Assignment 1

Name: Ruchitha Kota, Dhivyashree Siva Prakasam

UBIT: rkota2, dsivapra

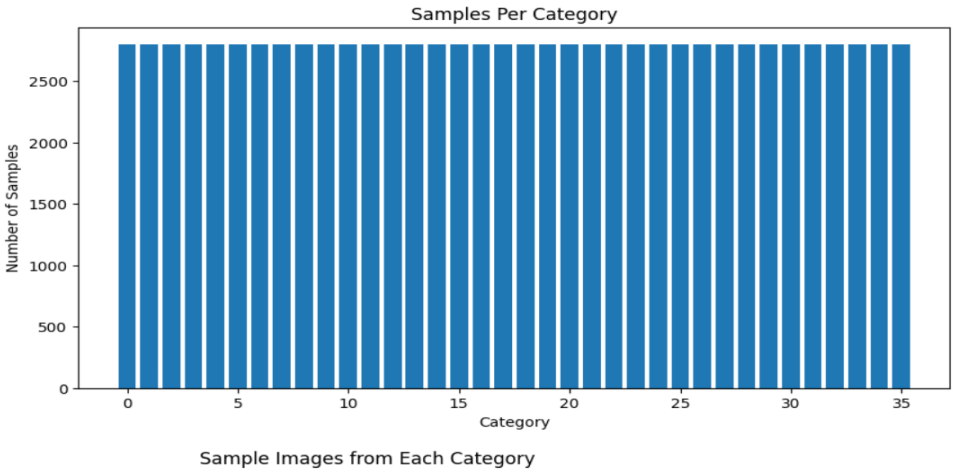
PART –1

1. **Dataset used:**
   Dataset consists of 36 categories and 2800 examples for each category, thus in total 100,800 samples. Each example is a 28x28 image. The dataset is a multi-class classification problem, where the objective is to correctly classify an image into one of the 36 categories.
   The images are 28x28 pixels in size and each image can be visualized as a 2D grid of pixel values

```
Number of categories: 36
Number of samples per category: [2800, 2800, 2800, 2800, 2800, 2800, 2800, 2800, 2
Total number of samples: 100800
```

2. **Visualization graphs:**

Bar plot of samples per category:

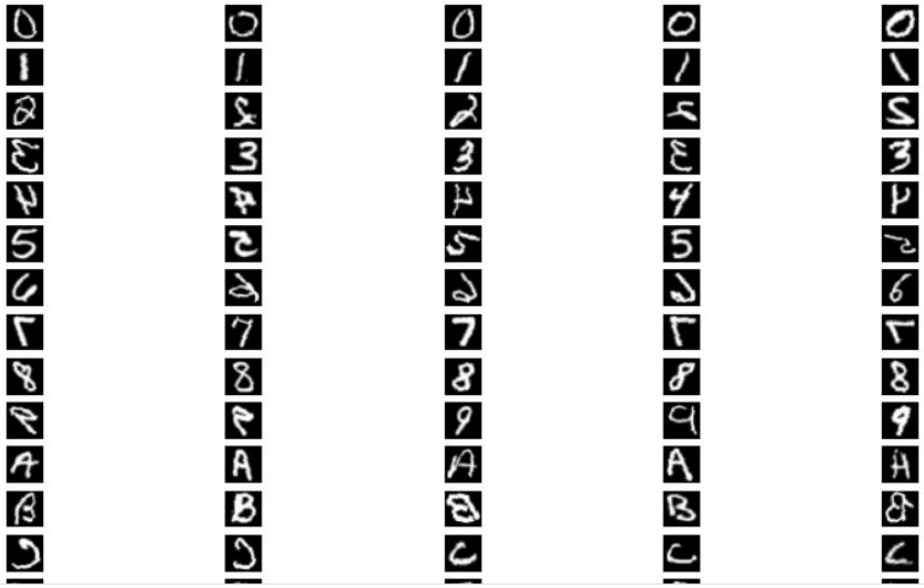


Sample Images from Each Category

This bar plot defines the distribution of data which contains the different categories.

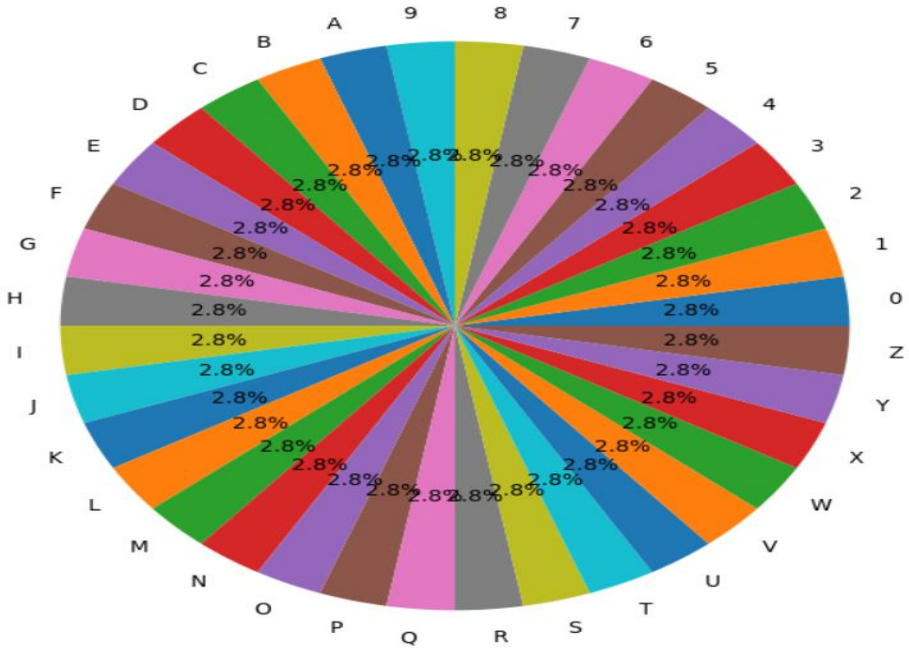
This visualization indicates whether the data in the dataset is well balanced or not.

Show sample images:



This visualization defines a grid of sample images from each category. Each row contains separate classes which are shown by selecting random sample images from each class. There are 36 classes in the dataset.

Pie chart of category distribution:



This pie chart defines the portion of each class in the dataset. The size of each slice pie chart indicates the percentage of each class in the dataset.

3. The model `CNN1` is a Convolutional Neural Network (CNN) designed for image classification.

   Input: 28x28 image with 3 channels (RGB).

   Convolutional Layers:
   Two layers with ReLU activation and 3x3 kernels.
   First layer: 1 input channel (grayscale), 32 output channels.
   Second layer: 32 input channels, 64 output channels.
   Pooling Layers:
   Two max pooling layers with 2x2 windows.
   Fully Connected Layers:
   First layer: 64 * 7 * 7 input features, 128 output features, ReLU activation, and 50% dropout.
   Second layer: 128 input features, 36 output features, Softmax activation.

4. **Dropout:**
   Effect: Prevents overfitting by randomly deactivating neurons, promoting robust feature learning.
   Implementation: self.dropout = nn.Dropout(p=0.5)

   Regularization (L2 Regularization):
   Effect: Adds a penalty to the loss function based on the magnitude of the weights.
   Impact: Reduces overfitting by discouraging large weights, leading to simpler, more generalizable models.
   optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-5)

   Early Stopping:

   Effect: Prevents overfitting by monitoring validation loss and stopping training when performance stagnates.
   Implementation: Monitors validation loss stops training if it does not improve

