Advancing Handwritten Digit Recognition: Machine Learning Techniques on the MNIST Dataset

1st Monica Suresh
Stevens Institute of Technology
Hoboken, USA
msuresh@stevens.edu

2nd Syed Aziz
Stevens Institute of Technology
Hoboken, USA
saziz2@stevens.edu

3rd Sarah Thuman
Stevens Institute of Technology
Hoboken, USA
sthuman@stevens.edu

***Abstract*—This study addresses the complex challenge of handwritten digit recognition, essential for automated processes in postal mail sorting, bank check processing, and form data entry. Utilizing the MNIST dataset, which comprises 60,000 training and 10,000 test grayscale images of digits 0-9, we conducted a comparative analysis of three prominent machine learning algorithms: k-Nearest Neighbors (k-NN), Convolutional Neural Network (CNN), and Support Vector Machine (SVM). Our research involved a meticulous implementation, thorough hyperparameter tuning, and comprehensive evaluation of each algorithm. A robust preprocessing pipeline was established to maintain data integrity and enhance model training quality. Principal Component Analysis (PCA) was employed to drastically reduce runtime for the SVM and k-NN models, demonstrating significant efficiency gains. Additionally, careful debugging ensured the CNN's input shape and dimensions were correctly configured, a critical step for seamless model operation. The study's outcome highlights the enhanced accuracy of digit recognition achieved by the SVM, which outperformed the CNN and k-NN in terms of test accuracy, demonstrating its superior ability to generalize from training to unseen data. The findings of our investigation provide significant contributions to the domain of digit classification by demonstrating the impact of algorithm selection, hyperparameter optimization, and effective debugging techniques on tackling complex classification tasks. The insights derived from this study pave the way for future research into the development of more sophisticated models, potentially incorporating ensemble methods and advanced dimensionality reduction techniques to further improve performance.**

I. Introduction

The recognition of handwritten digits holds paramount importance in diverse applications, including postal mail sorting, bank check processing, and form data entry. The inherent variability in handwriting styles presents a challenge for accurate digit recognition. Our study seeks to address this challenge by implementing and evaluating three machine learning algorithms: the k-Nearest Neighbors (k-NN), Convolutional Neural Network (CNN), and Support Vector Machine (SVM). Through this exploration, our goal is to identify the efficacy of these algorithms in classifying images of handwritten digits, identifying their strengths, weaknesses, and potential areas of enhancement.

The central problem addressed by this study is the accurate recognition of handwritten digits. The variability in handwriting styles poses a significant challenge. This necessitates the exploration of robust machine learning algorithms. The study will assess the k-NN, CNN, and SVM algorithms to determine their effectiveness in classifying images of handwritten digits, with the main goal of improving accuracy in real-world scenarios. In the pursuit of enhancing the accuracy of handwritten digit recognition, the study places a particular emphasis on the fine-tuning of the three machine learning algorithms. An integral aspect of this fine-tuning process involves the optimization of hyperparameters for each algorithm under consideration.

The MNIST (Modified National Institute of Standards and Technology) database, comprising 60,000 training images and 10,000 test images, will be employed in this study. Each image represents a 28x28 pixel grayscale depiction of a single digit ranging from 0 to 9. To ensure the integrity and reliability of the dataset, a comprehensive data preprocessing pipeline has been established. This includes normalization of pixel values, handling missing values, verifying that all values are integers, verifying that there are no duplicate images, standardizing image color, centering digits, verifying that all values are non-negative, and verifying that all images are black and white. Each step is carefully designed to enhance the quality and suitability of the dataset for training and testing.

The study employs k-NN, SVM, and CNN algorithms, each chosen for its unique strengths in classification tasks. The implementation includes algorithm-specific preprocessing, hyperparameter tuning, and performance evaluation. These algorithms are trained and tested on the MNIST dataset. They are then compared against each other to assess how well each of them performs.

Advancing beyond existing solutions, this study surpasses conventional algorithm implementation by conducting a thorough performance analysis of various methodologies for handwritten digit recognition. Addressing inherent challenges in digit classification and optimizing k-NN, CNN, and SVM, our solution aims to outperform current benchmarks. While feed-forward neural networks typically achieve 90% accuracy [1] our systematic exploration of diverse algorithms, combined with a robust dataset and meticulous preprocessing, positions this study to make significant strides in improving accuracy and reliability.

II. Related Work

Handwritten digit recognition is a well-explored area in machine learning, with various methodologies ranging from basic image processing techniques to advanced machine learning models. These approaches provide valuable insights into effective strategies for digit classification, each contributing unique perspectives and strengths. The work of Hafiz Ahamed,

Ishraq Alam, and Md. Manirul Islam in handwritten digit recognition [2] makes use of the polynomial kernel for SVM which resulted in a commendable 97.83% accuracy. By leveraging the power of the RBF kernel for SVM, our implementation achieves an even higher test accuracy of 98.03%.

