Machine Learning Project 3 Report
Handwritten Digit Recognition Using Machine Learning

Introduction

- The project aims to classify handwritten digits (0-9) from the MNIST dataset, which includes 60,000 training and 10,000 test images, split equally between employees and students.
- The training set contains 30,000 images from employees and 30,000 from students, while the test set contains 5,000 images from employees and 5,000 from students. This project aims to develop a prediction model that classifies images of handwritten digits to their corresponding numbers (0 to 9).

1. **Your code takes the labeled data and returns a model that can be used to make predictions for the test data. The code should be submitted as an iPython notebook or a Python file.** *(added in the zip folder)*

2. **A. Introduction to approaches considered and why:**

I. Neural Network: Neural Networks are some of the most effective methods for classifying image data.

Why a neural network was considered:
- Deep learning, particularly neural networks, excels in image recognition tasks such as handwriting recognition by effectively capturing intricate patterns in pixel data, which traditional machine learning algorithms often overlook.
- Employing techniques like cross-validation and dropout enhances the model's ability to generalize to new data, thereby preventing overfitting to the training dataset.

Ii. Support Vector Machines: Support Vector Machine (SVM) with a Radial Basis Function (RBF) kernel, designed to classify handwritten digits from the MNIST dataset. SVMs are powerful classifiers that work by finding a hyperplane that best separates the classes in a high-dimensional space.

- **Why SVM has considered:** The SVM model is particularly suitable for image classification tasks like the MNIST due to its effectiveness in dealing with high-dimensional data and its capability to capture complex patterns through the RBF kernel.

iii. K-Nearest Neighbor: The k-NN algorithm is a simple, instance-based learning method that classifies new cases based on a majority vote of the k-closest examples in the feature space.

- **Why KNN was chosen:**
  The model's simplicity and the non-parametric nature make it robust and versatile across different datasets.

B. A description of your submitted solution, including any data processing algorithms used:

Submitted solutions: Neural Network, Support Vector Machines, K-Nearest Neighbour

I. **Neural Network:** The implementation uses a Keras neural network to classify MNIST handwritten digits.
- It uses a Sequential model with layers that include a Flatten layer to convert 28x28 images to a 1D array, a Dense layer with 128 ReLU-activated neurons, a Dropout layer to prevent overfitting, and another Dense layer with softmax to output class probabilities.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss, which is suitable for multi-class classification.

Methodology:

Data Loading and Preprocessing

a. **Loading**: The MNIST dataset, which contains images of handwritten digits (0-9), is loaded. Each image is 28x28 pixels.

b. **Normalization**: Pixel values are normalized to a range of 0 to 1 to aid in the training process by ensuring consistent scale.
c. **One hot encoding:** Labels (digits) are one-hot encoded, transforming them from a single number to a binary vector for classification.

Model Architecture (Algorithm)

- **Sequential Model**: Uses a linear stack of layers in Keras.
- **Flatten Layer:** Transforms 28x28 pixel images into a 1D array.
- **Dense Layer:** Contains 128 neurons with ReLU activation for complex pattern learning.
- **Dropout Layer:** Reduces overfitting by randomly setting input units to 0.
- **Output Layer**: Dense layer with 10 neurons, each representing a digit, using softmax for probability outputs.

Compilation of the Model: The model is compiled with the Adam optimizer and categorical cross-entropy loss function, which is suitable for multi-class classification problems.

Cross-Validation Setup: K-Fold cross-validation (with five splits) is employed to evaluate the model's performance more robustly. This technique involves dividing the training data into five sets, using 4 for training and 1 for validation in each iteration, which helps ensure that the model's performance is consistent across different subsets of data.

Training: During training, an EarlyStopping callback is used to stop training if the validation loss does not improve for three consecutive epochs which prevents overfitting and reduces unnecessary computation by halting when performance plateaus.

Evaluation

- After training on each fold, the model is evaluated on the validation set, and the accuracy scores are recorded.
- These accuracy scores' mean and standard deviation estimate the model's performance and variance across different folds.

Final Model Training and Testing: Once cross-validation is complete, the model is trained on the whole training dataset and then evaluated on the separate test dataset to estimate how well it will perform on new, unseen data.

