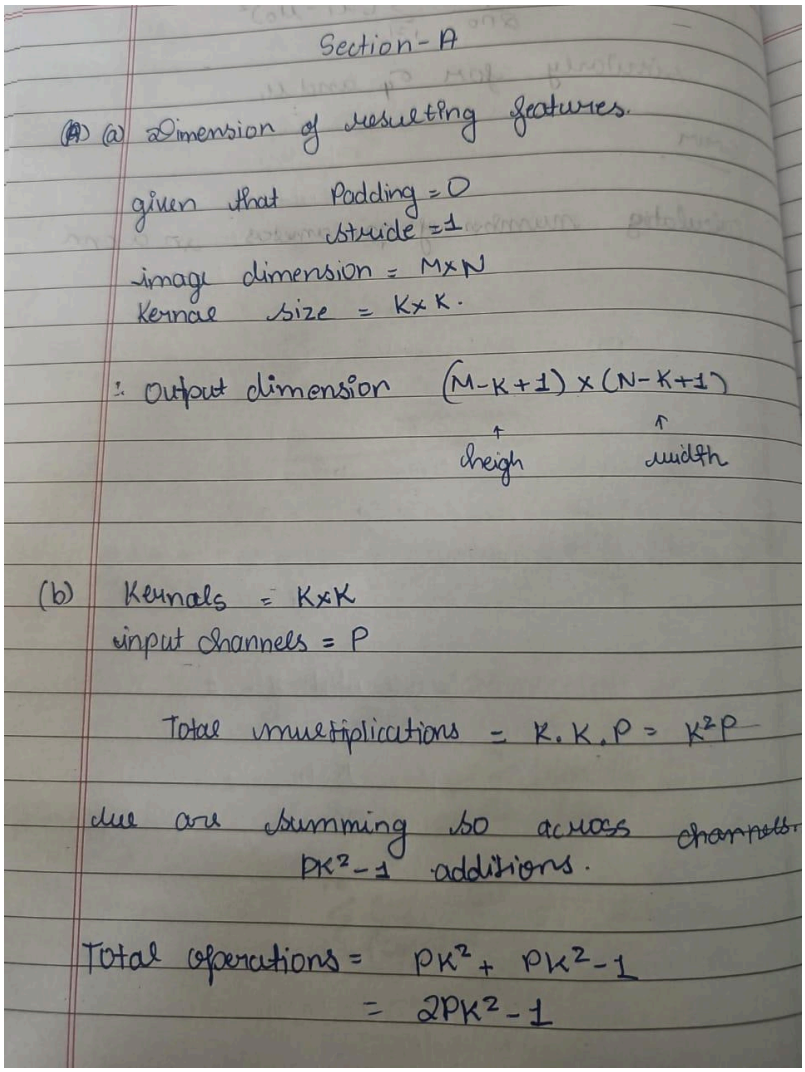
SECTION-A

PART-A



Section-A

(a) Dimension of resulting features.

given that Padding = 0
stride = 1
image dimension = M×N
Kernal size = K×K.

∴ Output dimension $(M-K+1) \times (N-K+1)$

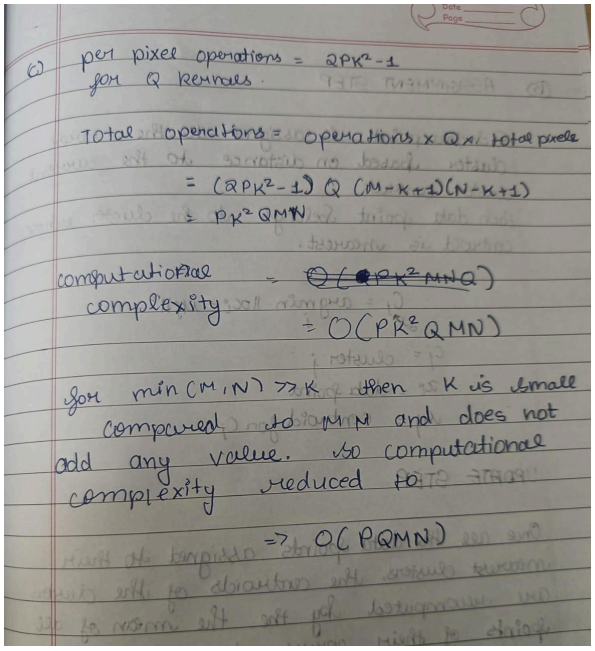
(height, width)

(b) Kernals = K×K
input channels = P

Total multiplications = K.K.P = K^2P

due are summing so across channels.
PK^2-1 additions.