In the paper NeuroWrite: A Multimodal Approach for Handwritten Digit Classification, NeuroWrite employs a multimodal approach, combining Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), to enhance digit recognition by leveraging diverse architectural strengths. This method has demonstrated high accuracy rates, such as those reported by Pandea and Haghighi (98.8%) and Asish (97.0%), achieved through training a CNN using TensorFlow and Keras on the MNIST dataset. The complexity of the combined CNN and RNN architecture, while computationally demanding, underscores the potential for improved recognition in complex digit classification tasks. The approach focuses on the Devanagari script, employing CNNs to automate and enhance digit recognition. Despite its tailored application, the methodology underscores the importance of adaptable neural network architectures, particularly in handling the intricacies of various scripts and handwriting styles [3].

K-Nearest Neighbor Classifier with Sliding Window Technique by Grover and Toghi introduces an innovative approach combining the K-nearest neighbor classifier with a sliding window technique. This method aims to improve accuracy without intensive pre-processing, thereby addressing spatial misalignment issues. However, the increased computational complexity and potential border information loss present challenges that need balancing against the accuracy gains [4].

Comparative Study of Neural Networks by Feiyang Chen et al, is a comparative study of CNN, ResNet, DenseNet, and CapsNet on the MNIST dataset, which highlights the emerging potential of CapsNet. With its unique architecture utilizing capsules, CapsNet offers a promising direction in image recognition, especially in terms of data efficiency and minimizing information loss [5].

Our study, while drawing on these existing methodologies, focuses on a comparative analysis of k-NN, CNN, and SVM algorithms, each with distinct operational mechanisms, to assess their applicability and effectiveness in handwritten digit recognition. The exploration and fine-tuning of these algorithms on the MNIST dataset aim to contribute further to the field by providing practical insights into algorithm selection and optimization for specific classification challenges.

III. Our Solution

In the following sections, we implement the three classification algorithms: k-NN, CNN, and SVM. For each algorithm, we compare a baseline to a version(s) with tuned hyperparameters and assess if the tuning affected the accuracy of the models. For SVM, the baseline model uses a linear kernel and gamma = "scale" where scale = 1/(number of features x the variance of the dataset). For CNN, the baseline uses two convolutional layers and one dense layer and the baseline for k-NN uses 5 nearest neighbors.

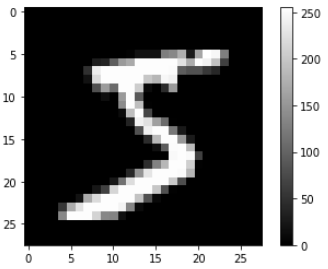


Fig. 1. Example image from MNIST dataset

A. Description of Dataset

This study will utilize the MNIST database which is a repository of 60,000 training images and 10,000 test images, each depicting a 28x28 pixel grayscale image of a single digit (0-9). This dataset was accessed through TensorFlow and provides labeled images, forming a solid foundation for training and evaluating our models.

The preprocessing of the MNIST dataset involved several critical steps to ensure the data's quality and readiness for effective model training:

- Duplicate Removal: duplicates were identified and removed to ensure the dataset's uniqueness and prevent the model from memorizing specific examples.
- Missing Value Checks: the dataset was thoroughly checked for missing values to avoid biases and inaccuracies in model training.
- Normalization: pixel values were scaled to a range of 0 to 1. This normalization is crucial as it prevents large gradient values during model training, leading to more stable and efficient learning.
- Integrity Checks: validation was done to ensure that all values are integers and non-negative, maintaining consistency and accuracy in representing pixel intensities.
- Image Standardization: the images were standardized to be black or white, and checks were conducted to ensure that the digits were centered. This standardization simplifies the data and ensures consistency across the dataset, which is vital for the performance of machine learning algorithms.

B. Machine Learning Algorithms

At the heart of our methodology is the deployment and fine-tuning of three key machine learning algorithms: k-NN, CNN, and SVM, all targeted at the recognition of handwritten digits. The k-NN algorithm operates by comparing input images with their closest counterparts. Its assumption that similar data points cluster together makes it particularly suited for classifying handwritten digits. On the other hand, CNN excel in discerning complex patterns and spatial relationships, thanks to their convolutional, pooling, and densely connected layers. Our iterative approach to optimizing CNN involves crafting its architecture, specifying filter attributes, and adjusting key parameters such as learning rates and dropout

rates. CNNs' ability to deliver high-quality predictions with minimal image preprocessing, combined with their resilience to spatial variance, renders them extremely effective in image classification tasks. In contrast, SVMs focus on identifying the optimal hyperplanes to distinctly separate different classes. Our iterative process with SVMs concentrates on refining hyperparameters like the cost parameter (C), gamma, and the kernel type. This involves experimenting with various kernel options, including non-linear ones like the RBF, to boost overall model efficacy. Such adjustments enable SVMs to tackle multi-class classification challenges, exemplified by the MNIST dataset, by methodically breaking down the problem into manageable components.

C. Implementation Details

The SVM algorithm was implemented using scikit-learn's SVC class. As mentioned previously the baseline parameters used for SVM were linear kernel and gamma equal scale. The CNN algorithm was implemented using tensorflow, specifically, Keras' Sequential model. The baseline model used two convolutional layers, one dense layer, and RMSprop as the optimizer. The k-NN algorithm was implemented using scikit-learn's KNeighborsClassifier class and 5 nearest neighbors.

