CSE343
Machine Learning

Monsoon 2024

Report for Assignment 4

Shamik Sinha - 2022468

Section A

(a) Dimensions of the resulting feature map

Given an input image with dimensions $M \times N$ (where $\min(M, N) \geq 1$), P channels ($P \geq 1$), and a single kernel of size $K \times K$ ($1 \leq K \leq \min(M, N)$), the dimensions of the resulting feature map can be determined as follows:

$$\text{Output Height} = M - K + 1$$
$$\text{Output Width} = N - K + 1$$

Since the stride is 1 and there is no padding, the kernel slides across the input image, reducing the dimensions of the input image by $K - 1$ in both height and width. Therefore, the resulting feature map dimensions are:

$$\text{Resulting feature map dimensions} = (M - K + 1) \times (N - K + 1)$$

(b) Number of elementary operations for a single output pixel

For a single output pixel in the feature map:

- **Kernel Operation:** The kernel of size $K \times K$ spans K^2 positions in the input image. Each kernel element performs one multiplication with the corresponding input pixel, resulting in a total of K^2 multiplications.
- **Summation:** After performing the multiplications, the results are summed up to produce a single output pixel. This requires $K^2 - 1$ additions.
- **For P Channels:** If the input image has P channels, the kernel spans all channels. Therefore, the total number of operations for P channels is:
  - Multiplications: $P \times K^2$
  - Additions: $P \times (K^2 - 1)$

Thus, the total elementary operations for one output pixel are:

$$\text{Total operations} = P \times K^2 \text{ (multiplications)} + P \times (K^2 - 1) \text{ (additions)}$$

$$\text{Total operations} = P \times (2K^2 - 1)$$

(c) Computational time complexity with Q kernels

To compute the computational time complexity of the forward pass with Q kernels, consider the following:

Number of Output Pixels in the Feature Map

From part (a), the resulting feature map dimensions for a single kernel are:

$$\text{Output Height} = M - K + 1 \quad \text{and} \quad \text{Output Width} = N - K + 1$$

Thus, the total number of output pixels in the feature map is:

$$\text{Number of output pixels} = (M - K + 1)(N - K + 1)$$

Computational Cost for a Single Output Pixel

From part (b), the cost to compute one output pixel for a single kernel is:

$$\text{Cost for one pixel} = P \times (2K^2 - 1)$$

- The total computation cost for one kernel is:

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

- For Q kernels, the total computation cost is:

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

  Expanding this:

  $$= O(QPK^2(MN - MK - NK + K^2))$$

- If $\min(M, N) \gg K$, the complexity simplifies to:

  $$O(QPK^2MN)$$

Part B.

K-Means Algorithm

The K-Means algorithm is an unsupervised machine learning technique used for clustering data into K distinct groups. It operates in two main steps:

1. Assignment Step

In this step, each data point x_i is assigned to the cluster whose centroid c_j is nearest to it. This is computed based on a distance metric (e.g., Euclidean distance):

$$\text{Cluster assignment for } x_i = \arg\min_j \|x_i - c_j\|$$

Thus, each data point is grouped with the closest centroid, forming K clusters.

2. Update Step

After assigning all points to clusters, the centroids of the clusters are recalculated. The new centroid for a cluster is the mean of all the data points assigned to that cluster:

$$c_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

Here, C_j is the set of data points in cluster j, and $|C_j|$ is the number of points in the cluster.

After updating the centroids, the process repeats until convergence.

Optimal Number of Clusters

Determining the optimal number of clusters K is not straightforward and often depends on the dataset. One common method to determine the optimal K is the Elbow Method:

- Run K-Means for different values of K (e.g., $K = 1, 2, 3, \ldots$).
- Compute the Within-Cluster Sum of Squares (WCSS), which measures the variance within clusters:

$$\text{WCSS} = \sum_{j=1}^{K} \sum_{x_i \in C_j} \|x_i - c_j\|^2$$

- Plot K vs. WCSS. As K increases, WCSS decreases because the clusters become smaller.
- Identify the "elbow point" in the plot, where the rate of decrease in WCSS slows down. This is the optimal K.

Randomly Assigning Cluster Centroids and Global Minima

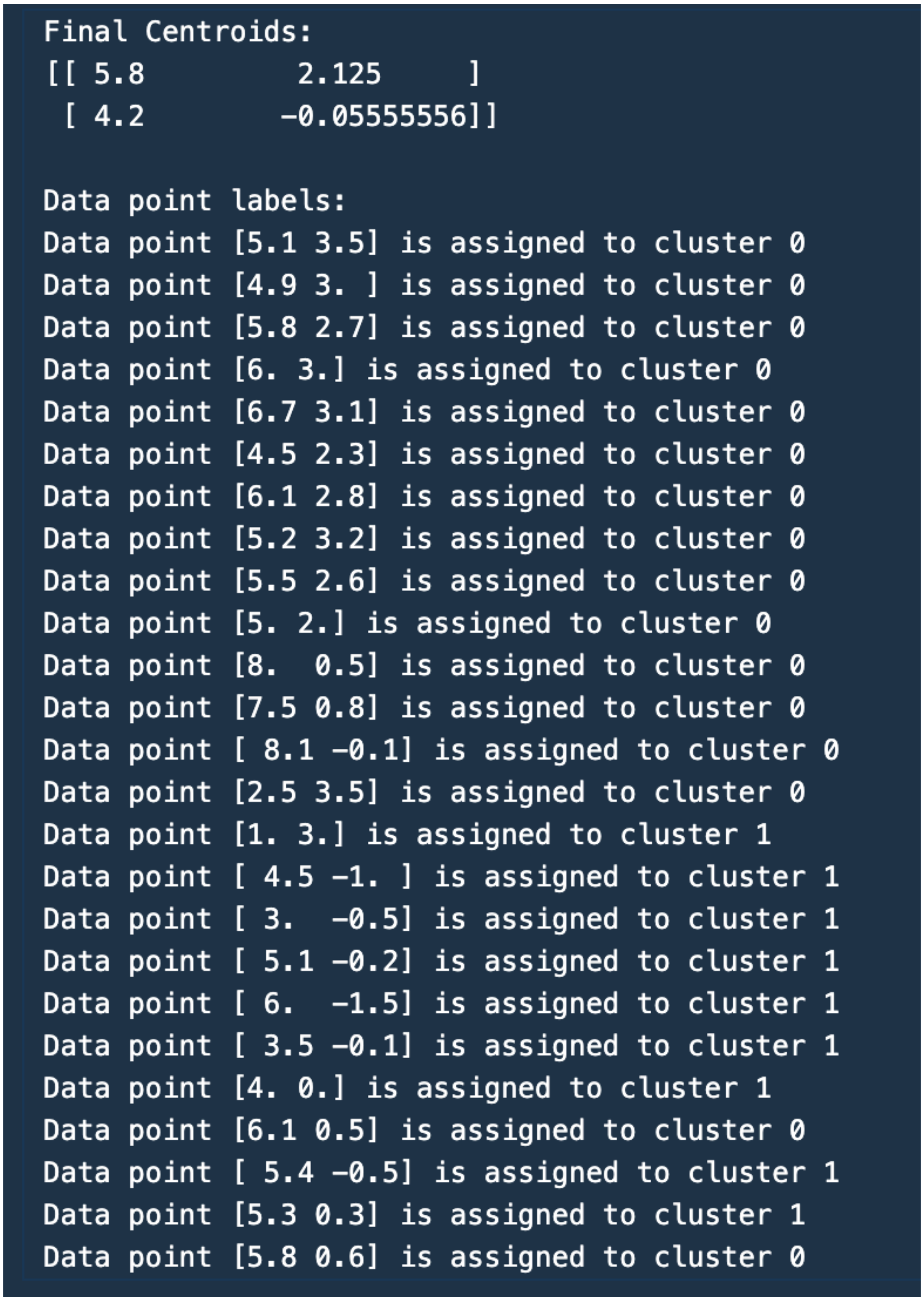
Can K-Means Reach a Global Minimum?