Total operations = $PK^2 + PK^2 - 1$
= $2PK^2 - 1$



(c) per pixel operations = $2PK^2 - 1$
for Q kernels.

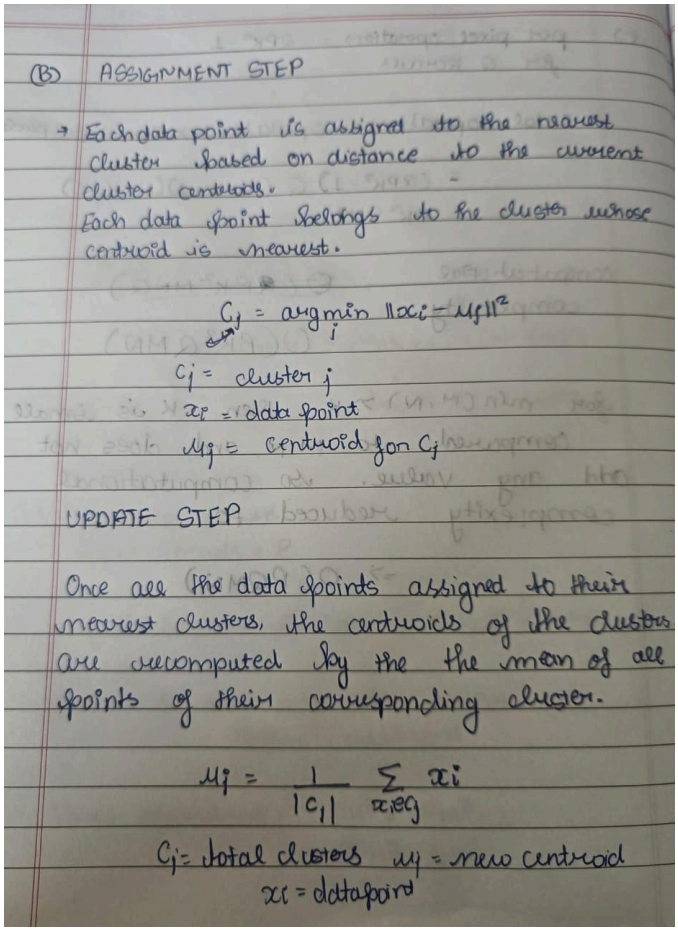
Total operations = operations x Q x total pixels
$= (2PK^2 - 1)\, Q\, (M-K+1)(N-K+1)$
$\approx PK^2QMN$

computational complexity = ~~$O(PK^2MNQ)$~~
$= O(PK^2QMN)$

for $\min(M, N) \gg K$ then K is small compared to M and does not add any value. so computational complexity reduced to

$\Rightarrow O(PQMN)$

PART-B



(B) ASSIGNMENT STEP

→ Each data point is assigned to the nearest cluster based on distance to the current cluster centroids.
Each data point belongs to the cluster whose centroid is nearest.

$$C_j = \arg\min_i \|x_i - \mu_j\|^2$$

C_j = cluster j
x_i = data point
μ_j = centroid for C_j

UPDATE STEP

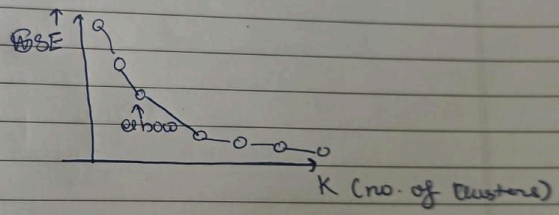
Once all the data points assigned to their nearest clusters, the centroids of the clusters are recomputed by the the mean of all points of their corresponding cluster.