In the process of hyperparameter tuning, our approach was guided by considerations of computational efficiency and practicality. We initially considered employing GridSearchCV for its exhaustive search capabilities. However, given the extensive size of the MNIST handwritten digits dataset, this method proved to be prohibitively time-consuming. In one instance, GridSearchCV exceeded 24 hours of runtime without completion. To address these challenges, we opted for manually adjusting hyperparameters and RandomizedSearchCV, which provided more efficient and feasible alternatives. This method allowed us to explore a wide range of hyperparameter combinations in a fraction of the time, striking a balance between thoroughness and computational practicality. This strategic decision facilitated a more agile and responsive tuning process, enabling us to efficiently identify optimal hyperparameters for our models

1) Use of Learning Curves in Model Assessment: Throughout our implementations, learning curves played a critical role in evaluating and fine-tuning our models. These curves plot the model's training and validation scores over various sizes of training data, providing insights into the model's learning progress and highlighting potential issues of overfitting or underfitting.

In the context of our study, learning curves were particularly valuable in identifying underfitting and overfitting scenarios. Underfitting was indicated by low training and validation scores, suggesting that our models were too simplistic and unable to capture the complexity of the data. On the other hand, overfitting was identified when we observed high training scores but significantly lower validation scores, a clear sign that the model was too complex and fitting noise in the training data rather than generalizable patterns.

For each of the algorithms (SVM, CNN, k-NN), we meticulously analyzed their learning curves. This analysis involved assessing the gap between the training and validation scores and how these scores evolved with increasing amounts of training data. By doing so, we could adjust the complexity of the models, either by tuning hyperparameters or modifying the model architecture, to strike an optimal balance between bias and variance. This iterative process of model refinement, guided by the insights gained from the learning curves, was instrumental in enhancing the predictive accuracy and generalizability of our machine learning models.

2) Debugging: In the development phase, a significant emphasis was placed on debugging and optimization techniques. Principal Component Analysis (PCA) played a pivotal role in enhancing the efficiency of our models. PCA is a statistical technique used for dimensionality reduction; it transforms the data into a new coordinate system, reducing the number of variables while retaining the most significant information. This method was instrumental when applied to our Support Vector Machine (SVM) and k-Nearest Neighbors (k-NN) models. By reducing the dimensionality of our dataset, PCA drastically decreased the computational burden, thereby reducing the runtime of these algorithms substantially. This not only accelerated the model training and testing phases but also improved the overall manageability of the dataset. Additionally, for our Convolutional Neural Network (CNN), a crucial aspect of debugging involved ensuring the correctness of input shapes and dimensions for all images. Meticulous attention to these details was essential for the CNN to function without errors. Ensuring that each image was correctly formatted and aligned with the network's input requirements was a critical step, which helped in avoiding common pitfalls associated with dimension mismatches in neural network architectures.

Classification Report:

	precision	recall	f1-score	support
0	0.94	0.94	0.94	980
1	0.81	0.99	0.89	1135
2	0.91	0.81	0.86	1032
3	0.74	0.85	0.79	1010
4	0.86	0.68	0.76	982
5	0.92	0.57	0.70	892
6	0.90	0.91	0.90	958
7	0.92	0.85	0.88	1028
8	0.80	0.80	0.80	974
9	0.67	0.89	0.77	1009
accuracy			0.83	10000
macro avg	0.85	0.83	0.83	10000
weighted avg	0.85	0.83	0.83	10000

Fig. 2. SVM Baseline statistics

Classification Report:

	precision	recall	f1-score	support
0	0.98	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.98	0.98	0.98	1010
4	0.98	0.98	0.98	982
5	0.98	0.98	0.98	892
6	0.99	0.98	0.99	958
7	0.98	0.98	0.98	1028
8	0.98	0.98	0.98	974
9	0.98	0.97	0.97	1009
accuracy			0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

