# Assignment 3 - ML

*Rishi Pendyala - 2022403*

## SECTION A

ML Assignment 4

SECTION-A.

1(a) CNN $\rightarrow$ Forward pass

Image $\rightarrow$ $M \times N$

P channels

$K \times K$ kernel

(a) Stride = 1, Padding = 0

$$\boxed{(M-K+1) \times (N-K+1)}$$

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

$K^2 + K(K-1)$

$(K-1)$ additions, $K$ mul

(b) Elementary Operations

Additions: $\boxed{P \cdot K^2 - 1}$ $(\because (K^2-1)P + (P-1))$

Multiplications: $\boxed{K^2 \cdot P}$

$$\boxed{\text{Operations per pixel} = K^2 P + P \cdot K^2 - 1 = 2K^2 P - 1}$$

(c) Q kernels

↳ Operations per pixel $= 2K^2 P - 1$

Output matrix dim $\longrightarrow (M-K+1) \times (N-K+1)$

Total op for a single kernel

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

Total op for Q kernels

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

$$\boxed{O(Q \cdot K^2 PMN)}$$

(b) Assignment:

- ~~the~~ Each datapoint is assigned to the cluster whose centroid is closest to the datapoint
- This is done by minimizing distance.

$$\arg\min_{j} \| x_i - \mu_j \|^2$$

- The dataset is partitioned into k clusters.

Update step

- Recompute the centroids of each cluster based on the mean of the datapoints (after including the newly assigned points) in the cluster.
- This is done by the formula (where $|C_j|$ is no. of points in cluster $C_j$)

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

These steps are repeated till convergence.

Elbow method is used to find the optimal no. of centroids

↳ Find a balance between minimizing within cluster variance and avoid overfitting (by ↑ clusters, we fit the data too well)

steps: * Run k-means for k = 1, 2, 3. ...

* Compute WCSS for each k

$$WCSS = \sum_{j=1}^{k} \sum_{x_i \in C_j} \| x_i - \mu_j \|^2$$

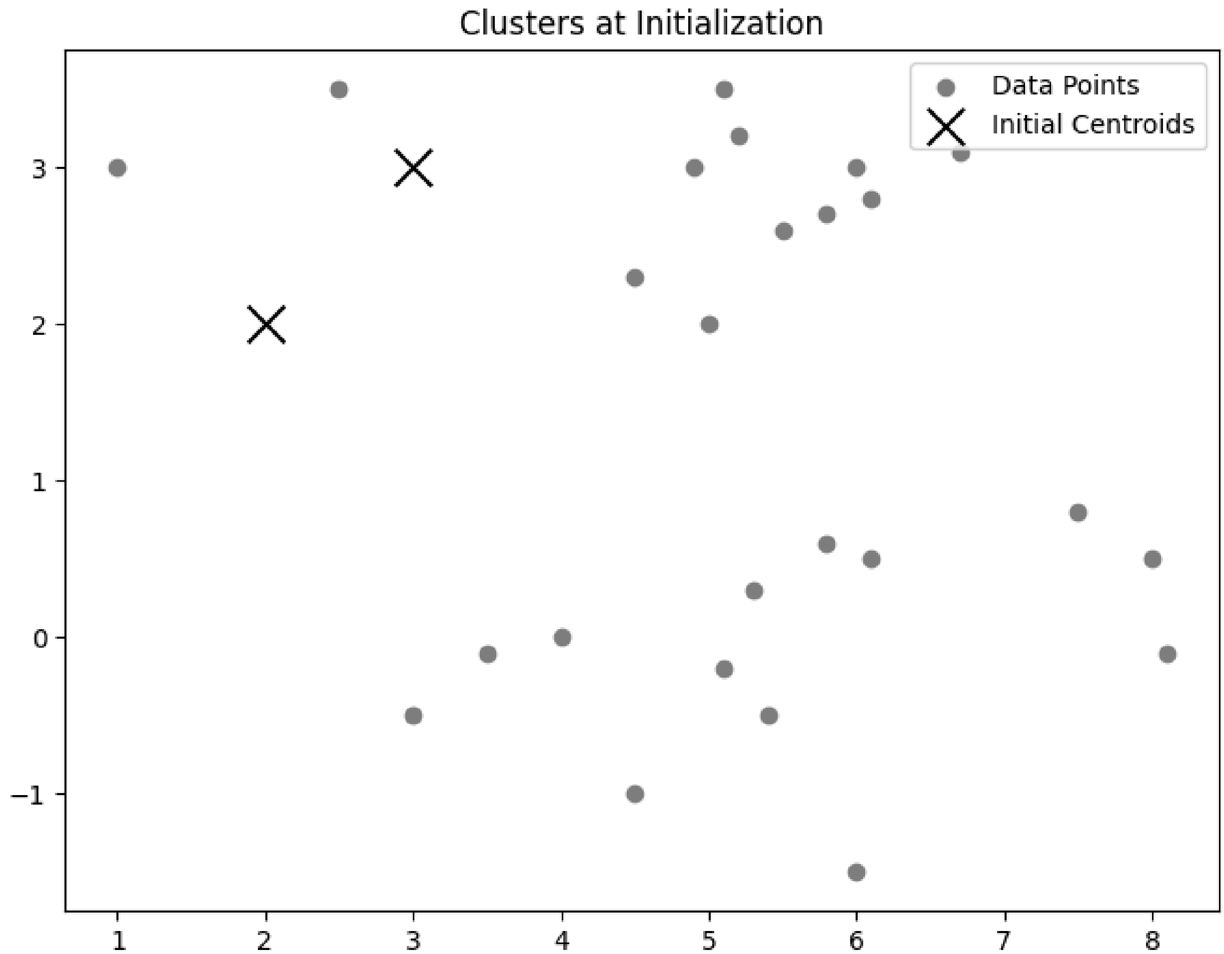
* Plot k vs. WCSS
* Identify the 'elbow point' → rate of decrease of WCSS decreases significantly.