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

C_j = total clusters μ_j = new centroid
x_i = datapoint

→ Method to determine optimal number of clusters is Elbow Method.
In this Method, As K increases the sum of squares within a group (SSE) decreases because when K↑ the points become closer to their centroids.

At a point, the rate decreases forming elbow in plot. which indicates the optimal number of clusters.



$$SSE = \sum_{i} \sum_{x_i = c_j} \|x_i - \mu_j\|^2$$

No, random initialization of cluster cent. does not guarantee convergence to global minim. In some cases it converges. K mean algo is sensitive to the initial points of centroids.

SECTION-B

This report explores:

- Implementation of K-Means from scratch.
- Comparison of clustering results using given versus random initial centroids.
- Evaluation of the clustering performance.
- WCSS

Given data:

```
X = np.array([
    [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]
])
```

Initial Centroids:

$$\mathbf{u}_1 = (3.0, 3.0)$$

$$\mathbf{u}_2 = (2.0, 2.0)$$

Euclidian k-means Algorithm

Algorithm Implementation

1. Initialization: Centroids are initialized as: u1=(3.0,3.0),u2=(2.0,2.0)
2. Assignment: Each point is assigned to the nearest centroid using Euclidean distance
3. Update: Centroids are recalculated as the mean of points in each cluster.
4. Convergence Check: The algorithm stops when centroid shifts are below a threshold (10−410^{-4}10−4) or after 100 iterations.

PART-A

The algorithm converged in **3 iterations**

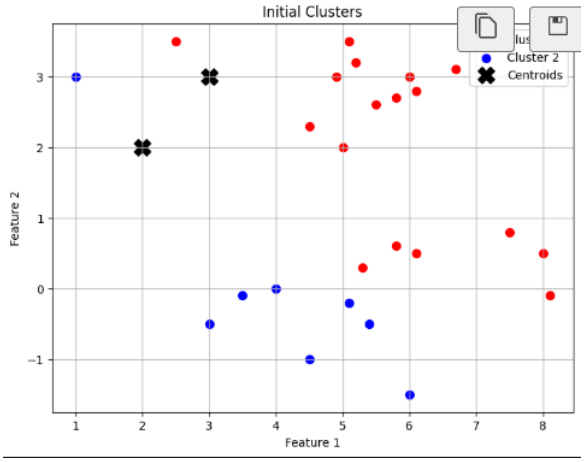
Initial Centroids: u1=(3.0,3.0), u1=(3.0,3.0)
Final Centroids: u1=(5.8,2.125), u2=(4.2,−0.0556)

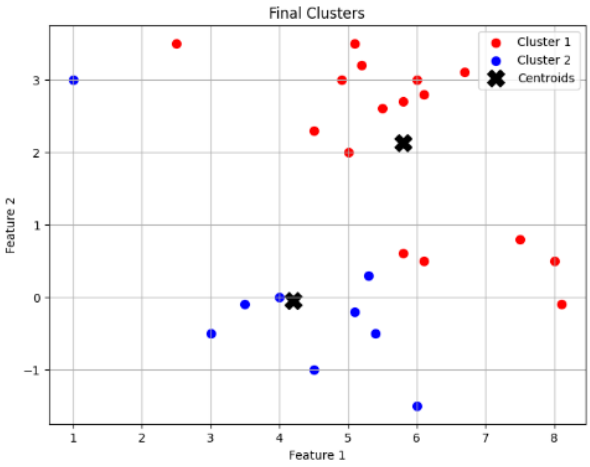
Finar Cluster Assignment

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

Data points were roughly split by the initial centroids. Improved separations achieved after convergence.
The clustering achieved a silhouette score of 0.354, indicating moderate separation between clusters.