Fig. 3. SVM Best Model statistics

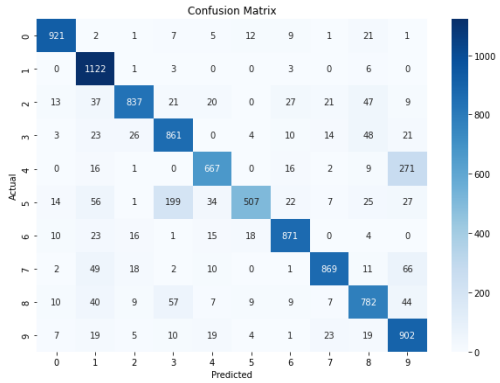


Fig. 4. SVM Baseline confusion matrix

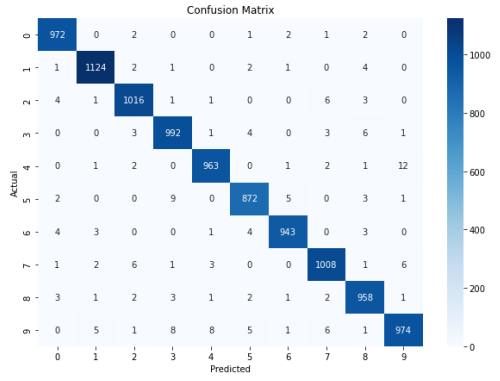


Fig. 5. SVM Best Model confusion matrix

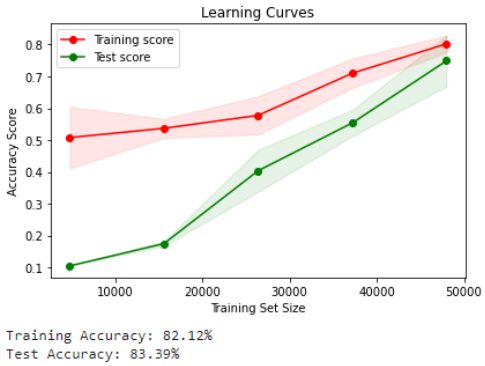


Fig. 6. SVM Baseline learning curve

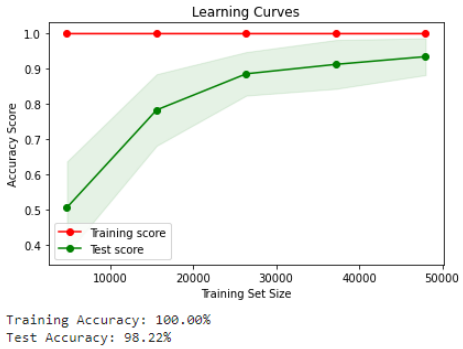


Fig. 7. SVM Best Model learning curve

D. SVM Implementation

1) SVM Baseline: The baseline performance of our SVM model on the MNIST dataset was established without hyperparameter tuning, achieving an overall accuracy of 83.39%. This baseline used a linear kernel, a C value of 1, and a scale for gamma. A detailed classification report showed varying performance across different digits. For instance, digit 0 had high precision and recall at 94%, indicating strong identification capability, while digit 1 had a precision of 81% but a higher recall of 99%. Conversely, digit 5's lower recall at 57% highlighted challenges in correctly identifying this digit. The macro and weighted averages across digits for precision, recall, and f1-score mirrored the overall accuracy, underlining the model's balanced performance across diverse classes.

2) Kernel Hyperparameter Tuning: Significant improvements were achieved by shifting from a linear to an RBF (radial basis function) kernel, better suited for the overlapping classification categories in MNIST. This change alone raised the model's accuracy from 83.39% to 98.03%. Subsequent adjustments to the gamma hyperparameter, which affects the decision boundary's curvature, revealed its sensitivity: a lower gamma of 0.01 drastically reduced accuracy to 11.35%, while higher gamma values improved performance. The final optimized model, with a gamma of 1000 and a C value of 100, achieved an accuracy of 98.22%. However, a consistently high training accuracy (100%) raised concerns about potential overfitting at this gamma level.

C determines how much we want to avoid misclassification. A smaller C value allows more points to be misclassified while a larger C value prohibits the misclassification of more points when fitting the hyperplanes. The initial value of C was set to 1, followed by two other values of 10 and 100, which resulted in increased model performance.

3) Enhancing SVM with PCA: Integrating Principal Component Analysis (PCA) for dimensionality reduction further refined the SVM's performance. Experimenting with different PCA component values, we found that setting $n\_components$ to 0.75 provided the best trade-off, enhancing accuracy to 98.03%. The learning curves validated the model's generalization ability, closely aligning a training accuracy of 98.92% with a test accuracy of 98.03%.

4) Gamma Hyperparameter Exploration: Further exploration into gamma values corroborated their impact on model accuracy. Our findings emphasized the need for higher gamma values to capture the intricacies of the MNIST dataset effectively. Despite initial concerns about overfitting with a gamma of 1000, the proximity of training and test accuracies suggested a well-calibrated model.

5) Final Remarks on SVM Optimization: Through careful hyperparameter tuning and the strategic application of PCA, our SVM model not only demonstrated a significant boost in accuracy but also maintained computational efficiency. This optimization process sets a new standard in MNIST digit classification using SVMs, showcasing the potential of methodical parameter adjustments and data preprocessing techniques in enhancing model performance.

Layer (type)	Output Shape	Param #
conv2d_22 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_22 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_23 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_23 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_11 (Flatten)	(None, 1600)	0
dropout_11 (Dropout)	(None, 1600)	0
dense_11 (Dense)	(None, 10)	16010

Total params: 34826 (136.04 KB)
Trainable params: 34826 (136.04 KB)
Non-trainable params: 0 (0.00 Byte)

Fig. 8. CNN Baseline architecture

Layer (type)	Output Shape	Param #
conv2d_38 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_38 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_39 (Conv2D)	(None, 11, 11, 64)	18496
batch_normalization_5 (BatchNormalization)	(None, 11, 11, 64)	256
activation_5 (Activation)	(None, 11, 11, 64)	0
max_pooling2d_39 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten_18 (Flatten)	(None, 1600)	0
dropout_21 (Dropout)	(None, 1600)	0
dense_19 (Dense)	(None, 10)	16010

Total params: 35082 (137.04 KB)
Trainable params: 34954 (136.54 KB)
Non-trainable params: 128 (512.00 Byte)

Fig. 9. CNN Final Model architecture

E. CNN Implementation

1) CNN Model Architecture: Our Convolutional Neural Network (CNN) model was meticulously architected to balance computational efficiency and high classification accuracy for the MNIST dataset. The model comprises two convolutional layers with ReLU activation functions, each succeeded by a max-pooling layer. The filter count was carefully chosen, starting with 32 filters of size 3×3 in the first convolutional layer, and doubling to 64 in the subsequent layer to progressively capture more complex features without overfitting. The output layer is a dense layer with 10 units, corresponding to the number of digit classes, utilizing softmax activation for probabilistic classification.

