香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

Assignment 6 Project
C/C++ (CSC3002)

122040012 – Filbert Cahyadi Hamijoyo

Introduction

Handwritten digit recognition is a fundamental problem in computer vision, playing a pivotal role in various applications such as postal automation, document digitization, and machine learning research. The objective is to create a robust system capable of accurately identifying handwritten digits from the MNIST dataset.

The chosen method for this project is the K-Nearest Neighbors (KNN) algorithm. KNN is a simple yet effective supervised machine learning algorithm that classifies objects based on the majority class of their k-nearest neighbors in the feature space. In the context of handwritten digit recognition, each digit image is treated as a point in a high-dimensional space, and the algorithm predicts the digit by considering the labels of its nearest neighbors.

Encompassing a diverse set of 60,000 images distributed among 10 classes (0-9), the MNIST dataset forms a comprehensive foundation for our investigation. Each image, constituting a 28x28 pixel grid, translates into 784-dimensional vectors. This extensive dataset, available online, features grayscale images with pixel values ranging from 0 to 255, representing the gradations of brightness and darkness in each pixel. This structured dataset serves as the cornerstone for our exploration into the intricacies of handwritten digit recognition.

Reproduce Project

To customize the KNN project based on the provided code, begin by making sure you have the necessary header files, including opencv2/core.hpp, opencv2/highgui.hpp, opencv2/ml.hpp, and opencv2/imgproc.hpp and standard libraries such as iostream, fstream, and vector. Following this, declare the constant IMAGE_SIZE with a value of 29, establishing a foundational parameter for image processing. Subsequently, include functions for loading images (LoadImages) and labels (LoadLabels) from binary files, pivotal components for training and evaluating the model.

Optionally, introduce the PreprocessImage function for image preprocessing. This function can be customized by uncommenting and adjusting specific preprocessing steps, such as Gaussian Blur, Grayscale conversion, and Thresholding, to enhance the model's performance.

The code further features the TrainKNN function, which facilitates the training of the K-Nearest Neighbors (KNN) model using input images and corresponding labels.

Complementing this, the RecognizeDigit function predicts the digit based on the trained KNN model.

To assess the model's accuracy, incorporate the EvaluateAccuracy function, which calculates and displays the accuracy on a set of test images. Finally, within the main function, load training and test data, train the KNN model, and evaluate its accuracy. Comments within the code are clear enough to guide further customization, allowing for the adjustment of preprocessing steps and other parameters based on specific preferences.

For execution, compile the code using a C++ compiler with the correct OpenCV library settings. Run the compiled executable (main.exe), and the program will output the training accuracy and test accuracy of the KNN model.

To modify the parameters of the TrainKNN function for improving the K-Nearest Neighbors (KNN) model's performance, access the function within the code and adjust the hyperparameters of the KNN model, such as the number of neighbors (k) and the distance metric. Additionally, explore and customize image preprocessing steps or fine-tune the preparation of training data within the function. Optimize the model training process by experimenting with parameters like the number of iterations. After making changes, utilize the EvaluateAccuracy function to assess the impact on the model's accuracy with test data, facilitating an iterative process of adjustment and improvement.

Code Implementation

The implementation is centered around handwritten digit recognition using the K-Nearest Neighbors (KNN) algorithm. Here's a breakdown of the sections:

1. **Data Loading**

   a. *LoadImages* Function:

   The LoadImages function initiates the process of data acquisition. Operating on IDX3-UBYTE files, it performs a meticulous dance of binary file handling. This dance involves opening files, deciphering headers (magic number, image count, rows, and columns), and constructing vectors to hold images. The result is a matrix of unsigned characters—each row representing an image.

   b. *LoadLabels* Function:

   The LoadLabels function handles IDX1-UBYTE files, focusing on label extraction. Alongside *LoadImages* function, it opens label files, decodes headers, and constructs vectors to store labels. Each label is encapsulated as a single unsigned character, forming a vector that parallels the images.

