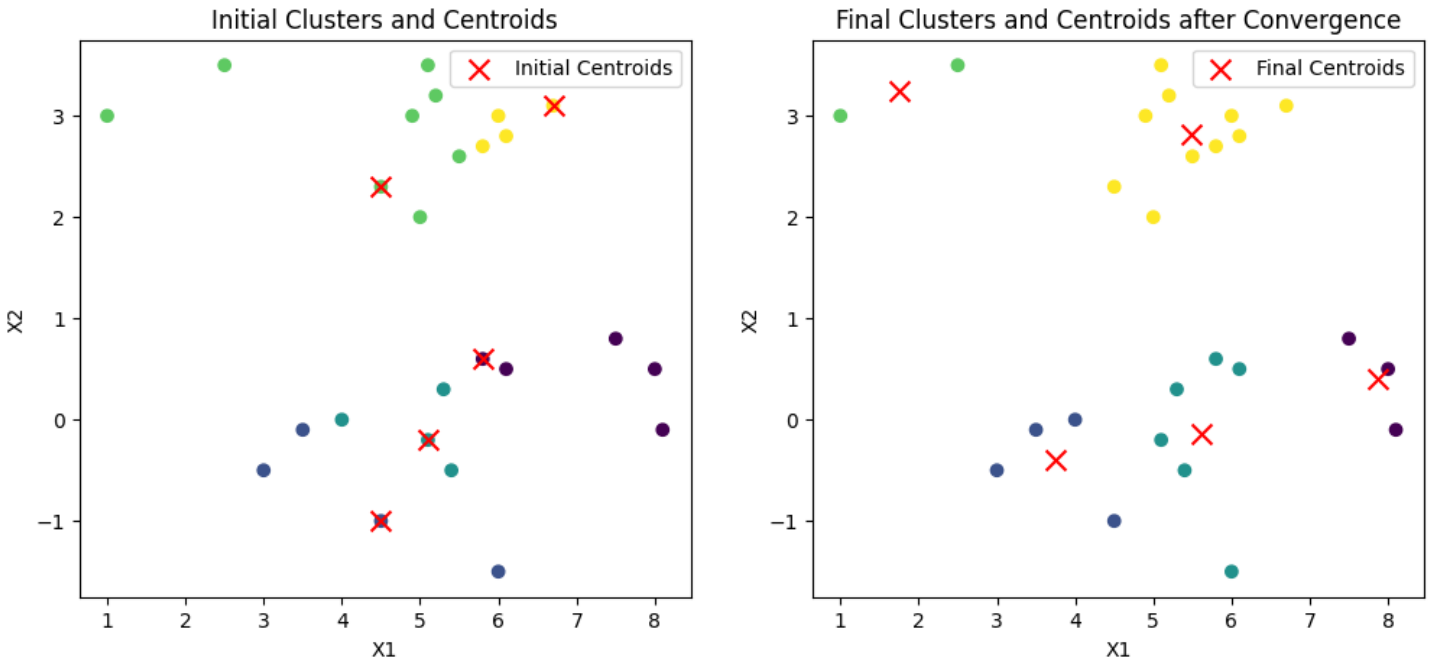


Figure 5: Visualization of clusters and centroids with optimal initialization

Loss Based Evaluation for M = 5:

The initial WCSS for M = 5 was **47.47**, which reduced significantly to **13.46** after convergence. This final loss is markedly lower compared to the losses obtained for K = 2 and random initialization, confirming that M = 5 provides a much better clustering structure by achieving tighter and more cohesive clusters. This further supports the effectiveness of the elbow method in selecting the optimal number of clusters.

3 Section C

We have been given the "CIFAR-10" dataset, which is a collection of 60,000 32x32 color images divided into 10 classes, with 6,000 images per class. It includes 50,000 training images and 10,000 testing images, with each image labeled with one of the 10 categories.