PART-B

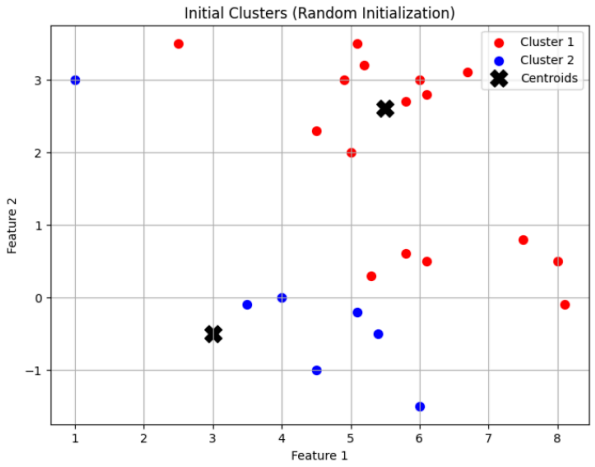


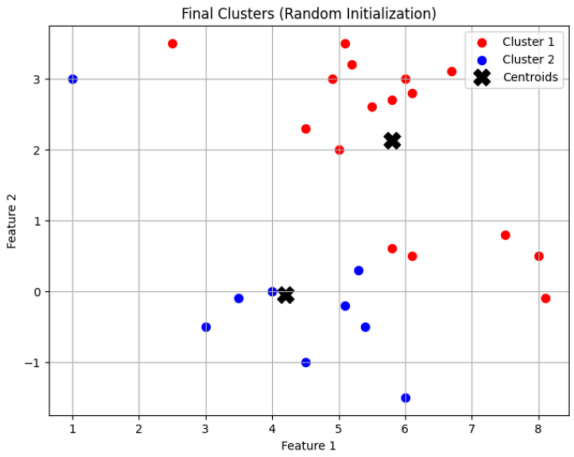


PART-C

On random Initialisation as in Picture The algorithm converged in **3 iterations**.

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





Silhouette Score: The score was slightly higher at **0.354**.

Observation:
On initial Centroids and on random initialization centroids both the initialization converged at the same centroids for the both that is:

```
Centroids with provided initialization:
[[ 5.8         2.125     ]
 [ 4.2        -0.05555556]]

Centroids with random initialization:
[[ 5.8         2.125     ]
 [ 4.2        -0.05555556]]
```

PART-D

WCSS is a metric used to evaluate the quality of clustering, representing the sum of squared distances between each data point and its closest cluster centroid.
As the number of clusters (k) increases, the WCSS generally decreases because data points are assigned to closer centroids, reducing the distances.

In this implementation, most cases converged within 2-3 iterations, indicating the algorithm's efficiency in finding stable cluster configurations.

For **k = 1**, all data points belong to a single cluster, with the centroid representing the mean of the entire dataset. This results in a high WCSS value since no distinctions are made between clusters. When **k = 2**, the data is divided into two clusters, significantly reducing the WCSS as

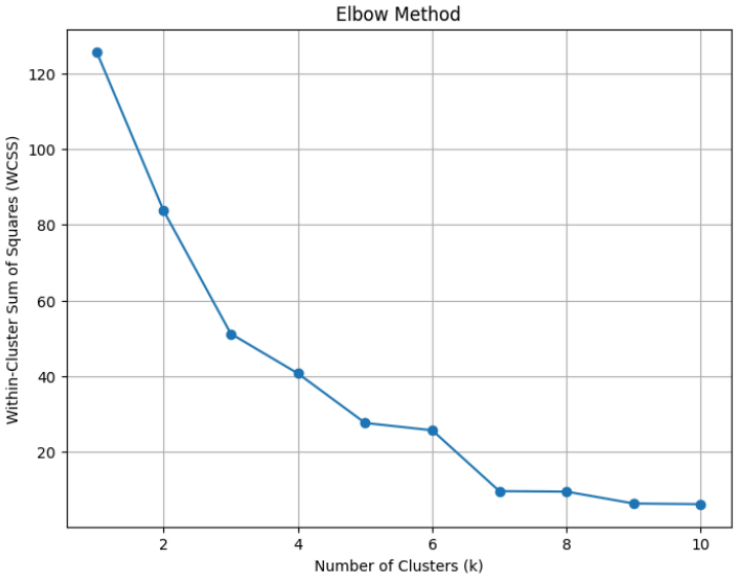
the algorithm identifies two main groups within the data. With **k = 3**, the dataset is further divided into three distinct clusters. One cluster captures the compact grouping of points, while the others cover more dispersed regions or outliers. As **k** increases to 4 and 5, the dataset is split into finer divisions, leading to a further reduction in WCSS. However, the decrease in WCSS becomes less significant, indicating diminishing returns with additional clusters.