II. Support Vector Machine: The RBF kernel allows the SVM to handle nonlinear relationships by mapping the input features into a higher-dimensional space where a linear separation is feasible.

Data Processing:

- **Data Reshaping:** MNIST images (28x28 pixels) are reshaped into 784-dimensional vectors using `.reshape(-1, 784)`, where `-1` lets numpy determine the appropriate number of rows.
- **Type Conversion:** Pixel values are converted to `float32` to ensure compatibility during normalization.
- **Normalization:** Pixel values are normalized by dividing by 255.0, scaling them to a range between 0 and 1, enhancing learning efficiency and stability.

Algorithm Implemented:

- The MNIST dataset's 28x28 pixel images are reshaped into 784-dimensional vectors and normalized for SVM processing.
- The SVM is then trained on the entire training dataset to distinguish among the ten-digit classes.
- Model performance is evaluated on a separate test dataset using accuracy to measure the prediction correctness of handwritten digits.

III. K-Nearest Neighbour:

The k-NN algorithm is a simple, instance-based learning method that classifies new cases based on a majority vote of the k-closest examples in the feature space.

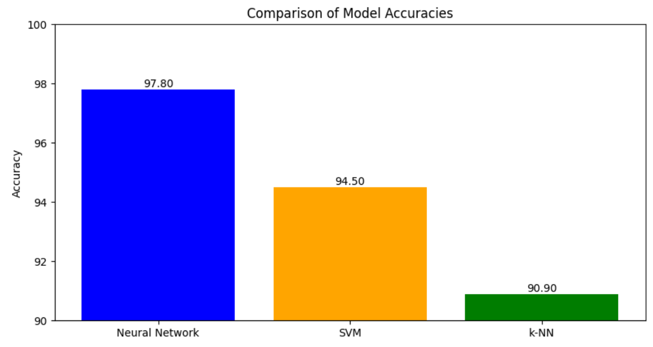
Data Preprocessing: Similar to Support vector machines, the following processing was done:

- **Normalization:** The pixel values are normalized by dividing each by 255.0, scaling them to a range of 0 to 1. Normalization is crucial for the k-NN classifier because it relies on distance calculations to determine the nearest neighbors.
- **Type Conversion:** The pixel values are converted to float32. This conversion ensures that subsequent operations on the pixel values, such as normalization, are performed correctly and efficiently.

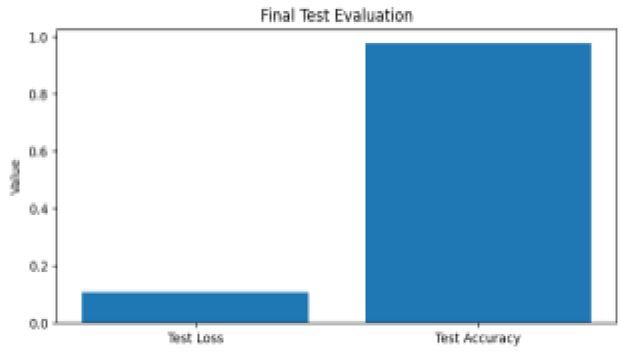
Algorithm:

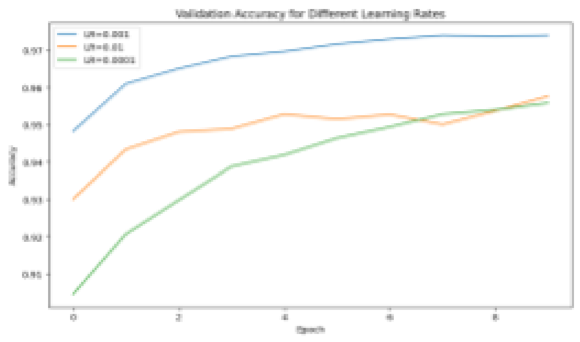
1. **Number of Neighbors (k=3)**: The algorithm uses the three nearest neighbors for predictions.
2. **Distance Calculation:** Calculates Euclidean distance (or other suitable metrics) between the test instance and training data.
3. **Identify Nearest Neighbors:** Identifies the three closest training instances based on distance.
4. **Majority Voting:** Assigns the test instance the class most common among its nearest neighbors.
5. **Output Prediction:** Outputs the classification based on majority voting.