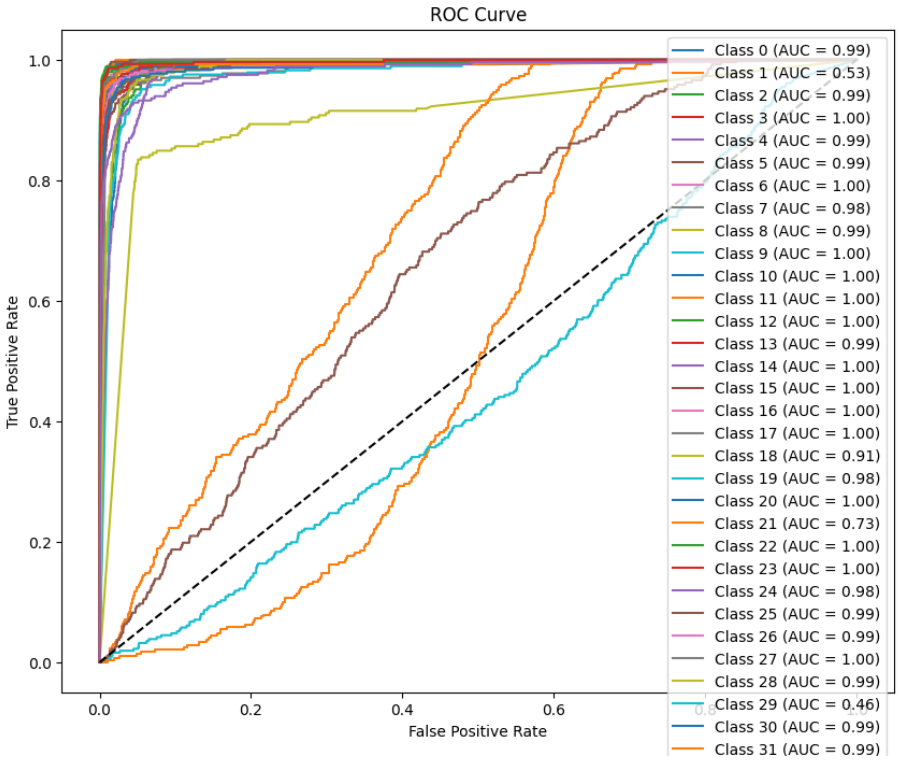
   The transform pipeline applies a series of preprocessing steps to each image: converting to grayscale, resizing to 28x28 pixels, applying random horizontal flips and rotations, converting to a tensor, and normalizing the pixel values. These steps enhance the dataset by augmenting the images and standardizing their format for improved training performance.

5. **Performance Metrics and analysis:**

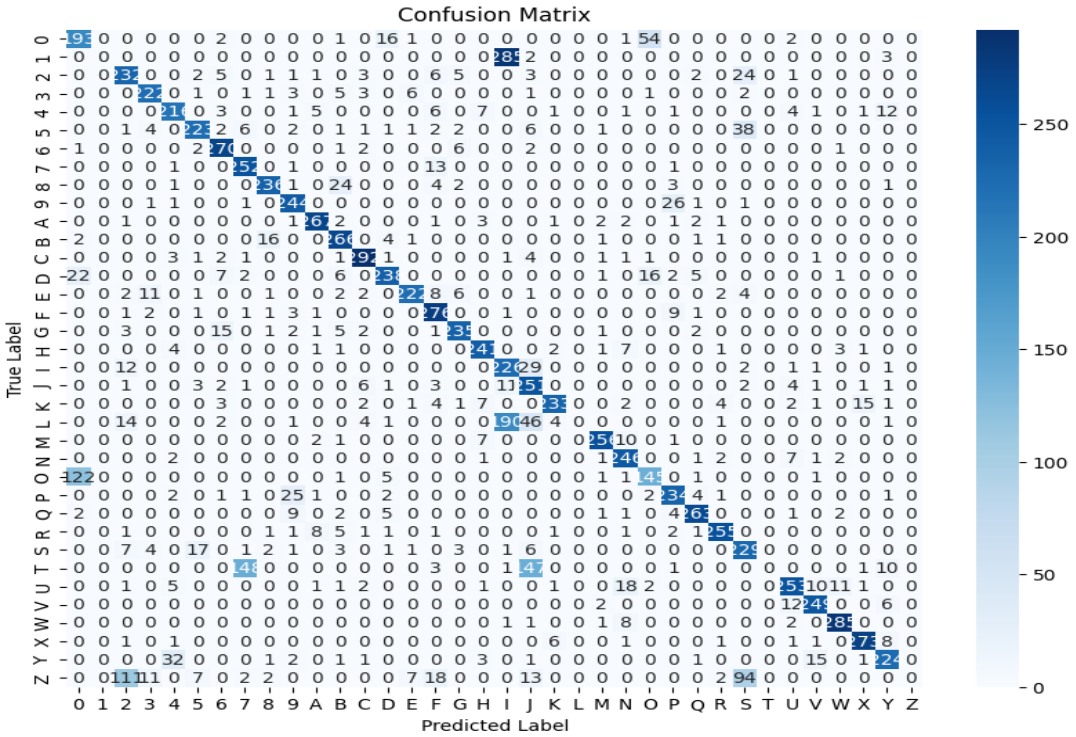
```
Train Accuracy: 72.9278, Validation Accuracy: 80.0198

Test Loss: 2.8355, Test Accuracy: 79.4643
Epoch 11
 Train Loss: 2.8827, Validation Loss: 2.8137
 Train Accuracy: 74.8313, Validation Accuracy: 81.6567

Test Loss: 2.8173, Test Accuracy: 81.2798
```

6. **Graphs:**

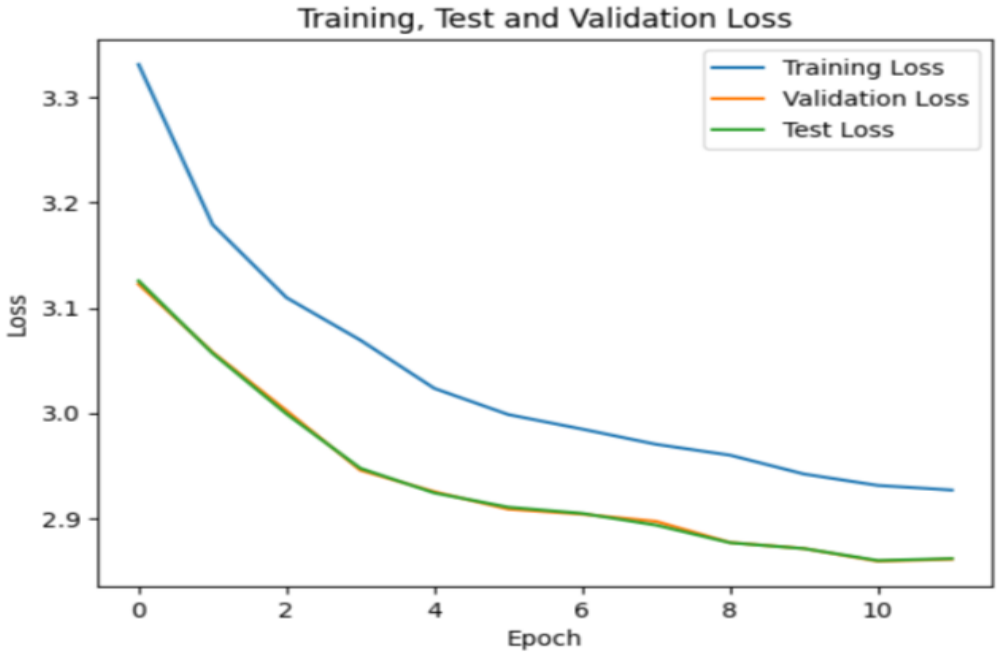
ROC curve: **(receiver operating characteristic curve)**



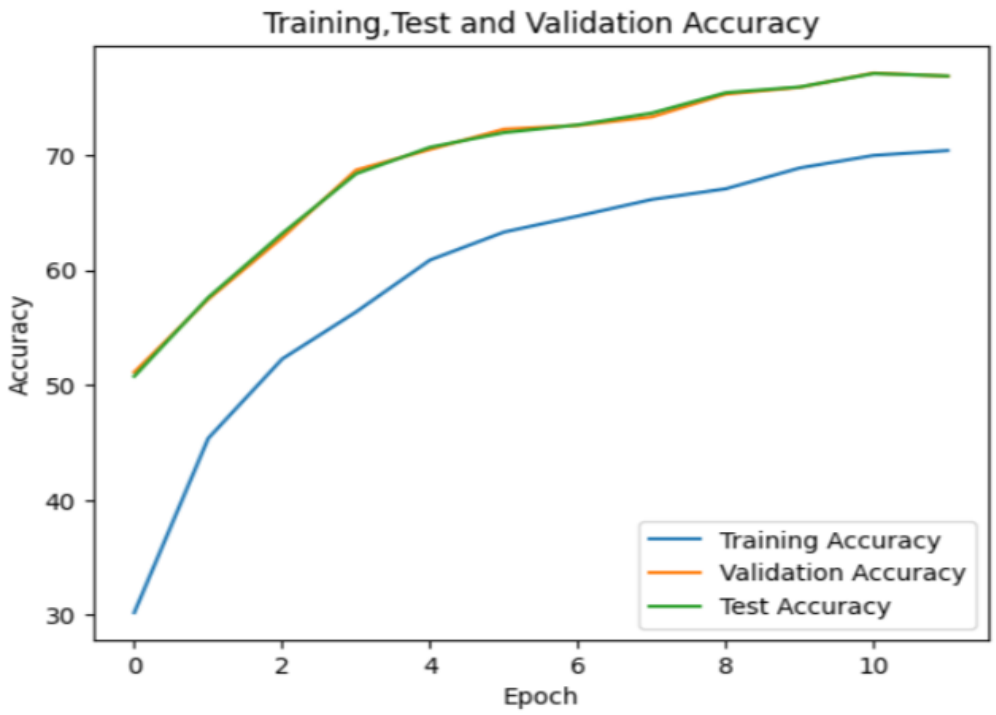
Confusion Matrix:



Plot for train, valid and test loss over epochs:



Plot for train, valid and test accuracy over epochs:



PART – 2

Dataset used: CNN dataset consists of 10,000 examples for each category, thus in total 30,000 samples. Each example is a 64x64 image.