No, K-Means does not guarantee convergence to a global minimum because it uses a greedy approach. The algorithm's performance heavily depends on the initial placement of the centroids. Poor initialization can lead to suboptimal clustering (local minima).

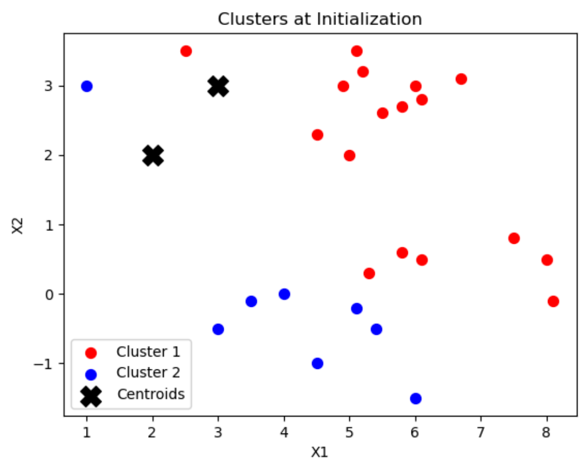
To mitigate poor initialization, the K-Means++ algorithm can be used:

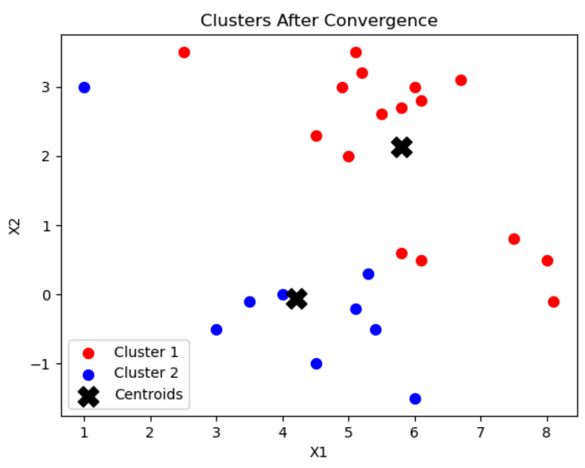
- Select the first centroid randomly.
- For subsequent centroids, choose data points with a probability proportional to their squared distance from the nearest already-selected centroid.
- This method spreads the initial centroids and often results in faster convergence and better clusters.

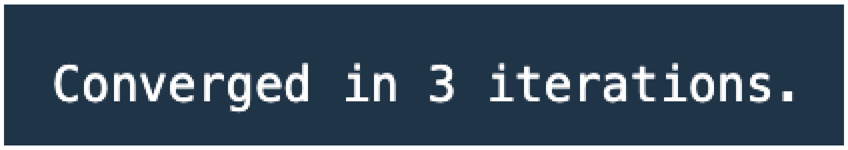
Section B



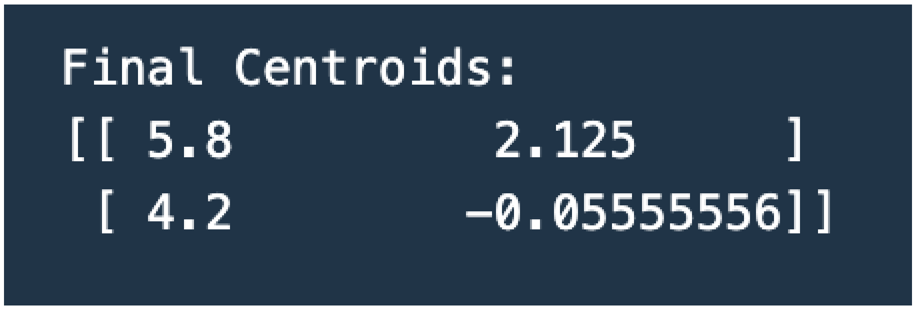
```
Final Centroids:
[[ 5.8         2.125     ]
 [ 4.2        -0.05555556]]

Data point labels:
Data point [5.1 3.5] is assigned to cluster 0
Data point [4.9 3. ] is assigned to cluster 0
Data point [5.8 2.7] is assigned to cluster 0
Data point [6. 3.] is assigned to cluster 0
Data point [6.7 3.1] is assigned to cluster 0
Data point [4.5 2.3] is assigned to cluster 0
Data point [6.1 2.8] is assigned to cluster 0
Data point [5.2 3.2] is assigned to cluster 0
Data point [5.5 2.6] is assigned to cluster 0
Data point [5. 2.] is assigned to cluster 0
Data point [8.  0.5] is assigned to cluster 0
Data point [7.5 0.8] is assigned to cluster 0
Data point [ 8.1 -0.1] is assigned to cluster 0
Data point [2.5 3.5] is assigned to cluster 0
Data point [1. 3.] is assigned to cluster 1
Data point [ 4.5 -1. ] is assigned to cluster 1
Data point [ 3.  -0.5] is assigned to cluster 1
Data point [ 5.1 -0.2] is assigned to cluster 1
Data point [ 6.  -1.5] is assigned to cluster 1
Data point [ 3.5 -0.1] is assigned to cluster 1
Data point [4. 0.] is assigned to cluster 1
Data point [6.1 0.5] is assigned to cluster 0
Data point [ 5.4 -0.5] is assigned to cluster 1
Data point [5.3 0.3] is assigned to cluster 1
Data point [5.8 0.6] is assigned to cluster 0
```



