ML Assignment 4

Parth Sandeep Rastogi[1]

[1]Indraprastha Institute of Information Technology

November 28, 2024

1 Section A) Theory

1.1 Q1) CNN

1.1.1 (1a) Dimensions of the Resulting Feature Map

For a convolutional layer with:

- Input dimensions: $M \times N$ with P channels,
- Kernel size: $K \times K$,
- Stride: 1,
- No padding,

the resulting feature map dimensions are calculated as follows:

$$\text{Output height} = M - K + 1, \quad \text{Output width} = N - K + 1$$

Thus, the dimensions of the resulting feature map are:

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

1.1.2 (1b) Number of Elementary Operations Required per Output Pixel

To compute a single output pixel in the resulting feature map:

- Each kernel has dimensions $K \times K$, and with P input channels, the kernel spans $K \times K \times P$.
- The computation involves:
  - $K \times K \times P$ multiplications (one for each element in the kernel and its corresponding input),
  - $K \times K \times P - 1$ additions (to sum the products).

Thus, the total number of operations for a single output pixel is:

$$\text{Number of operations} = K^2 \cdot P \text{ (multiplications) } + (K^2 \cdot P - 1) \text{ (additions)}$$
$$= 2 \cdot K^2 \cdot P - 1$$

1.1.3 (1c) Computational Time Complexity for Q Kernels

Now consider:

- Q kernels, each of size $K \times K$,
- Input image dimensions: $M \times N$,
- Feature map dimensions: $(M - K + 1) \times (N - K + 1)$.

For each kernel:

- The number of output pixels in the feature map is:

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

- Each pixel requires $2 \cdot K^2 \cdot P - 1$ operations.
- Total operations per kernel:

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

For Q kernels, the total number of operations is:

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

In Big-O notation, ignoring constants and lower-order terms:

$$O(Q \cdot M \cdot N \cdot K^2 \cdot P)$$

1.2 K-Means Algorithm: Assignment and Update Steps

The K-Means algorithm is a popular technique in unsupervised machine learning used to group data into K distinct clusters. It works iteratively to refine cluster assignments and centroids. The algorithm alternates between two main steps: the Assignment Step and the Update Step. Additionally, selecting the right number of clusters K is crucial for effective clustering. This section discusses how these steps work, as well as methods for determining the optimal number of clusters.

1.2.1 Assignment Step

In the Assignment Step, each data point is assigned to the nearest cluster centroid. The nearest centroid is typically chosen using the Euclidean distance, which measures how far a data point is from each centroid. For each point, we calculate the distance to each centroid and assign it to the centroid that is closest.

Mathematically, for each data point x_i, we find the cluster centroid that minimizes the Euclidean distance. This means that for a dataset $\mathcal{X}$, which consists of N data points, and a set of K centroids $\mathcal{C}$, we assign each point x_i to the nearest centroid. The assignment rule looks like this:

$$c_i = \arg\min_k \|x_i - \mu_k\|$$

Where c_i represents the index of the assigned centroid for point x_i, and $\|\cdot\|$ is the Euclidean distance.

1.2.2 Update Step

After assigning all data points to their closest centroids, the Update Step recalculates the centroids based on the points assigned to each cluster. The centroid of each cluster is updated by taking the mean of all the points in that cluster. For each cluster k, the updated centroid μ_k is computed as:

$$\mu_k = \frac{\sum_{x_i \in C_k} x_i}{|C_k|}$$

1.2.3 Determining the Optimal Number of Clusters

Choosing the right number of clusters K is critical for good clustering results. A common method for determining K is the Elbow Method. This method involves running the K-Means algorithm with different values of K and calculating the Within-Cluster Sum of Squares (WCSS) for each value of K. The WCSS measures how compact the clusters are by summing up the squared distances between each point and its assigned centroid.

The WCSS for a given K can be written as:

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

As K increases, the WCSS decreases because the clusters become smaller and more compact. The goal is to plot WCSS against different values of K and look for an elbow in the graph, where the rate of decrease slows down. The value of K at this elbow point is typically the optimal number of clusters.