The "elbow" is typically considered the optimal number of clusters since it balances model complexity and the quality of clustering.

From the observations, the elbow point likely appears around **k=3** or **k=4**.

Trade-Off Between Model Complexity and Overfitting

- Increasing the number of clusters (k) increases model complexity, potentially leading to overfitting, where clusters are too fine-grained and lose generalization.
- On the other hand, too few clusters may oversimplify the dataset's structure, failing to capture meaningful patterns.



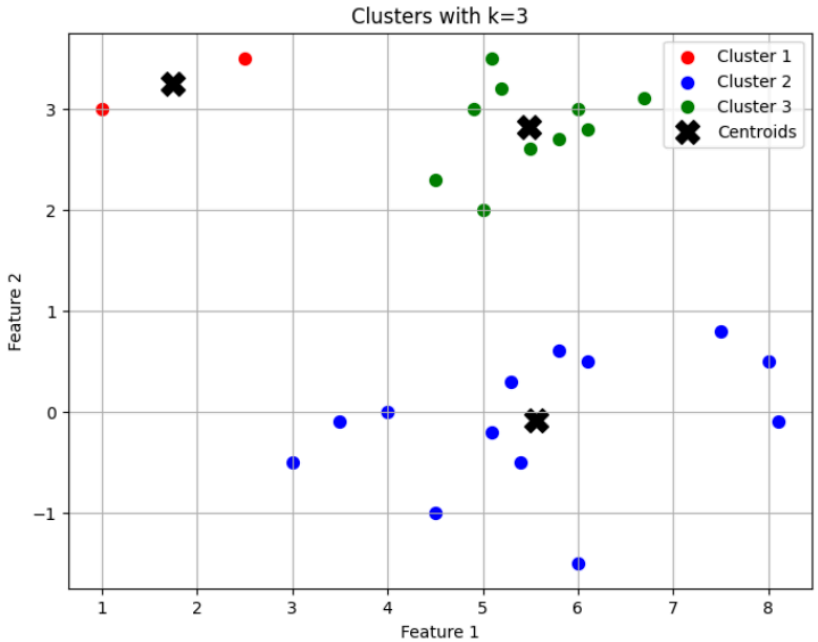
Applying K-means clustering on the optimal number of clusters through Random Initialization. In this I observed that Convergence reached at iteration 5.

```
Cluster 1: [[2.5 3.5]
 [1.  3. ]]
  Cluster 2: [[ 8.   0.5]
 [ 7.5  0.8]
 [ 8.1 -0.1]
 [ 4.5 -1. ]
 [ 3.  -0.5]
```

```
[ 5.1 -0.2]
[ 6.  -1.5]
[ 3.5 -0.1]
[ 4.   0. ]
[ 6.1  0.5]
[ 5.4 -0.5]
[ 5.3  0.3]
[ 5.8  0.6]]
 Cluster 3: [[5.1 3.5]
[4.9 3. ]
[5.8 2.7]
[6.  3. ]
[6.7 3.1]
[4.5 2.3]
[6.1 2.8]
[5.2 3.2]
[5.5 2.6]
[5.  2. ]]
```

Final Clusters

```
 Updated Centroids: [[ 1.75       3.25      ]
[ 5.56153846 -0.09230769]
[ 5.48       2.82      ]]
```



Silhouette Score: 0.4945359834656429

As expected, the above score is greater than when the value of k = 2. Because of the reason mentioned Above in part D only.