It looks like an elbow.

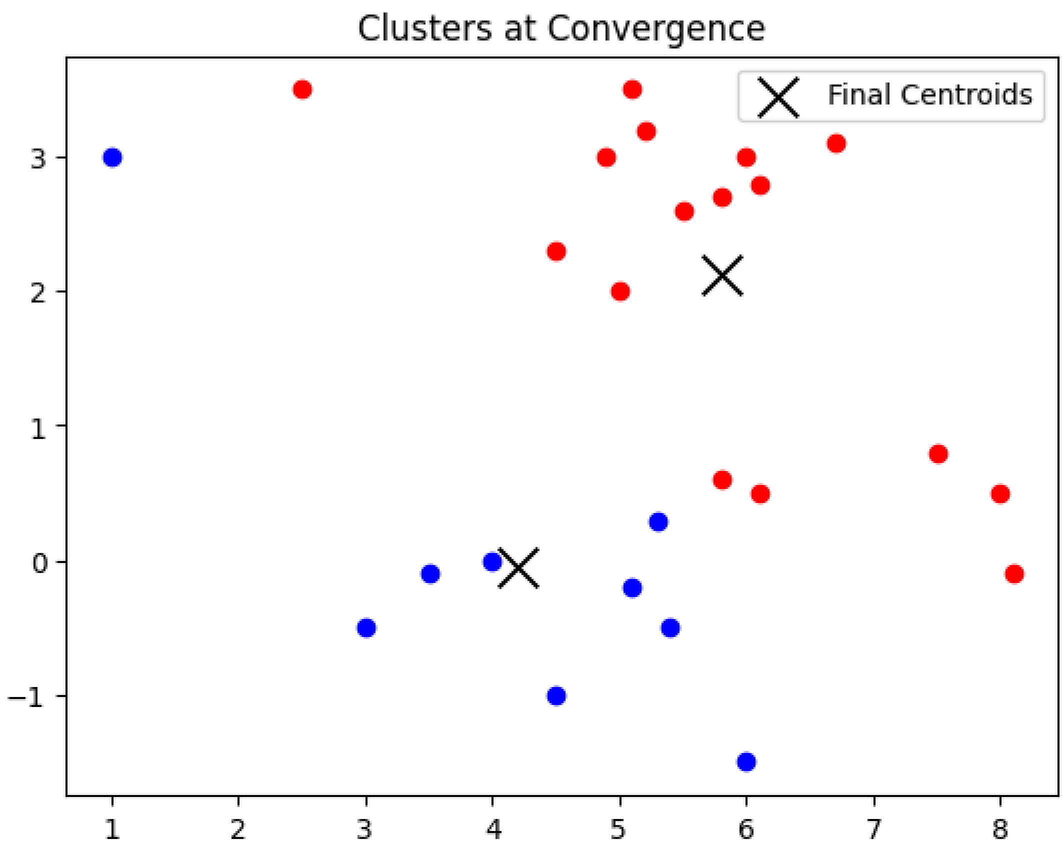
## SECTION B

```python
def kmeans(X, init_centroids, k=2, max_iter=100, threshold=1e-4):
    centroids = np.array(init_centroids)
    for iter in range(max_iter):
        # assignment
        clusters = [[] for i in range(k)]
        for point in X:
            distances = [euclidean_distance(point, centroid) for centroid in centroids]
            cluster_idx = np.argmin(distances)
            clusters[cluster_idx].append(point)

        # update
        new_centroids = []
        for cluster in clusters:
            if cluster:
                new_centroids.append(np.mean(cluster, axis=0))
            else:
                new_centroids.append(np.random.uniform(np.min(X, axis=0), np.max(X, axis=0)))
        new_centroids = np.array(new_centroids)

        # converge
        if np.all(np.abs(new_centroids - centroids) < threshold):
            break

        centroids = new_centroids
```