1.2.4 Initialization of Centroids and Local Minima

The K-Means algorithm is sensitive to the initial placement of centroids. If the initial centroids are chosen randomly, the algorithm may converge to a local minimum, which may not be the best solution. This happens because the objective function we want to minimize is:

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Since this function is non-convex, the algorithm can end up in different local minima depending on the starting centroids. This is why it's common to run the K-Means algorithm multiple times with different random initializations and choose the solution with the lowest objective value.

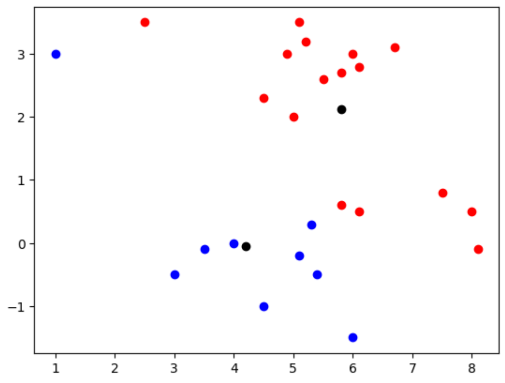
2 Section B) Scratch Implementation

2.1 Algorithm

Algorithm 1 K-Means Clustering with 2 Centroids

```
Require: Dataset $X$, initial centroids $c1$ and $c2$, max iterations $max\_iter = 100$, tolerance $\epsilon = 10^{-4}$
 1: Initialize $i \leftarrow 0$
 2: $c1\_current \leftarrow c1$, $c2\_current \leftarrow c2$
 3: $c1\_prev \leftarrow c1$, $c2\_prev \leftarrow c2$
 4: while $i < max\_iter$ do
 5:    Initialize empty point sets $pt1 \leftarrow \emptyset$, $pt2 \leftarrow \emptyset$
 6:    for each point $x \in X$ do
 7:       Compute distances:
 8:          $d1 \leftarrow \text{distance}(c1\_current, x)$
 9:          $d2 \leftarrow \text{distance}(c2\_current, x)$
10:       if $d1 < d2$ then
11:          Assign $x$ to cluster 1: $pt1 \leftarrow pt1 \cup \{x\}$
12:       else
13:          Assign $x$ to cluster 2: $pt2 \leftarrow pt2 \cup \{x\}$
14:       end if
15:    end for
16:    Update previous centroids:
17:       $c1\_prev \leftarrow c1\_current$, $c2\_prev \leftarrow c2\_current$
18:    Recalculate centroids:
19:       $c1\_current \leftarrow \text{mean}(pt1)$
20:       $c2\_current \leftarrow \text{mean}(pt2)$
21:    Increment iteration: $i \leftarrow i + 1$
22:    Print current state:
23:    if $c1\_current = c1\_prev$ and $c2\_current = c2\_prev$ then
24:       Break
25:    end if
26:    if $\text{distance}(c1\_current, c1\_prev) < \epsilon$ and $\text{distance}(c2\_current, c2\_prev) < \epsilon$ then
27:       Break
28:    end if
29: end while
```

2.2 Plot For Random and Preset Initialization


(a) Given initialization

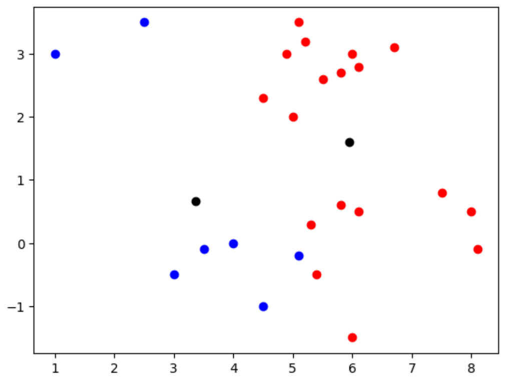

(b) Random initialization

Figure 1: Cluster plots

2.3 Final Centroid from both random and Preset

Initialization	**Centroid 1**	**Centroid 2**	**Convergence Steps**
Preset	$5.8, 2.125$	$4.2, -0.0556$	3
Random	$5.9444, 1.6$	$3.3714, 0.6714$	4