3.1 Q3. 1) Data Preparation

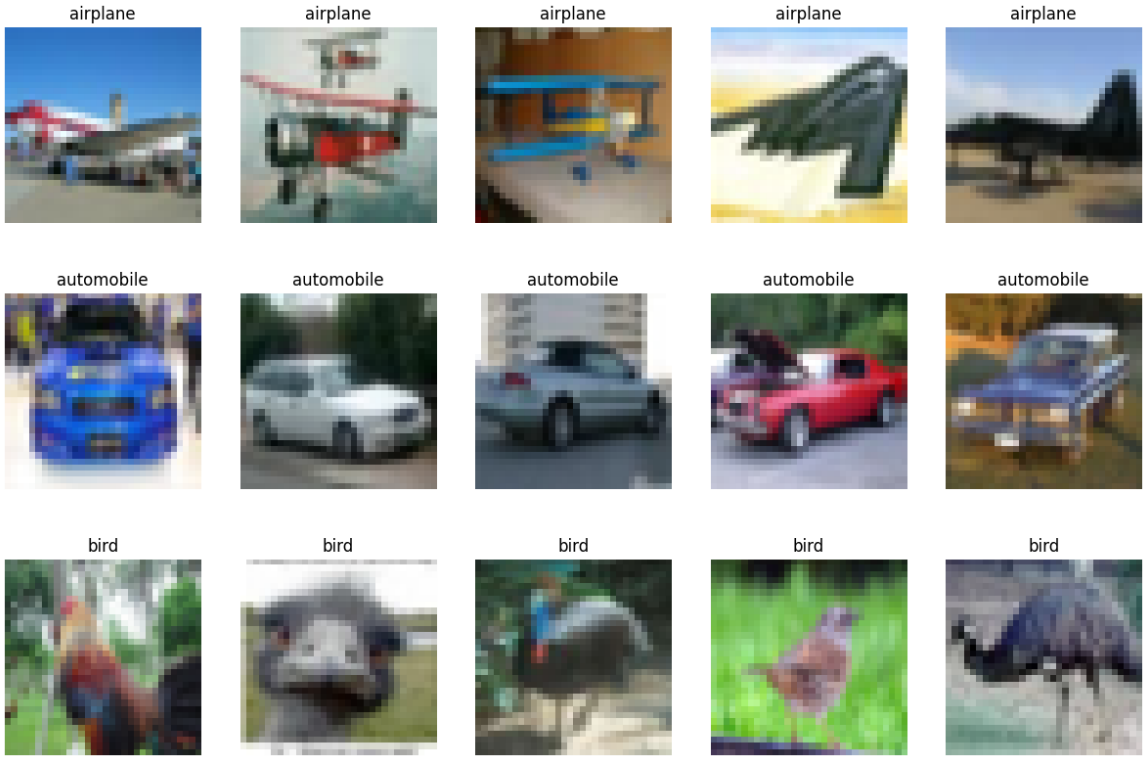
In this task, the CIFAR-10 dataset was used to create a curated dataset consisting of three classes: airplane (class 0), automobile (class 1) and bird (class 2). These classes were selected from the available 10 classes in the CIFAR-10 dataset. The transformation pipeline converts the original PIL images into tensors and normalizes the pixel values to the range $[-1, 1]$.

The dataset preparation process included filtering the CIFAR-10 dataset to retain only the selected classes, splitting the dataset into three subsets—training, validation, and testing—and creating PyTorch DataLoaders for each of the resulting subsets. The training dataset consists of 15,000 images, with an 80% split for training (12,000 images) and a 20% split for validation (3,000 images). This split was performed using a stratified random method to ensure balanced representation across the selected classes. The test dataset comprises exactly 3,000 images, with 1,000 images per class, extracted from the original CIFAR-10 test set. To handle these operations, a custom PyTorch dataset class, `CustomCIFAR10Dataset`, was implemented.