Initialization Centroids: [array([3., 3.]), array([2., 2.])]




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

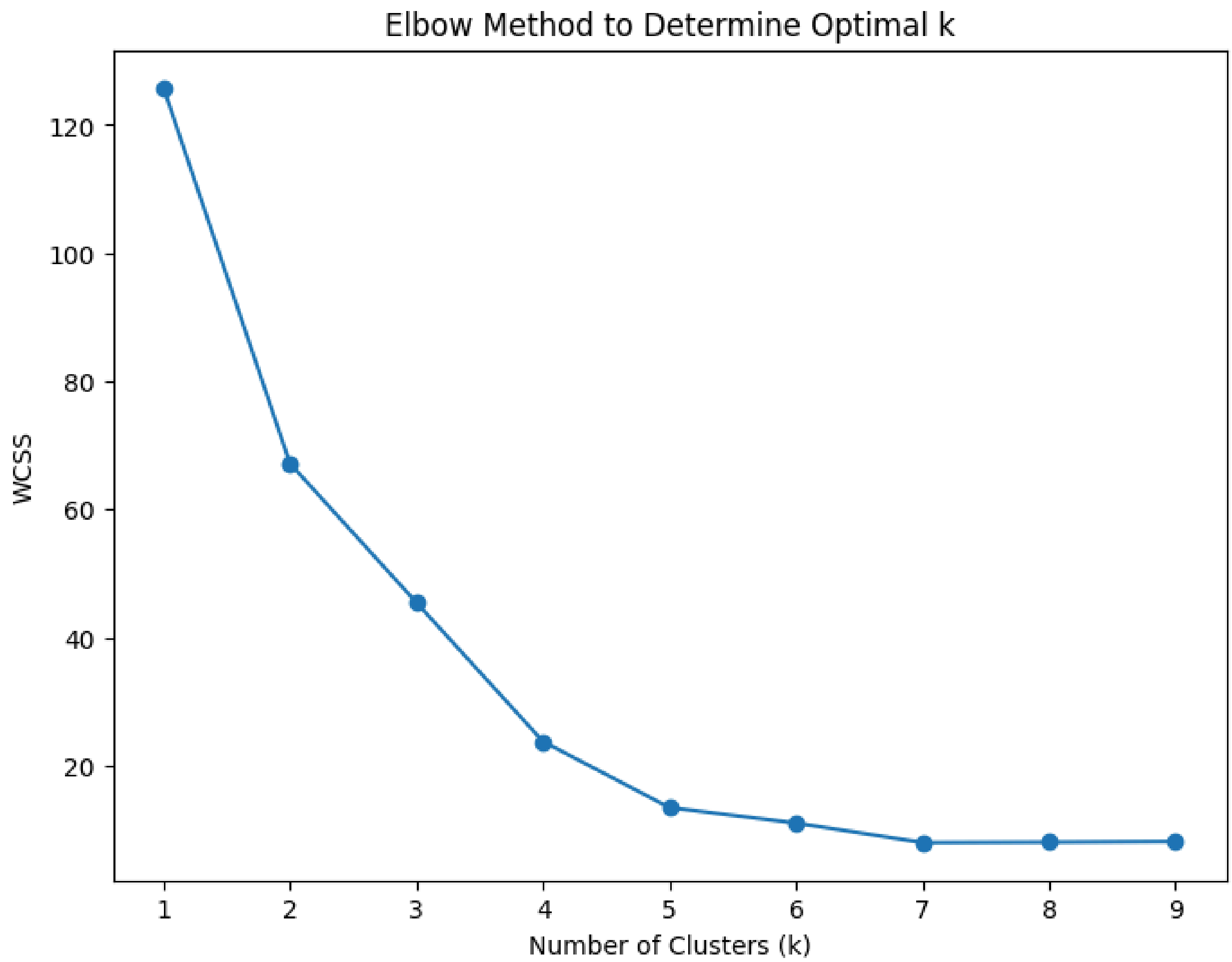



Initialization Centroids: Larger shift in centroid positions - Centroids moved significantly - Indicating suboptimal initial placement - Slower convergence - Higher likelihood of poor local optima

Random Initialization: Smaller shift in centroid positions - Closer to final cluster positions - Faster convergence - Potentially better-defined clusters - Less dramatic movement of centroids

