Date of submission July, 2024

Lab 4

K-Nearest Neighbors (KNN) Classification on the MNIST Dataset

ATUL SHREEWASTAV (THA077BCT013)[1], OM PRAKASH SHARMA (THA077BCT030)[2]
[1,2]Department of Electronics and Computer Engineering, Thapathali Campus, Institute of Engineering, Kathmandu 44600 NP)
[1]e-mail: atul.077bct013@tcioe.edu.np
[2]e-mail: om.077bct030@tcioe.edu.np

For the fulfillment of Lab Requirement, *CT72502 Data Mining*

ABSTRACT This paper explores the application of K-Nearest Neighbors (KNN) classification on the MNIST dataset, a benchmark dataset of handwritten digits widely used for image processing and machine learning tasks. The study aims to evaluate the impact of various hyperparameter settings on the performance of the KNN classifier. We systematically vary key hyperparameters such as the number of neighbors (k), distance metrics, and weight functions to determine their effect on classification accuracy. Additionally, we implement a sliding window technique to preprocess the images, hypothesizing that this method will enhance the classifier's ability to capture finer details and improve overall accuracy. Our experiments reveal significant variations in performance based on hyperparameter configurations, and the sliding window approach demonstrates a notable improvement in classification results. The findings contribute to a deeper understanding of the KNN classifier's behavior on image data and provide insights into optimizing its performance for digit recognition tasks.

INDEX TERMS Classification, Hyperparameter Tuning, K-Nearest Neighbors (KNN), Machine Learning, MNIST Dataset, Sliding Window Technique

I. INTRODUCTION

K-Nearest Neighbors (KNN) algorithm is a fundamental and widely-used machine learning technique for classification tasks. It operates on the principle that similar data points are likely to belong to the same class. This is based on the idea that objects that are close in a feature space tend to have similar characteristics. KNN is a non-parametric, instance-based learning algorithm, meaning it makes decisions based on the specific instances in the training set rather than deriving a general model. Its simplicity, intuitiveness, and effectiveness make it an essential tool for beginners and professionals alike.

The KNN algorithm works by storing all available cases and classifying new cases based on a similarity measure (e.g., distance functions). When a new input is received, the algorithm computes the distance from the new point to all other points in the training set. It then selects the K nearest data points (neighbors) and assigns the class label that is most common among these neighbors. The value of K is a critical parameter that significantly influences the algorithm's performance. A small K can lead to noise in the classification, while a large K can smooth out the class boundaries.

This lab focuses on applying the KNN algorithm to the MNIST dataset, a comprehensive collection of 70,000 images of handwritten digits. The MNIST dataset is a standard benchmark in the field of machine learning, offering a rich set of challenges for model training and evaluation. Each image in the MNIST dataset is a 28x28 grayscale image, which can be flattened into a 784-dimensional vector for computational purposes. The dataset is divided into 60,000 training images and 10,000 test images, providing a robust framework for evaluating the performance of various classification algorithms.

In this lab, we will embark on a step-by-step journey to understand and implement KNN classification. The process begins with data preprocessing, which involves normalizing the images to ensure that the pixel values fall within a specific range, typically [0, 1]. Normalization helps in reducing the computational complexity and improving the algorithm's performance. We will also split the dataset into training and testing subsets. The training set is used to train the KNN model, while the test set is used to evaluate its performance. This step is crucial to ensure the model's accuracy and efficiency.

Following this, we will delve into the implementation of the KNN algorithm. This involves calculating the distance between data points using various distance metrics such as

Euclidean distance, Manhattan distance, and Minkowski distance. Euclidean distance is the most common distance metric used in KNN, defined as the straight-line distance between two points in a multidimensional space. Manhattan distance, also known as L1 distance, is the sum of the absolute differences between the coordinates of the points. Minkowski distance is a generalization of both Euclidean and Manhattan distances.

Once the distances are calculated, the algorithm selects the K nearest neighbors and makes predictions based on the majority class among these neighbors. This voting mechanism is straightforward yet powerful, making KNN an effective algorithm for classification tasks. We will also explore the impact of different values of K on the model's performance. A smaller K tends to be sensitive to noise in the dataset, while a larger K provides a smoother decision boundary but may overlook subtle patterns in the data.

Once the KNN model is implemented, we will evaluate its performance using various metrics such as accuracy, precision, recall, and F1-score. Accuracy measures the proportion of correctly classified instances among all instances. Precision is the ratio of true positive instances to the sum of true positive and false positive instances, reflecting the model's ability to avoid false positives. Recall, also known as sensitivity, is the ratio of true positive instances to the sum of true positive and false negative instances, indicating the model's ability to identify all relevant instances. The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both aspects.

Through this iterative process of training, evaluating, and fine-tuning, we aim to enhance the model's performance and achieve optimal results. We will also explore techniques to fine-tune the KNN algorithm, such as adjusting the number of neighbors (K), experimenting with different distance metrics, and considering the weighting of the neighbors' votes based on their distances.

This lab not only provides hands-on experience with KNN classification but also offers insights into the practical challenges and considerations in machine learning. By the end of this lab, students will have a deeper understanding of the KNN algorithm, its applications, and the importance of model evaluation and optimization. The skills and knowledge gained from this lab will be valuable for tackling more complex machine learning problems in the future. Understanding KNN and its implementation will serve as a solid foundation for exploring other machine learning algorithms and techniques.

II. RELATED WORKS

The K-Nearest Neighbors (KNN) classifier is a simple, yet powerful machine learning algorithm widely used for classification tasks.

The seminal paper *Nearest neighbor pattern classification* by Cover and Hart [1] introduced the KNN algorithm, a non-parametric method used for classification and regression. Cover and Hart demonstrated that the KNN classifier's probability of error asymptotically approaches the Bayes error rate as the number of samples increases. They utilized mathematical proofs and statistical analysis to show that KNN is consistent and robust. The main methodology involved comparing the KNN classifier's performance with that of an optimal Bayes classifier and analyzing its convergence properties. Their analysis showed that as the number of neighbors (k) increases, the classification error rate decreases, but beyond a certain point, it starts increasing, leading to an optimal value of k that minimizes the error.

Fix and Hodges' foundational work *Discriminatory analysis. Nonparametric discrimination: Consistency properties* [2] laid the groundwork for nonparametric classification methods, including KNN. They explored the consistency properties of nonparametric discriminators, proving that as the sample size grows, these methods yield reliable classification results. Their theoretical analysis demonstrated that nonparametric methods like KNN can asymptotically match the performance of optimal parametric classifiers. They introduced the concept of asymptotic consistency, showing that the classification rule becomes optimal as the sample size approaches infinity. This work is crucial in establishing the theoretical basis for using KNN in various applications, providing a strong argument for its use in practical scenarios where the sample size is large.