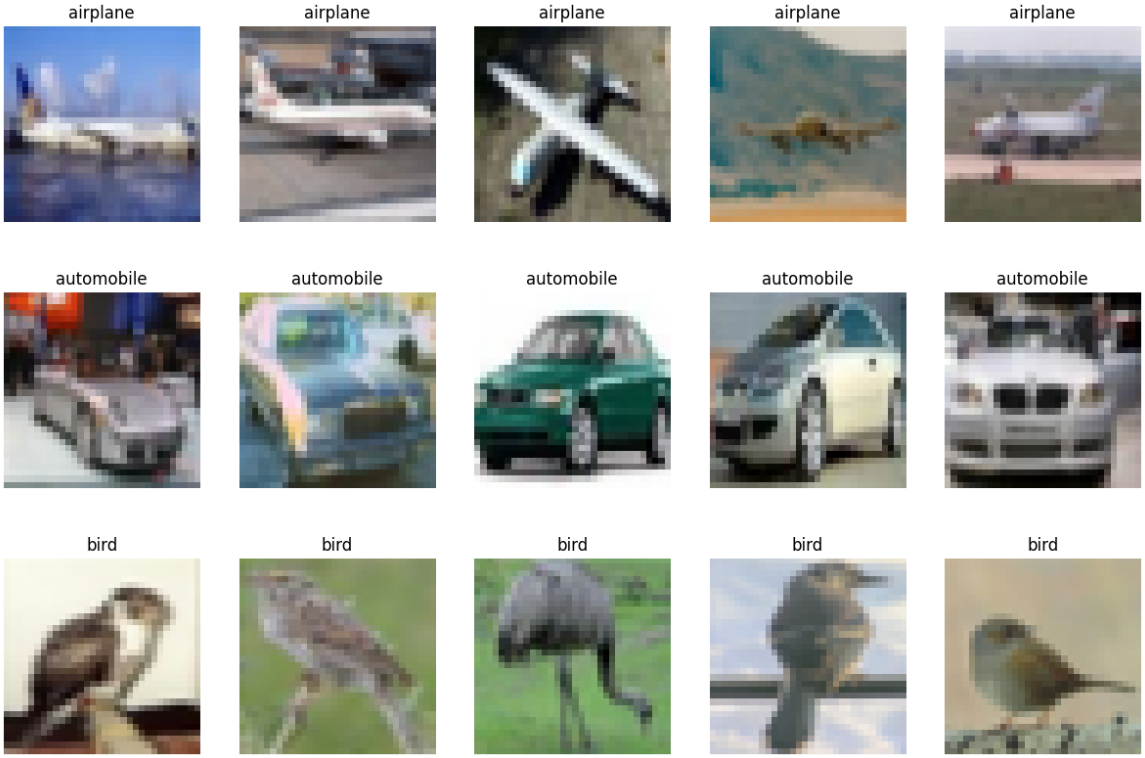
The implementation was guided by PyTorch's official tutorials for handling datasets and DataLoaders, which can be found in the following links: CIFAR-10 Tutorial (PyTorch) and Datasets & Dataloaders (PyTorch).

3.2 Q3. 2) Visualization

The visualization below displays 5 sample images from each selected class (airplane, automobile, bird) from both the training and validation datasets. The code for image visualization was adapted from the PyTorch CIFAR-10 Tutorial.