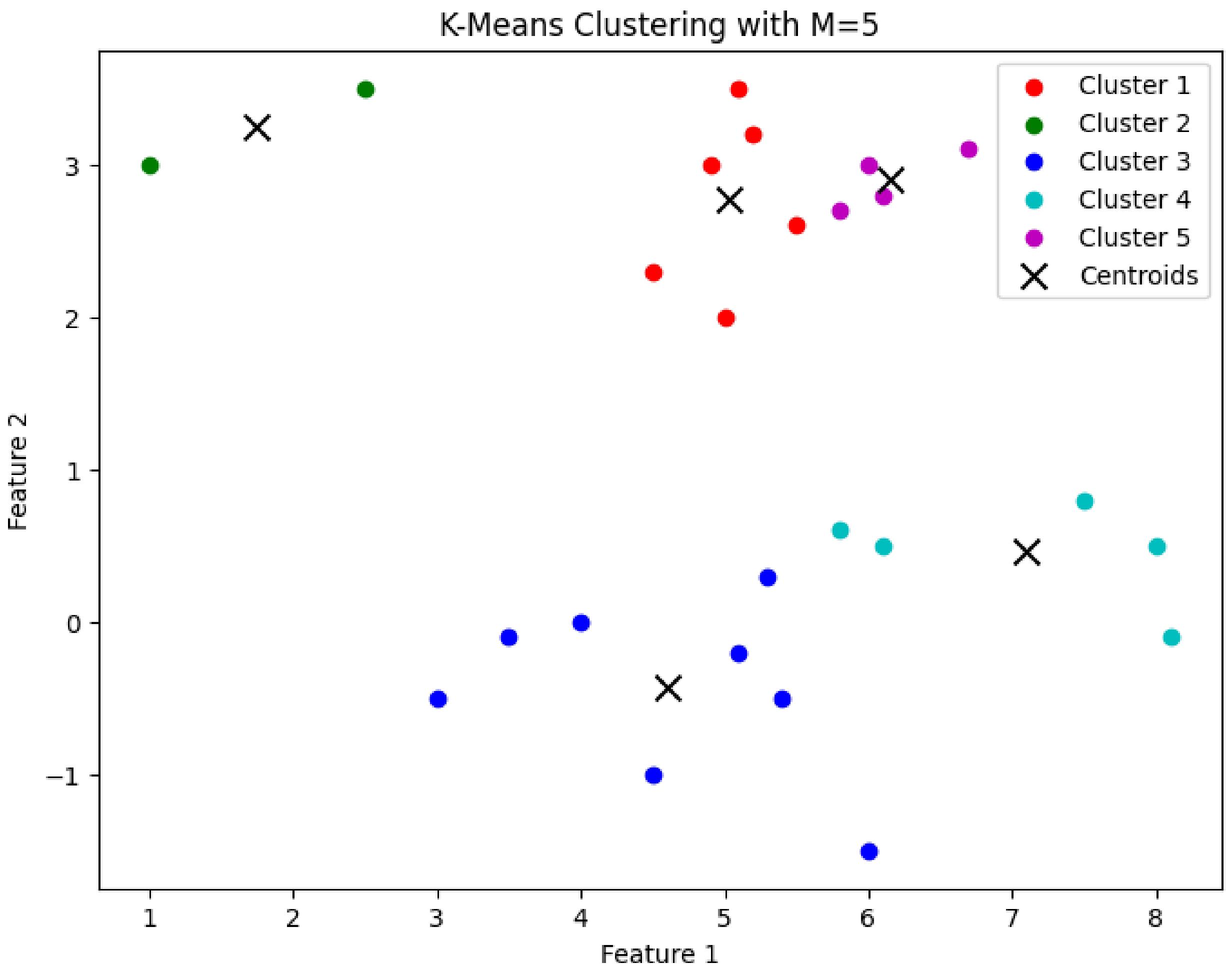
Clustering Performance: Random initialization likely leads to better clustering - Faster convergence - Better-defined clusters

Conclusion: Random initialization preferred for k-means - Helps avoid poor local minima - More stable convergence - Better clustering performance



