NEURAL NETWORKS - DEEP LEARNING FIRST ASSIGNMENT

Dimitrios Diakoloukas 10642, Electrical and Computer Engineering, *AUTH, Student*

***Abstract*—This project implements and compares three models—k-Nearest Neighbor (k-NN), Convolutional Neural Network (CNN), and Multi-Layer Perceptron (MLP)—for multi-class classification on the CIFAR-10 dataset. Each model's performance was evaluated based on accuracy, using both training and testing data. The CNN demonstrated superior feature extraction and classification capabilities, while the MLP provided insight into performance with fully connected layers. The k-NN model served as a baseline comparison. Results highlight each model's strengths and limitations, offering a detailed analysis of their effectiveness on CIFAR-10.**

I. Introduction

IMAGE classification is an essential task in computer vision, used in areas like autonomous driving and facial recognition. This project tests variations of three models—k-Nearest Neighbor (k-NN), Convolutional Neural Network (CNN), and Multi-Layer Perceptron (MLP)—on the CIFAR-10 dataset, which has 60,000 images in 10 classes.

The k-NN models serve as a simple, distance-based baseline. The CNNs use convolutional layers to capture patterns in images, making them especially effective for this task. The MLPs, a fully connected network, show how traditional neural networks handle image classification.

Each model's accuracy is evaluated on CIFAR-10 to compare their strengths and weaknesses, highlighting the advantages of more advanced models like CNNs over simpler ones like k-NN.

II. Data Loading and Preprocessing

This section explains how we loaded the CIFAR-10 dataset and prepared it so it could be used with the k-Nearest Neighbor (k-NN) algorithm.

A. *Loading CIFAR-10 Data*

The CIFAR-10 dataset contains 60,000 small color images (32x32 pixels) across 10 different classes. There are 50,000 images for training and 10,000 images for testing. Each image is stored as a set of pixel values, and the data is divided into multiple files.

We used the following steps to load the data:

- **Unpacking Data Files:** The training data is stored in five files, each containing 10,000 images. We created a helper function called unpickle to open each file and load the data inside.
- **Combining Training Files:** We combined the data from all five training files into one large set. This gave us a total of 50,000 images for training. We did the same for the labels, so each image has a label showing which class it belongs to.
- **Loading Test Data:** The test data is stored in a separate file. We loaded this file in the same way to get 10,000 test images and their labels.

B. *Data Preprocessing*

After loading the data, we prepared it for use with machine learning by following these steps:

- **Reshaping Images:** Each image was originally stored as a long row of numbers (a flat array). We reshaped these numbers back into the original 32x32 size with 3 color channels (Red, Green, and Blue).
- **Normalizing Pixel Values:** The pixel values in each image were originally between 0 and 255. To make the data easier to work with, we scaled these values to be between 0 and 1 by dividing each value by 255. This helps the model perform better and faster.

The final output of our data loading and preprocessing is:

- `x_train`: A set of 50,000 training images, each with shape (32, 32, 3) and pixel values between 0 and 1.
- `y_train`: A set of 50,000 labels for the training images.
- `x_test`: A set of 10,000 test images, also with shape (32, 32, 3) and pixel values between 0 and 1.
- `y_test`: A set of 10,000 labels for the test images.

This data is now ready to be used in our CNN and MLP algorithms as well as the k-NN algorithm, making it easy to classify each image based on its pixel values.

III. The k-Nearest Neighbor (k-NN) Algorithm

The k-Nearest Neighbor (k-NN) algorithm is a simple, method used for classification tasks. In classification, the algorithm works by identifying the ”k” closest data points (or neighbors) to a given input and predicting the class based on these neighbors.

The steps of the k-NN algorithm are as follows:

1) **Choosing the Number of Neighbors (k):** The user selects the number of nearest neighbors (k) that will be used to make the prediction. A small k value can make the model more sensitive to noise and a larger k value may make it too generalized.
2) **Calculating Distance:** For a new data point, the algorithm calculates the distance between this point and all

points in the training data. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance, with Euclidean distance being the most widely used in k-NN classification and the one we used in our implementations later on.

