C/C++ Programming (CSC 3002)

122040010 – Edward Sean Alexander

Assignment 6 Project Report

## Introduction

The challenge of recognizing handwritten data holds significant importance in today's world, with practical applications permeating our daily lives. One notable area is the automation of tasks such as processing bank checks and postal addresses. For instance, in this assignment, we employed Artificial Neural Network models to discern handwritten digits from 0 to 9. In the realm of neural networks, where nodes operate akin to neurons, intricate connections and weight adjustments occur during forward propagation, mirroring the complexity of the human brain. This process allows the system to compare its final output with the desired result, adjusting weights through backpropagation to refine predictions. The incorporation of various layers, including input, output, and hidden layers, enhances the system's overall accuracy and capability. Understanding and addressing the nuances of handwritten data recognition is pivotal, given its widespread applications and the constant need for efficient, automated processes.

## Code Implementation

### 1. Data Extraction and Preprocessing:

#### a. ReadImages Function:

- Open the File and Read Header:

  The function starts by opening the specified file in binary mode (ios::binary) using an input file stream (ifstream). The first four bytes of the file contain the magic number, the number of images, the number of rows in each image, and the number of columns in each image. These values are stored in separate arrays (magicNumber, nImagesBytes, nRowsBytes, and nColsBytes). The read function is then used to read these values from the file into their respective arrays.
- Decode Header Information:

  The magic number is not directly used in this function, but it is a unique identifier for the file format. The number of images, rows, and columns are extracted from their respective byte arrays (nImagesBytes, nRowsBytes, nColsBytes) using bitwise operations and combined to form integers (nImages, nRows, nCols).
- Read Image Data:

  The function then initializes an empty vector result to store the images.

It enters a loop that iterates over the number of images specified in the file header. For each image, a vector image is created with a size equal to the number of rows times the number of columns. The image data is read from the file into the image vector. The image vector is then added to the result vector, effectively creating a vector of vectors where each inner vector represents an image.

- Normalize Pixel Values:

    The pixel values in each image are read as unsigned characters and stored in the range [0, 255]. To normalize these values to the range [0, 1], the values are divided by 255.0.

- Close the File and Return Result:

    After reading all images, the file is closed using the file.close() statement.

The function returns the result vector, which now contains all the images from the file in a suitable format for further processing.

**b. ReadLabels Function:**

- Open the File and Read Header:

    Similar to the ReadImages function, ReadLabels begins by opening the specified IDX1-UBYTE label file in binary mode using an input file stream (ifstream). It reads the first four bytes of the file, which represent the magic number and the number of labels, storing them in arrays (magicNumber and nImagesBytes).

- Decode Header Information:

    The magic number is retained for file format identification, while the number of labels is extracted using bitwise operations to form the integer nImages.

- Read Label Data:

    The function initializes an empty vector result to store the labels. It enters a loop that iterates over the number of labels specified in the file header. For each label, a vector label is created with a size of 1 (since each label is a single unsigned character). The label data is then read from the file into the label vector. The label vector is added to the result vector, creating a vector of vectors where each inner vector represents a label.

- Close the File and Return Result:

    After reading all labels, the file is closed using file.close(). The function returns the result vector, now containing all the labels from the file in a suitable format for further processing.

**2. Main function**

**a. Reading Training and Test Data:**

The main function starts by calling the ReadImages and ReadLabels functions to read the training and test data from the MNIST dataset files (train-images.idx3-ubyte, train-labels.idx1-ubyte, t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte). The image data is stored in trainImages and testImages vectors, and the label data is stored in trainLabels and testLabels vectors.

**b. Preparing Data for Machine Learning:**

- Matrices (trainData, trainLabelsMat, testData, testLabelsMat) are created using OpenCV's Mat class to store the formatted data suitable for machine learning models.
- The image data is normalized by dividing pixel values by 255.0, and the label data is converted to the appropriate data types

**c. Machine Learning Models:**

**1. Random Forest (RTrees):**

- Initializes and trains a Random Forest classifier.
- Parameters: None (uses default hyperparameters).
- Uses OpenCV's RTrees class.
- Sets parameters: max depth, min sample count, use surrogates, max categories, termination criteria.
- Trains the model using the training data (trainData and trainLabelsMat).
- Predicts on the test data (testData) and evaluates accuracy.
- Outputs and displays predictions.

**2. K-Nearest Neighbors (KNearest):**

- Initializes and trains a K-Nearest Neighbors classifier.
- Parameters: None (uses default hyperparameters).
- Uses OpenCV's KNearest class.
- Sets parameters: is classifier, algorithm type, default k, emax.
- Trains the model using the training data (trainData and trainLabelsMat).
- Predicts on the test data (testData) and evaluates accuracy.
- Outputs and displays predictions.