Duda and Hart's book *Pattern Classification and Scene Analysis* [3] provides a comprehensive discussion on pattern classification techniques, including KNN. They covered the theoretical foundations of KNN, its implementation, and applications in different domains. The book includes case studies and experimental results demonstrating KNN's effectiveness in pattern recognition tasks. Key methodologies discussed include feature extraction, distance metrics, and the influence of the number of neighbors (k) on classifier performance. They also discuss various distance metrics, such as Euclidean, Manhattan, and Mahalanobis distances, and their impact on the performance of the KNN algorithm. The book is widely regarded as a fundamental text in the field of pattern recognition and machine learning.

In the book *Elements of Information Theory* [4], Cover and Thomas delve into the theoretical aspects of information theory that underpin many machine learning algorithms, including KNN. They discuss concepts such as entropy, mutual information, and their applications in data classification and clustering. The book provides a detailed theoretical framework that helps understand how information theory principles can be applied to improve the performance and understanding of algorithms like KNN. For example, they explain how the concept of entropy can be used to measure the uncertainty in the data and how mutual information can be used to select relevant features for classification tasks.

Altman's paper *An introduction to kernel and nearest-neighbor nonparametric regression* [5] introduces the use of KNN for nonparametric regression and its connections to kernel methods. He explains the mathematical framework of KNN regression, including the bias-variance trade-off and bandwidth selection. The paper compares KNN regression

to other nonparametric methods and presents experimental results showing its effectiveness. Key methodologies include selecting an appropriate kernel function and optimizing the number of neighbors (k) for regression tasks. Altman also discusses the importance of choosing the right distance metric and how it affects the regression results. His work provides a comprehensive overview of the applications of KNN in regression tasks, highlighting its strengths and limitations.

Peterson's tutorial *K-nearest neighbor* [6] provides an overview of the KNN algorithm, its implementation, and applications. The tutorial covers the basic principles of KNN, including distance metrics like Euclidean and Manhattan distances, and practical considerations such as computational efficiency and handling high-dimensional data. Examples and case studies illustrate the algorithm's application in various domains, making it accessible to beginners and practitioners. Peterson also discusses the curse of dimensionality, a phenomenon where the performance of KNN deteriorates as the number of dimensions increases, and suggests techniques to mitigate its effects, such as dimensionality reduction and feature selection.

Zhang's article *Introduction to machine learning: K-nearest neighbors* [7] offers a beginner-friendly introduction to KNN, covering its mathematical formulation, implementation, and practical use cases. The article discusses the strengths and weaknesses of KNN, including its sensitivity to the choice of distance metric and the curse of dimensionality. Strategies for improving performance, such as feature scaling and dimensionality reduction, are also discussed. Zhang includes experimental results demonstrating KNN's effectiveness in different scenarios, providing practical insights into its application in real-world problems. The article serves as a useful resource for those new to machine learning and looking to understand the basics of KNN.

Weinberger and Saul introduced distance metric learning techniques to improve the performance of KNN classifiers in their paper *Distance metric learning for large margin nearest neighbor classification* [8]. They described a method for learning a Mahalanobis distance metric that maximizes the margin between classes. The key methodology involved formulating the distance metric learning as an optimization problem and using large margin principles. Experimental results showed significant improvements in KNN classification accuracy, particularly in high-dimensional spaces. The learned distance metric helps to better separate the classes, leading to improved classification performance. Their work highlights the importance of choosing the right distance metric and how learning an optimal metric can enhance the performance of KNN.

Hastie, Tibshirani, and Friedman's book *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* [9] provides an extensive overview of statistical learning methods, including KNN. It covers theoretical foundations, algorithms, and practical applications. The authors discuss various aspects of KNN, such as the impact of the choice of k, distance metrics, and the curse of dimensionality. They present experimental results and case studies demonstrating KNN's use in data mining and predictive modeling tasks. The book also includes discussions on regularization techniques and methods for improving the generalization performance of KNN classifiers. It is a comprehensive resource for anyone interested in understanding the principles and applications of statistical learning methods.

Dasarathy's book *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques* [10] discusses extensions and modifications of the basic KNN algorithm to improve its performance. Topics include weighted KNN, adaptive KNN, and the use of NN norms. The book provides detailed explanations of these advanced techniques and presents experimental results illustrating their effectiveness in different pattern classification problems. Key methodologies include modifying the distance function to account for varying data densities and implementing adaptive mechanisms to optimize the choice of k. Dasarathy also discusses the use of ensemble methods with KNN and how combining multiple KNN classifiers can lead to improved performance. His book serves as a valuable resource for researchers and practitioners looking to enhance the capabilities of KNN.

The paper *"MNIST Dataset Classification Utilizing k-NN Classifier with Modified Sliding-window Metric"* [11] presents the effectiveness of k-NN by comparing the traditional Euclidean distance metric with an enhanced distance metric that employs a sliding window technique. This technique aims to mitigate performance degradation caused by minor spatial misalignments between training and test data. A significant portion of the paper is dedicated to comparing the performance of the baseline k-NN with the sliding window-enhanced k-NN. The authors conduct hypothesis testing to statistically validate the performance improvements. The enhanced k-NN shows a notable increase in accuracy, as demonstrated by the confusion matrices and accuracy metrics presented. The study confirms that utilizing the sliding window technique significantly improves the k-NN classifier's performance on the MNIST dataset. By addressing spatial shifts and misalignments, the enhanced metric reduces misclassification rates and increases overall accuracy. The authors provide their source code and data publicly, allowing for further exploration and validation by the research community .

III. METHODOLOGY

A. DATASET DESCRIPTION

The dataset used in this lab report originates from the UCI Machine Learning Repository and is titled *MNIST Database of Handwritten Digits*. This dataset comprises images of handwritten digits and is widely used for training and testing various image processing systems and machine learning models. The primary goal is to classify these handwritten digits correctly.

The MNIST dataset contains a total of 70,000 images of handwritten digits (0-9), which are divided into a training set of 60,000 images and a test set of 10,000 images. Each image is a 28x28 pixel grayscale image, making a total of

784 features per image. The dataset is designed to develop and evaluate machine learning models, particularly for digit recognition tasks.