SECTION-C

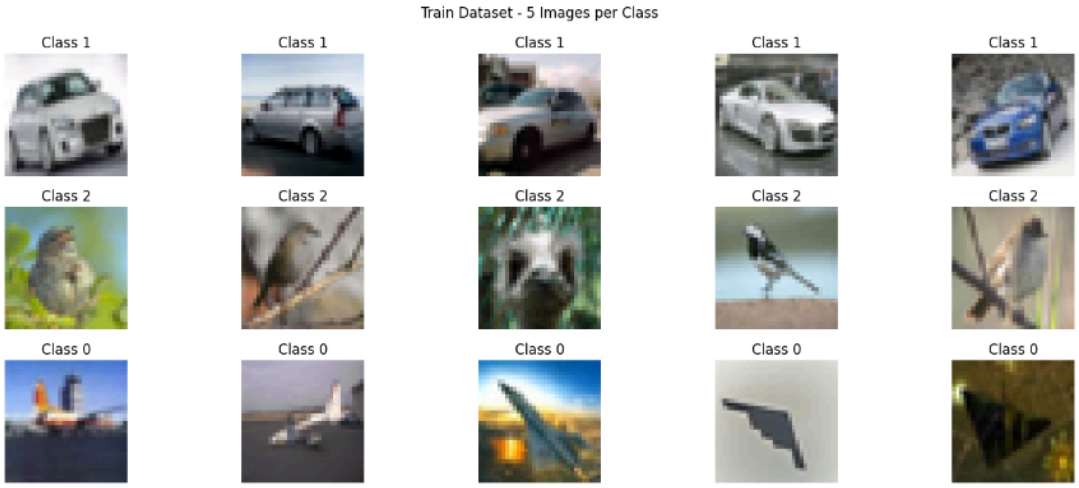
PART-1
Dataset Downloaded successfully and created custom class.

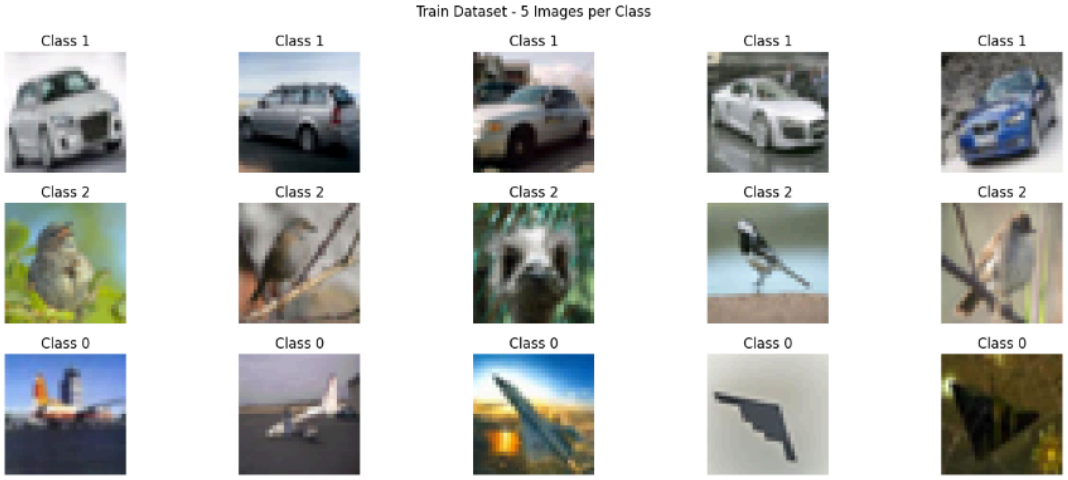
PART-2

Class Distribution:

```
Train class distribution: Counter({2: 4000, 1: 4000, 0: 4000})
Val class distribution: Counter({1: 1000, 0: 1000, 2: 1000})
Test class distribution: Counter({0: 1000, 1: 1000, 2: 1000})
```

5 Sample images from the three classes for both the train set and validation set.





PART-3

Created CNN architecture with following configuration:

- **Conv1:** Kernel size 5x5, 16 channels, stride 1, padding 1.
- **Conv2:** Kernel size 3x3, 32 channels, stride 1, padding 0.
- Max-pooling layers followed each convolutional layer, with strides of 2 and 3, respectively.
- Fully connected layers: [Flatten → 16 neurons → Output layer (classification head)]
- Activation: ReLU after each layer, except the output layer.