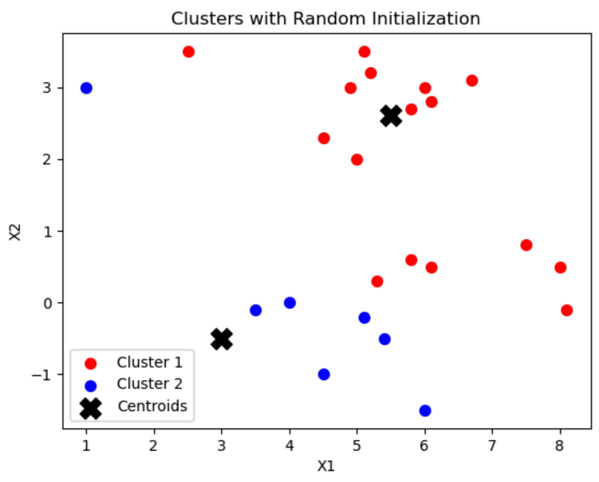


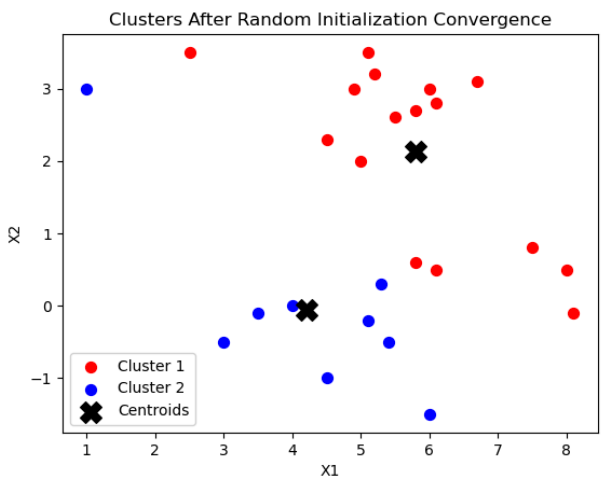


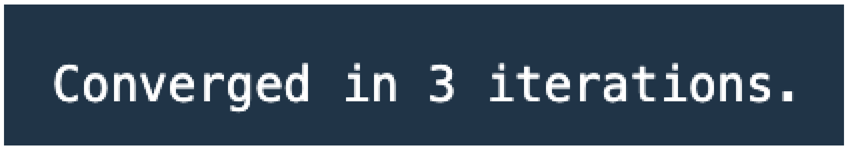
```
Converged in 3 iterations.
```



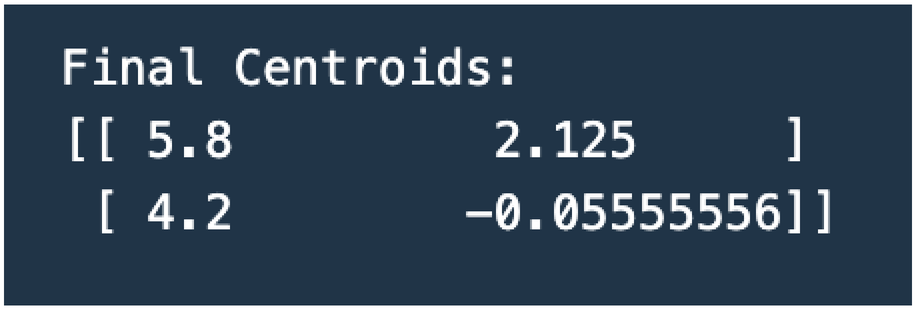
```
Final Centroids:
[[ 5.8         2.125     ]
 [ 4.2        -0.05555556]]
```