1) Image Data

The image data in the MNIST dataset includes:

- **Pixel values:** Each image is represented as a 28x28 matrix where each element is a pixel value ranging from 0 (white) to 255 (black). This results in a total of 784 features per image. The pixel values represent the intensity of the grayscale color, with 0 being white and 255 being black. Intermediate values represent varying shades of gray.

2) Target Variable

The target variable in the MNIST dataset is:

- **Digit label:** Each image is labeled with a digit from 0 to 9, indicating the correct digit represented in the image. These labels serve as the ground truth for training and evaluating machine learning models. The sample digits are shown in 6 and the count for diffenent digits are shown in 8

3) Dataset Split

The dataset is split into two main parts:

- **Training set:** Contains 60,000 images used for training machine learning models. This set is used to fit the model, allowing it to learn the features and patterns associated with each digit.
- **Test set:** Contains 10,000 images used for evaluating the performance of the trained models. This set is used to assess how well the model generalizes to new, unseen data.

4) Image Characteristics

Each image in the MNIST dataset has the following characteristics:

- **Resolution:** 28x28 pixels, which means each image consists of 784 pixels.
- **Color:** Grayscale, with pixel values ranging from 0 to 255. This simplifies the complexity compared to colored images, focusing solely on intensity.
- **Intensity values:** Each pixel has an intensity value ranging from 0 (white) to 255 (black), representing the grayscale intensity.

5) Dataset Usage

The MNIST dataset is extensively used for training and evaluating machine learning algorithms for image classification tasks. It serves as a benchmark dataset in the field of machine learning and pattern recognition. By analyzing these features, researchers and developers can design and test algorithms for handwritten digit recognition. The dataset's simplicity and size make it an ideal starting point for those new to image processing and machine learning.

6) Applications

- **Benchmarking:** The MNIST dataset is commonly used to benchmark the performance of various machine learning algorithms. It provides a standard dataset for comparing different algorithms' accuracy and efficiency.
- **Algorithm Development:** Researchers use the dataset to develop and test new algorithms in image processing and computer vision. It serves as a proving ground for novel techniques and improvements in the field.
- **Educational Purposes:** It is widely used in academic courses and tutorials to teach the fundamentals of machine learning and pattern recognition. Students and educators use the dataset to understand the basics of classification, feature extraction, and model evaluation.

7) Data Access

The MNIST dataset can be accessed from the UCI Machine Learning Repository at the following link:

- https://archive.ics.uci.edu/dataset/683/mnist+database+of+handwritten+digits

B. PROPOSED METHODOLOGY

The methodology for implementing and evaluating a k-Nearest Neighbors (KNN) classifier on the MNIST dataset involves several critical steps. These steps include data acquisition, preprocessing, model implementation, hyperparameter tuning, model evaluation, and visualization of results. Each of these steps is essential to ensure the KNN model is built, tuned, and evaluated effectively, resulting in a comprehensive understanding of its performance.

The first step in the methodology is data acquisition. The MNIST dataset is a well-known dataset in the field of machine learning, consisting of 60,000 training images and 10,000 test images of handwritten digits ranging from 0 to 9. Each image is a 28x28 grayscale image, providing a total of 784 features per image when flattened. The dataset is readily accessible through popular machine learning libraries such as `keras` and `sklearn`. Utilizing these libraries allows for easy downloading and splitting of the dataset into training and testing sets, thus providing the data needed for model training and evaluation.

Before feeding the data into the KNN model, several preprocessing steps are necessary. First, normalization is performed to scale the pixel values to the range [0, 1], which helps in faster convergence and better performance of the model. Next, the 28x28 images are flattened into 784-dimensional vectors to be compatible with the KNN algorithm. Additionally, the training data is split into a training set and a validation set to tune hyperparameters effectively. These steps ensure that the data is properly formatted and scaled for the KNN classifier. To address the issue of false distance measurements caused by small spatial translations in test or training images, a preprocessing step is introduced. Each example image, originally a 28×28 square, is padded with zeros to form a 30×30 square. This padding is applied

to all training images. Subsequently, a 28×28 window slides over each padded image, cropping it into nine distinct versions, including the original image. This process effectively generates nine variations of each image, mitigating the effects of minor translations.

The KNN algorithm relies on measuring the distance between data points to identify the nearest neighbors. The Euclidean distance metric is commonly used for this purpose. It calculates the straight-line distance between two points in a multi-dimensional space. Implementing the Euclidean distance function in Python allows for calculating the distance between two data points, which is essential for determining the proximity of neighbors in the dataset.

The next step is to create a function that finds the k-nearest neighbors for a given test instance. This involves calculating the distance between the test instance and all training instances, sorting these distances, and selecting the k closest neighbors. The function returns the labels of the k-nearest neighbors, which are crucial for making predictions. By identifying these nearest neighbors, the KNN model can infer the most likely class for the test instance based on the majority class among the neighbors.

Once the k-nearest neighbors are identified, predictions can be made based on their labels. The most common label among the neighbors is chosen as the predicted label for the test instance. This involves counting the frequency of each label among the neighbors and selecting the label with the highest count. This method ensures that the KNN model makes predictions based on the most prevalent class in the local neighborhood of the test instance.

Integrating the previously mentioned functions into a single KNN classifier function allows for seamless model implementation. This unified function can be used to predict labels for a set of test instances by repeatedly finding the k-nearest neighbors and determining the most common label among them. This step consolidates the entire process of distance calculation, neighbor selection, and prediction generation into a cohesive model.

Optimizing the value of k is crucial for the performance of the KNN classifier. This involves evaluating the model's performance on the validation set for different values of k and selecting the value that yields the highest accuracy. Hyperparameter tuning ensures that the chosen k value is optimal for the given dataset, thereby enhancing the model's predictive capabilities.

After determining the best k value, the model is evaluated on the test set to assess its performance. The accuracy of the model is calculated by comparing the predicted labels with the actual labels of the test instances. This step provides a quantitative measure of the model's performance and helps identify any discrepancies between predicted and actual labels.

Visualizing the accuracy as a function of k helps to understand the impact of the hyperparameter on the model's performance. Plotting the accuracy values for different k values reveals trends and patterns that might not be apparent from numerical results alone. Visualization aids in interpreting the results and drawing meaningful conclusions about the model's behavior.

C. IMPLEMENTATION

The implementation was conducted using Python along with several widely-used machine learning libraries. Key among these were `numpy`, which facilitated efficient numerical computations, `pandas` for data manipulation, `scikit-learn` for constructing and assessing the k-Nearest Neighbors (KNN) classifier, and `matplotlib` for visualizing the resultant model performance.