2) Regularization and Optimization Techniques: A dropout rate of 0.5 was strategically integrated before the dense layer to mitigate overfitting, creating a robust model that generalizes well to new data. RMSprop, known for its adaptive learning rates, was initially chosen as the optimizer. Through hyperparameter tuning, we identified a learning rate of 0.001 to be optimal. However, further experimentation revealed that the Adam optimizer slightly outperformed RMSprop at this learning rate, leading to its adoption in the final model iteration.

3) Hyperparameter Tuning and Model Enhancement: Extensive hyperparameter tuning was undertaken to refine our

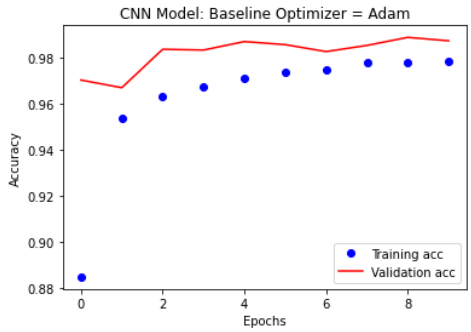


Fig. 10. CNN Baseline tuned hyperparameter learning curve

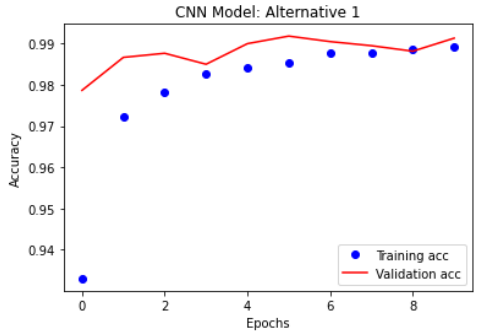


Fig. 11. CNN learning curves with batch normalization

CNN model. This included varying the learning rate for RMSprop from 0.1 downwards by an order of magnitude and comparing the performance with the Adam optimizer. Furthermore, we experimented with batch normalization, additional convolutional layers, and additional dense layers paired with dropout layers, to assess their impact on model complexity and performance. While most of these variations performed comparably to the baseline model with Adam optimizer they were not selected due to being more complex than the baseline. However, adding a batch normalization layer after the second convolutional layer did result in an increase in model performance to 98.93% training accuracy compared to 97.86% for the baseline model with the optimizer set to Adam. Because of this, the model with batch normalization layer was used as the final model to evaluate performance on test data.

4) Final Model Selection and Performance: Our final model selection was the alternative CNN architecture with an Adam optimizer and a batch normalization layer, which achieved a remarkable test accuracy of 99.31%. This model did not exhibit signs of overfitting, a testament to the effectiveness of the dropout layer and batch normalization layer, and was validated by a consistently higher validation accuracy.

5) Model Evaluation Metrics and Comparison: The accuracy metric, as recommended by the Keras documentation, was the primary measure used for model evaluation due to the multi-class classification nature of our problem. Our CNN model demonstrates superior performance, warranting its selection for the MNIST digit classification task.

6) Final Model Evaluation: The final CNN model was evaluated on the test set, achieving an accuracy of 99.31%, demonstrating the model's effectiveness in classifying unseen

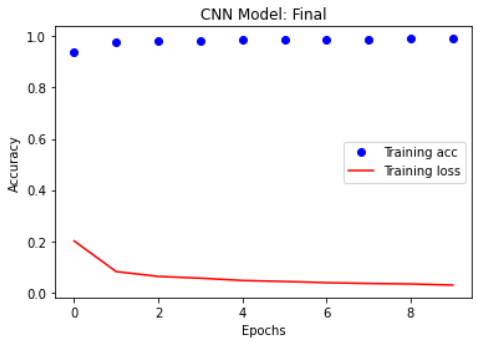


Fig. 12. CNN Final Model learning and loss curves

data. This represents a substantial improvement over the baseline CNN model. Future work will explore further optimizations and architectural changes to enhance performance.

F. K-Nearest Neighbors (k-NN) Implementation

1) k-NN Model Description: The K-Nearest Neighbors (k-NN) algorithm was employed to classify the MNIST dataset based on the proximity of data points in the feature space. This non-parametric method is widely recognized for its simplicity and effectiveness in classification tasks.

2) Data Preprocessing and Dimensionality Reduction: In our study, Principal Component Analysis (PCA) with an n_components value of 0.77 proved to be the optimal choice for maximizing test accuracy. This application of PCA aimed to reduce the dimensionality of our dataset while retaining a significant proportion of variance. By doing so, we sought to simplify our model, enhancing computational efficiency and improving predictive accuracy.