1. **CNN model:**
   Architecture: VGG-13 with 13 weight layers.

   5 blocks of convolutional layers with ReLU activations and MaxPooling.

   Convolutional filters progress from 64 to 512.

   2 fully connected layers with 4096 units each, followed by ReLU and dropout.

   Final output layer with 3 units for 3 classes.

```
VGG13(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (15): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (16): ReLU(inplace=True)
    (17): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (20): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (21): ReLU(inplace=True)
    (22): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (23): ReLU(inplace=True)
    (24): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Linear(in_features=2048, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=3, bias=True)
  )
)
```

2. **The techniques impacted the model's performance:**

   Regularization (Weight Decay):
   Implementation: weight_decay=1e-5 in Adam optimizer
   Impact: Prevents overfitting by penalizing large weights, promoting generalization

   Dropout:
   Implementation: Dropout layers in the classifier
   Impact: Reduces overfitting by ignoring random neurons during training, enhancing generalization

Early Stopping:
Implementation: Stops training if validation loss doesn't improve
Impact: Prevents overfitting by halting training once performance on validation data ceases to improve

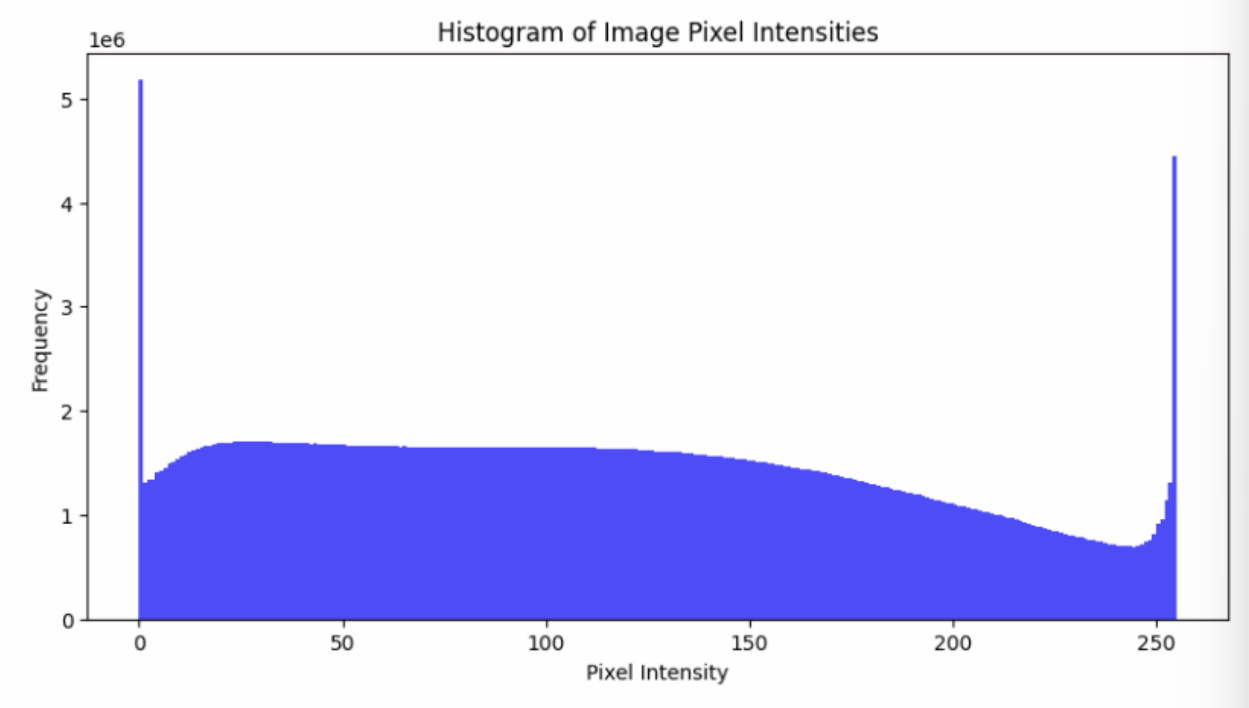
Sample input:

Main statistics of the dataset:

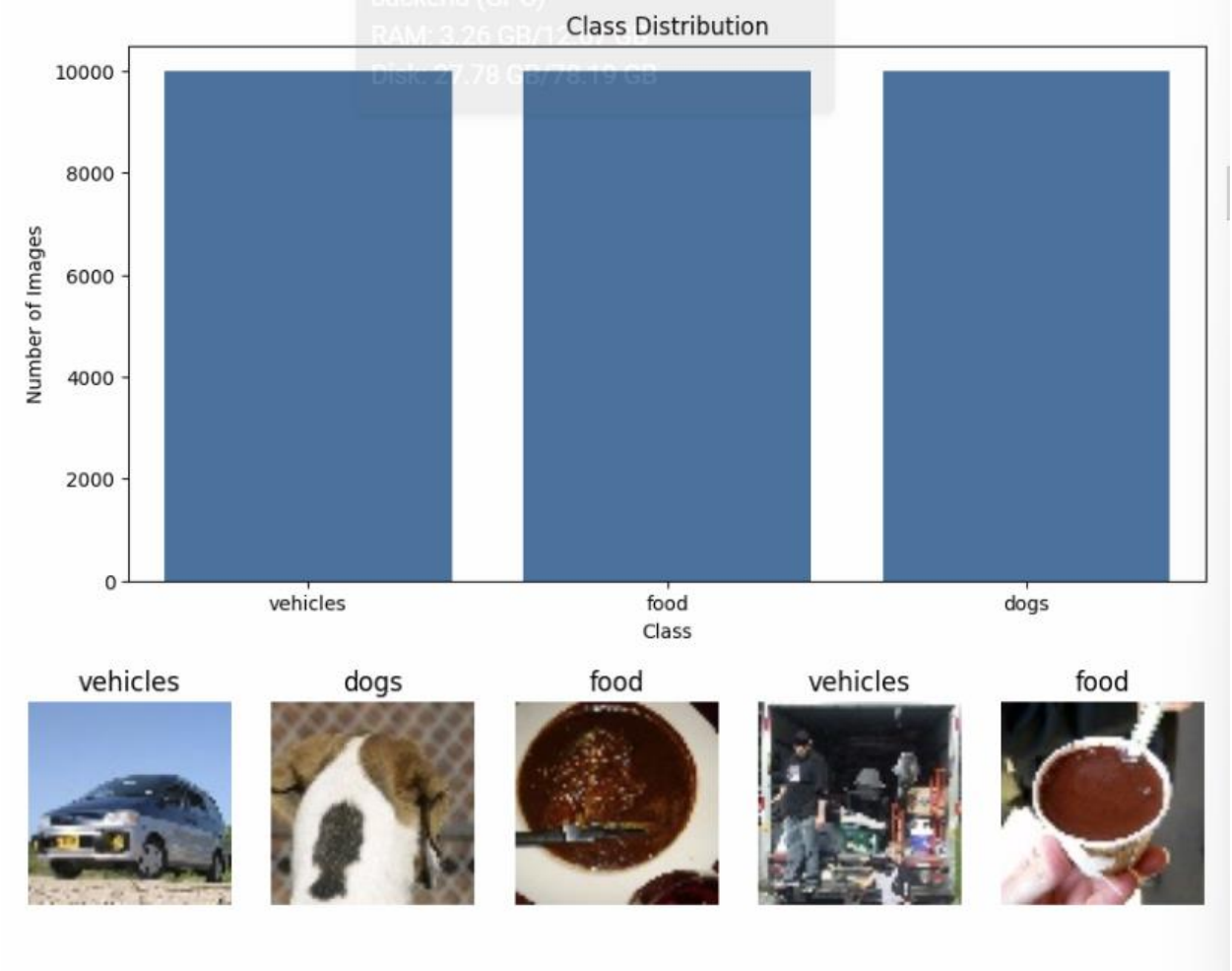
```
Total number of images: 30000
Image array shape: (30000, 64, 64, 3)
Label array shape: (30000,)
Number of images per class:
vehicles: 10000
food: 10000
dogs: 10000
```

