ML Assignment 4

Vikranth Udandarao
2022570

Section A

ML Assignment 4

1.a- a- Kernel size $= K \times K$
Input Image Dimensions $= M \times N$

Resulting feature Map dimensions $= H_{out} \times W_{out}$

$H_{out} = (M-K+1)$, $W_{out} = (N-K+1)$

b- The convolution operation involves -

-> multiply each pixel in $K \times K$ space with kernel weight.

-> Summing up over all channels

No. of elementary operations

-> for a single pixel is $K \times K \times P$.

c- Given Q kernels, Q features map are computed.

we know from

a- $(M-K+1) \times (N-K+1)$ = No. of pixels for one kernel.

b- Given Q kernels, -> $d = (M-K+1) \times (N-K+1)$.

Involved

For each pixel - $K \times K \times P$ operations are ~~iterated~~ involved

So total operations - $Q \times (M-K+1) \times (N-K+1) \times K \times K \times P$

If $\min(M,N) \sim K - O(QK^2P)$

If $\min(M,N) > K$

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

b - In the assignment step, we assign each point in the dataset to the nearest cluster centroid based on chosen distance metrics.

Eg - Euclidean distance, Manhattan, Mahalanobis distance.

Cluster assignment - argmin (dist (x_i, centroid))

However, in the update step, new centroids are calculated using the mean of the data points reassigned to each ~~other~~ cluster.

$$\text{centroid}_j = \frac{1}{\text{No. of pts in centroid}_j} \times \frac{\Sigma x_i}{x_i \in \text{centroid}_j}$$

And now the assignment step is repeated after the update step the assigns the dataset to the new cluster.

A- A ~~[illegible]~~ common method for determining the optimal number ~~[illegible]~~ centroids is the elbow method.

A- We plot the WCSS (Within Cluster Sum squares) is the number of clusters and we find the 'elbow' in the where the WCSS reduces significantly.

→ No, it is not confirmed that convergence will be reached. For this are more optimized methods such as kmeans.

Section A

(A) a)

Input image dimensions: $M \times N$
P channels

Kernel Size : $K \times K$
Stride : 1
Padding : No Padding

Output height : $M - K + 1$
Output width : $N - K + 1$

⇒ Kernel slides across the input image, without any zero padding.
⇒ The kernel reduces dimension of input image by $K - 1$, in both height and width

∴ Dimension of the resulting feature map is:

~~$[M-K+1] \times [N-K+1]$~~

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

Ans

WRT condition: $\min(M, N) >> K$:

we have: $(M-K+1) \simeq M$

$(N-K+1) \simeq N$.

Time complexity: $\boxed{O(Q \times M \times N \times P \times K^2)}$

Ans

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

$|C_j| \rightarrow$ number of data points in cluster C_j

∴ Centroids move ~~to~~ to the center of their respective clusters, this process repeats until convergence.

Optimal Number of Clusters:

"Elbow Method" helps determine optimal number of clusters (K).

Steps:

① Run K-means for different values of K $[K = 1, 2, 3, \ldots]$

② Compute WCSS i.e. "Within Cluster Sum of Squares" which measures variance within the cluster.

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

③ Plot ~~K v/s WCSS~~ "WCSS v/s K."

As K increases ~~K~~, in general, WCSS decreases because clusters becomes smaller

(b) # Assignment Step :

① For each data point, the K-Means Clustering algorithm computes distance between the data point and each of the centroids.

② The data point is assigned to the cluster whose centroid is closer to it.

③ The distance metrics used ~~is majorly~~ say → "Euclidean Distance."

For datapoint x_i & cluster C_j

$$\text{Cluster Assignment} = \underset{j}{\operatorname{argmin}} \|x_i - C_j\|$$

for the x_i

∴ Each of the datapoint is grouped to its closest centroid, forming "k" clusters.

These "k" clusters are either manually chosen or methods like Elbow method can be used.

Update Step :

① The centroids of the clusters are updated as the mean of all data points assigned to each cluster.

② The new centroid of a cluster is calculated as:

(b) With respect to a single output pixel in feature map:

(1) Kernel Operation:

Kernel size = $K \times K$ which spans K^2 positions in the input image

Each Kernel element performs:
→ ~~not~~ one multiplication with the corresponding input pixel
⇒ Total of $\boxed{K^2}$ multiplications

(2) Summation: The results of K^2 multiplication are ~~taken up~~ summed up to produce a single output pixel
⇒ $(K^2 - 1)$ additions

(3) Given P channels: If the input image has P channels, the Kernel will span all channels

∴ For P channels, the number of operations
Multiplication: $P \times K^2$
Addition: $P \times (K^2 - 1)$

Total: $(P \times K^2) + P \times (K^2 - 1)$
$= \boxed{P \times [2K^2 - 1]}$ Ans

(4) Elbow point is where the rate of decrease in WCSS slows down. This point is the optimal value of "K".