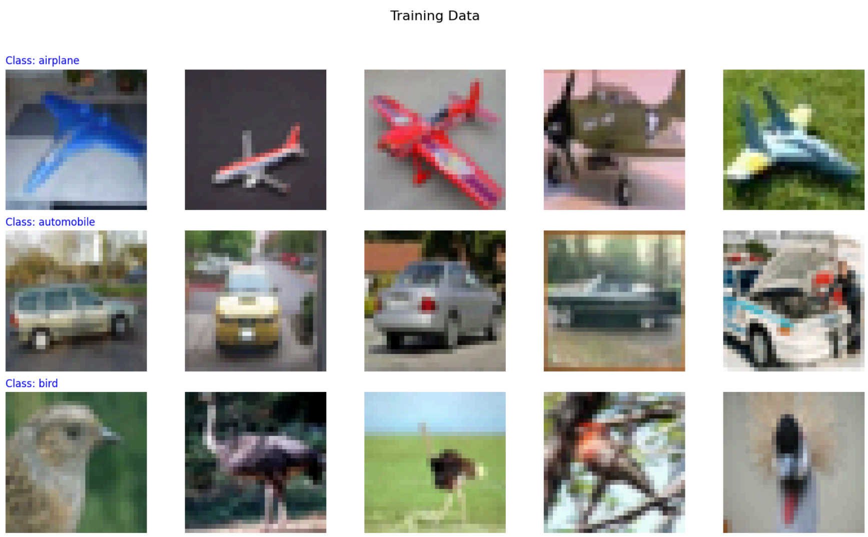

The elbow method suggests that 5 clusters are best

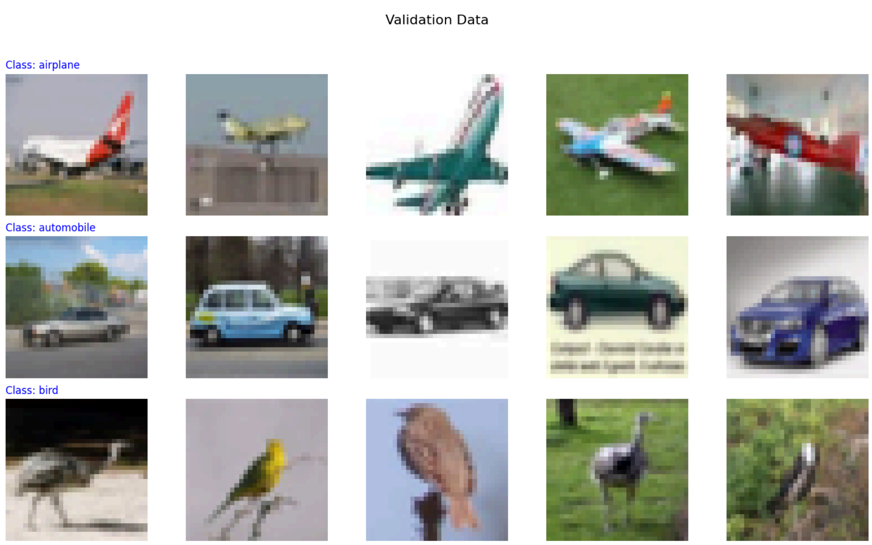
Running k means with 5 clusters


K-Means Clustering with M=5
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Cluster 5
Centroids
Feature 1
Feature 2
1
2
3
4
5
6
7
8
−1
0
1
2
3