## Reproduce Project

Steps:

1. **Download MNIST Dataset:**

To recreate this project, you'll start by downloading the MNIST dataset, from: http://yann.lecun.com//exdb/mnist/, which consists of images of handwritten digits. Once downloaded, create a new folder named mnist_data and place the dataset files (train-images.idx3-ubyte, train-labels.idx1-ubyte, t10k-images.idx3-ubyte, t10k-labels.idx1-ubyte) inside.

2. **Update File Paths**

Now, open the code in a text editor or an Integrated Development Environment (IDE). Look for lines that contain file paths like vector<vector<unsigned char>> trainImages = ReadImages("D:/...") and update these paths to point to the location where you've saved the mnist_data folder.

3. **Build the Image and Label Data**

In the main part of the code, there are functions called ReadImages and ReadLabels. These functions read the image and label data from the dataset files. You don't need to change anything here; these functions are like helpers that get the data ready for the program.

4. **Convert Data for Machine Learning**

The next step involves converting the image and label data into a format suitable for machine learning. The code reads the images, normalizes the pixel values to a range between 0 and 1, and sets up matrices to hold the data in a way that the computer can understand for machine learning purposes.

5. **Train the Random Forest Model**

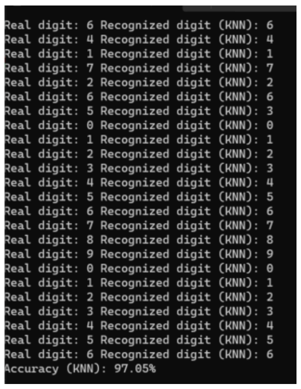
With the data prepared, the program proceeds to train a Random Forest model. Think of a Random Forest as a smart computer that learns from examples. The training process utilizes the image and label data, and you generally don't need to change anything here unless you want to experiment with different settings. The current settings are configured to work effectively.

6. **Make Predictions and Evaluate**

After training, our program tests the trained Random Forest and K-Nearest Neighbors models using a separate dataset. These models predict the digits of new, unseen images. We then compare these predictions with the actual labels of the test dataset to measure the accuracy of both models.

K-Nearest Neighbors (KNN) is a technique that classifies an unknown digit by considering its similarity to known digits. The program prints the predictions made by both models, providing insights into how well the computer recognizes handwritten digits based on their training. It's a way of checking how effective our models have become at identifying handwritten numbers.

## Results



```
Real digit: 6 Recognized digit (KNN): 6
Real digit: 4 Recognized digit (KNN): 4
Real digit: 1 Recognized digit (KNN): 1
Real digit: 7 Recognized digit (KNN): 7
Real digit: 2 Recognized digit (KNN): 2
Real digit: 6 Recognized digit (KNN): 6
Real digit: 5 Recognized digit (KNN): 3
Real digit: 0 Recognized digit (KNN): 0
Real digit: 1 Recognized digit (KNN): 1
Real digit: 2 Recognized digit (KNN): 2
Real digit: 3 Recognized digit (KNN): 3
Real digit: 4 Recognized digit (KNN): 4
Real digit: 5 Recognized digit (KNN): 5
Real digit: 6 Recognized digit (KNN): 6
Real digit: 7 Recognized digit (KNN): 7
Real digit: 8 Recognized digit (KNN): 8
Real digit: 9 Recognized digit (KNN): 9
Real digit: 0 Recognized digit (KNN): 0
Real digit: 1 Recognized digit (KNN): 1
Real digit: 2 Recognized digit (KNN): 2
Real digit: 3 Recognized digit (KNN): 3
Real digit: 4 Recognized digit (KNN): 4
Real digit: 5 Recognized digit (KNN): 5
Real digit: 6 Recognized digit (KNN): 6
Accuracy (KNN): 97.05%
```

This image shows an example of how the output pane is like. This program was executed with the accuracy of 97.05% in terms of recognizing the digits.

## Conclusions

This project explores handwritten digit recognition using Artificial Neural Network models, specifically Random Forest and K-Nearest Neighbors algorithms. Addressing real-world applications like check processing, the project emphasizes the significance of understanding handwritten data recognition. The code implementation details the data preprocessing steps and machine learning model application. The Reproduce Project section provides clear steps for replication, empowering users to experiment and understand the models. In summary, the project offers a concise yet comprehensive guide for implementing neural network models in the context of handwritten digit recognition.