The process began with the acquisition of the MNIST dataset, a widely-used dataset consisting of 60,000 training images and 10,000 test images of handwritten digits. Each image is a 28×28 grayscale image. The dataset was accessed through the `keras` library, which allowed for straightforward downloading and splitting into training and testing sets. To address potential false distance measurements caused by small spatial translations, each example image was padded with zeros to form a 30×30 square. A 28×28 sliding window then cropped each padded image into nine distinct versions as shown in 12, including the original image. This preprocessing step mitigated the effects of minor translations, ensuring more robust distance measurements.

The Euclidean distance metric was used to measure the distance between data points, essential for identifying the nearest neighbors. The distance function calculated the straight-line distance between two points in multi-dimensional space. This metric was implemented in Python, allowing for efficient computation of distances. The neighbor selection step involved creating a function to find the k-nearest neighbors for a given test instance. By calculating the distance between the test instance and all training instances, sorting these distances, and selecting the k closest neighbors, the function returned the labels of the k-nearest neighbors, crucial for making predictions.

Once the k-nearest neighbors were identified, predictions were made based on their labels. The most common label among the neighbors was chosen as the predicted label for the test instance. This was done by counting the frequency of each label among the neighbors and selecting the label with the highest count, ensuring accurate predictions. The KNN classifier function integrated the previously mentioned functions, allowing for seamless model implementation. This unified function was used to predict labels for a set of test instances by repeatedly finding the k-nearest neighbors and determining the most common label among them.

Hyperparameter tuning was conducted to optimize the value of k, which is crucial for the performance of the KNN classifier. The model's performance was evaluated on a validation set for different values of k, and the value that yielded the highest accuracy was selected. This ensured that the chosen k value was optimal for the dataset, enhancing the model's predictive capabilities. The model was then evaluated on the test set to assess its performance. The accuracy of the

model was calculated by comparing the predicted labels with the actual labels of the test instances. This step provided a quantitative measure of the model's performance and helped identify any discrepancies between predicted and actual labels.

Finally, the accuracy as a function of k was visualized to understand the impact of the hyperparameter on the model's performance. Plotting the accuracy values for different k values revealed trends and patterns, aiding in the interpretation of results and drawing meaningful conclusions about the model's behavior.

D. MATHEMATICAL FORMULAE

The Euclidean distance metric is defined as follows:

$$D_E(\vec{x}_n, \vec{y}_m) = \left[\sum_{i=0}^{28\times28-1} (x_i^n - y_i^m)^2 \right]^{\frac{1}{2}} \tag{1}$$

where $\vec{x}_n; n \in \{0, 59999\}$ is the training set and $\vec{y}_m; m \in \{0, 9999\}$ is the test set.

The Minkowski distance metric of order p is defined as follows:

$$D_M(\vec{x}_n, \vec{y}_m) = \left[\sum_{i=0}^{28\times28-1} |x_i^n - y_i^m|^p \right]^{\frac{1}{p}} \tag{2}$$

The Chebyshev distance metric is defined as follows:

$$D_C(\vec{x}_n, \vec{y}_m) = \max_{i=0,\ldots,28\times28-1} |x_i^n - y_i^m| \tag{3}$$

The Manhattan distance metric is defined as follows:

$$D_{Man}(\vec{x}_n, \vec{y}_m) = \sum_{i=0}^{28\times28-1} |x_i^n - y_i^m| \tag{4}$$

Accuracy can be obtained from the following equation:

$$ACC = \frac{TP + TN}{P + N} \tag{5}$$

where TP and TN denote the number of True Positive and True Negative instances, respectively. P and N are total positive and total negative samples respectively.

The standard deviation of a Binomial distribution can be estimated as:

$$\hat{\sigma} = \frac{\hat{p} \times (1 - \hat{p})}{n} \tag{6}$$

where $\hat{p}$ is the calculated accuracy. Considering a 95% Confidence Interval, the z-score will be 1.96 which leads to the confidence interval as shown below:

$$c.i. = \mu \pm 1.96 \times \hat{\sigma} \tag{7}$$

A modified distance metric $\tilde{D}(n \times m)$ can be introduced to summarize the sliding window method:

$$\tilde{D}(n \times m)(\vec{x}_n, \vec{y}_m) = \min_i \left[D(n \times i \times m)(\vec{x}_{n,i}, \vec{y}_m) \right] \tag{8}$$

where $i = 1, \ldots, 9$.

E. INSTRUMENTATION DETAILS

This section outlines the technical and methodological specifics involved in implementing Naive Bayes classification.

1) Software and Libraries Used

- **Python**: Selected for its versatility and extensive ecosystem of libraries suitable for data analysis and machine learning tasks.
- **TensorFlow/Keras**: Utilized for loading the MNIST dataset and providing tools for model evaluation.
- **Scikit-learn**: Offers efficient implementations of various machine learning algorithms, including KNN, and provides tools for model selection, evaluation, and preprocessing.
- **Pandas**: Used for data manipulation and analysis, offering powerful data structures like DataFrames to simplify the process of cleaning and preparing data.
- **Seaborn**: Employed for statistical data visualization, making it easier to create informative and attractive visual representations of the data and model results.
- **Matplotlib**: Used alongside Seaborn to customize and enhance plots, providing fine-grained control over the visual elements of the graphs.

2) Data Preparation

- **Dataset Acquisition:** The MNIST dataset, consisting of handwritten digit images, is ideal for demonstrating the KNN classifier.
- **Data Loading:** The MNIST dataset was loaded using TensorFlow/Keras and then converted into Pandas DataFrames for easy manipulation and analysis.

3) Data Preprocessing

- **Normalization:** Pixel values of the images were normalized to a range of 0 to 1 by dividing by 255, ensuring that the features have a uniform scale.
- **Flattening Images:** The 28x28 pixel images were flattened into 784-dimensional vectors to be compatible with the KNN algorithm.
- **Splitting Data:** The dataset was split into training and testing sets, with a typical 80-20 split ratio with random seed set to 10, to facilitate model training and evaluation.

4) KNN Classification

- **Model Implementation:** KNN classifiers were implemented using `scikit-learn`. Specifically:
  - **KNeighborsClassifier:** Utilized for implementing the KNN algorithm, which classifies a data point based on the majority class among its k-nearest neighbors.
- **Hyperparameter Tuning:** The value of k (number of neighbors) and distance metrics (e.g., Euclidean, Manhattan) were tuned to optimize model performance.