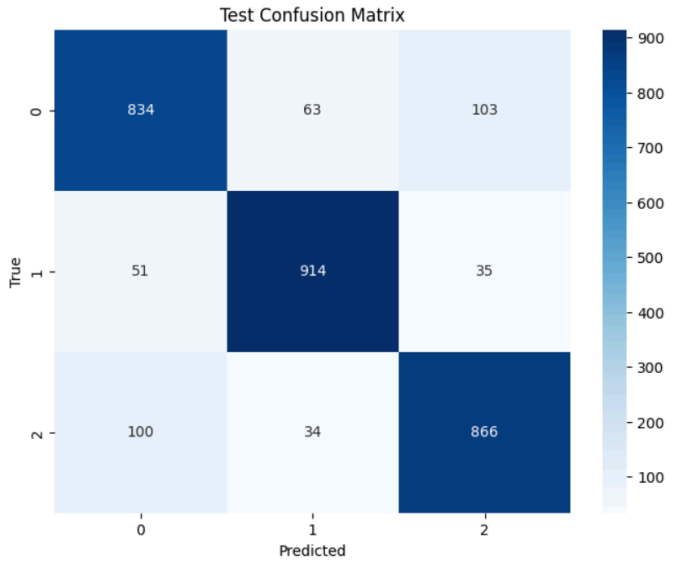
PART-4

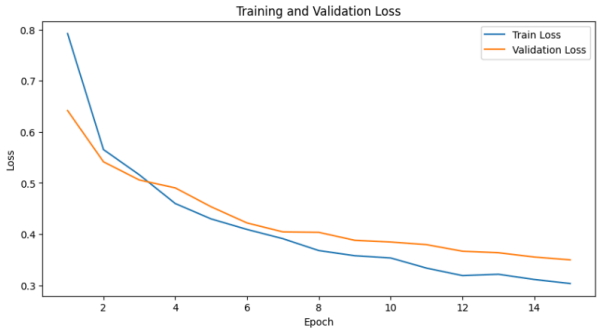
Trained the CNN model using the cross-entropy loss function and the Adam optimizer for 15 epochs.
Training loss steadily decreased across epochs, indicating effective learning. Training loss steadily decreased across epochs, indicating effective learning. The final training accuracy reached **88.23%**. The convergence of the loss and improvement in validation accuracy demonstrate that the CNN effectively learned patterns from the training data. No overfitting as such.

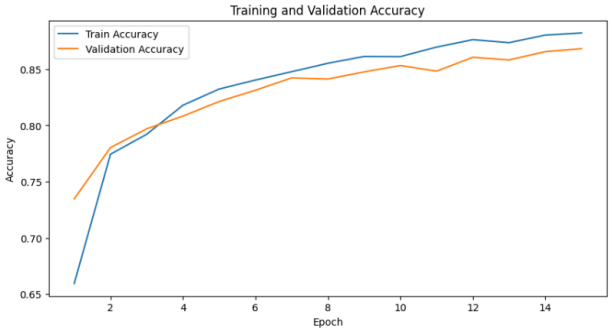
```
Epoch 1/15, Train Loss: 0.7922, Train Acc: 0.6594, Val Loss: 0.6417, Val Acc: 0.7347
Epoch 2/15, Train Loss: 0.5654, Train Acc: 0.7744, Val Loss: 0.5415, Val Acc: 0.7803
Epoch 3/15, Train Loss: 0.5160, Train Acc: 0.7922, Val Loss: 0.5059, Val Acc: 0.7970
Epoch 4/15, Train Loss: 0.4599, Train Acc: 0.8180, Val Loss: 0.4905, Val Acc: 0.8083
Epoch 5/15, Train Loss: 0.4298, Train Acc: 0.8324, Val Loss: 0.4535, Val Acc: 0.8213
Epoch 6/15, Train Loss: 0.4092, Train Acc: 0.8404, Val Loss: 0.4219, Val Acc: 0.8313
Epoch 7/15, Train Loss: 0.3911, Train Acc: 0.8478, Val Loss: 0.4044, Val Acc: 0.8423
Epoch 8/15, Train Loss: 0.3678, Train Acc: 0.8553, Val Loss: 0.4035, Val Acc: 0.8413
Epoch 9/15, Train Loss: 0.3579, Train Acc: 0.8614, Val Loss: 0.3879, Val Acc: 0.8477
Epoch 10/15, Train Loss: 0.3534, Train Acc: 0.8612, Val Loss: 0.3847, Val Acc: 0.8533
Epoch 11/15, Train Loss: 0.3336, Train Acc: 0.8698, Val Loss: 0.3795, Val Acc: 0.8483
Epoch 12/15, Train Loss: 0.3189, Train Acc: 0.8763, Val Loss: 0.3666, Val Acc: 0.8607
Epoch 13/15, Train Loss: 0.3215, Train Acc: 0.8737, Val Loss: 0.3637, Val Acc: 0.8583
Epoch 14/15, Train Loss: 0.3113, Train Acc: 0.8804, Val Loss: 0.3553, Val Acc: 0.8657
Epoch 15/15, Train Loss: 0.3035, Train Acc: 0.8823, Val Loss: 0.3497, Val Acc: 0.8683
```