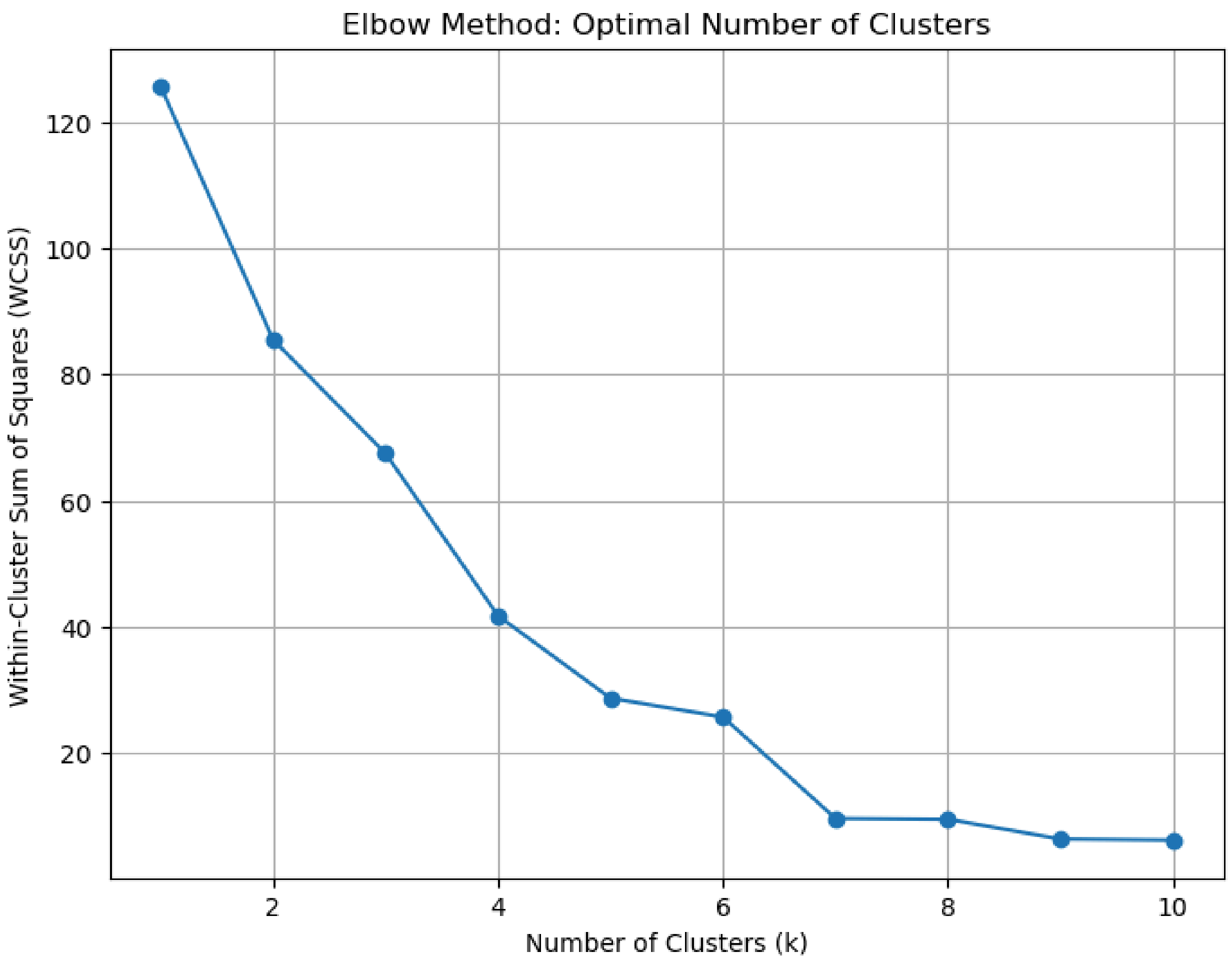


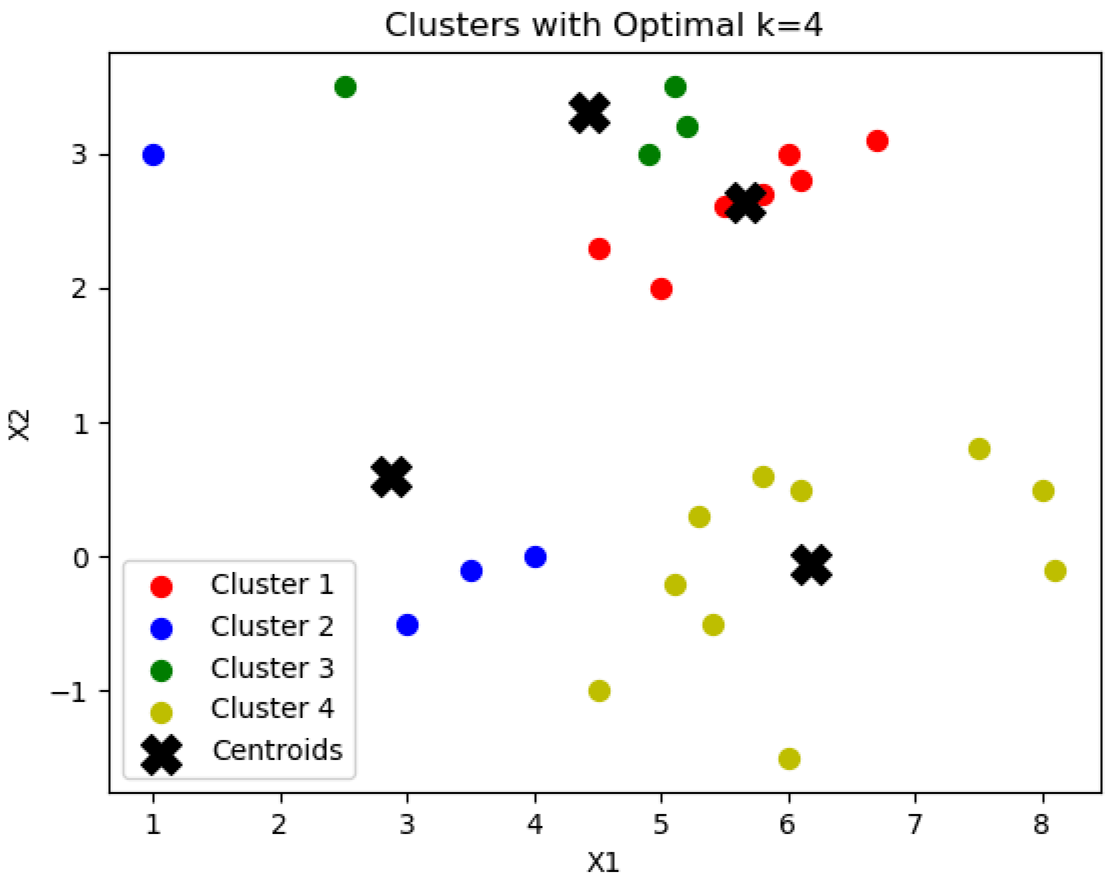


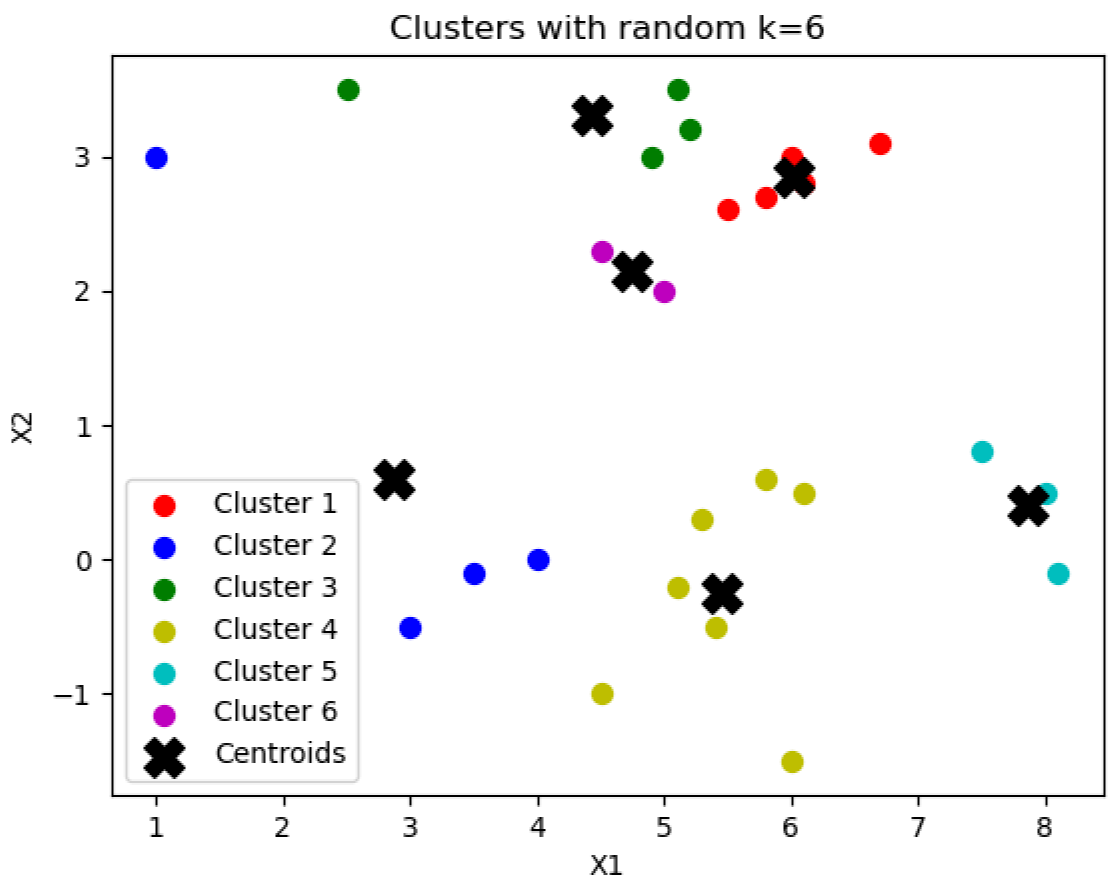
```
Converged in 3 iterations.
```



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


Elbow Method: Optimal Number of Clusters
Within-Cluster Sum of Squares (WCSS)
Number of Clusters (k)
20
40
60
80
100
120
2
4
6
8
10


Clusters with Optimal k=4
X2
X1
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Centroids