Table I: Final Centroids and Number of Convergence Steps

2.4 Elbow plot

From Given Curve we find that Around 5, 6 elbow occurs

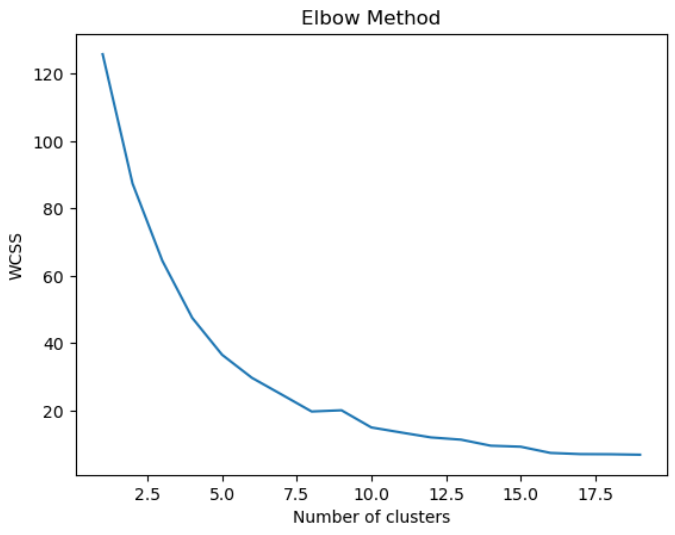


Figure 2: Elbow Curve

2.5 Plot after setting k = 5

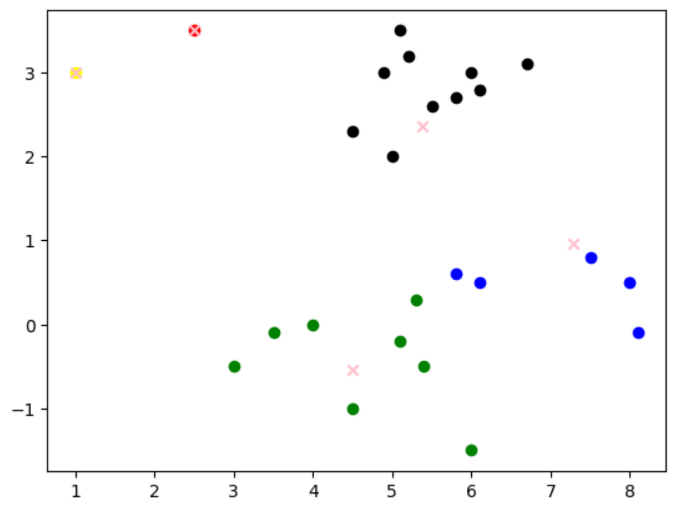


Figure 3: clusters where pink cross are final centroids

3 Section C) library Implementation

3.1 Data Preparation

For this section, the CIFAR-10 dataset, consisting of 60,000 32x32 RGB images of 10 classes, was used. Three classes were selected (bird, car and plane) , resulting in a curated dataset with 18,000 images (15,000 for training and 3,000 for testing). A custom PyTorch Dataset class was created to handle loading, splitting. The dataset was split as follows:

- Training: 12,000 images (0.8 of 15,000 images)
- Validation: 3,000 images (0.2 of 15,000 images)
- Testing: 3,000 images

Custom PyTorch dataloaders were implemented to streamline batch loading.

3.1.1 Visualizations

Five images per class were visualized from both the training and validation datasets to ensure correct preprocessing and labeling. Sample visualizations are provided in Figures 4 and 5.

3.2 CNN Implementation

The CNN model was constructed using two convolutional layers and max-pooling layers as follows:

- **Convolutional Layer 1:** Kernel size 5x5, 16 channels, stride 1, padding 1.
- **Max Pooling Layer 1:** Kernel size 3x3, stride 2.
- **Convolutional Layer 2:** Kernel size 3x3, 32 channels, stride 1, padding 0.
- **Max Pooling Layer 2:** Kernel size 3x3, stride 3.
- **Fully Connected Layers:** First layer with 16 neurons; second (classification head) layer with 3 neurons.

ReLU activation was applied after each layer except the classification head.

The final test accuracy and F1-score are presented in Table **??**.