Visualisation of dataset:

- *Histogram plot of Pixel Intensities:*



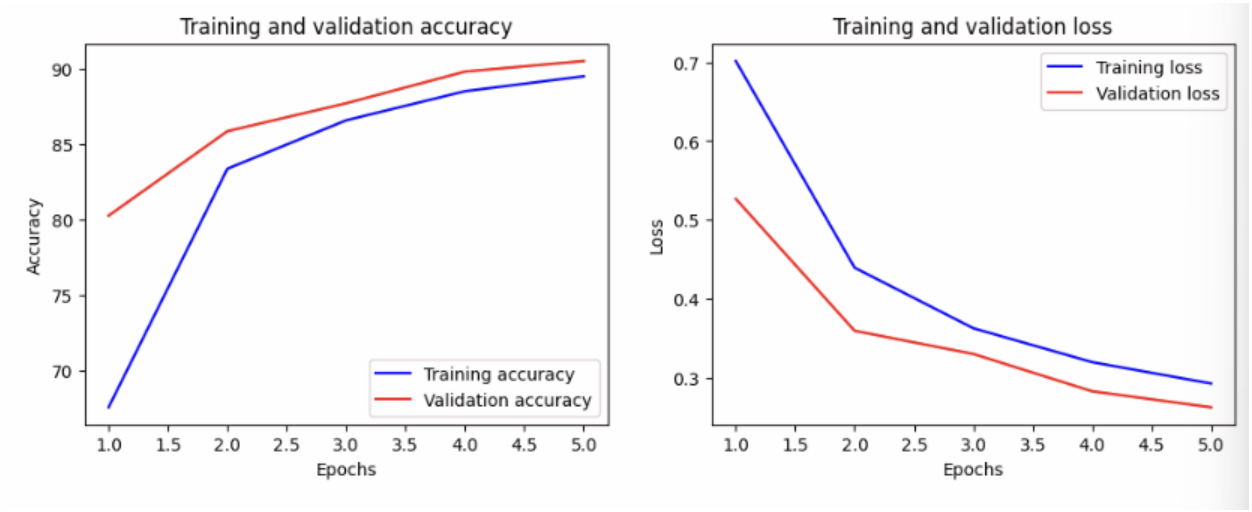
- *Plot of the Class distribution:*



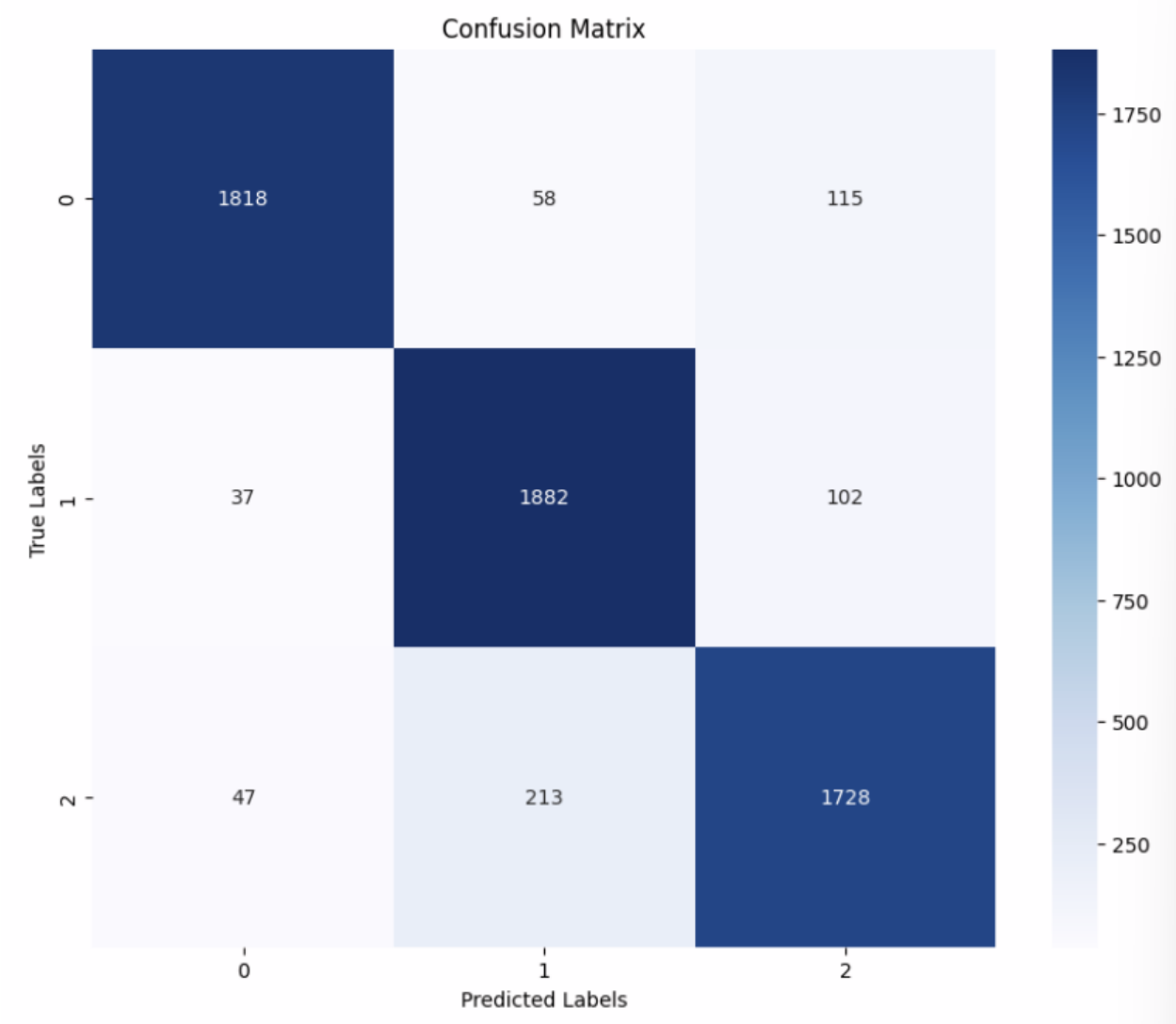
3. **a) Training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss is as below:**

```
Epoch 1,
 Train Loss: 0.7015, Train Accuracy: 67.58%,
 Validation Loss: 0.5267, Validation Accuracy: 80.28%

Epoch 2,
 Train Loss: 0.4393, Train Accuracy: 83.40%,
 Validation Loss: 0.3593, Validation Accuracy: 85.88%

Epoch 3,
 Train Loss: 0.3625, Train Accuracy: 86.60%,
 Validation Loss: 0.3299, Validation Accuracy: 87.73%

Epoch 4,
 Train Loss: 0.3195, Train Accuracy: 88.53%,
 Validation Loss: 0.2825, Validation Accuracy: 89.83%

Epoch 5,
 Train Loss: 0.2924, Train Accuracy: 89.53%,
 Validation Loss: 0.2622, Validation Accuracy: 90.53%

Test Loss: 0.2645, Test Accuracy: 90.47%
```

b) & c) Plots for training and validation losses and accuracies over epochs:



d) Confusion Matrix:



e) Other metrices used to analyze the model's performance on the test set:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.91      0.93      1991
           1       0.87      0.93      0.90      2021
           2       0.89      0.87      0.88      1988

    accuracy                           0.90      6000
   macro avg       0.91      0.90      0.90      6000
weighted avg       0.91      0.90      0.90      6000
```

CSE 676: Deep Learning

Assignment 0

NAME: Dhivyashree Siva Prakasam
UBIT: 50560688

PART –1:

1. The dataset "Electric Vehicle Population Size History By County" is used. The prediction of the percentage of electric vehicles is done using various models. It contains information about the population of electric vehicles (EVs) across different counties and states. It helps in understanding the adoption and growth trends of electric vehicles in various regions.