The selection of an n_components value of 0.77 was a result of finding a balance between dimensionality reduction and information retention. Decreasing the n_components value beyond this threshold led to a decline in test accuracy, indicating that too much essential information was being discarded. Conversely, opting for an n_components value higher than 0.77 resulted in lower test accuracy, suggesting the inclusion of potentially redundant or noisy components.

Our decision to employ PCA was driven by the desire to streamline the model and make it more computationally efficient. The retained significant variance in the transformed features ensured that the essential patterns and relationships present in the original data were preserved. This is critical for our model's ability to generalize effectively to new, unseen data.

3) Hyperparameter Tuning: Extensive hyperparameter tuning was conducted to optimize the k-NN model's performance. The exploration included:

- Adjusting the number of neighbors, with 6 identified as the optimal count for balancing overfitting and generalization capabilities.
- Experimenting with different distance metrics such as Euclidean and Manhattan, where Euclidean demonstrated superior performance.

```
Neighbors: 6, Metric: euclidean, Weighting Scheme: distance, Test Accuracy: 97.18%

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.96      0.96      0.96      1010
           4       0.98      0.96      0.97       982
           5       0.98      0.96      0.97       892
           6       0.97      0.99      0.98       958
           7       0.97      0.96      0.97      1028
           8       0.97      0.96      0.97       974
           9       0.94      0.96      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Fig. 13. k-NN Baseline statistics

```
Hyperparameters: Weights=distance, Neighbors=6, Metric=euclidean
Accuracy: 97.18%

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.97      0.98      1032
           3       0.96      0.96      0.96      1010
           4       0.98      0.96      0.97       982
           5       0.98      0.96      0.97       892
           6       0.97      0.99      0.98       958
           7       0.97      0.96      0.97      1028
           8       0.97      0.96      0.97       974
           9       0.94      0.96      0.95      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

Fig. 14. k-NN Best Model statistics

- Applying various weighting schemes, with the distance-based weighting scheme yielding the highest test accuracy.
- Iteratively selecting the number of PCA components to maintain a substantial variance proportion, leading to improved model accuracy.

The implementation of the k-Nearest Neighbors algorithm initially employed default parameters and resulted in a test accuracy of 97.10%, serving as a benchmark for subsequent optimization. We delved into the default settings, encompassing n_neighbors (5), weights ('uniform'), algorithm ('auto'), leaf size (30), p value (2), metric ('minkowski'), and metric_params (None), to comprehend their impact on model performance.

To refine the model, we conducted targeted experiments on key parameters, particularly n_neighbors, metric, and weights. N_neighbors, determining the number of nearest points considered for predicting the class of a new datapoint, played a pivotal role in balancing bias and variance. In our experiments, a value of 5 for n_neighbors along with other default parameters yielded the highest test accuracy (97.10%), striking a balance between model performance and overfitting. While test accuracies continued to decrease beyond n_neighbors of 5 with default parameters, overfitting was less pronounced.

The metric parameter, defining the distance metric for neighbor computation, emerged as another crucial aspect of our exploration. We evaluated various distance metrics, including Euclidean, Manhattan, Cosine, Chebyshev, Minkowski, and Hamming distances. The Euclidean distance metric, measuring straight-line distance in pixel space, proved most effective with a test accuracy of 97.10%. This metric excels in capturing geometric relationships within image-based datasets like MNIST, where pixel spatial arrangement holds significance.

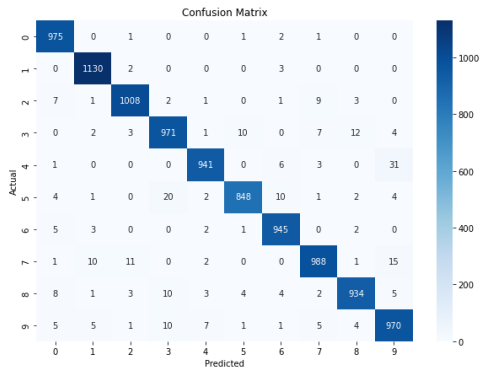


Fig. 15. k-NN Baseline confusion matrix

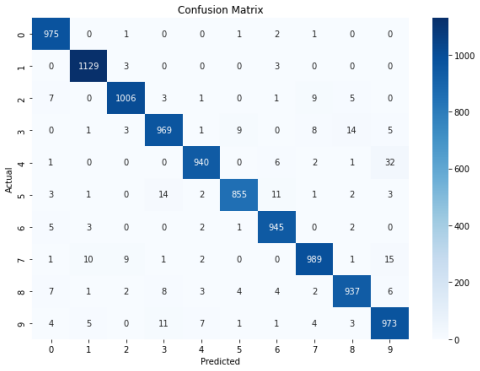


Fig. 16. k-NN Best Model confusion matrix

The weights parameter, influencing the weight assigned to each neighbor, was equally significant. We experimented with both 'uniform' and 'distance' weight schemes. Notably, the 'distance' weight scheme, coupled with an n_neighbors value of 6 and the Euclidean metric, resulted in the highest test accuracy (97.18%). This underscored the impact of weight selection on model performance, particularly in image data such as the MNIST dataset where nearby pixels contribute significantly to local structure and patterns of a digit.