## SECTION C

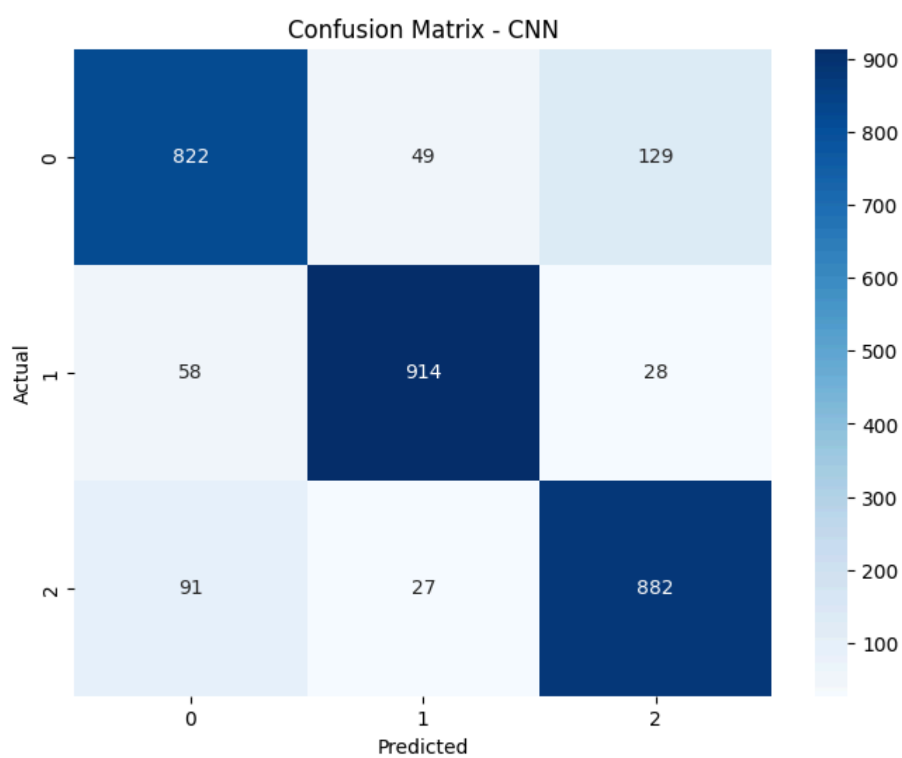
Loaded the dataset



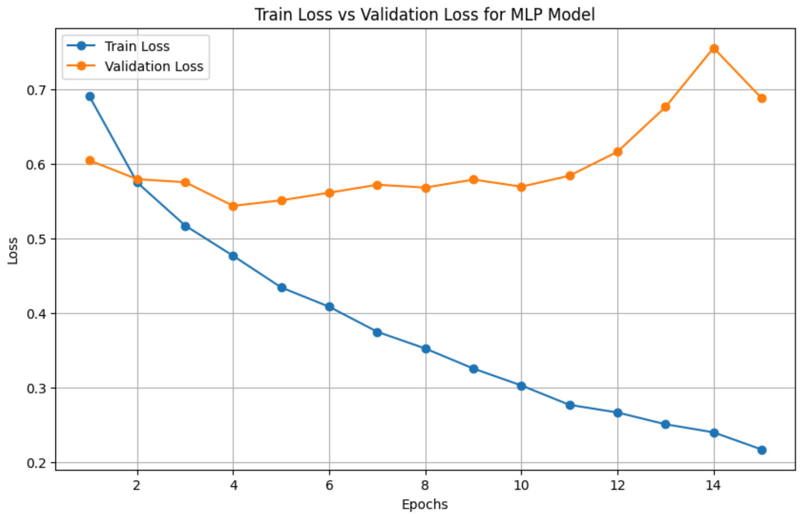



Using CNN

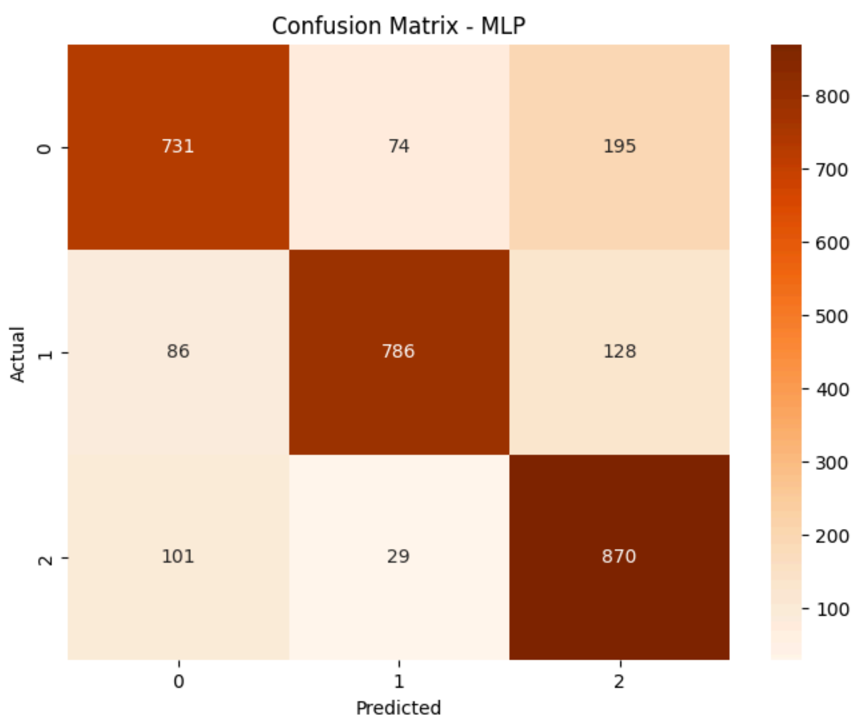

Test Loss: 0.3392, Test Accuracy: 86.87%, Test F1 Score: 0.8678



Using MLP




MLP Test Loss: 0.5881, Test Accuracy: 81.20%, Test F1 Score: 0.8120



Comparison of CNN and MLP Models:

CNN Test Accuracy: 87.27%, Test F1 Score: 0.8726

MLP Test Accuracy: 79.57%, Test F1 Score: 0.7960

Inferences:

- The CNN performed better than MLP model based on accuracy and F1 Score
- This is probably because CNN architecture is specifically designed to handle images.