C. Results and Evaluation: Neural Network Outperforms SVM and K-NN models.



Neural Networks:





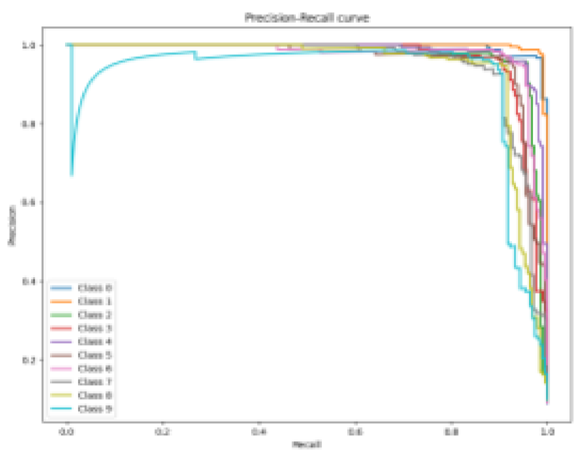
Final Test Evaluation

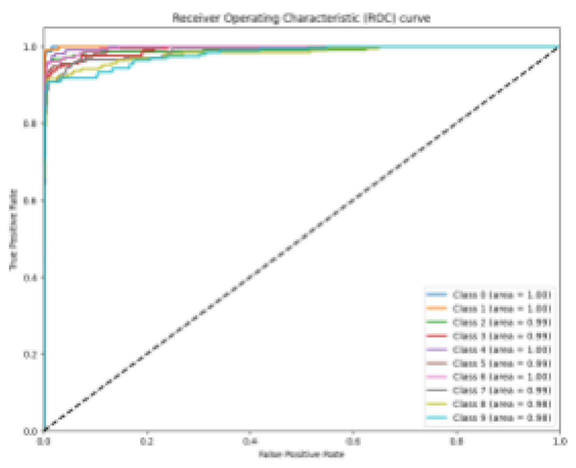
- The 'Test Loss' is relatively low (0.015), indicating that the model's predictions are, on average, quite close to the actual values.
- The 'Test Accuracy' is high, close to 1 (or 100%), suggesting that the model performs very well on the test dataset.

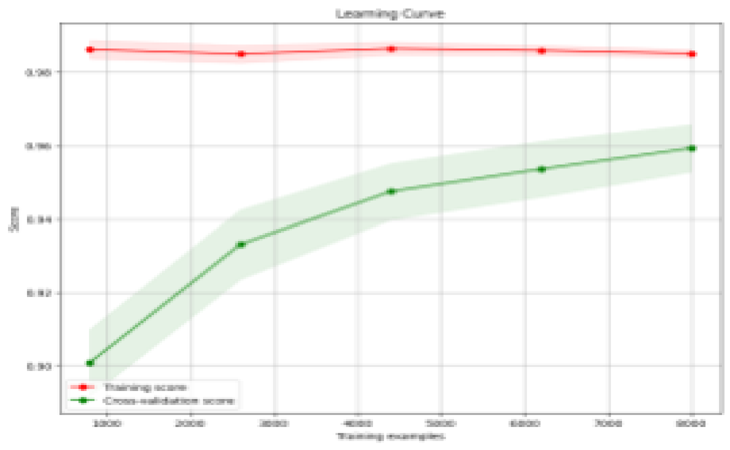
Hyperparameter tuning (Learning Rates):

- The 0.001 learning rate model (blue) leads in accuracy with consistent gains.
- The 0.01 rate model (orange) begins strong but soon levels off.
- The 0.0001 rate model (green) improves gradually and may need more epochs to match others' accuracy.