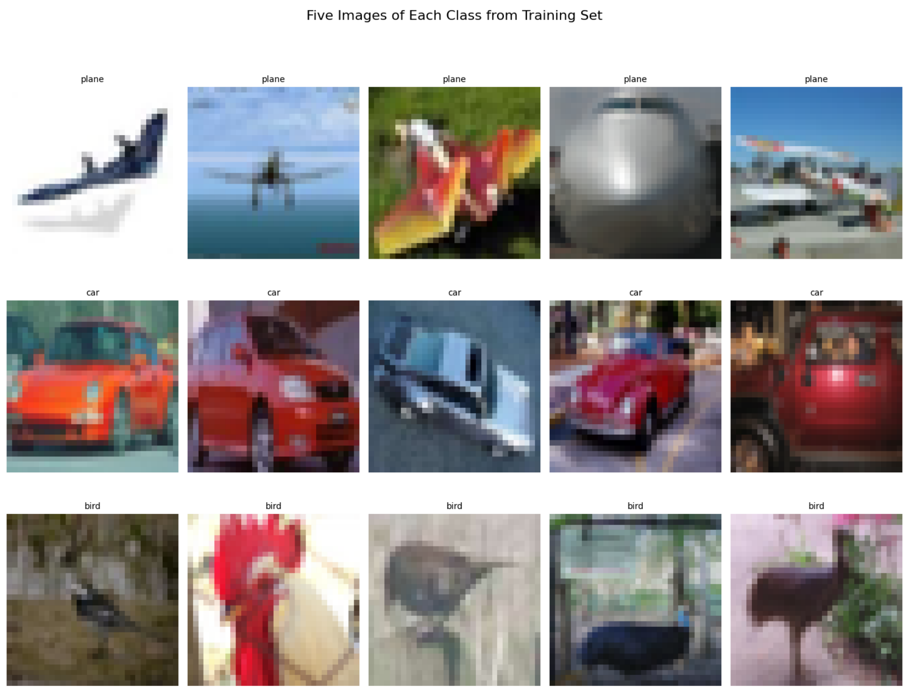


Figure 4: Sample images from the training dataset (5 per class).

3.3 MLP Implementation

The MLP model was implemented with two fully connected layers:

- **Fully Connected Layer 1:** 64 neurons with ReLU activation.
- **Fully Connected Layer 2:** Classification head with 3 neurons.

The model was trained for 15 epochs using the cross-entropy loss function and Adam optimizer. Loss and accuracy plots are shown in Figures

3.4 Performance Comparison

The performance of the CNN and MLP models was compared based on accuracy, F1-score

Metric	CNN Model	MLP Model
Accuracy	87%	75%
Macro F1-Score	0.88	0.75
Weighted F1- Score	0.88	0.75

Table II: Test accuracy and F1-score for CNN and MLP models.

Key Observations:

- CNN model outperformed the MLP in both accuracy and F1-score, benefiting from hierarchical feature extraction through convolutional layers.
- The confusion matrices revealed that the CNN model better distinguished between similar classes, while the MLP struggled with overlapping features we can see both are good in predicting car but the mlp gets comfused in the case of bird and plane and sometimes false identify it .