PART-5

Test Accuracy: 0.8713333333333333, Test F1-Score: 0.8711914071177965
The model's strong performance on the test set validates its ability to generalize. The confusion matrix reveals specific areas where the model struggled, particularly between similar classes or images with ambiguous features







PART-6

Train an MLP with 2 fully connected layers for 15 epochs and log metrics.

Despite achieving nearly perfect training accuracy, the validation accuracy was substantially lower, revealing the model's inability to generalize as well as the CNN. This highlights the limitations of MLPs for image data, which lack the spatial feature extraction capabilities inherent to CNNs.

```
Epoch [1/15] - Train Loss: 0.5177, Train Acc: 0.7984 - Val Loss: 0.5829, Val Acc: 0.7737
Epoch [2/15] - Train Loss: 0.4750, Train Acc: 0.8157 - Val Loss: 0.6076, Val Acc: 0.7687
Epoch [3/15] - Train Loss: 0.4482, Train Acc: 0.8274 - Val Loss: 0.5795, Val Acc: 0.7820
Epoch [4/15] - Train Loss: 0.4117, Train Acc: 0.8445 - Val Loss: 0.5843, Val Acc: 0.7750
Epoch [5/15] - Train Loss: 0.3795, Train Acc: 0.8547 - Val Loss: 0.5829, Val Acc: 0.7753
Epoch [6/15] - Train Loss: 0.3566, Train Acc: 0.8653 - Val Loss: 0.5877, Val Acc: 0.7877
Epoch [7/15] - Train Loss: 0.3270, Train Acc: 0.8807 - Val Loss: 0.5939, Val Acc: 0.7790
Epoch [8/15] - Train Loss: 0.3123, Train Acc: 0.8842 - Val Loss: 0.6131, Val Acc: 0.7843
Epoch [9/15] - Train Loss: 0.2923, Train Acc: 0.8931 - Val Loss: 0.6270, Val Acc: 0.7870
Epoch [10/15] - Train Loss: 0.2693, Train Acc: 0.9028 - Val Loss: 0.6601, Val Acc: 0.7640
Epoch [11/15] - Train Loss: 0.2565, Train Acc: 0.9073 - Val Loss: 0.6287, Val Acc: 0.7893
Epoch [12/15] - Train Loss: 0.2304, Train Acc: 0.9225 - Val Loss: 0.6379, Val Acc: 0.7900
Epoch [13/15] - Train Loss: 0.2213, Train Acc: 0.9208 - Val Loss: 0.6728, Val Acc: 0.7753
Epoch [14/15] - Train Loss: 0.2006, Train Acc: 0.9303 - Val Loss: 0.6897, Val Acc: 0.7830
Epoch [15/15] - Train Loss: 0.1831, Train Acc: 0.9393 - Val Loss: 0.7016, Val Acc: 0.7773
```

PART-7

Test Accuracy (MLP): 0.7963
F1-Score (MLP): 0.7970

CNN vs. MLP: The CNN significantly outperformed the MLP on both test accuracy (~87% vs. ~79%) and F1-score. The CNN's ability to capture spatial features explains its superior performance on image data.
Training and Validation Trends: The MLP showed pronounced overfitting, while the CNN maintained a better balance between training and validation metrics.

The results demonstrate the superiority of CNNs for image classification tasks, as they are specifically designed to capture hierarchical spatial features. MLPs, while simpler, struggle with the high-dimensional nature of raw image data.