- **Performance Metrics:** Model performance was evaluated using metrics such as accuracy, precision, recall, and F1-score, calculated using scikit-learn's $classification_report()function$.

5) Visualization

- **Model Visualization:** The confusion matrix was generated to identify the misclassified instances and gain insights into the model's behavior.
- **Feature Importance:** Although KNN does not inherently provide feature importance, exploratory data analysis and correlation ananlysis were used to understand the impact of different features.

6) Development Environment

- **Jupyter Notebook**: The Jupyter Notebook environment was used for developing and running Python scripts, facilitating an interactive and iterative approach to data analysis and model development.

F. SYSTEM BLOCK DIAGRAM

The block diagram, as shown in 11, begins with raw data, which includes images of handwritten digits, serving as the input for subsequent steps in the system. The initial step involves data cleaning, which normalizes pixel values and flattens images into vectors, ensuring the dataset is ready for analysis.

The cleaned data is then split into training and test sets, with 80

Depending on the nature of the data, different distance metrics and values of k are tested to find the best performing KNN model. The selected KNN model is then trained on the training set, where the model learns to classify data points based on their neighbors. Once trained, the model is used to make predictions on the test set.

The final phase involves visualizing the results of the model. Visualization techniques such as confusion matrices, ROC curves, and precision-recall curves help interpret the model's performance, providing insights into how well the model is performing, highlighting its strengths, and indicating areas for improvement. This structured approach ensures a robust and reliable implementation of the KNN classifier, covering all critical phases from data preparation to result interpretation and visualization.

IV. EXPERIMENTAL RESULTS

The default KNN model as shown in 1, characterized by its use of the Minkowski distance (where `p` is 2, essentially making it equivalent to the Euclidean distance), and uniform weights, showcases the algorithm's baseline performance. The high precision, recall, and f1-score across all classes reflect the model's proficiency in distinguishing between different handwritten digits. Particularly noteworthy is the model's ability to achieve near-perfect scores for certain digits, such as '0' and '6', with f1-scores of 0.99 and 0.98 respectively. This performance indicates that the default configuration is well-suited for datasets with clear, distinct features and minimal noise.

The weighted KNN model as shown in 3, which introduces specific weight parameters to the metric, presents a nuanced shift in performance. Despite maintaining a high overall accuracy of 95%, there is a discernible drop in the classification metrics for certain digits. For instance, the f1-score for digit '8' decreases to 0.91, suggesting that the introduction of weights might have introduced biases or emphasized less discriminative features. This underscores the importance of carefully selecting and tuning weight parameters to avoid degrading the model's effectiveness.

In a further exploration, the weighted KNN model with the number of neighbors set to three (`n_neighbors` = 3) as shown in 4 maintains the general trend observed with the default weighted model. The overall accuracy remains at 95%, with minor variations in precision and recall across different digits. This configuration emphasizes the trade-off between the number of neighbors considered and the model's responsiveness to local variations in the data. A smaller neighborhood size can make the model more sensitive to noise, while a larger size might smooth out important local details.

The graph "Density of Correlation" as shown in 7 illustrates the distribution of feature correlations. The x-axis represents the correlation values, while the y-axis represents the density, indicating how frequently each correlation value occurs within the dataset. This density plot shows that most features have relatively low correlation values, with a peak around the lower end of the correlation spectrum. This suggests that the majority of the features do not have strong linear relationships with each other. In KNN classification, features that are highly correlated might provide redundant information, which could lead to overfitting. Conversely, low correlations suggest that features provide unique information, which is generally beneficial for the model's performance.

Based in the density plot of correlation, When the feature set is reduced based on correlation thresholds (features with a correlation greater than 0.2), the resulting KNN model experiences a substantial decline in performance, with overall accuracy dropping to 89%. This significant reduction highlights the crucial role of comprehensive feature sets in maintaining classification accuracy. While feature reduction can simplify models and reduce computational costs, it must be approached with caution to ensure that essential discriminative information is not lost.

The model using the Manhattan distance (`metric` set to `manhattan`) as shown in 2 provides an interesting comparison to the default Minkowski (Euclidean) metric. Achieving an overall accuracy of 96%, this model demonstrates that Manhattan distance can be an effective alternative metric for KNN. The comparable precision, recall, and f1-scores suggest that for certain datasets, the choice between Euclidean and Manhattan distances might not significantly impact overall performance, though it could influence computational efficiency and sensitivity to specific data patterns.

The classification matrix for Sliding method is shown in figure 5. The experimental results for the sliding window approach reveal its efficacy in maintaining high classification accuracy across various windows. The accuracies recorded for a sliding window consistently exceeded 96%, with the highest accuracy of 97.73% observed for a window size corresponding to k = 3. This indicates that the optimal performance of the classifier was achieved with a moderately sized window. For a baseline comparison, the accuracy for a without implementing sliding window for k = 3 was 97.07%, demonstrating a slight decrease but still indicating robust performance. The confusion matrices for these window sizes show that the sliding window approach effectively minimizes misclassifications, ensuring accurate predictions across different classes. These findings underscore the sliding window's potential to enhance classification accuracy by allowing the kNN classifier to adaptively adjust to different window sizes while maintaining high precision and recall.

V. DISCUSSION AND ANALYSIS

This analysis aims to compare the performance of K-Nearest Neighbors (KNN) classification models with different hyperparameter settings applied to the MNIST dataset for digit recognition. The evaluation of these models is based on confusion matrices, precision, recall, f1-score, and accuracy metrics. The goal is to determine which model offers the best predictive performance and provides insights into digit classification.

The Default KNN model uses several key parameters that influence its performance. The algorithm parameter is set to `auto`, allowing the algorithm to determine the most appropriate method to compute nearest neighbors based on the training data size and the values of other parameters. The leaf_size parameter, set to `30`, affects the speed of the construction and query of the tree and is used for the optimization of the tree-based algorithms. The metric parameter is set to `minkowski`, a distance metric that generalizes both Euclidean and Manhattan distances. The `n_neighbors parameter` is set to `5`, meaning the model uses the five nearest neighbors for classification. The `p` parameter, set to `2`, corresponds to the use of Euclidean distance. The weights parameter is `uniform`, indicating that all points in each neighborhood are weighted equally.