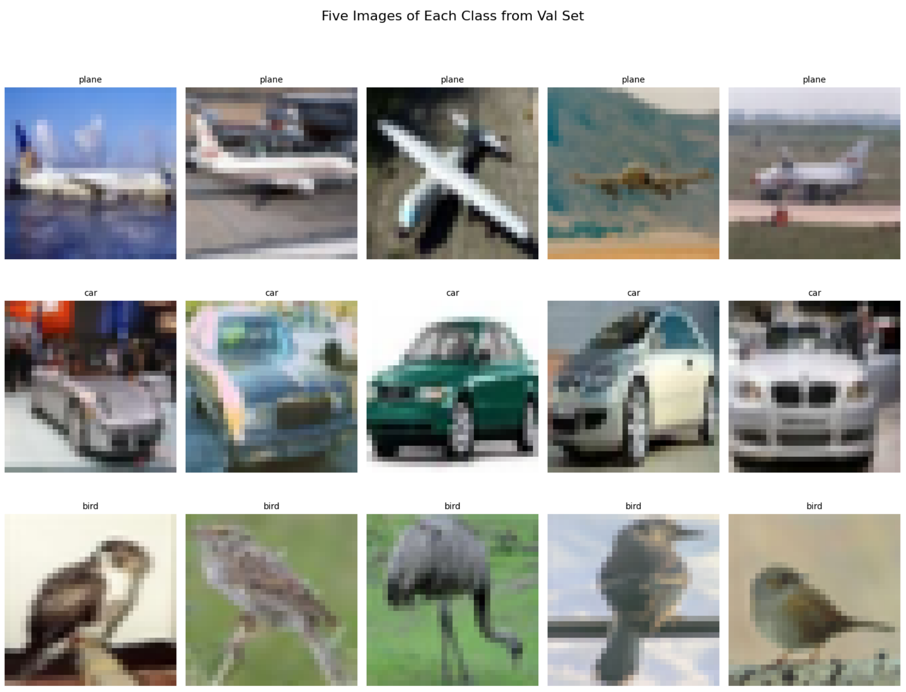


Figure 5: Sample images from the validation dataset (5 per class).

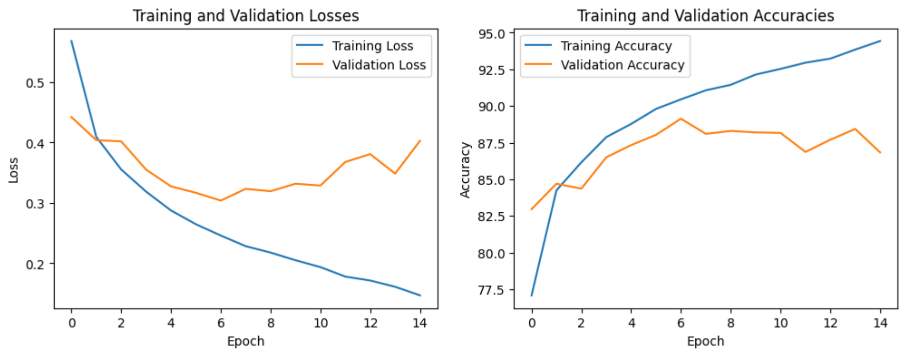


Figure 6: Training and validation loss and accuracy over 15 epochs for CNN.

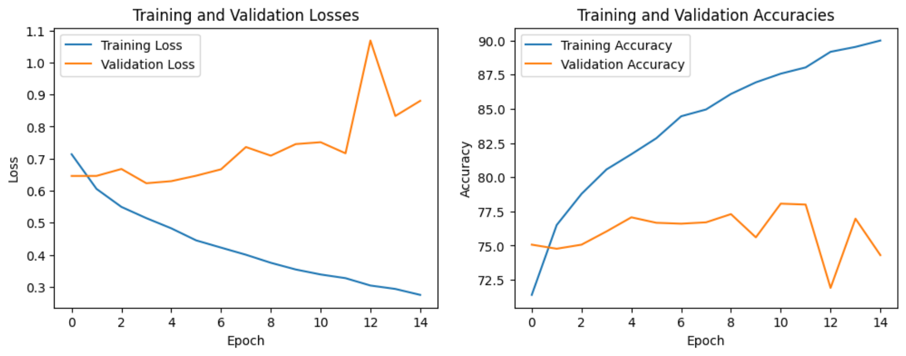
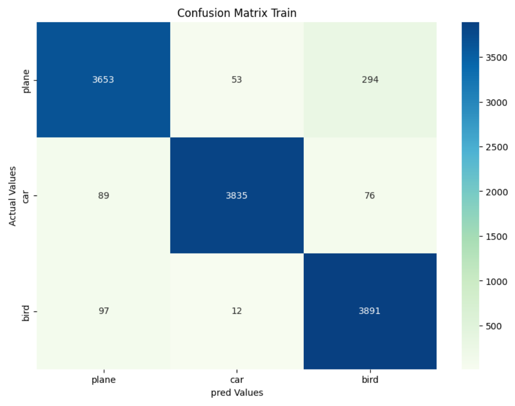
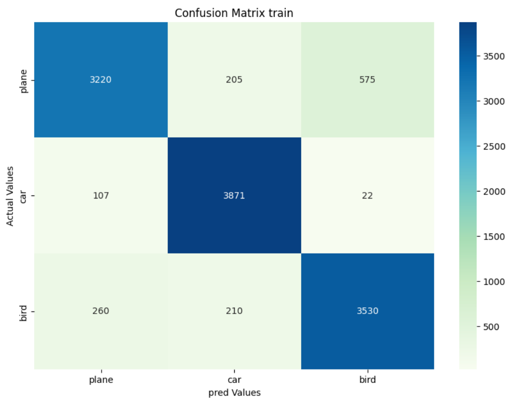