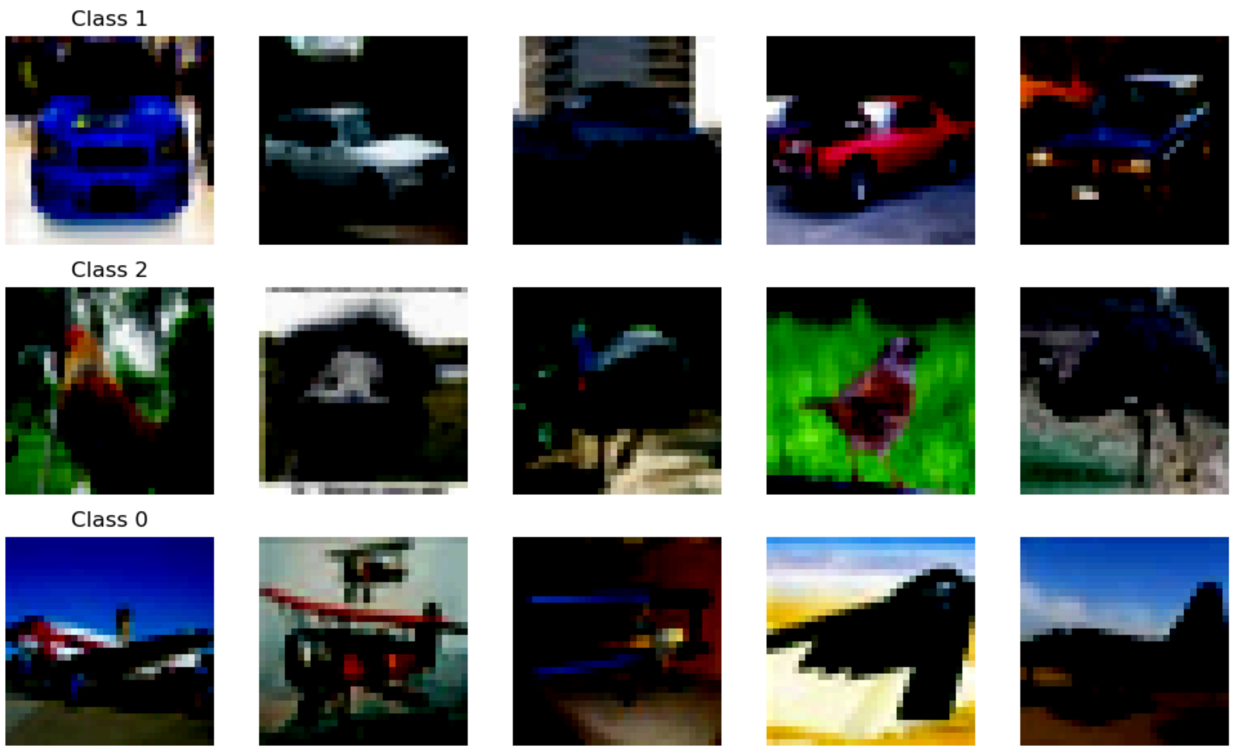
Comparing Initial Centroids vs. Random Initialization

When using K-Means clustering, the choice of initial centroids can significantly impact the final clustering result. Using initial centroids can potentially lead to faster convergence and a better solution, especially if they are well-placed. However, this requires prior knowledge or domain expertise.

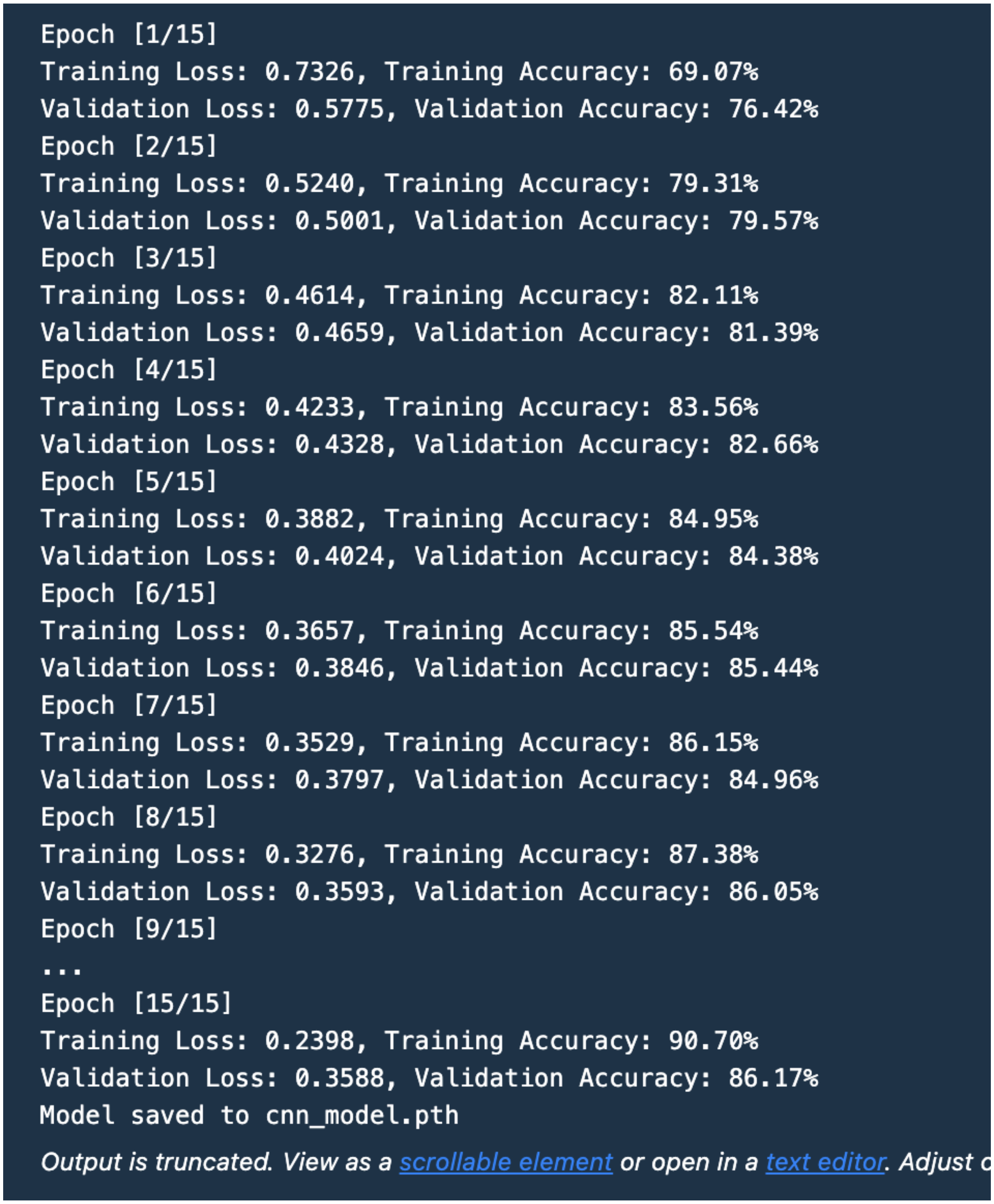
On the other hand, random initialisation is simpler and more robust, as it doesn't rely on any prior information. While it may take more iterations to converge, it can still yield satisfactory results. The best approach depends on the specific dataset and the desired level of accuracy and efficiency. Factors like data distribution, the number of clusters, and the initialisation technique can influence the performance of K-Means.

Section C

Part 2.



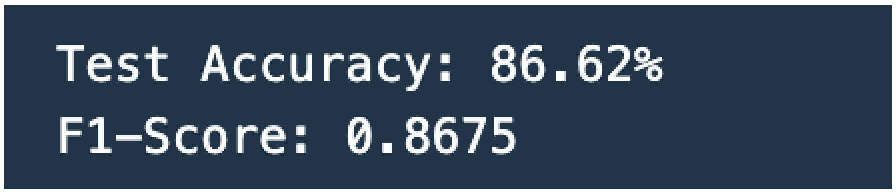
Part 3.