The 'distance' weight scheme, considering the inverse of distances, proved effective in image datasets with varying feature densities. It allowed the model to prioritize closer neighbors, focusing on local details. Additionally, this scheme accommodated varying feature densities by emphasizing neighbors based on their distances, particularly beneficial in image datasets where relevant features concentrate in specific regions of the input space.

In our hyperparameter tuning process, we prioritized optimizing n_neighbors, recognizing its pivotal role in the bias-variance tradeoff. Subsequently, fine-tuning the distance metric, with Euclidean distance proving most effective, and exploring different weight schemes, highlighted the superiority of the 'distance' scheme in enhancing model accuracy. In summary, our findings emphasize the profound influence of a thoughtfully selected combination of n_neighbors, distance metric, and weight scheme on the k-NN algorithm's performance. The results presented offer valuable insights into optimal parameter configurations for the scikit-learn KNeighborsClassifier when applied to the MNIST dataset.

4) Performance Evaluation of k-NN with Bagging Ensemble Technique: After arriving at an optimal configuration (n_neighbors=6, metric='euclidean', weights='distance'), we sought to explore the potential of further improving our k-NN model through ensemble techniques. One such technique we employed was the Bagging Classifier which combines multiple instances of a base learner to reduce overfitting and enhance generalization. We applied this classifier to our newly tuned k-NN model with the aforementioned optimal hyperparameters.

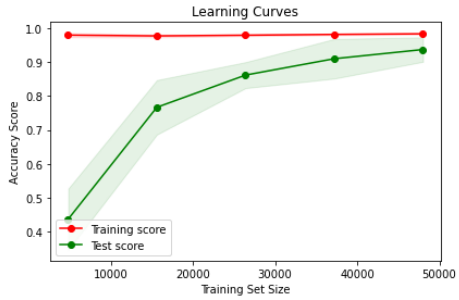
While the non-bagging k-NN model, with the finely tuned hyperparameters exhibited a commendable test accuracy of 97.18%, the bagging enhanced version lagged slightly behind at a test accuracy of 97.10%. Several factors may contribute to this phenomenon. Bagging is effective when used to reduce variance. Since the k-NN model is a robust and non-parametric algorithm, it may not benefit significantly from the variance reduction provided by bagging. Moreover, the nature of k-NN, which relies on local neighborhood information, might limit the potential gains from combining multiple instances of the same algorithm.

While our Bagging Classifier did not outperform the standalone k-NN model in this instance, the exploration of ensemble techniques remains an invaluable aspect of model refinement, providing insights into the interplay between algorithmic choices and performance outcomes. Future work could delve deeper into understanding the interplay between the characteristics of k-NN and ensemble methods to extract the maximum potential for model improvement.

5) Model Evaluation: The finalized k-Nearest Neighbors (k-NN) model for MNIST classification achieved an impressive 97.18% accuracy on the test dataset. The selected hyperparameters, including a Euclidean distance metric, distance-weighted neighbors, and a choice of 6 nearest neighbors, highlight the model's capacity to generalize effectively to unseen data. The Euclidean metric is well-suited for capturing relationships in multi-dimensional spaces, and assigning weight based on distance enhances sensitivity to local patterns. A k value of 6 strikes a balance between capturing intricate details and avoiding overfitting. This outcome underscores the success of the systematic tuning process, demonstrating the model's robustness in recognizing handwritten digits and its ability to extend performance to previously unseen instances. This study has reaffirmed the KNN algorithm's effectiveness in handwritten digit classification, highlighting its enduring value as a fundamental tool in machine learning for such tasks.

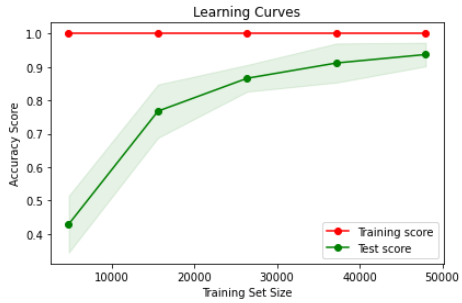
IV. Comparison

In our investigation into digit classification using the MNIST dataset, the performances of Support Vector Machine (SVM), Convolutional Neural Network (CNN), and k-Nearest Neighbors (k-NN) were thoroughly analyzed and compared. The SVM model, initially benchmarked at 83.39% accuracy with a linear kernel, showed remarkable improvement upon tuning, especially with the transition to an RBF kernel and the introduction of PCA for dimensionality reduction. This led to its best performance of 98.22% accuracy, demonstrating the efficacy of methodical hyperparameter optimization and strategic data processing. In contrast, the CNN model, designed with a




```
Training Accuracy: 98.20%
Test Accuracy: 97.10%
```

Fig. 17. k-NN Baseline learning curve




```
Training Accuracy: 100.00%
Test Accuracy: 97.18%
```