Figure 7: Training and validation loss and accuracy over 15 epochs for MLP.

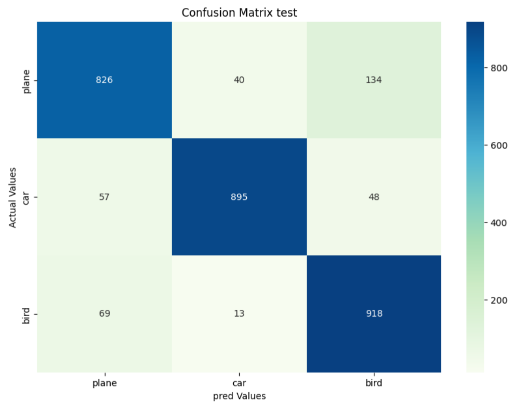


(a) Confusion matrix for CNN model (train).

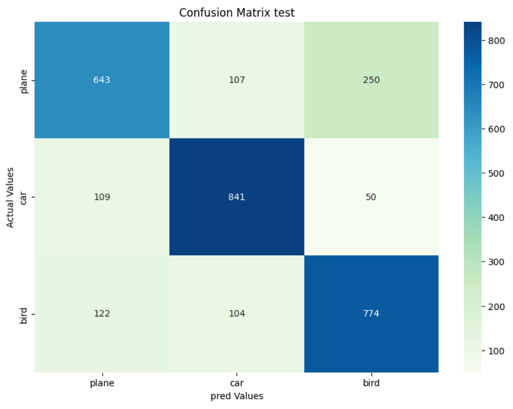


(b) Confusion matrix for MLP model (train).

Figure 8: Confusion matrices for CNN and MLP models on train dataset.

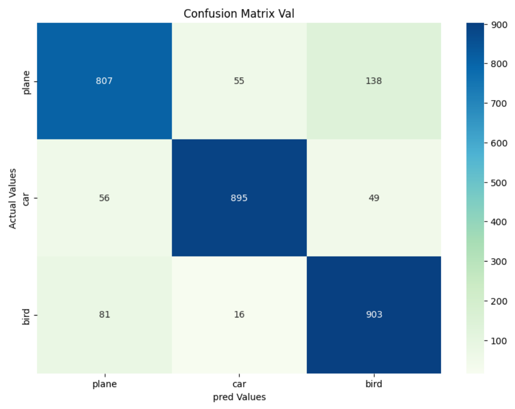


(a) Confusion matrix for CNN model (test).

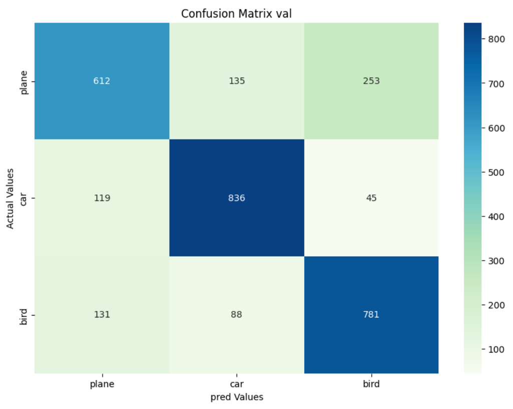


(b) Confusion matrix for MLP model (test).

Figure 9: Confusion matrices for CNN and MLP models on test dataset.



(a) Confusion matrix for CNN model (val).



(b) Confusion matrix for MLP model (val).

Figure 10: Confusion matrices for CNN and MLP models on validation dataset.