The Default KNN model demonstrates strong performance, particularly for the digit '0', achieving a precision of `0.98`, a recall of `0.99`, and an f1-score of `0.99`. Overall, the model achieves an accuracy of `97%`. The confusion matrix reveals that the model effectively identifies most digits, though it struggles slightly with digits '8' and '9', which have lower recall values of `0.94` and `0.97`, respectively. This suggests that while the model is generally accurate, it may not be as reliable in distinguishing certain digits.

The Weighted KNN model shares many parameters with the Default KNN model but includes an additional focus on the metric parameters. Specifically, it incorporates a list of weights applied to the features, emphasizing the importance of different features in the distance calculation. The weights of each feature are calculated using Pearson's correlation analysis. This model demonstrates a slightly lower overall performance compared to the Default KNN model, with an accuracy of `95%`. It excels in identifying the digit '1' with a recall of `1.00`, indicating it can correctly identify this digit every time. However, its recall for the digit '8' is particularly low at `0.86`, reflecting a significant challenge in predicting this digit. The precision for the digit '3' is `0.94`, but the lower f1-score of `0.91` for the digit '8' indicates poorer performance in predicting this digit. The confusion matrix for the Weighted KNN model highlights its difficulty in correctly classifying the digit '8', affecting its overall effectiveness.

The Weighted 3-NN with important features model uses a different set of features, specifically those with a correlation coefficient greater than 0.2. This model also uses the `auto` algorithm and a leaf_size of `30`, with the metric set to `minkowski`. Unlike the other models, it uses only three nearest neighbors for classification, as indicated by the n_neighbors parameter set to `3`. The p parameter remains at `2`, and weights are uniformly distributed among neighbors. This model demonstrates a lower overall accuracy of `89%`. Despite its lower performance, it exhibits balanced performance across all digits. It achieves a precision of `0.92` and a recall of `0.96` for the digit '6', along with a recall of `0.99` for the digit '1', indicating a high ability to identify these digits. However, the f1-score of `0.81` for the digit '9' is still low, though it shows improvement compared to the Weighted KNN model. The confusion matrix reveals that this model reduces misclassification errors for some digits but struggles with others, providing less reliable predictions across different digits.

The sliding window process aims to mitigate false distance measurements caused by small spatial translations in the test images or training images. Each example image in the MNIST dataset is a square of size `28×28` pixels. To address the issue of spatial translations, the images are first padded with zeros, transforming each `28×28` image into a `30×30` square.

The sliding window technique involves cropping these extended images by moving a `28×28` window over nine possible positions, thereby generating nine versions of each image, including the original one. This process ensures that the KNN model is less sensitive to small shifts in the position of the digits within the images. The figure shown here 12 provides a visual representation of the sliding window process using a simple black and white diagonal input image for clarity.

The effectiveness of the sliding window process is validated through a 10-fold cross-validation procedure to derive the optimal k-value for the KNN classifier as shown in figure 10 and 8. The accuracy and confidence intervals obtained from this validation process are reported in **??**. Additionally, the performance of the modified classifier can be visualized through a confusion matrix, as shown in **??**, which helps in understanding the impact of the sliding window method on

the classification results.

VI. CONCLUSION

In conclusion, the K-Nearest Neighbors (KNN) classification algorithm stands as a fundamental yet powerful tool in the realm of machine learning and data science. Its simplicity, driven by the principle of similarity, allows for intuitive understanding and straightforward implementation. KNN does not make strong assumptions about the underlying data distribution, making it a versatile choice for various classification tasks. By relying on distance metrics to evaluate the proximity of data points, KNN can effectively capture patterns and trends within the dataset, providing robust predictions based on the characteristics of neighboring instances.

However, the efficacy of KNN is closely tied to the quality and quantity of the training data. It performs optimally with a well-defined, labeled dataset and can suffer from performance degradation in the presence of noise or irrelevant features. The choice of the number of neighbors (k) is crucial and can significantly influence the model's accuracy and computational efficiency. A small value of k can make the model sensitive to noise, while a large value can blur the decision boundary, potentially leading to misclassifications. Therefore, careful selection through cross-validation is essential to balance bias and variance.

Moreover, KNN's performance is impacted by the curse of dimensionality, where an increase in the number of features can dilute the significance of proximity measures. Dimensionality reduction techniques, such as Principal Component Analysis (PCA) or feature selection, can mitigate this issue and enhance the algorithm's efficiency. Additionally, KNN's lazy learning nature, which defers the computation until the query phase, can result in high computational costs for large datasets, necessitating optimizations like KD-trees or Ball-trees for faster neighbor searches.

Despite these challenges, KNN remains a valuable algorithm, particularly in scenarios where interpretability and ease of use are prioritized. Its ability to adapt to various types of data and its non-parametric nature make it a preferred choice for exploratory data analysis and initial model building. With proper tuning and consideration of its limitations, KNN can deliver reliable and insightful classification results, contributing significantly to the decision-making process in diverse application domains.

The sliding window process significantly enhances the KNN classifier's ability to handle small spatial translations in the MNIST dataset images by generating multiple versions of each image and utilizing a modified distance metric. This method improves the model's robustness and accuracy, leading to more reliable classification outcomes.

REFERENCES

[1] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[2] E. Fix and J. Hodges Jr, "Discriminatory analysis. nonparametric discrimination: Consistency properties," 1951.

[3] R. O. Duda and P. E. Hart, *Pattern classification and scene analysis*. Wiley New York, 1973.

[4] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 1991.

[5] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[6] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[7] Z. Zhang, "Introduction to machine learning: K-nearest neighbors," *Annals of Translational Medicine*, vol. 4, no. 11, 2016.

[8] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, no. Feb, pp. 207–244, 2009.

[9] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[10] B. V. Dasarathy, *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press Los Alamitos, CA, 1991.