```
Number of entries: 21494
Number of features: 9
<class 'pandas.core.frame.DataFrame'>
Index: 21494 entries, 0 to 21581
Data columns (total 9 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   County                                21494 non-null  string
 1   State                                 21494 non-null  string
 2   Vehicle Primary Use                   21494 non-null  string
 3   Battery Electric Vehicles (BEVs)      21494 non-null  Int64
 4   Plug-In Hybrid Electric Vehicles (PHEVs)  21494 non-null  Int64
 5   Electric Vehicle (EV) Total           21494 non-null  Int64
 6   Non-Electric Vehicle Total            21494 non-null  Int64
 7   Total Vehicles                        21494 non-null  Int64
 8   Percent Electric Vehicles             21494 non-null  Float64
dtypes: Float64(1), Int64(5), string(3)
memory usage: 1.8 MB
None

First few rows of the dataset:
Summary statistics for numerical features:
       Battery Electric Vehicles (BEVs)  \
count                           21494.0
mean                          224.078766
std                          2369.606795
min                                 0.0
25%                                 0.0
50%                                 1.0
75%                                 3.0
max                             75487.0
Distribution of categorical features:
Count of unique values for each feature:
County                                  315
State                                    50
Vehicle Primary Use                       2
Battery Electric Vehicles (BEVs)       1238
Plug-In Hybrid Electric Vehicles (PHEVs)    950
Electric Vehicle (EV) Total            1432
Non-Electric Vehicle Total             7211
Total Vehicles                         7227
Percent Electric Vehicles                574
dtype: int64