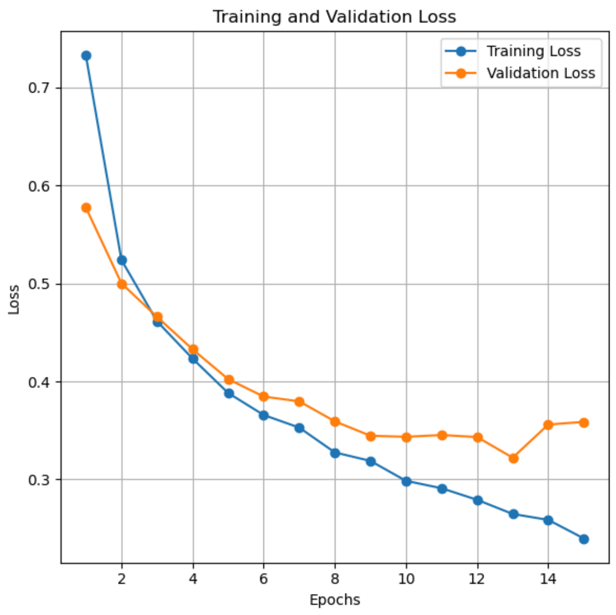
```
Epoch [1/15]
Training Loss: 0.7326, Training Accuracy: 69.07%
Validation Loss: 0.5775, Validation Accuracy: 76.42%
Epoch [2/15]
Training Loss: 0.5240, Training Accuracy: 79.31%
Validation Loss: 0.5001, Validation Accuracy: 79.57%
Epoch [3/15]
Training Loss: 0.4614, Training Accuracy: 82.11%
Validation Loss: 0.4659, Validation Accuracy: 81.39%
Epoch [4/15]
Training Loss: 0.4233, Training Accuracy: 83.56%
Validation Loss: 0.4328, Validation Accuracy: 82.66%
Epoch [5/15]
Training Loss: 0.3882, Training Accuracy: 84.95%
Validation Loss: 0.4024, Validation Accuracy: 84.38%
Epoch [6/15]
Training Loss: 0.3657, Training Accuracy: 85.54%
Validation Loss: 0.3846, Validation Accuracy: 85.44%
Epoch [7/15]
Training Loss: 0.3529, Training Accuracy: 86.15%
Validation Loss: 0.3797, Validation Accuracy: 84.96%
Epoch [8/15]
Training Loss: 0.3276, Training Accuracy: 87.38%
Validation Loss: 0.3593, Validation Accuracy: 86.05%
Epoch [9/15]
...
Epoch [15/15]
Training Loss: 0.2398, Training Accuracy: 90.70%
Validation Loss: 0.3588, Validation Accuracy: 86.17%
Model saved to cnn_model.pth
```

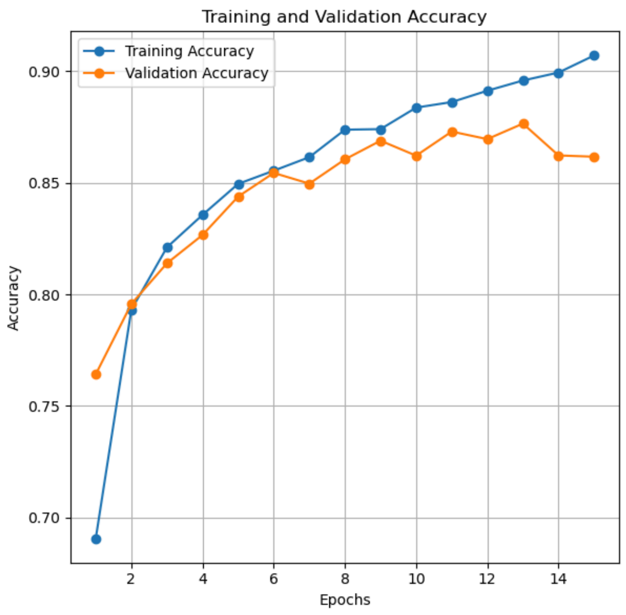
Output is truncated. View as a scrollable element or open in a text editor. Adjust

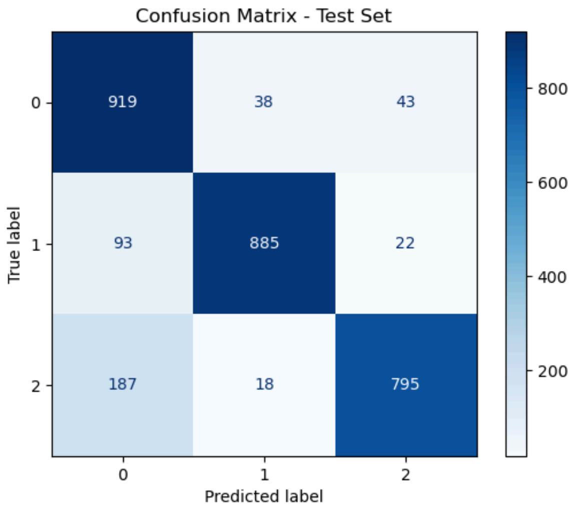
Part 5.

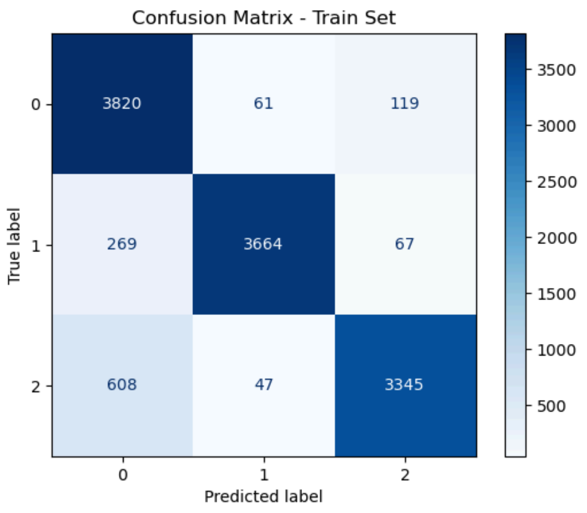


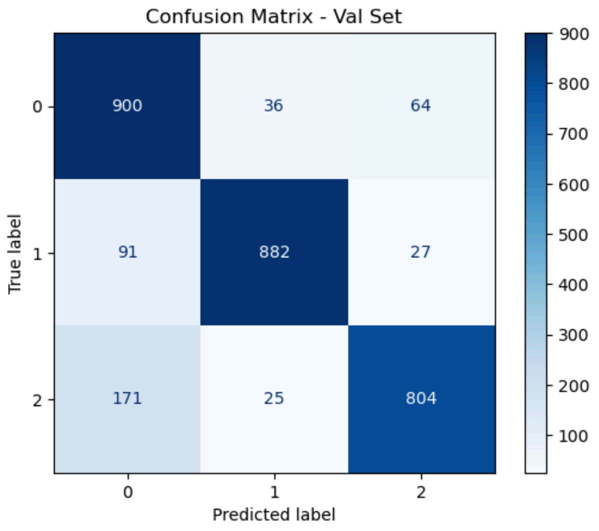
```
Test Accuracy: 86.62%
F1-Score: 0.8675
```