Support Vector Machine







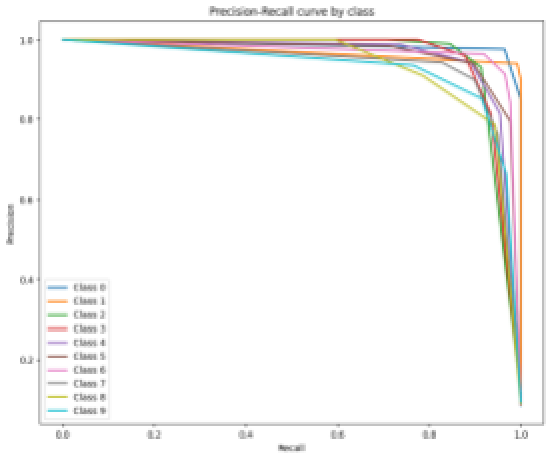
Hyperparameter Tuning

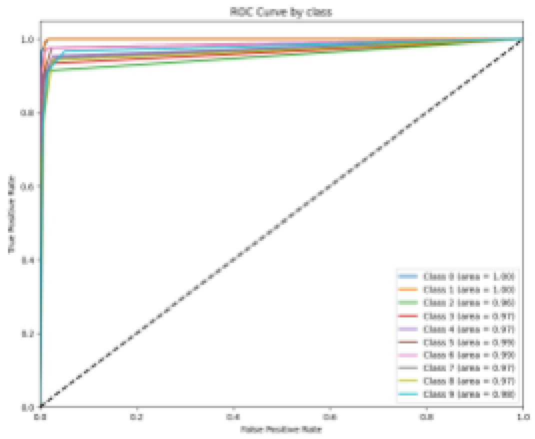
To train a model, GridSearchCV uses a dictionary of potential model parameters. The parameters are the keys in the parameter grid, which is defined as a dictionary with the settings to be tested as the values.

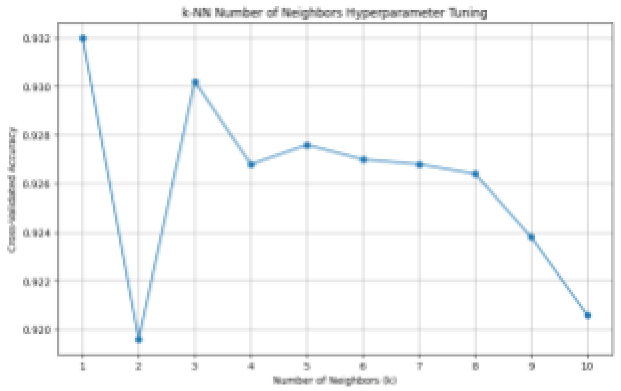
Final Results:

- **Best parameters:** {'C': 10, 'gamma': 0.01}
- **Best cross-validation accuracy**: 0.94
- **Test set accuracy:** 93.90%

K-Nearest Neighbor







- The Precision-Recall curve demonstrates high precision across all classes for most recall levels, indicating that the model reliably identifies relevant instances with few false positives.
- The ROC curve graph indicates that the classification model performs exceptionally well in distinguishing between the different classes, with near-perfect area under the curve (AUC) values for each class, showing high true positive rates with very low false positive rates.
- **Hyperparameter tuning:** The KNN model gives an accuracy of 90.9% which can be achieved with K = 1 (1 nearest neighbor).

d. Conclusion

- The models applied to the MNIST dataset—Neural Networks, Support Vector Machines (SVM), and K-Nearest Neighbors (k-NN)—show high-digit classification effectiveness.

- The Neural Network, with its deep learning capabilities, exhibits impressive and consistent cross-validation accuracy of around 97%, hinting at excellent generalizability and slight variations in loss, suggesting room for optimization.

- SVM performs strongly with a 94% accuracy and balanced precision, recall, and F1-scores, despite minor confusions between classes like 3 and 5; ROC analysis shows AUCs between 0.95 and 0.99, indicating almost perfect classification for some digits.

- K-NN also demonstrates excellent classification ability with Precision-Recall curves showing high precision across all recall levels and ROC curves with high AUC values, underscoring its strong discriminative power.

- In summary, all models are highly capable with specific strengths, and with further fine-tuning, they can be optimized for even more reliable performance.

e. References

1. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/
2. https://www.tensorflow.org/datasets/keras_example
3. https://keras.io/api/datasets/mnist/
4. https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-keras.md
5. https://medium.com/analytics-vidhya/a-beginners-guide-to-knn-and-mnist-handwritten-digits-recognition-using-knn-from-scratch-df6fb982748a
6. https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/

f. Any special instructions that are required to run your code.

- None, we used **Google Colab,** so it should run on every device.
- **Note**: The runtime for the Neural network may exceed 15 minutes, although the hyperparameters were tuned (epochs, batch size) for maximizing model performance
- **Note**: The report is actually 6 pages only, since indexing begins from page 2.