Date                                    False
County                                   True
State                                    True
Vehicle Primary Use                     False
Battery Electric Vehicles (BEVs)        False
Plug-In Hybrid Electric Vehicles (PHEVs)    False
Electric Vehicle (EV) Total             False
Non-Electric Vehicle Total              False
Total Vehicles                          False
Percent Electric Vehicles               False
dtype: bool
```

2. **Mean Imputation**: Missing values in the 'Battery Electric Vehicles (BEVs)' column were imputed using the mean of the column.

Dropped rows with missing values in 'County' and 'State' columns

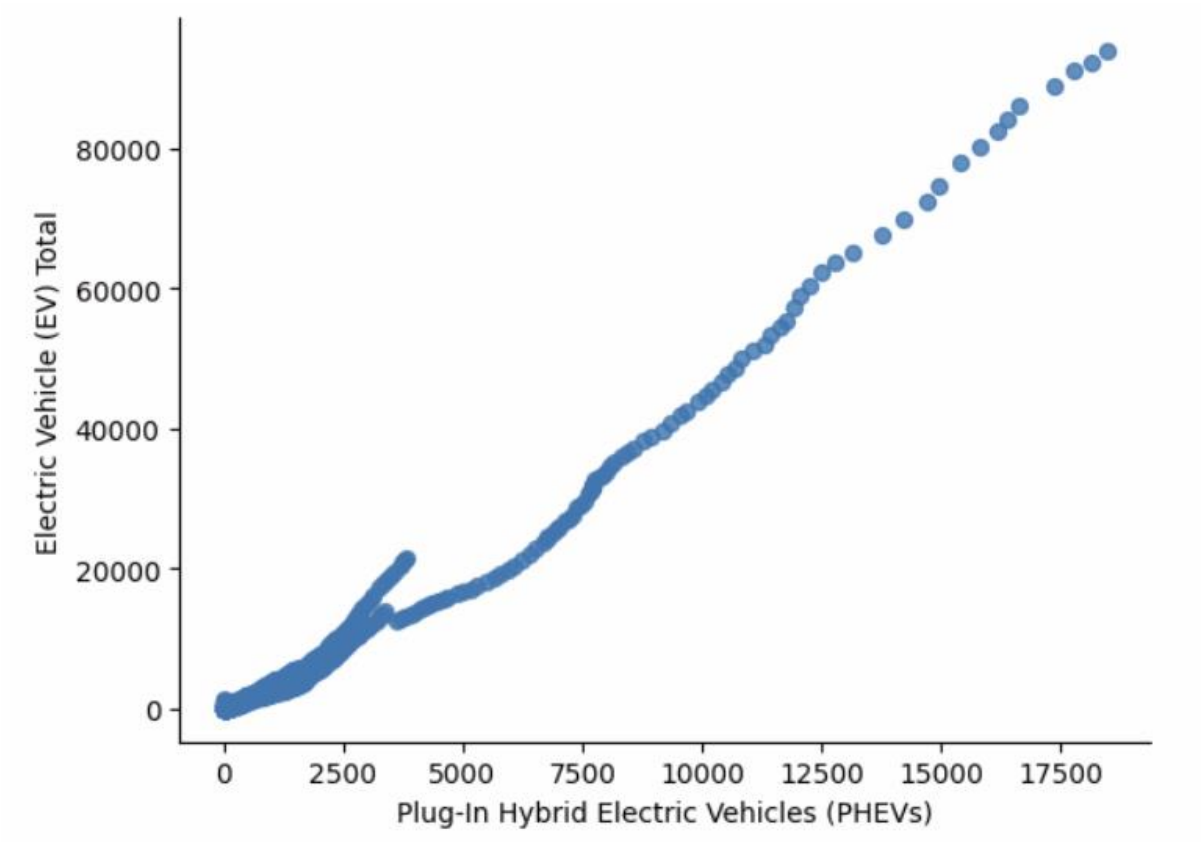
Feature Engineering: Date Conversion: Converted 'Date' column to datetime format and then dropped it as it wasn't used in the analysis.

Encoding Categorical Variables: One-Hot Encoding: Applied to categorical features like 'County', 'State', and 'Vehicle Primary Use'.

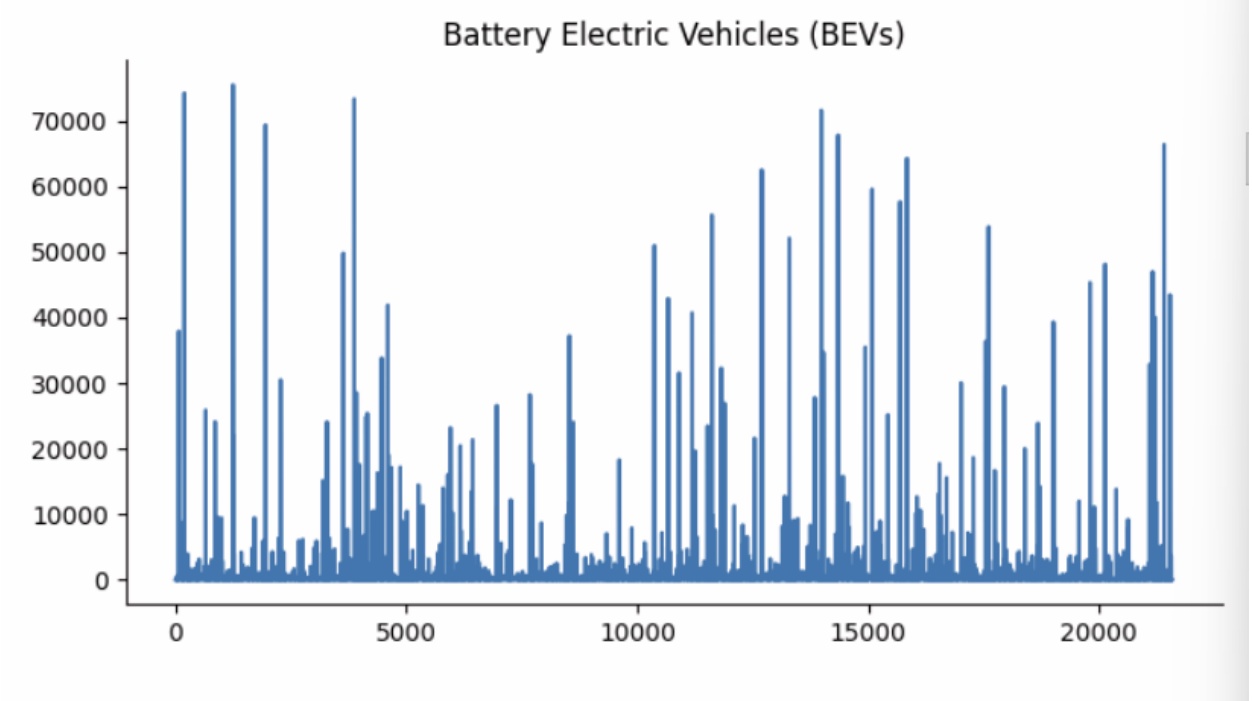
Standardized the features to have zero mean and unit variance.

Based on correlation matrix, the uncorrelated feature removed

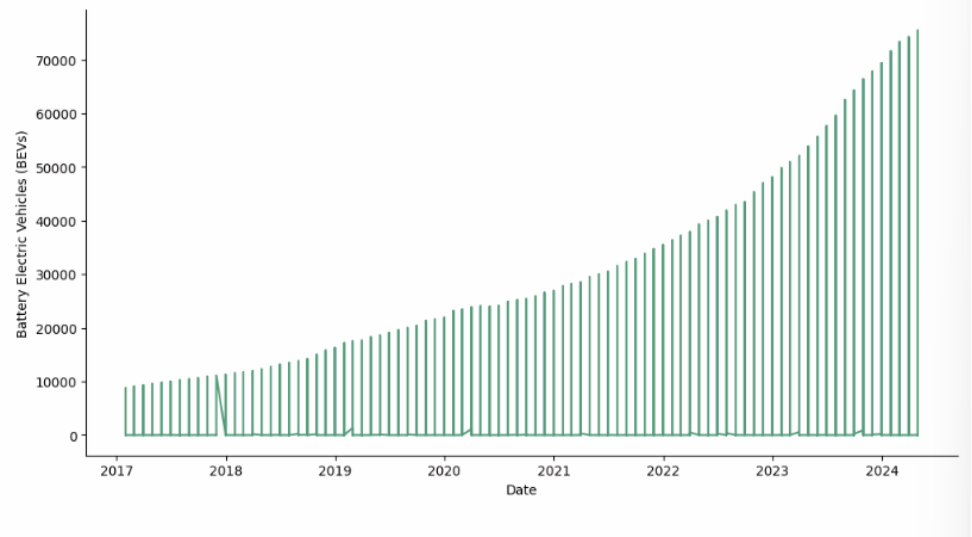
3. **1.Plot between Plug-In Hybrid Electric Vehicles and Electric Vehicles:**



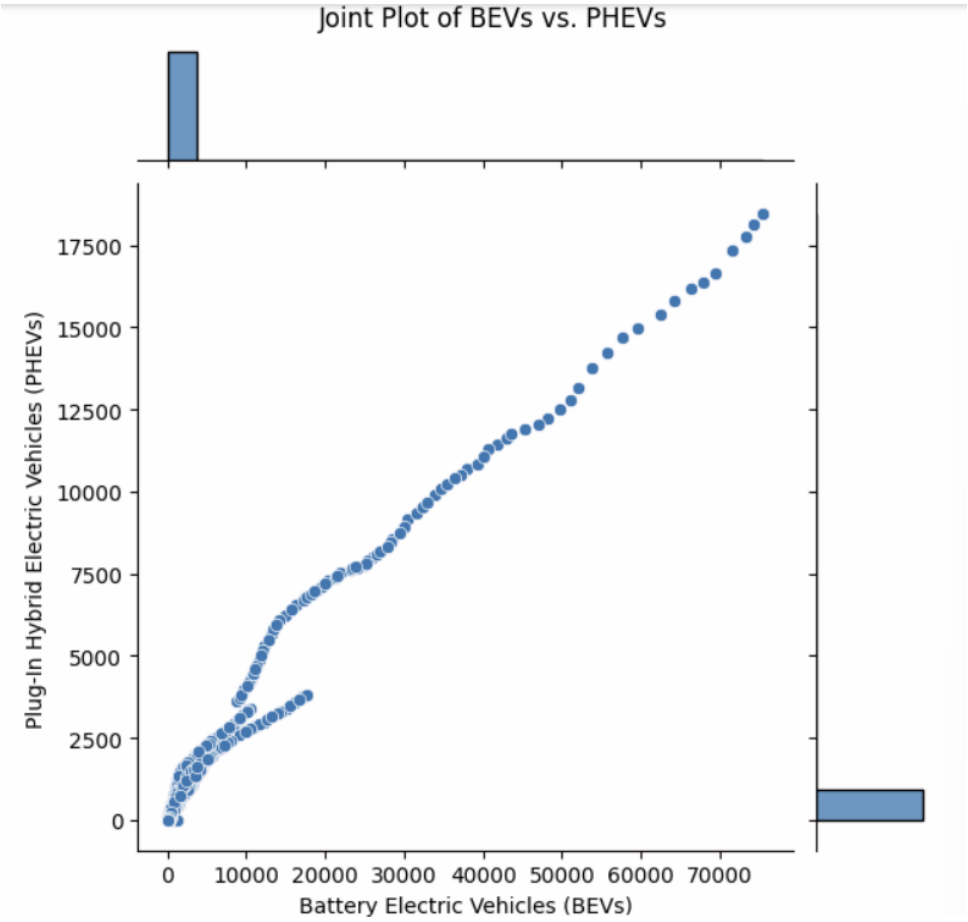
2.Plot of Battery Electric Vehicles:



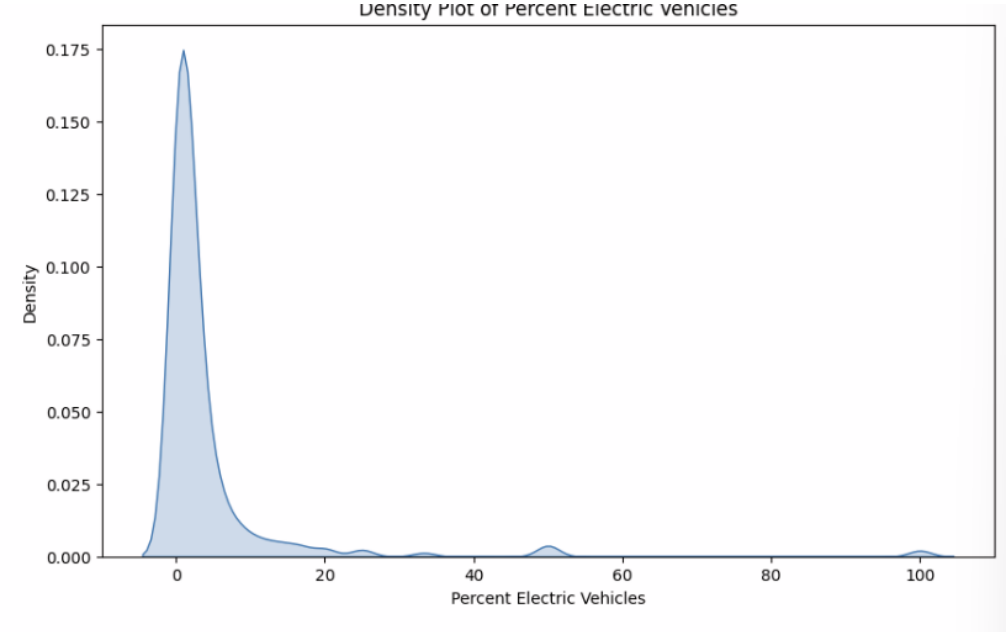
3.**Plot between Date and BEVs:**



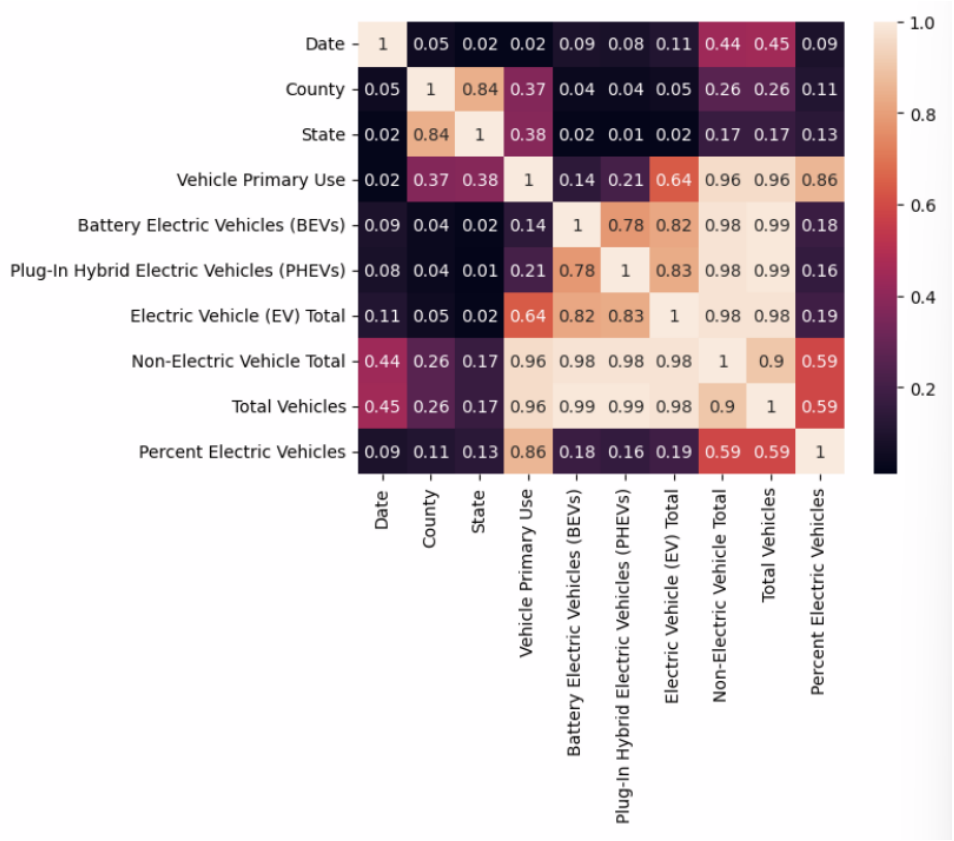
4.Joint Plot between BEVs and PHEVs:



5. Plot between percent electric vehicles and its density:



6. Correlation Matrix HeatMap:



4. **1. Decision Tree Regressor**
   Mathematical Representation: Uses a tree structure to model decisions and their possible consequences.
   Advantages: Simple to understand and interpret, requires little data preprocessing.
   Disadvantages: Prone to overfitting, especially with many features

   ***2. Random Forest* Regressor**

   Mathematical Representation: An ensemble method that builds multiple decision trees and merges them together to get a more accurate and stable prediction.
   Advantages: Reduces overfitting, handles large datasets with higher dimensionality.
   Disadvantages: Computationally expensive, less interpretable than a single decision tree.

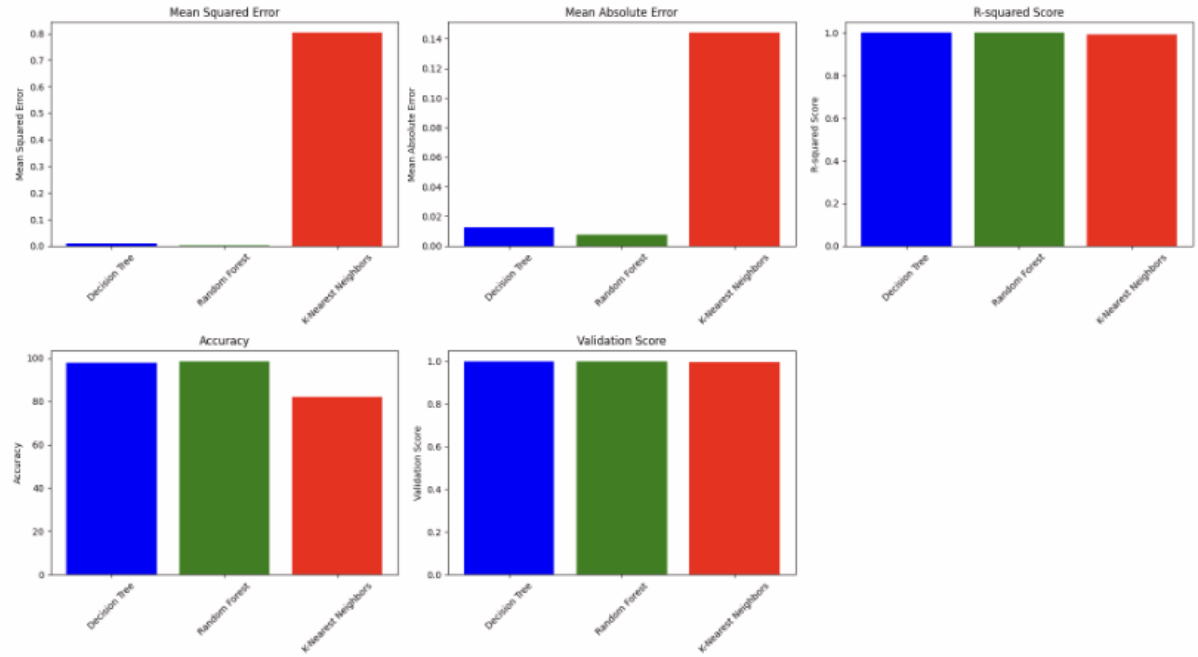
3. K-Nearest Neighbors Regressor (KNN)

Mathematical Representation: $y=1/k \sum yi \text{ where } i=1 \text{ to } k$, where yi are the target values of the k nearest neighbors.
Advantages: Simple to implement, makes no assumptions about the data.
Disadvantages: Computationally expensive during prediction, performance depends on the choice of k and distance metric.

5. Loss and accuracy of 3 models:

```
Metrics for Decision Tree:
Mean Squared Error: 0.0080
Mean Absolute Error: 0.0120
R-squared (R²) Score: 0.9999
Accuracy: 97.91%
Validation Score: 1.0000

Metrics for Random Forest:
Mean Squared Error: 0.0023
Mean Absolute Error: 0.0079
R-squared (R²) Score: 1.0000
Accuracy: 98.60%
Validation Score: 0.9999

Metrics for K-Nearest Neighbors:
Mean Squared Error: 0.8035
Mean Absolute Error: 0.1441
R-squared (R²) Score: 0.9928
Accuracy: 81.95%
Validation Score: 0.9954
```

6. **Plot comparing the predictions vs the actual test data for all methods used:**



7. This neural network consists of three fully connected (dense) layers:

   Input Layer:

   a. The input layer has the same number of neurons as the input dimension.
   b. It takes the input features and passes them to the next layer.

   Hidden Layers:

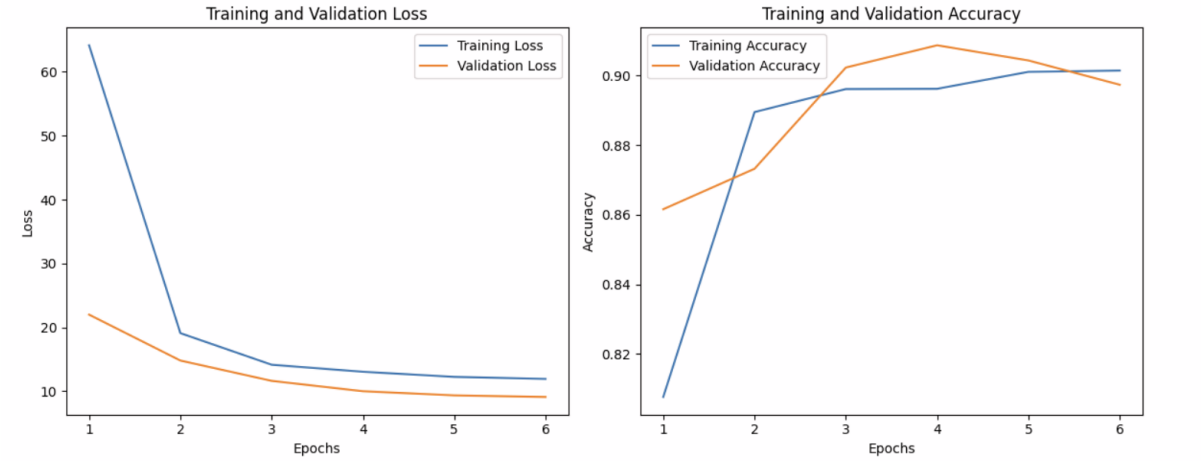
   c. Two hidden layers with 64 and 32 neurons, respectively.
   d. Each hidden layer applies the ReLU activation function to introduce non-linearity.

   Output Layer:

   e. The output layer has a single neuron since it's a regression task (predicting a continuous value).
   f. It provides the final output prediction.

   Accuracy, Loss and plots:

```
Epoch 2/6, Training Loss: 19.1089, Training Accuracy: 0.8895, Validation Loss: 14.8224, Validation Accuracy: 0.8732
Epoch 3/6, Training Loss: 14.1662, Training Accuracy: 0.8961, Validation Loss: 11.6429, Validation Accuracy: 0.9023
Epoch 4/6, Training Loss: 13.0662, Training Accuracy: 0.8962, Validation Loss: 10.0216, Validation Accuracy: 0.9087
Epoch 5/6, Training Loss: 12.2694, Training Accuracy: 0.9011, Validation Loss: 9.3739, Validation Accuracy: 0.9043
Epoch 6/6, Training Loss: 11.9477, Training Accuracy: 0.9014, Validation Loss: 9.1237, Validation Accuracy: 0.8974
Test Loss: 11.1500, Test Accuracy: 0.9051
```