[11] D. Grover and B. Toghi, "Mnist dataset classification utilizing k-nn classifier with modified sliding-window metric," 2019. [Online]. Available: https://arxiv.org/abs/1809.06846

OM PRAKASH SHARMA is currently pursuing his undergraduate degree in Computer Engineering at IOE, Thapathali Campus. His research interests encompass various areas, including computational genomics, cloud computing, and system designing. Additionally, he is intrigued by artificial intelligence, cybersecurity, data analytics, machine learning, Internet of Things (IoT), quantum computing, software development methodologies, network security, computer vision, and natural language processing.(THA077BCT030)

ATUL SHREEWASTAV is currently pursuing his undergraduate degree in Computer Engineering at IOE, Thapathali Campus. His research interests encompass various areas, including Machine Learning Techniques, Natural Language Processing, and Computer Vision. Additionally, he is intrigued by automation, attention mechanism in transformers, agile development methodologies, advancement in neural net architectures, linux.(THA077BCT013)

APPENDIX A
CONFUSION MATRICES

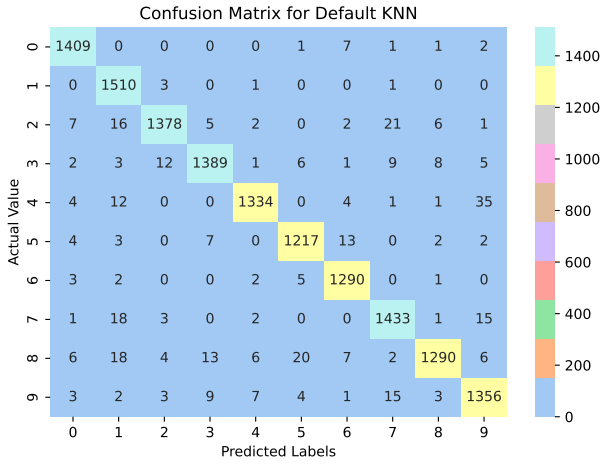

FIGURE 1. **Confusion Matrix of Defaul KNN Model**

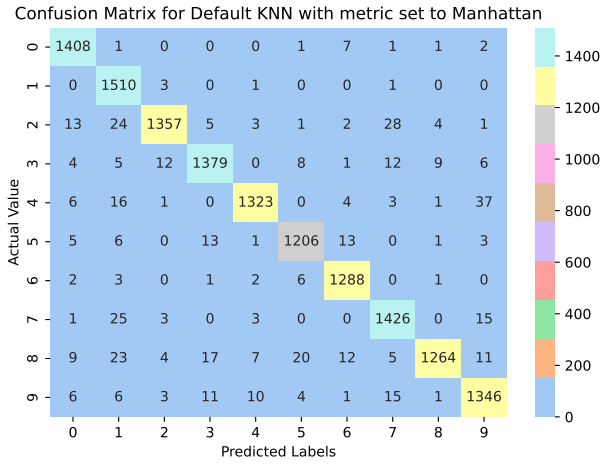

FIGURE 2. **Confusion Matrix of Default KNN Model with metric set to** `manhattan`