Minima:

Randomly assigning cluster centroids and global ~~Minima~~

NO; K-means does not guarantee convergence to global ~~max~~ minima because it is sensitive to initial centroids positions and uses a greedy approach.

* Based on the initialisation can ~~be~~ cause:
→ There can be convergence to local minima.
→ Clustering results to be poor.

– Mitigation steps:

Technique: K-means ++ initializations
it selects initial centroids more strategically to improve the likelihood of reaching the global minima.

Section B

(a)

1. Initialization:

The initial centroids were set as:

u1=(3.0,3.0), u2=(2.0,2.0)

These centroids serve as starting points for clustering.

2. Assignment:

Each data point was assigned to the nearest centroid using the Euclidean distance:

Distance=(x1−c1)2+(x2−c2)2 {Distance} = sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2}Distance=(x1−c1)2+(x2−c2)2

The algorithm iteratively recalculated the distances and assigned the data points to the clusters defined by the closest centroids.

3. Update:

After each assignment, the centroids were updated by computing the mean of the points in each cluster:

uj=1Nj∑i=1Njxi\{u}_j = \frac{1}{N_j} \sum_{i=1}^{N_j} \{x}_iuj=Nj1i=1∑Njxi

where NjN_jNj is the number of points in cluster jjj.

4. Convergence Check:

The algorithm terminated when the centroids' change was smaller than the threshold tol=10−4\text{tol} = 10^{-4}tol=10−4 or after 100 iterations.

(b)

Final Centroids:

After convergence, the centroids were:

u1=[5.8,2.125],u2=[4.2,−0.0556]{u}_1 = [5.8, 2.125], \quad {u}_2 = [4.2, -0.0556]u1=[5.8,2.125],u2=[4.2,−0.0556]

Cluster Assignments:

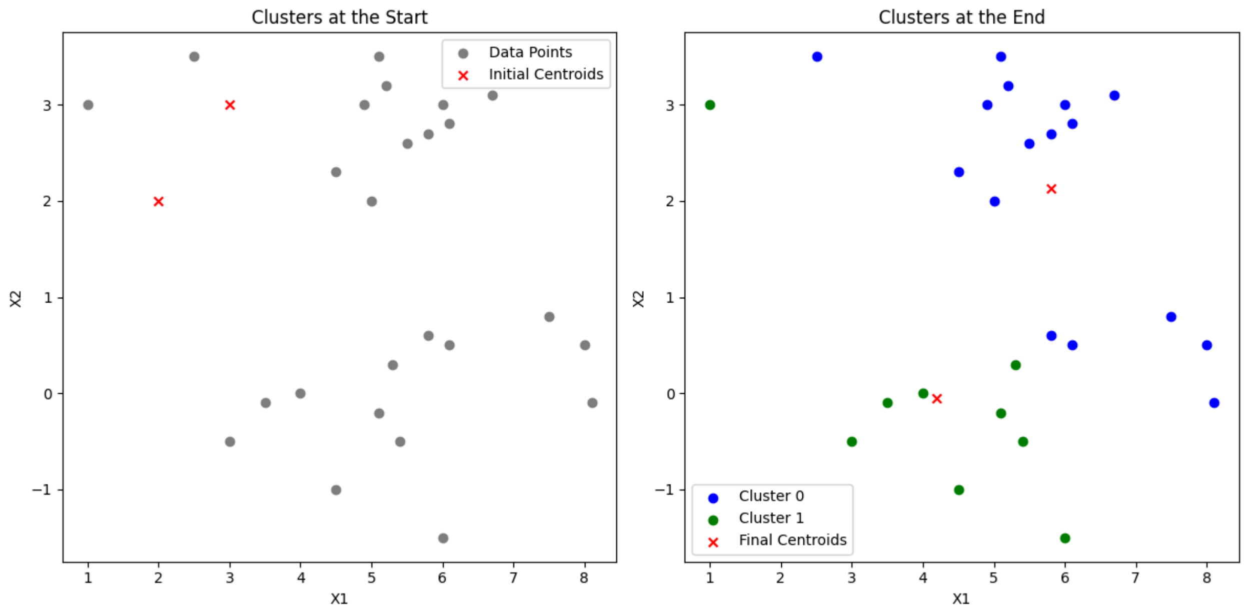
The data points were divided into two clusters as follows:

- Cluster 0: Points closer to u1\mathbf{u}_1u1
- Cluster 1: Points closer to u2\mathbf{u}_2u2

Visualization:

- The left plot shows the data points with the initial centroids.
- The right plot illustrates the final clusters with the updated centroids after convergence.