(a) Sample images from the training dataset for the selected classes



(a) Sample images from the validation dataset for the selected classes

3.3 Q3. 3) CNN Implementation

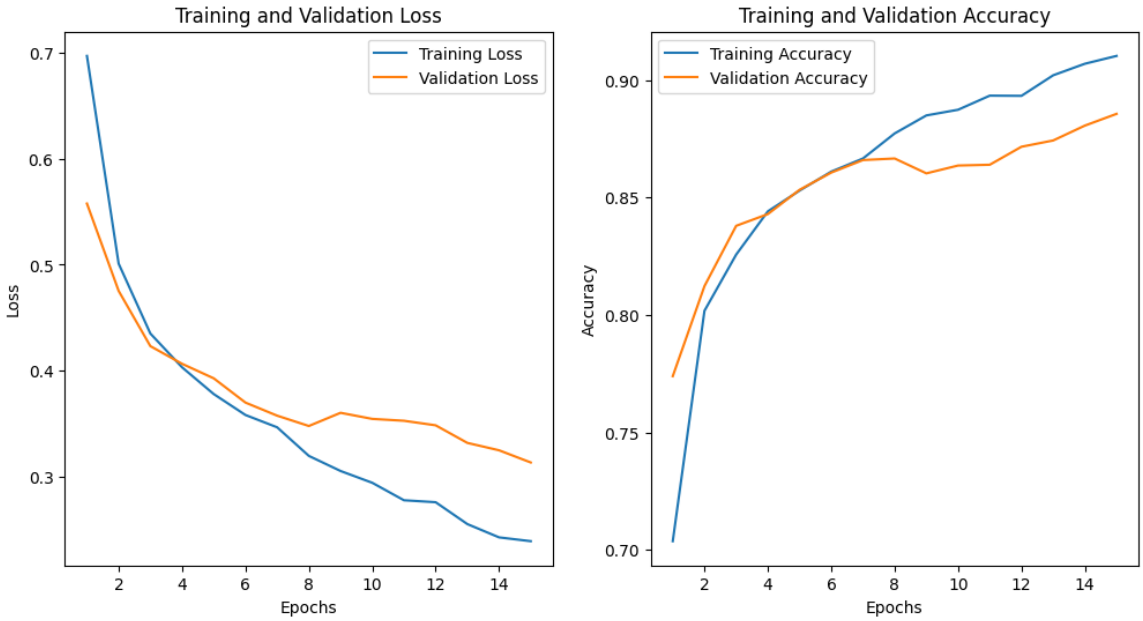
CNN model with 2 convolutional layers, followed by max-pooling, and a multi-layer perceptron was implemented for classification. The model uses ReLU activations and outputs predictions for 3 selected classes. The code for CNN implementation, training the model & testing was adapted from the PyTorch CIFAR-10 Tutorial.

3.4 Q3. 4) Training the model

The model was trained using the cross-entropy loss function and the Adam optimizer for 15 epochs. During training, inputs were fed through the network, the loss was calculated, and gradients were backpropagated to optimize the model parameters. Metrics such as training and validation loss, along with accuracy, were logged after each epoch to monitor performance. Validation was conducted without gradient updates to assess the model's generalization capability. The final trained model was saved as `CNNmodel.pth` for future use.

3.5 Q3. 5) Testing

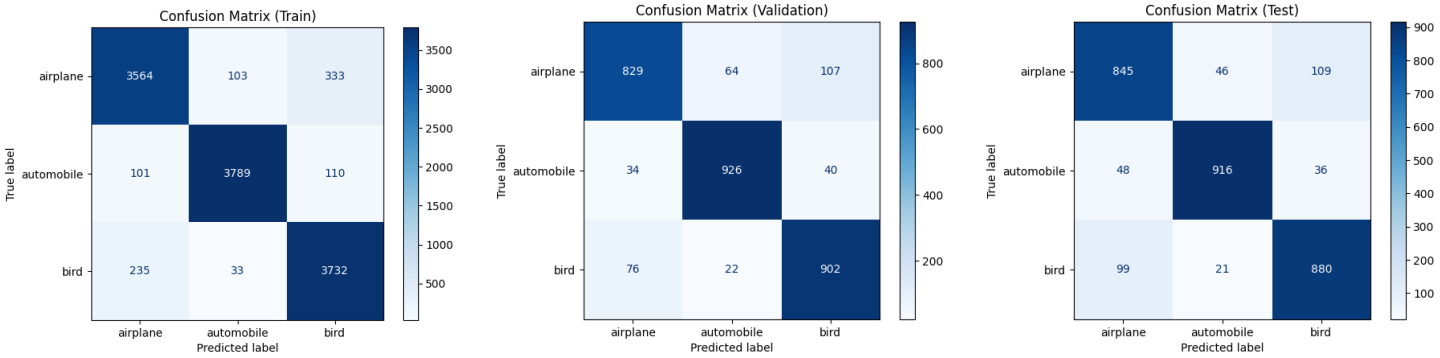
The training and validation loss, along with accuracy, were plotted to analyze the model's performance over the 15 epochs.