2. **KNN Model Training**

   TrainKNN Function:

   With data in hand, the TrainKNN function engineers the training process. It established the necessary matrices for training data and labels.

   a. Data Preparation:

      It sets the stage by creating two matrices: trainingData for images and labelsMat for corresponding labels. The matrices are structured to align with KNN's requirements for training.

   b. Image Matrices and Flattening:

For each training image, it creates an OpenCV matrix (imageMat) and converts it to the float format (imageMat32F). The image matrix is flattened to a one-dimensional structure, ensuring compatibility with the KNN model.

c. Assembling Training Data:
The flattened image matrices are collected into trainingData. Labels are organized into labelsMat, ensuring a clear mapping between images and their respective labels.

d. Model Creation and Training:
It creates a KNN model using KNearest::create() and trains it with the prepared data. The model is set to recognize patterns based on the training images and labels.

3. **Digit Recognition**

RecognizeDigit Function:
The heart of recognition lies in RecognizeDigit. It's the cognitive center where a test image, prepped for KNN's discernment, undergoes evaluation. Through a series of operations—format conversion, reshaping, and finding the nearest neighbors—the model reveals its predictions. This function encapsulates the decoding process, translating pixel patterns into digit classifications.

a. Image Transformation:
It receives a test image (testImage) and converts it to the float format (testImage32F). The transformation is a preparation step to align the test image with the KNN model's understanding.

b. Reshaping and Normalization:
The image is reshaped into a one-dimensional matrix (testImageReshaped), adhering to the KNN model's flattened structure. This reshaping ensures a consistent format, essential for meaningful comparisons.

c. Prediction and Insight:
Utilizing the trained KNN model, it predicts the digit represented by the test image.
The result is extracted and encapsulated in the results matrix, along with additional information like neighbor responses and distances.

d. Deciphering the Outcome:
The function retrieves the predicted digit from the results matrix. The digit is converted to an integer and returned as the final output.

4. **Accuracy Evaluation**

EvaluateAccuracy Function:
The EvaluateAccuracy function acts as the judge, rigorously examining the model's performance.

a. Iteration Through Test Images:
The function traverses through each test image, one by one. For each test image, it creates an OpenCV matrix (testImageMat) and resizes it to the required dimensions.

b. Recognition and Comparison:

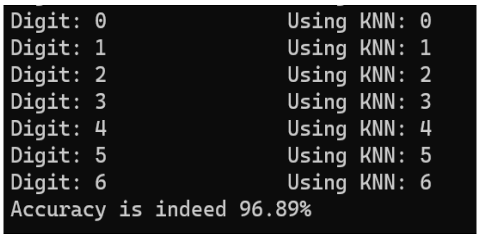
It calls the RecognizeDigit function to predict the digit using the trained KNN model. The predicted digit and the actual digit from the test labels are compared.

c. Visualization and Correct Prediction Count:
Using OpenCV, it displays the test image alongside the predicted digit. If the predicted digit matches the actual digit, it's considered a correct prediction, and the count increases.

d. Calculation and Display of Accuracy:
The accuracy is calculated as the ratio of correct predictions to the total number of test images and then displayed as a percentage.

5. **Main Function**
In the main function, the program begins by loading training and testing datasets. It then calls the TrainKNN function, a pivotal process where raw image data is transformed into a trained KNN model. The heart of evaluation lies in the EvaluateAccuracy function, which assesses the model's performance against test images and labels, providing real-time recognition displays and accuracy metrics. Finally, the main function concludes the program, encapsulating the entire process of digit recognition from data ingestion to model evaluation.

Results



```
Digit: 0            Using KNN: 0
Digit: 1            Using KNN: 1
Digit: 2            Using KNN: 2
Digit: 3            Using KNN: 3
Digit: 4            Using KNN: 4
Digit: 5            Using KNN: 5
Digit: 6            Using KNN: 6
Accuracy is indeed 96.89%
```

In the evaluation phase, the K-Nearest Neighbors (KNN) model demonstrated commendable performance, achieving an accuracy of 96.89%. This result signifies the model's proficiency in recognizing and classifying digits based on the provided test images.

Conclusion

The K-Nearest Neighbors (KNN) model obtained 96.89% accuracy in recognizing handwritten digits. This underscores the model's effectiveness in digit classification, confirming its suitability for diverse applications requiring accurate and efficient digit recognition.