Plots:



(c)

Provided Initialization:

Using the given initial centroids:

u1=[3.0,3.0],u2=[2.0,2.0]\mathbf{u}_1 = [3.0, 3.0], \quad \mathbf{u}_2 = [2.0, 2.0]u1=[3.0,3.0],u2=[2.0,2.0]

- Final centroids after convergence:

u1=[5.8,2.125],u2=[4.2,−0.0556]\mathbf{u}_1 = [5.8, 2.125], \quad \mathbf{u}_2 = [4.2, -0.0556]u1=[5.8,2.125],u2=[4.2,−0.0556]

- The data points were assigned to clusters based on proximity to these centroids.

Random Initialization:

With randomly initialized centroids:

Random initial centroids depend on the specific run.\text{Random initial centroids depend on the specific run.}Random initial centroids depend on the specific run.

Example from one run:

u1=[4.03,0.3],u2=[6.02,2.033]\mathbf{u}_1 = [4.03, 0.3], \quad \mathbf{u}_2 = [6.02, 2.033]u1=[4.03,0.3],u2=[6.02,2.033]

Final centroids after convergence:

u1=[4.03,0.3],u2=[6.02,2.033]\mathbf{u}_1 = [4.03, 0.3], \quad \mathbf{u}_2 = [6.02, 2.033]u1=[4.03,0.3],u2=[6.02,2.033]

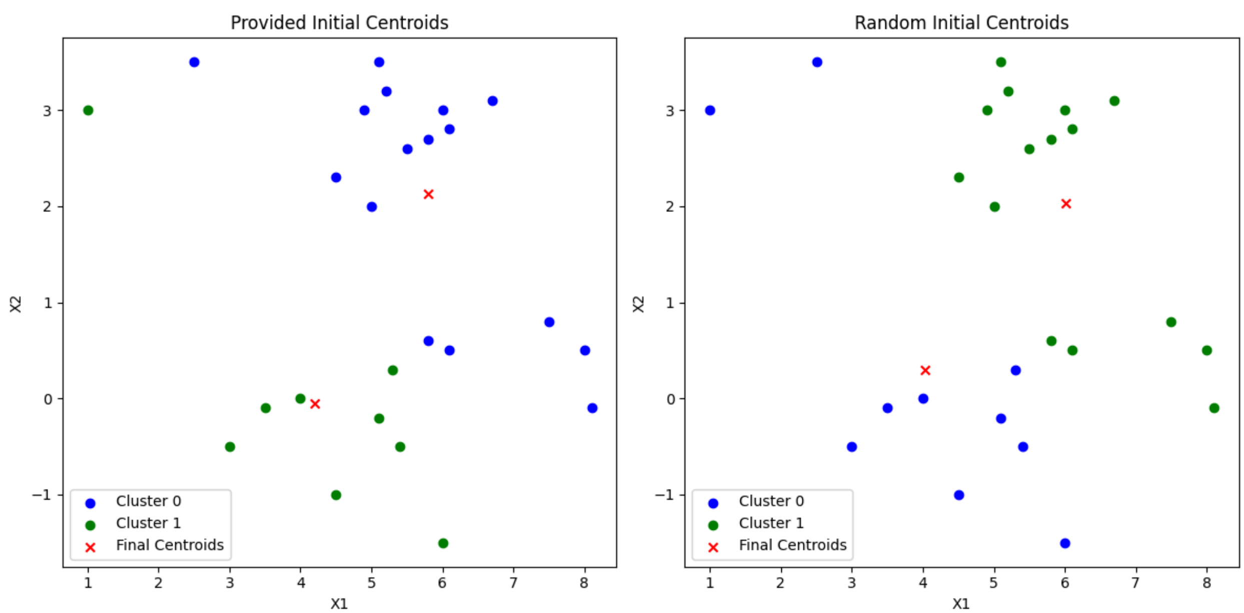
Key Observations:

1. The final cluster configurations depend significantly on the initialization of centroids.
2. With provided initialization, clustering follows the bias introduced by the given starting centroids, yielding consistent results.
3. Random initialization introduces variability in clustering, potentially leading to different outcomes, as seen in the random initialization plot.

Visualization:

- The left plot shows clustering with provided centroids.
- The right plot illustrates clustering with random initialization.

Plots:



(d)

Method:

To determine the optimal number of clusters (MMM), the **Elbow Method** was used. This involved:

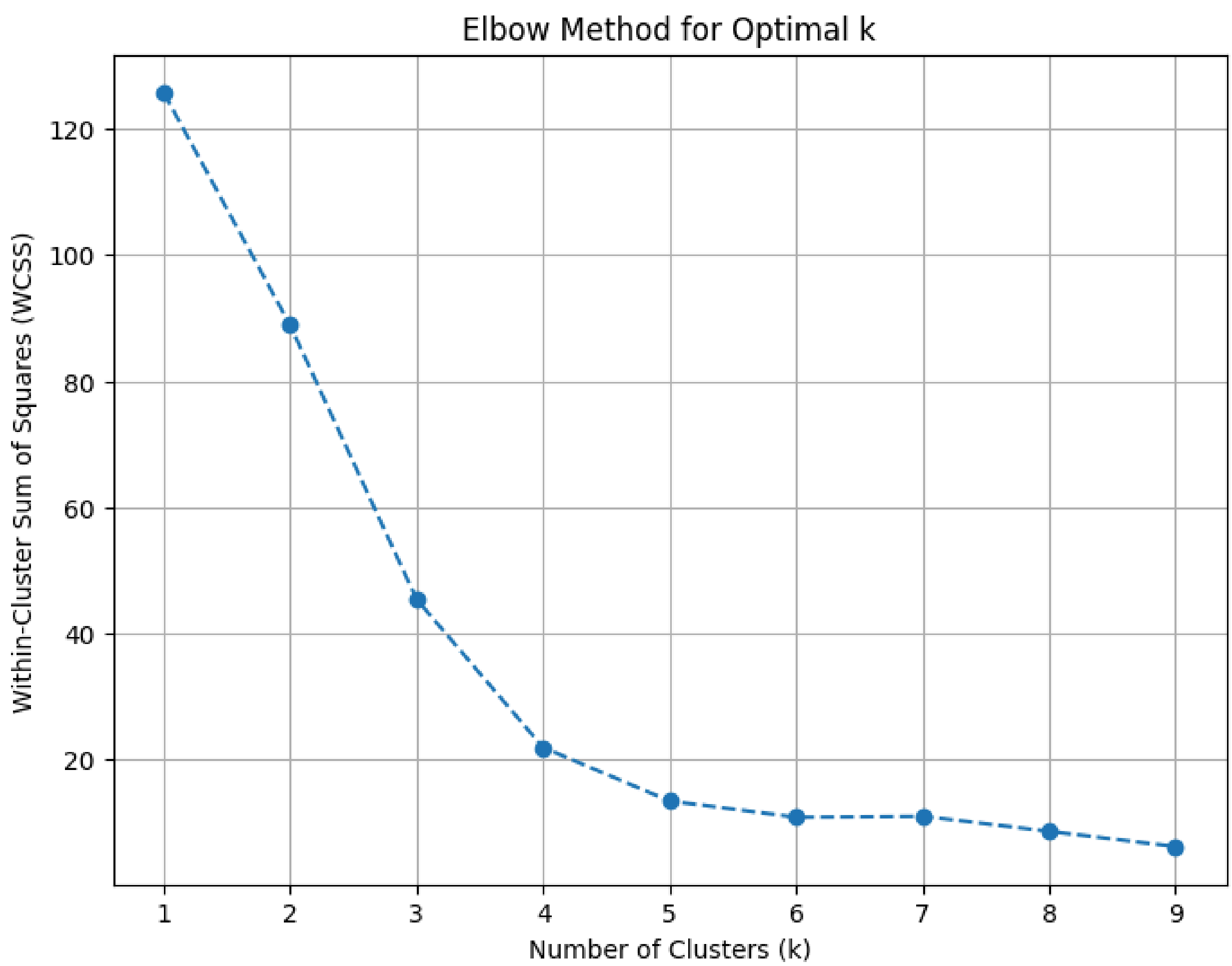
1. Calculating the **Within-Cluster Sum of Squares (WCSS)** for different values of kkk (number of clusters).
2. Plotting kkk against WCSS and identifying the "elbow point," where the rate of decrease slows significantly.

Results:

- The elbow point was observed at k=3k = 3k=3, indicating that three clusters best represent the data.

Visualization:

- **Elbow Method Plot**:

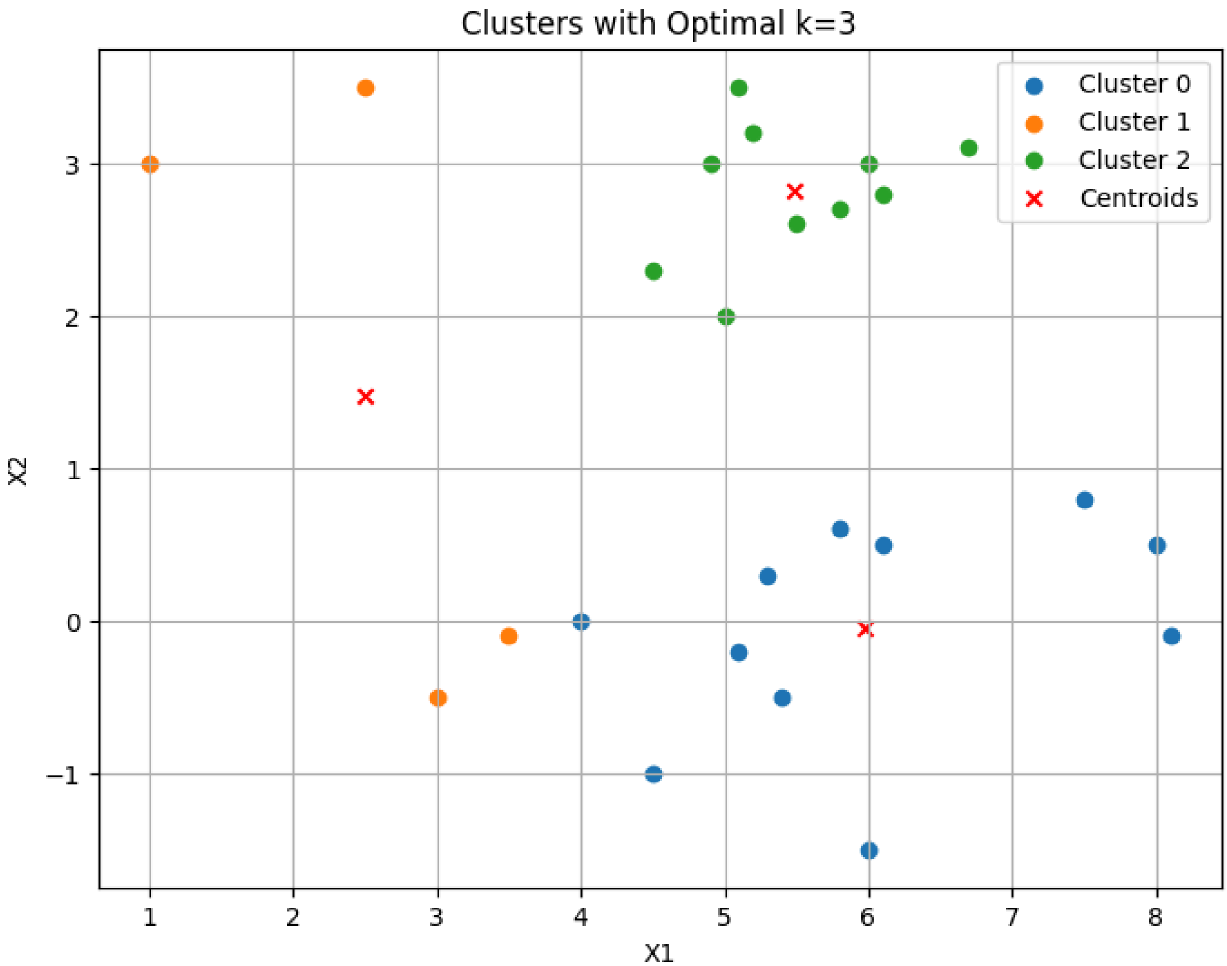


  - The plot below shows WCSS values for k=1k = 1k=1 to k=9k = 9k=9, with the elbow point at k=3k = 3k=3.

Clustering with k=3k = 3k=3:

After identifying k=3k = 3k=3 as the optimal number of clusters, clustering was performed with random initialization. The results are illustrated in the plot below:

- Three distinct clusters are shown, each marked with a unique color.
- The centroids of the clusters are marked in red.



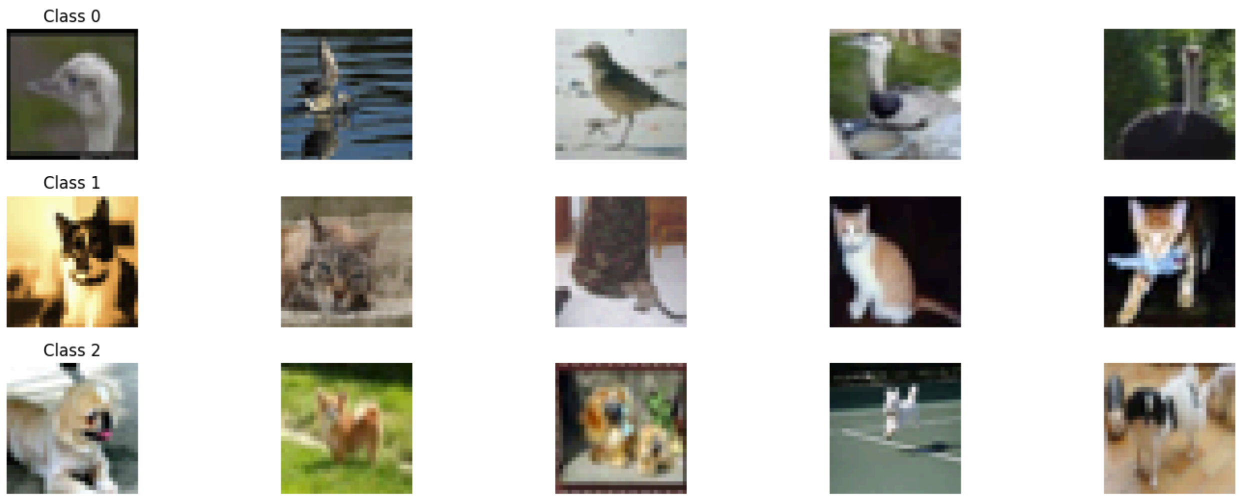
Key Observations:

1. Increasing the number of clusters (kkk) reduces WCSS but leads to diminishing returns after k=3k = 3k=3.
2. The clustering results effectively group similar data points, with each cluster representing a distinct region in the feature space.