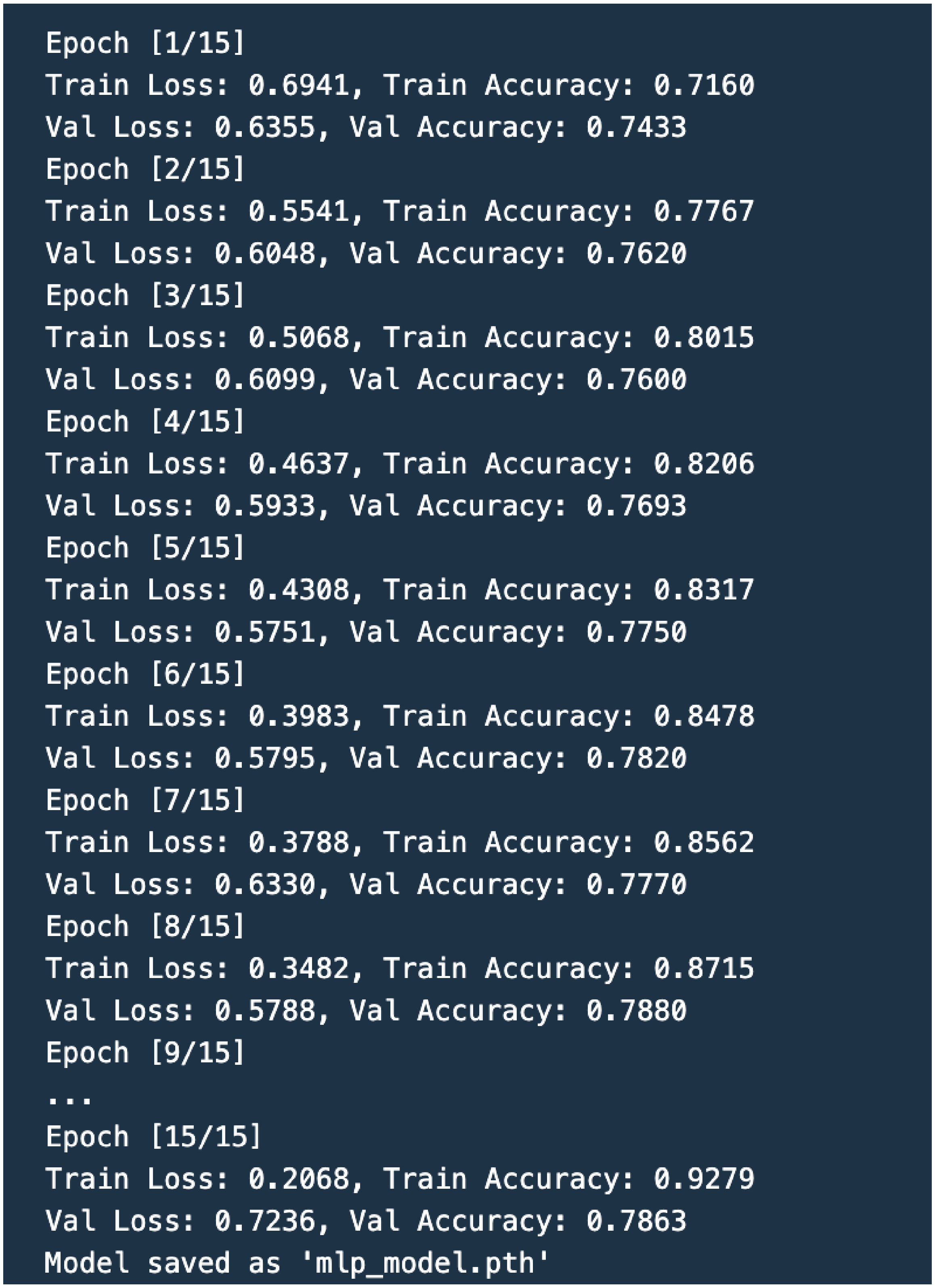



Confusion Matrix - Test Set
919
38
43
93
885
22
187
18
795
True label
Predicted label
0
1
2
200
400
600
800

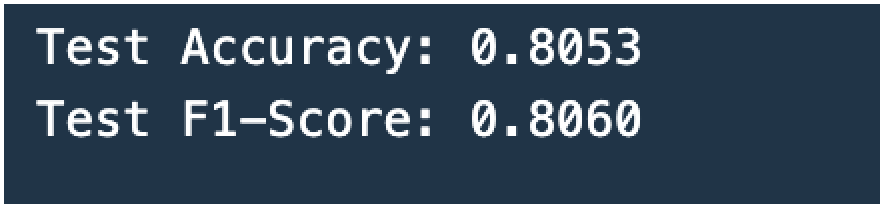

Confusion Matrix - Train Set
3820
61
119
269
3664
67
608
47
3345
True label
Predicted label
0
1
2
500
1000
1500
2000
2500
3000
3500


Confusion Matrix - Val Set
900
36
64
91
882
27
171
25
804
True label
Predicted label
0
1
2
100
200
300
400
500
600
700
800
900

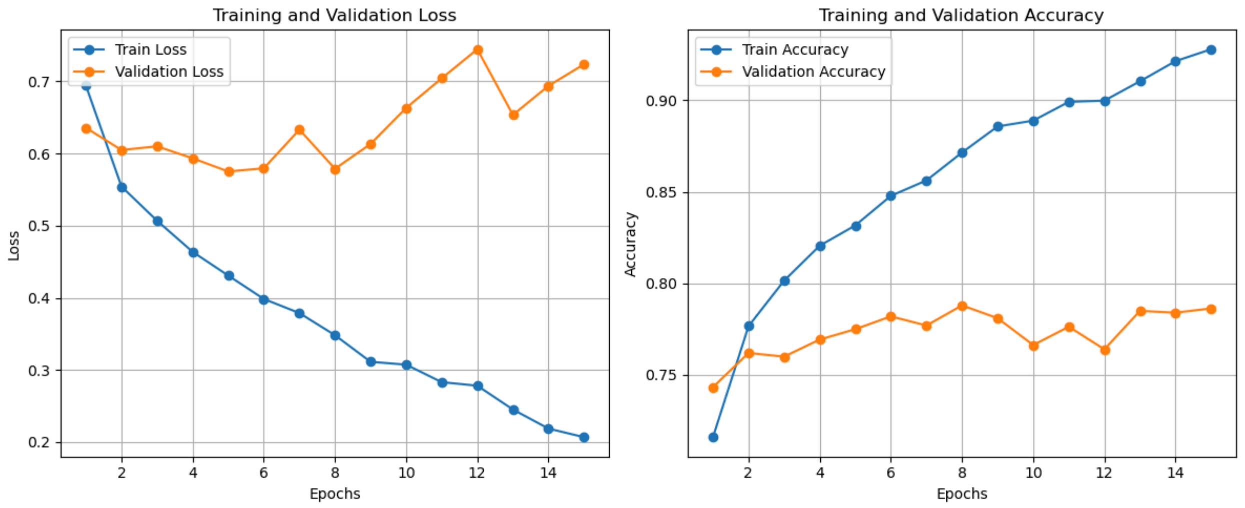
Part 7.