3) **Finding the k Nearest Neighbors:** After calculating the distances, the algorithm identifies the "k" closest data points in the training set.
4) **Assigning the Class Label:** The algorithm assigns the new data point to the class that appears most frequently among the k neighbors. For example, if $k = 5$ and three neighbors are in class A while two are in class B, the new point is classified as class A.

A. *Advantages and Disadvantages of k-NN*

Advantages:

- Simplicity: k-NN is easy to implement and understand.
- **No Training Phase:** Not complicated because the computation happens during prediction.
- **Flexibility:** Works well with various distance metrics (like Euclidean, Manhattan, etc) and can be adjusted for specific tasks by changing the value of k.

Disadvantages:

- **Computational Cost:** Since k-NN requires calculating distances for all points in the training set, it can be slow and computationally expensive with large datasets. Since this is a major disadvantage we used only 2000 training examples in the implementation without PCA (Principal Component Analysis). In our implementation with PCA we used the entire CIFAR-10 dataset (60000 32x32 colour images in 10 classes) since PCA can reduce its dimensionality while keeping as much variation as possible making the k-NN algorithm operate quicker.
- **Sensitive to Scale:** k-NN needs preprocessing (like normalization) for better performance.
- **Sensitive to Imbalanced Data:** If one class has many more examples than others, k-NN might favor that class in its predictions.

That is why k-NN gave us worse accuracy results overall than the Convolutional Neural Network (CNN), and Multi-Layer Perceptron (MLP). For this project though it provides a simple comparison to evaluate the effectiveness of more complex neural network models like CNN and MLP.

B. *Approach of the Task*

1. Custom Implementation of k-NN Without PCA

In this approach, we implemented k-NN from scratch, using Euclidean distance to measure similarity between points. Each image was flattened into a 1D vector, and we used a subset of 2000 samples for both training and testing to keep it manageable, as the algorithm was quite slow.

- **Distance Calculation**: The distance between each test image and all training images is calculated.
- **Classification**: Each test image is labeled based on the most common label among its k nearest neighbors.
- **Nearest Centroid**: We also implemented a centroid-based classifier, which assigns each test image to the nearest class centroid.
- **Results**:
  1) **k-NN with $k = 1$ Accuracy:** 0.244
  2) **k-NN with $k = 3$ Accuracy:** 0.255
  3) **Nearest Centroid Accuracy:** 0.271

2. Custom Implementation of k-NN with PCA

In this approach, we used Principal Component Analysis (PCA) to reduce the data to 100 features, making the algorithm faster. Both k-NN and centroid classification were then applied to this simplified dataset.

- **Dimensionality Reduction**: PCA was used to reduce each image to 100 features.
- **k-NN and Centroid Classification**: Both k-NN and nearest centroid classifiers were used on the reduced data.
- **Results**:
  1) **k-NN with $k = 1$ Accuracy:** 0.3863
  2) **k-NN with $k = 3$ Accuracy:** 0.3893
  3) **Nearest Centroid Accuracy:** 0.2767

3. Using sklearn Library for k-NN and Centroid Classifier with PCA

In this approach, we used the KNeighborsClassifier and NearestCentroid from sklearn. PCA was applied to reduce each image to 100 features, making it faster to process. The optimized classifiers from sklearn gave us an efficient and scalable implementation.

- **Optimized Implementation**: KNeighborsClassifier and NearestCentroid offer fast, built-in methods for k-NN and centroid classification.
- **Parameter Tuning**: sklearn makes it easy to adjust settings like the number of neighbors k.
- **Efficiency**: Using sklearn improved performance and sped up the calculations on large datasets.
- **Results**:
  1) **k-NN with $k = 1$ Accuracy:** 0.3856
  2) **k-NN with $k = 3$ Accuracy:** 0.3691
  3) **Nearest Centroid Accuracy:** 0.2767

Each approach offers a unique perspective, from manual implementation to more efficient, library-based methods suitable for practical use and are widely known.

IV. References Section

References

[1] https://www.ibm.com/topics/knn