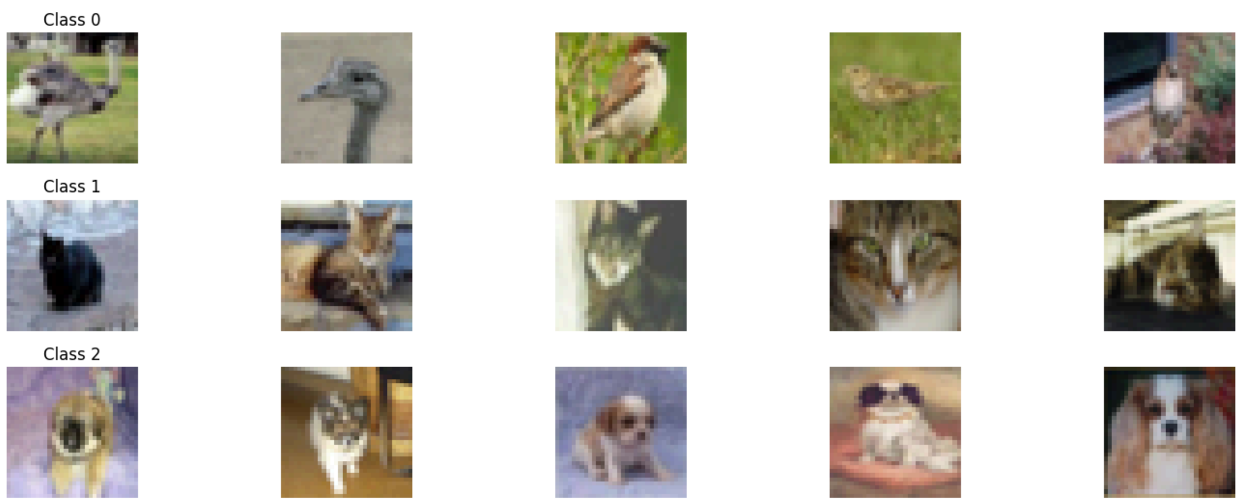
Section C

2.

Training Dataset Visualization



Validation Dataset Visualization



3.

Convolutional Layer 1:

- Kernel size: 5×55 \times 55×5
- Number of filters: 16
- Stride: 1
- Padding: 1

Max-Pooling Layer 1:

- Kernel size: 3×33 \times 33×3

- Stride: 2

Convolutional Layer 2:

- Kernel size: 3×33 \times 33×3
- Number of filters: 32
- Stride: 1
- Padding: 0

Max-Pooling Layer 2:

- Kernel size: 3×33 \times 33×3
- Stride: 3

Fully Connected Layers:

- FC1: 512→16512 \to 16512→16 neurons with ReLU activation.
- FC2: 16→316 \to 316→3 neurons for classification (3 classes).

4.

1. **Training Progress**:
   - Training loss decreased steadily from 0.98320.98320.9832 in epoch 1 to 0.57460.57460.5746 in epoch 15.
   - Training accuracy improved from 50.59%50.59\%50.59% to 75.82%75.82\%75.82% over 15 epochs.
2. **Validation Progress**:
   - Validation loss initially decreased, stabilizing in later epochs. This indicates the model was generalizing well.
   - Validation accuracy improved from 59.23%59.23\%59.23% in epoch 1 to 70.90%70.90\%70.90% by epoch 15.
3. **Final Model**:
   - The trained model achieved good generalization performance, as evidenced by the close gap between training and validation metrics.

Output Logs

The training logs provide a detailed view of the model's performance over each epoch:

```

Using device: cpu

Epoch 1/15
```