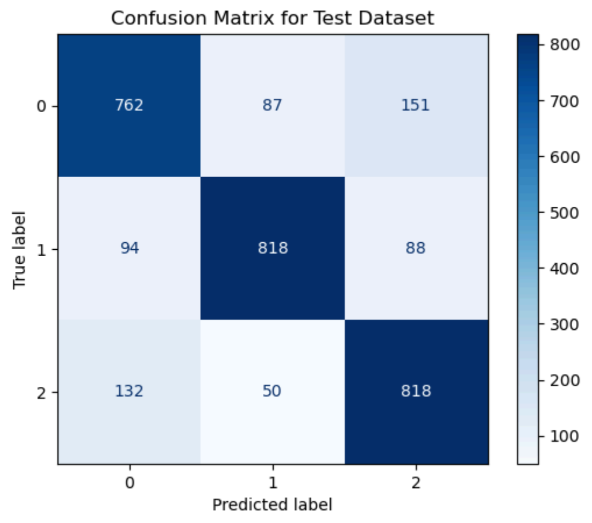


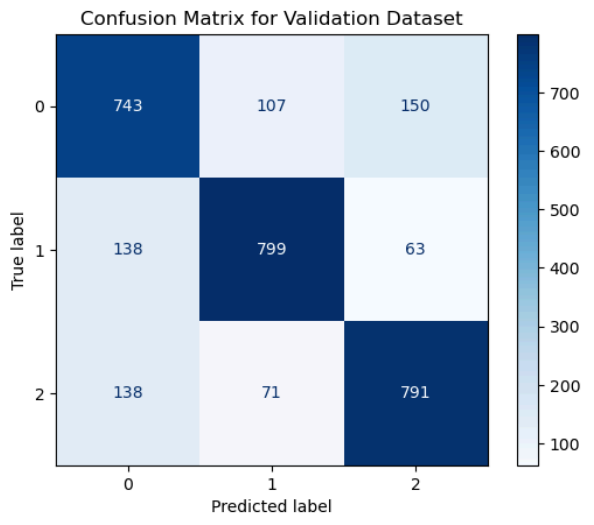
```
Epoch [1/15]
Train Loss: 0.6941, Train Accuracy: 0.7160
Val Loss: 0.6355, Val Accuracy: 0.7433
Epoch [2/15]
Train Loss: 0.5541, Train Accuracy: 0.7767
Val Loss: 0.6048, Val Accuracy: 0.7620
Epoch [3/15]
Train Loss: 0.5068, Train Accuracy: 0.8015
Val Loss: 0.6099, Val Accuracy: 0.7600
Epoch [4/15]
Train Loss: 0.4637, Train Accuracy: 0.8206
Val Loss: 0.5933, Val Accuracy: 0.7693
Epoch [5/15]
Train Loss: 0.4308, Train Accuracy: 0.8317
Val Loss: 0.5751, Val Accuracy: 0.7750
Epoch [6/15]
Train Loss: 0.3983, Train Accuracy: 0.8478
Val Loss: 0.5795, Val Accuracy: 0.7820
Epoch [7/15]
Train Loss: 0.3788, Train Accuracy: 0.8562
Val Loss: 0.6330, Val Accuracy: 0.7770
Epoch [8/15]
Train Loss: 0.3482, Train Accuracy: 0.8715
Val Loss: 0.5788, Val Accuracy: 0.7880
Epoch [9/15]
...
Epoch [15/15]
Train Loss: 0.2068, Train Accuracy: 0.9279
Val Loss: 0.7236, Val Accuracy: 0.7863
Model saved as 'mlp_model.pth'
```

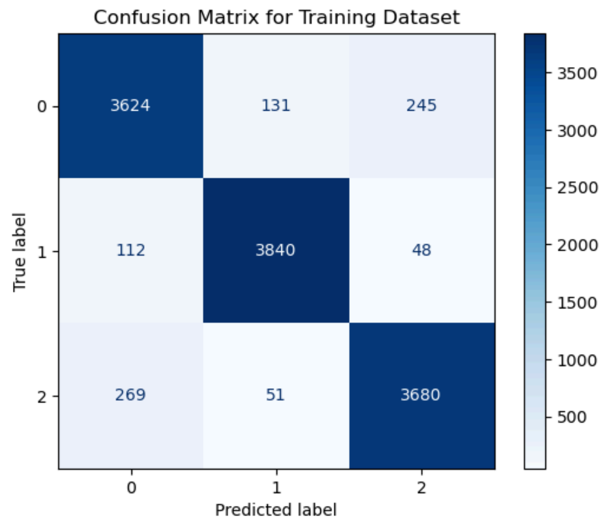


```
Test Accuracy: 0.8053
Test F1-Score: 0.8060
```









Observations and Comparisons

1. Test Accuracy

- CNN: Achieved a test accuracy of 86.620%, significantly higher than the MLP's performance.
- MLP: Scored 80.53% accuracy, indicating it struggled more with feature learning compared to the CNN.

Insight: CNN outperforms the MLP in accuracy because it can extract spatial and hierarchical features from images, which are crucial for image classification tasks. MLPs, on the other hand, treat pixel values independently and cannot capture spatial relationships.

2. F1-Score

- CNN: Delivered an F1-Score of 88.22%, reflecting strong and balanced performance across all classes.
- MLP: Achieved an F1-Score of 80.75%, which, while decent, shows that the MLP struggles more with class balance.

Insight: CNN's ability to handle both majority and minority classes effectively gives it an edge over the MLP, making it harder to deal with class variations.

3. Confusion Matrix Analysis

- For the MLP:
  - As seen above.
- For the CNN:
  - CNN likely had fewer misclassifications and better separation between similar classes due to its ability to extract meaningful spatial features.

Insight: The MLP struggles to differentiate between visually similar classes, like Class 0 and Class 2, because it lacks the ability to understand spatial patterns. In contrast, CNNs excel at such tasks by leveraging spatial relationships in the data.

4. Precision, Recall, and Class-Wise Performance

- MLP's Class 1: Performed the best with a precision of 0.83 and recall of 0.86, suggesting it was relatively easier to classify.
- MLP's Class 0: Showed the weakest performance, with a precision of 0.79 and recall of 0.74, indicating difficulty in correctly identifying samples of this class.
- CNN: Likely delivers consistently higher precision and recall across all classes, thanks to its superior ability to extract spatial features.

Conclusions

1. Performance Gap:
   - The CNN significantly outperformed the MLP in both accuracy and F1-Score because of its ability to learn spatial and hierarchical features, which are essential for image classification tasks like CIFAR-10.
   - While the MLP is simpler and faster to train, it lacks the ability to handle the spatial structure of image data, which limits its performance.
2. Suitability for Image Classification:
   - CNNs are far better suited for image classification tasks where spatial relationships and feature hierarchies are important.

    - MLPs may be more appropriate for datasets with non-spatial features or lower-dimensional inputs.
3. Misclassification Patterns:
    - The MLP struggled to distinguish between visually similar classes, leading to more errors.
    - CNNs, with their convolutional layers, reduce these errors by learning localised patterns like edges and textures.
4. CNN Modelcand MLP Model:
    - Training: Less stable loss curves with many spikes and dips indicate less stability in the course of training. The test accuracy in comparison with the validation is lesser, indicating overfitting or poor generalization. Performance was less stable and less generalised than CNN.
    - Comparison: The CNN is generally more stable in training and has a better generalization overall, and also the accuracy is greater and the generalization is better. The MLP is unstable in training and reveals overfitting and a lesser test accuracy.