Fig. 18. k-NN Best Model learning curve

layered architecture including convolutional and max-pooling layers, dropout for regularization, and batch normalization for performance enhancement, excelled with a test accuracy of 99.31%. This superiority in performance underscores CNN's strength in handling complex image data, benefiting from its deep learning capabilities to extract nuanced features from the digit images. On the other hand, the k-NN model, recognized for its simplicity, achieved a commendable 97.18% accuracy. The application of PCA proved pivotal in managing the dimensionality of the data, enhancing both computational efficiency and predictive accuracy. The model's performance was further optimized through careful selection of the number of neighbors and the distance metric, showcasing the effectiveness of k-NN in image classification when combined with appropriate feature reduction and parameter tuning. Each model demonstrated unique strengths: SVM's adaptability through hyperparameter tuning, CNN's exceptional accuracy with advanced architectural design, and k-NN's robust performance with simpler, yet effective, optimizations. This comparative analysis not only highlights the importance of algorithm selection and tuning in machine learning tasks but also sheds light on the distinct approaches and challenges associated with each model in the realm of digit classification.

V. Future Directions

There remains potential for further enhancement of the algorithms. For SVM, exploring parallel computing options in order to perform more in-depth hyperparameter tuning is an avenue that could be explored to enhance the performance of the model.

For CNN, with additional time, we could explore a wider array of architectures and delve into tuning other hyperparameters such as batch size and epochs. Moreover, data augmentation presents an opportunity to expand our training dataset, potentially leading to the development of a deeper and more sophisticated CNN architecture that could surpass our current model's performance.

For k-NN further enhancements that could be tried are investigating alternative distance metrics and weighting schemes to further refine the k-NN model's accuracy, incorporating additional dimensionality reduction techniques like t-SNE or UMAP for potentially better feature extraction and computational efficiency, and exploring the implementation of ensemble methods, such as boosting, to improve the model's predictive power.

VI. Conclusion

In conclusion, this comprehensive investigation into machine learning techniques for handwritten digit recognition using the MNIST dataset has yielded insightful results. Through rigorous preprocessing, experimentation with hyperparameters, and implementation of three distinct algorithms—k-NN, CNN, and SVM—we have demonstrated the critical importance of selecting and tuning the right algorithm for specific data challenges.

The SVM algorithm, initially exhibiting a moderate performance, was significantly enhanced through the strategic application of an RBF kernel and careful hyperparameter optimization. The introduction of PCA for dimensionality reduction further improved the model's efficiency and test accuracy, culminating in an impressive accuracy of 98.22%. These enhancements were validated through the use of learning curves, which indicated robust generalization without signs of overfitting. https://www.overleaf.com/project/6578a31158325cf98670c6bd Our CNN model, with its well-architected layers and incorporation of batch normalization, achieved outstanding performance. The model reached a test accuracy of 99.31%, reflecting its ability to extract and learn complex features from the digit images. The learning and loss curves presented evidence of the model's stability and high learning capacity throughout the training process.

The k-NN algorithm's simplicity, combined with PCA and optimal hyperparameter selection, also showcased commendable performance, achieving a test accuracy of 97.18%. The exploration of ensemble techniques like bagging offered additional perspectives on model refinement, although the standalone k-NN model already demonstrated a fine balance between bias and variance.

Overall, the study underscores the efficacy of machine learning in automating the task of digit recognition—a task that has numerous practical applications in the modern world. By pushing the boundaries of algorithmic performance on the MNIST dataset, this study contributes to the ongoing discourse in the field and sets a benchmark for future research endeavors.

Moreover, the study reflects on the dynamic nature of machine learning, where continuous evaluation, adaptation, and optimization of models are integral to achieving high

accuracy and reliability. The findings from this study not only enhance the accuracy of digit recognition but also pave the way for applying similar methodologies to more complex image and pattern recognition tasks in various domains.

As we look to the future, we envision further exploration into deep learning architectures, dimensionality reduction techniques, and ensemble methods. There is potential to expand the dataset, experiment with real-world noisy data, and adapt the models for broader applications beyond digit recognition.

This journey through data preprocessing, algorithm selection, and hyperparameter tuning has been a testament to the power of machine learning. It has showcased the innovation, learning, and advancement of technology that can interpret the world with the nuance and precision of the human eye, yet with the scalability and efficiency that only machines can offer.

REFERENCES

[1] S. Ali, Z. Shaukat, M. Azeem, Z. Sakhawat, T. Mahmood, and K. Rehman, "An efficient and improved scheme for handwritten digit recognition based on convolutional neural network," *SN Applied Sciences*, vol. 1, 09 2019.

[2] M. H. Ahamed, S. M. I. Alam, and M. Islam, "Svm based real time hand-written digit recognition system," 01 2019. [Online]. Available: https://www.researchgate.net/publication/330684489_SVM_Based_Real_Time_Hand-Written_Digit_Recognition_System

[3] K. Asish, P. S. Teja, R. K. Chander, and D. D. D. Hema, "Neurowrite: Predictive handwritten digit classification using deep neural networks," 2023.

[4] D. Grover and B. Toghi, "Mnist dataset classification utilizing k-nn classifier with modified sliding-window metric," 2019.

[5] F. Chen, N. Chen, H. Mao, and H. Hu, "Assessing four neural networks on handwritten digit recognition dataset (mnist)," 2019.