Train Loss: 0.9832, Train Accuracy: 0.5059

Val Loss: 0.8917, Val Accuracy: 0.5923

Epoch 2/15

Train Loss: 0.8733, Train Accuracy: 0.5982

Val Loss: 0.8191, Val Accuracy: 0.6247

Epoch 3/15

Train Loss: 0.8231, Train Accuracy: 0.6211

Val Loss: 0.7826, Val Accuracy: 0.6500

Epoch 4/15

Train Loss: 0.7918, Train Accuracy: 0.6398

Val Loss: 0.7597, Val Accuracy: 0.6620

Epoch 5/15

Train Loss: 0.7597, Train Accuracy: 0.6581

Val Loss: 0.7328, Val Accuracy: 0.6727

Epoch 6/15

Train Loss: 0.7330, Train Accuracy: 0.6768

Val Loss: 0.7214, Val Accuracy: 0.6810

Epoch 7/15

Train Loss: 0.7040, Train Accuracy: 0.6949

Val Loss: 0.6968, Val Accuracy: 0.6910

Epoch 8/15

Train Loss: 0.6822, Train Accuracy: 0.7042

Val Loss: 0.6941, Val Accuracy: 0.6860

Epoch 9/15

  Train Loss: 0.6649, Train Accuracy: 0.7115

  Val Loss: 0.6976, Val Accuracy: 0.6960

Epoch 10/15

  Train Loss: 0.6504, Train Accuracy: 0.7174

  Val Loss: 0.6916, Val Accuracy: 0.6933

Epoch 11/15

  Train Loss: 0.6260, Train Accuracy: 0.7307

  Val Loss: 0.6688, Val Accuracy: 0.7037

Epoch 12/15

  Train Loss: 0.6138, Train Accuracy: 0.7388

  Val Loss: 0.7192, Val Accuracy: 0.6863

Epoch 13/15

  Train Loss: 0.5926, Train Accuracy: 0.7479

  Val Loss: 0.6789, Val Accuracy: 0.7010

Epoch 14/15

  Train Loss: 0.5847, Train Accuracy: 0.7543

  Val Loss: 0.6934, Val Accuracy: 0.7053

Epoch 15/15

  Train Loss: 0.5746, Train Accuracy: 0.7582

  Val Loss: 0.6784, Val Accuracy: 0.7090

Model saved as 'simple_cnn_model.pth'

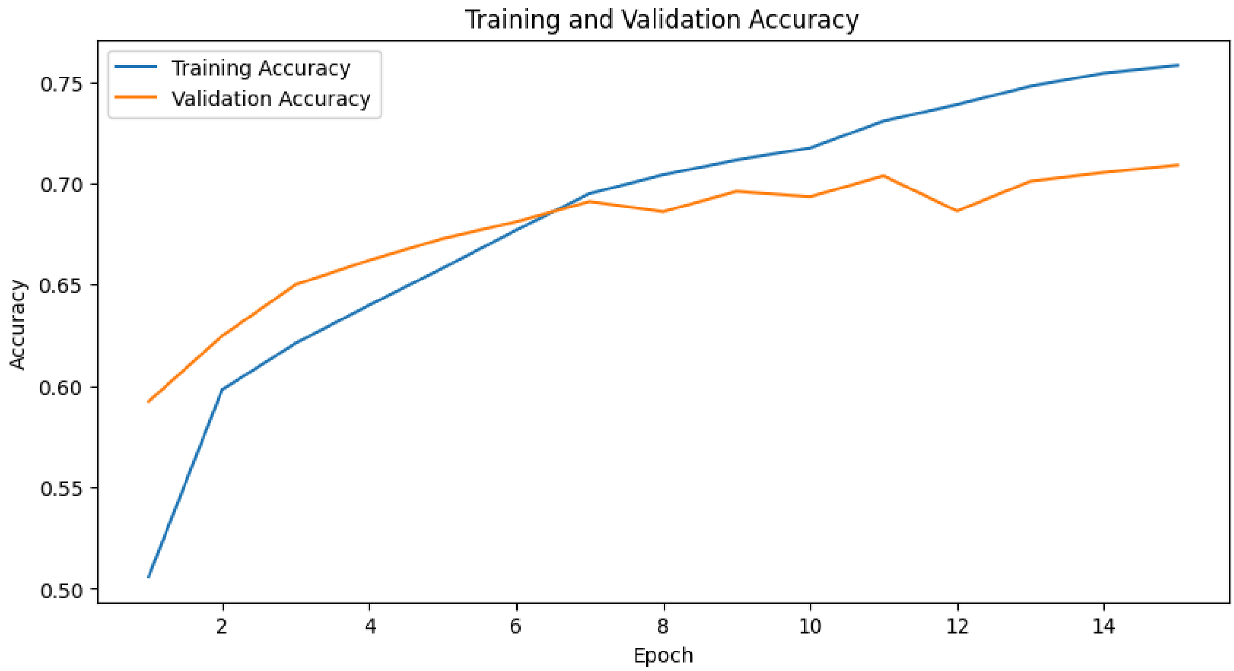
```
```

5.

Training and Validation Loss Plot

- **Trend**:
  - Both training and validation loss consistently decreased during the 15 epochs.
  - Training loss shows a smooth decline, suggesting that the model effectively learned from the data.
  - Validation loss stabilizes and fluctuates slightly after epoch 10, indicating the model's generalization capability.
- **Interpretation**:
  - The model avoids overfitting as validation loss does not significantly diverge from training loss.
  - Slight fluctuations in validation loss are expected due to the nature of mini-batch training.

Training and Validation Accuracy Plot



- **Trend**:
  - Training accuracy steadily increases from 50.59%50.59\%50.59% in epoch 1 to 75.82%75.82\%75.82% in epoch 15.
  - Validation accuracy improves from 59.23%59.23\%59.23% in epoch 1 to 70.90%70.90\%70.90%, closely tracking training accuracy.
- **Interpretation**:
  - The small gap between training and validation accuracy indicates good generalization.
  - The model has sufficient capacity to capture patterns in the data without overfitting.