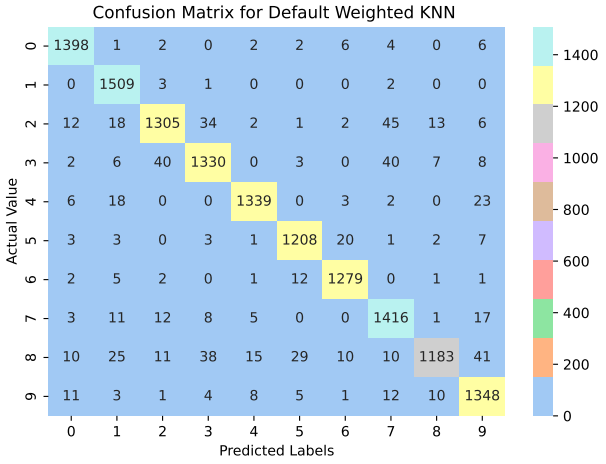

FIGURE 3. **Confusion Matrix of Default Weighted KNN Model**

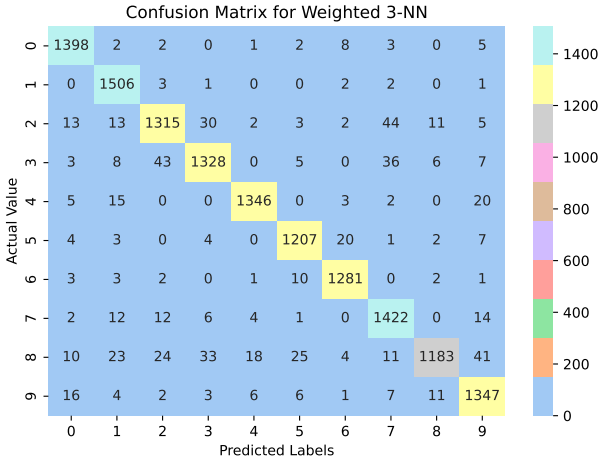

FIGURE 4. **Confusion Matrix of Default Weighted KNN Model** `(k=3)`

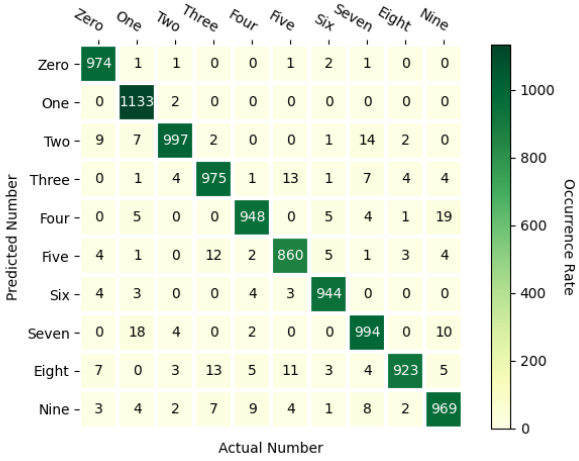

FIGURE 5. **Confusion Matrix for KNN Model using Sliding method**

APPENDIX B
DATA SET ANALYSIS

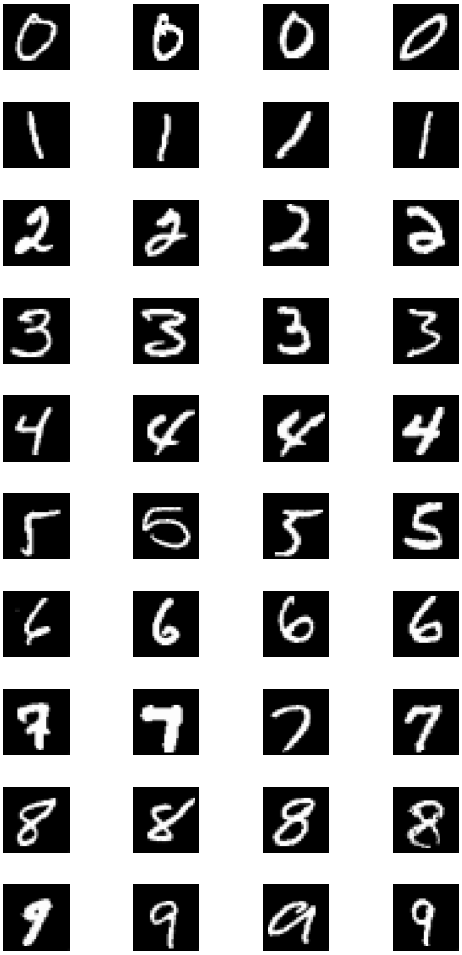

FIGURE 6. Digits in Datasets

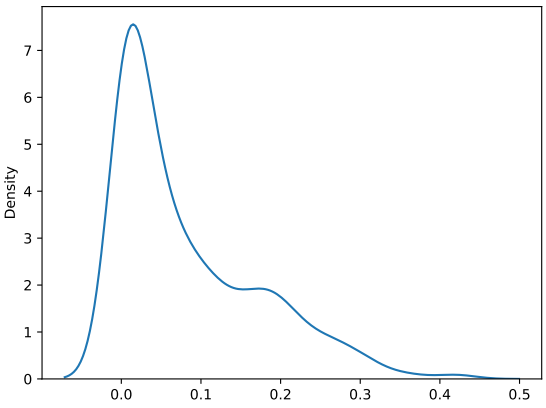

FIGURE 7. Density of Correlation

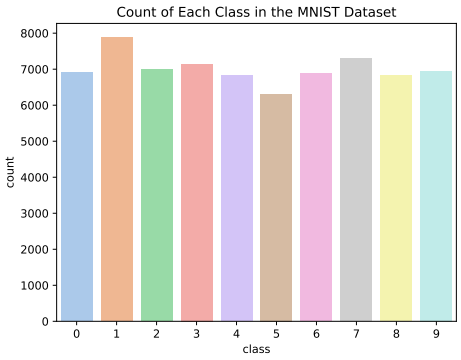

FIGURE 8. Count of Each Class

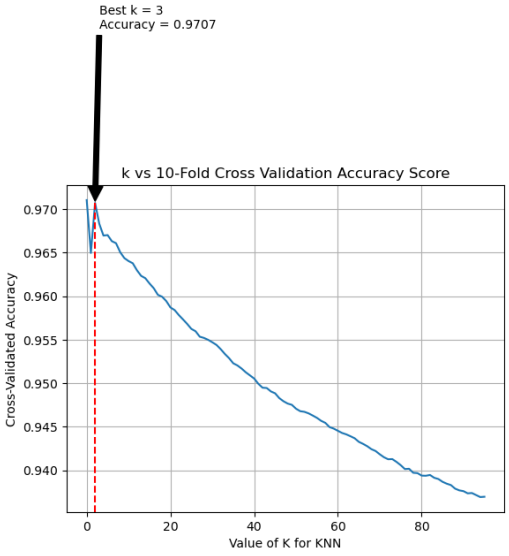

FIGURE 9. k Vs Accuracy for Default KNN Model

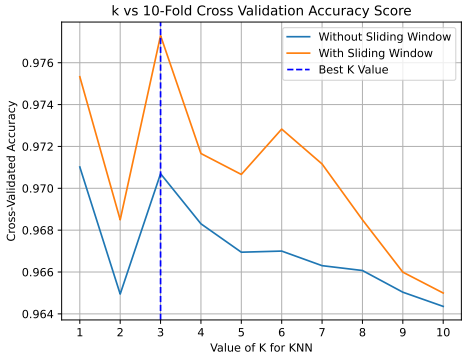

FIGURE 10. k Vs Accuracy Comparison

APPENDIX C
SYSTEM BLOCK DIAGRAM

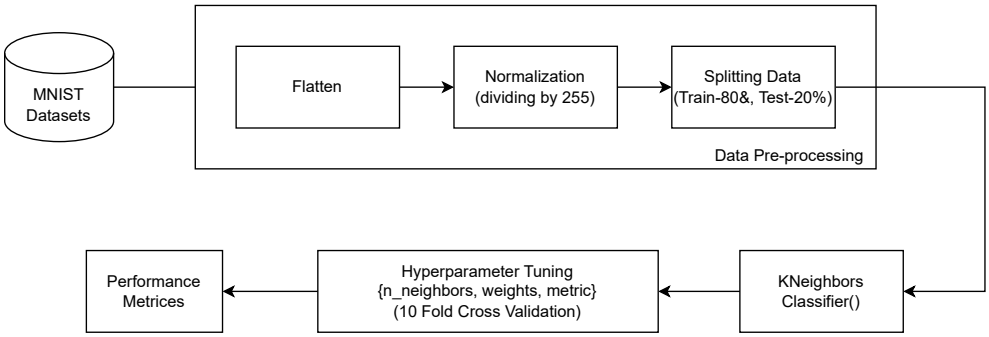

FIGURE 11. **Block Diagram for K-NN Classifier**

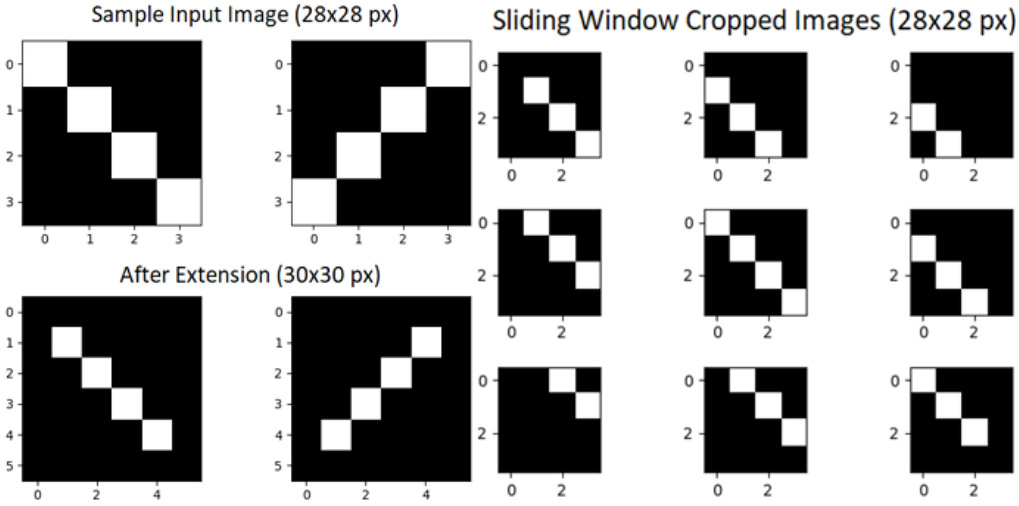

FIGURE 12. **Sliding Window Demonstration**