(a) Training and Validation Losses & Accuracies

The plots show steady decreases in training and validation loss, with no significant divergence, indicating minimal overfitting. Both training and validation accuracy improve consistently, suggesting the model is learning effectively and generalizing well to the validation data.

The final model achieved a Test Accuracy of **88.03%** and a Test F1-Score of **88.05%**, indicating a well performing model with balanced precision and recall across the selected classes (automobile, bird, horse). The confusion matrices for the train, validation, and test datasets are given below :



(a) Training Dataset (b) Validation Dataset (c) Test Dataset

Figure 9: Confusion Matrices for Training, Validation, and Test Datasets.

3.6 Q3. 6) Training an MLP

The MLP model was trained for 15 epochs using cross-entropy loss and Adam optimizer. It consists of two fully connected layers: the first with 64 neurons and a ReLU activation, and the second as the classification head. The model's performance was tracked by logging training and validation loss and accuracy after each epoch. The trained model is saved as `MLPmodel.pth`.

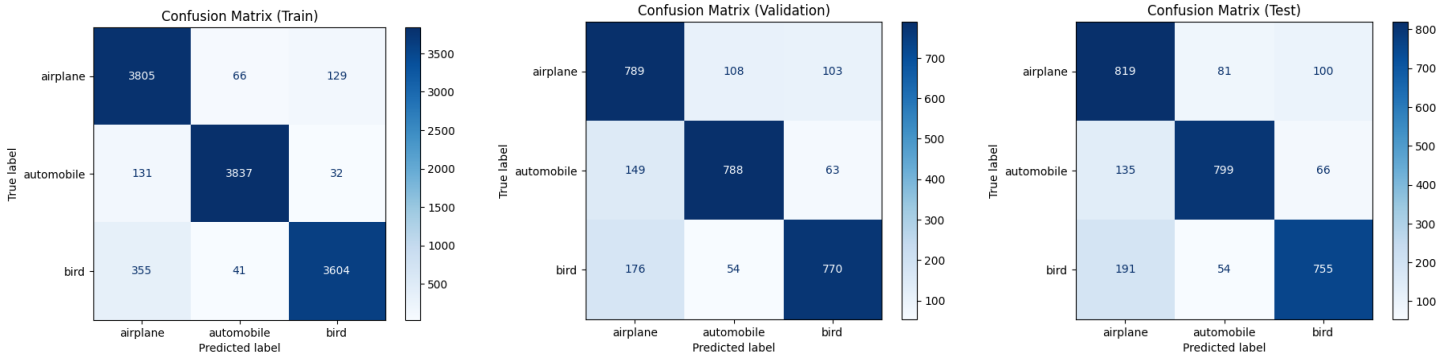
3.7 Q3. 7) Infer and Compare

The training and validation loss, along with accuracy, were plotted to analyze the model's performance over the 15 epochs.

(a) Training and Validation Losses & Accuracies

Comparing the CNN and MLP plots, the CNN shows better generalization with consistent validation loss and accuracy trends. In contrast, the MLP overfits as indicated by increasing validation loss despite decreasing training loss, and its validation accuracy fluctuates without significant improvement. This highlights the CNN's superior performance and robustness.

The final model achieved a Test Accuracy of **79.10%** and a Test F1-Score of **79.20%**, indicate CNN performance was better than MLP.The confusion matrices for the train, validation, and test datasets are given below :



(a) Training Dataset (b) Validation Dataset (c) Test Dataset

Figure 11: Confusion Matrices for Training, Validation, and Test Datasets.