6.

Training Progress

The model was trained for 15 epochs. Key observations include:

1. **Training Loss**:
   - Decreased steadily from 0.97000.97000.9700 in epoch 1 to 0.57230.57230.5723 in epoch 15.
   - Indicates the model successfully minimized the loss during training.
2. **Training Accuracy**:

- Improved from 52.25%52.25\%52.25% to 76.64%76.64\%76.64%, demonstrating the model's ability to learn from the training data.

Validation Progress

1. **Validation Loss**:
   - Decreased initially but started increasing from epoch 6 (0.90580.90580.9058) to epoch 15 (1.09241.09241.0924).
   - Suggests overfitting, as the model struggled to generalize to validation data.
2. **Validation Accuracy**:
   - Peaked at 59.37%59.37\%59.37% (epoch 9) but fluctuated and eventually dropped to 57.07%57.07\%57.07% (epoch 15).
   - Indicates the model failed to generalize effectively compared to CNN.

7.

Training and Validation Loss

CNN Observations:

- The **training loss** steadily decreases from 0.9832 to 0.5746 over 15 epochs, indicating consistent improvement during training.
- The **validation loss** initially decreases but stabilizes around epoch 11, indicating that the model is beginning to generalize well.
- Final Validation Loss: **0.6784**

MLP Observations:

- The **training loss** decreases more gradually from 0.9700 to 0.5723.
- The **validation loss** decreases initially but starts increasing after epoch 6, which could indicate overfitting.
- Final Validation Loss: **1.0924**

Inference:

- CNN outperforms MLP in terms of loss reduction.
- MLP shows signs of overfitting as the validation loss diverges from the training loss in later epochs.

Training and Validation Accuracy

CNN Observations:

- The **training accuracy** improves consistently, reaching **75.82%** by epoch 15.

- The **validation accuracy** also improves, stabilizing at **70.90%**.
- The close gap between training and validation accuracy suggests good generalization.

MLP Observations:

- The **training accuracy** improves to **76.64%**, slightly higher than CNN.
- The **validation accuracy**, however, stagnates around **57%** in the later epochs, showing poor generalization.

Inference:

- CNN achieves better validation accuracy and generalizes well to unseen data, whereas MLP overfits and struggles to generalize.

Test Dataset Performance

CNN Observations:

- **Test Accuracy**: **70.23%**
- **F1-Score**: Balanced across classes, with an average of **70%**.
  - **Class 0**: F1 = **0.78**
  - **Class 1**: F1 = **0.62**
  - **Class 2**: F1 = **0.70**

MLP Observations:

- **Test Accuracy**: **55.90%**
- **F1-Score**: Weaker and imbalanced, with an average of **56%**.
  - **Class 0**: F1 = **0.64**
  - **Class 1**: F1 = **0.51**
  - **Class 2**: F1 = **0.52**

Inference:

- CNN significantly outperforms MLP in both accuracy and F1-score, particularly for complex classes (Class 1 and Class 2).

Confusion Matrices

Training Confusion Matrix

- **CNN**: Strong diagonal dominance, indicating excellent classification during training.
- **MLP**: Moderate diagonal dominance with more misclassifications.

Validation Confusion Matrix

- **CNN**: Better performance with fewer off-diagonal elements compared to MLP.
- **MLP**: Higher confusion between similar classes (e.g., Class 1 and Class 2).

Test Confusion Matrix

- **CNN**: More balanced predictions across all classes.
  - E.g., Class 0: **802/1000**, Class 1: **548/1000**, Class 2: **757/1000**.
- **MLP**: Higher misclassifications, especially for Class 2.
  - E.g., Class 0: **682/1000**, Class 1: **501/1000**, Class 2: **494/1000**.

Inference:

- CNN shows stronger classification ability across all datasets, while MLP struggles, particularly for the more complex test dataset.

Key Takeaways

1. **Model Performance**:
   - CNN is better suited for image classification tasks because of its ability to capture spatial features using convolutional layers.
   - MLP, lacking convolutional layers, treats all pixels independently, making it less effective for image data.
2. **Generalization**:
   - CNN generalizes well to unseen data, as seen in the small gap between training, validation, and test metrics.
   - MLP shows poor generalization, with validation and test accuracy significantly lower than training accuracy.
3. **Class-Specific Performance**:
   - CNN handles all classes relatively well, with balanced F1-scores.
   - MLP struggles, especially for Class 1 and Class 2, which likely have more complex patterns.
4. **Confusion Matrices**:
   - CNN shows better diagonal dominance, reflecting fewer misclassifications.
   - MLP confusion matrices show more errors, especially for similar or complex classes.