```
Training MSE: 11.2364, RMSE: 3.3521, MAE: 1.2036, R^2: 0.9140
Validation MSE: 9.1237, RMSE: 3.0205, MAE: 1.2067, R^2: 0.9107
Test MSE: 11.1500, RMSE: 3.3392, MAE: 1.1784, R^2: 0.9006
```

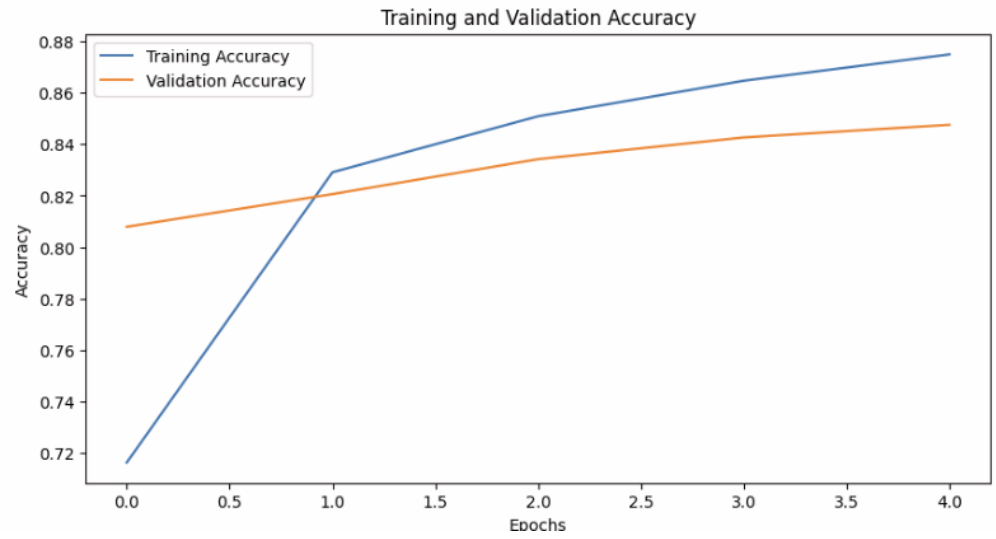
PART – 3:

1. I have used CNN model with ReLU activation function. The CNN model is a deep learning architecture used for image classification, comprising convolutional and fully connected layers to extract hierarchical features from input images. By incorporating ReLU activation functions and dropout regularization, the model enhances non-linearity and prevents overfitting, respectively, to achieve accurate classification results.
2. I have used L2 **regularisation**(optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-4), implemented **dropout** by randomly disabling neurons, dropout helps the model generalize better to unseen data, resulting in improved validation and test accuracy and **early stopping** monitors the model's performance on a validation set and stops training when the validation loss stops improving.

3. a)

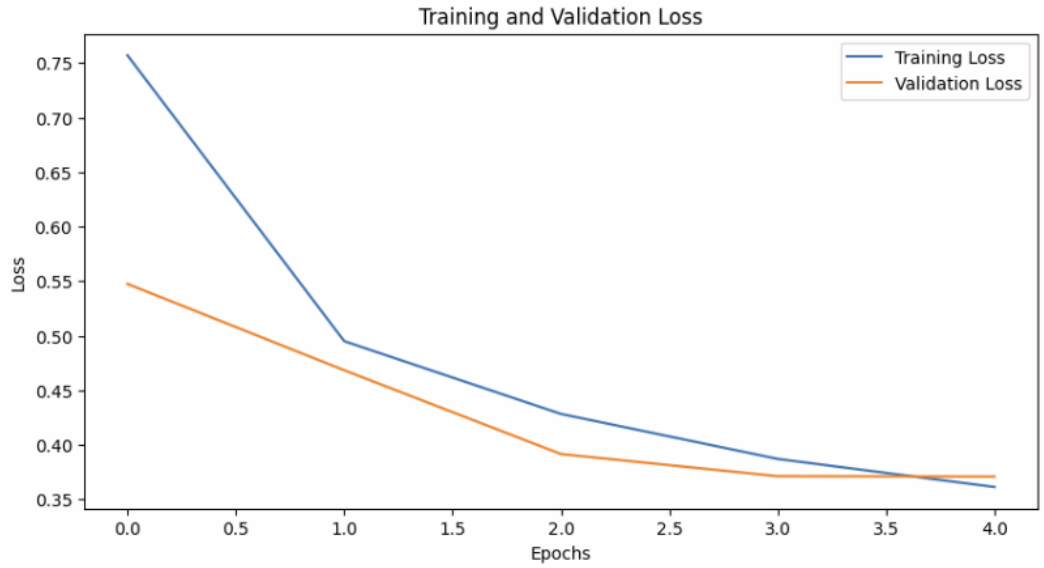
```
Epoch 1/5, Training Loss: 0.8466, Validation Loss: 0.5960
Epoch 2/5, Training Loss: 0.5447, Validation Loss: 0.4886
Epoch 3/5, Training Loss: 0.4805, Validation Loss: 0.4364
Epoch 4/5, Training Loss: 0.4308, Validation Loss: 0.4058
Epoch 5/5, Training Loss: 0.3930, Validation Loss: 0.3576
CNNModel(
  (conv1): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (fc1): Linear(in_features=1152, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=4, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
)

Training Accuracy [0.7163338386742643, 0.8291076376182759, 0.8508808615191594, 0.86
Validation Accuracy [0.8079409612100512, 0.8206269821907782, 0.8342075302919412, 0.

Test Loss: 0.3808, Test Accuracy: 0.8656
```

B) **Plot the training and validation accuracy over time (epochs).**

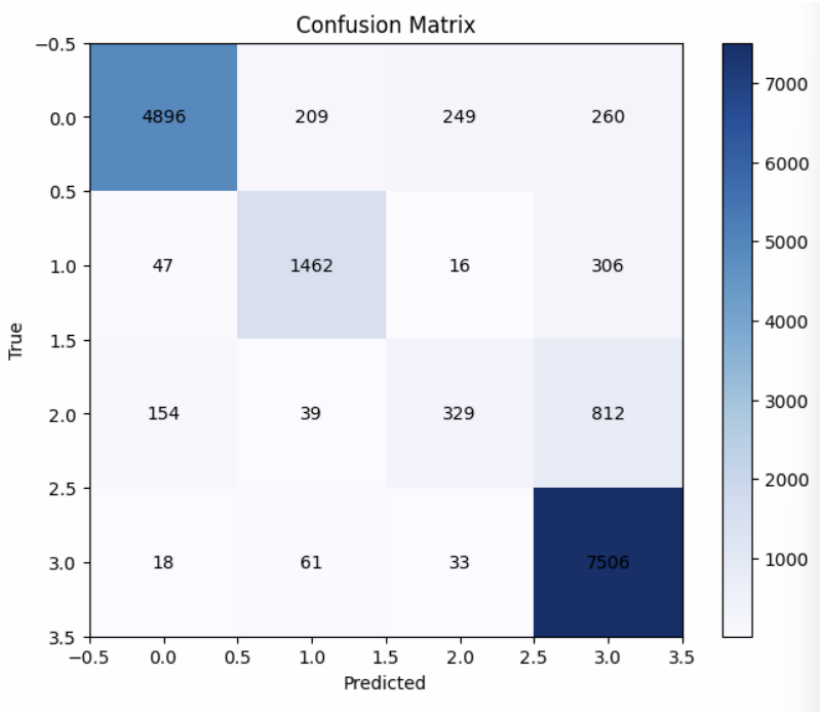


c. **Plot the training and validation loss over time (epochs)**

	precision	recall	f1-score	support
Class 0	0.96	0.87	0.91	5614
Class 1	0.83	0.80	0.81	1831
Class 2	0.52	0.25	0.34	1334
Class 3	0.84	0.99	0.91	7618
accuracy			0.87	16397
macro avg	0.79	0.73	0.74	16397
weighted avg	0.86	0.87	0.85	16397



D) **Confusion Matrix:**



e) **other evaluation metrics:**

	precision	recall	f1-score	support
Class 0	0.96	0.87	0.91	5614
Class 1	0.83	0.80	0.81	1831
Class 2	0.52	0.25	0.34	1334
Class 3	0.84	0.99	0.91	7618
accuracy			0.87	16397
macro avg	0.79	0.73	0.74	16397
weighted avg	0.86	0.87	0.85	16397