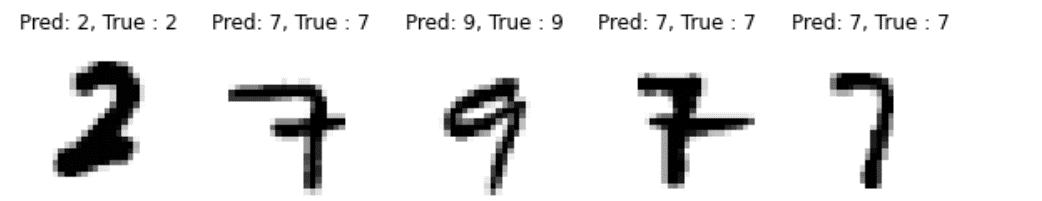
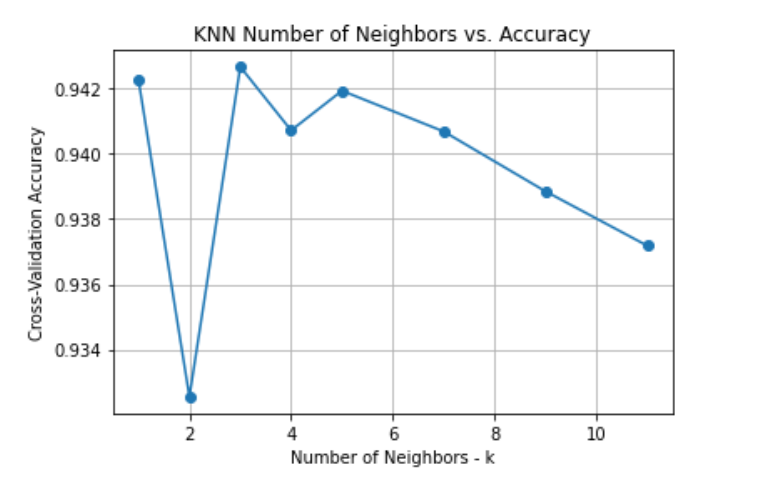
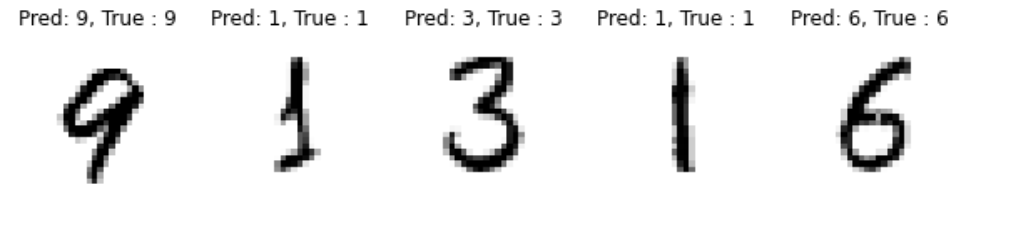
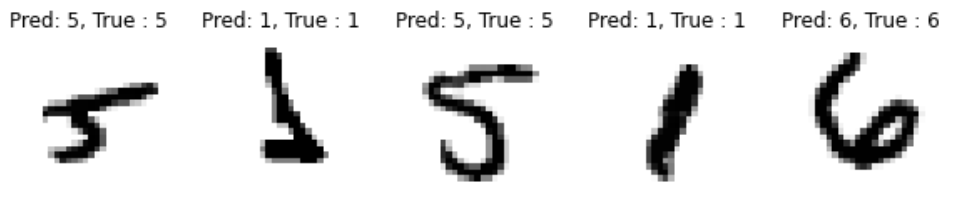
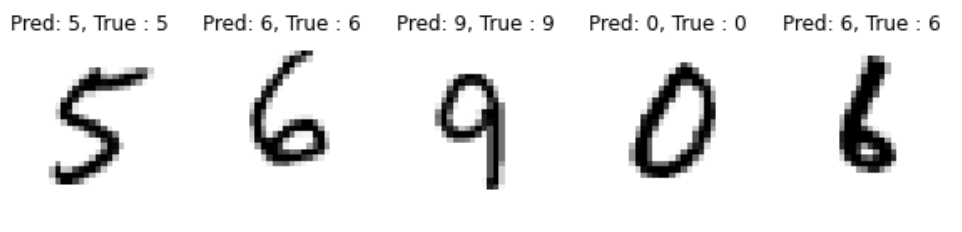
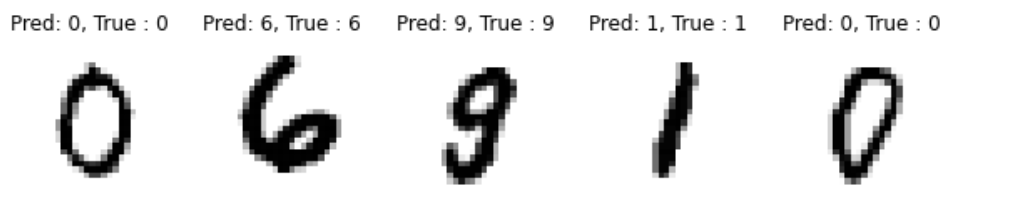
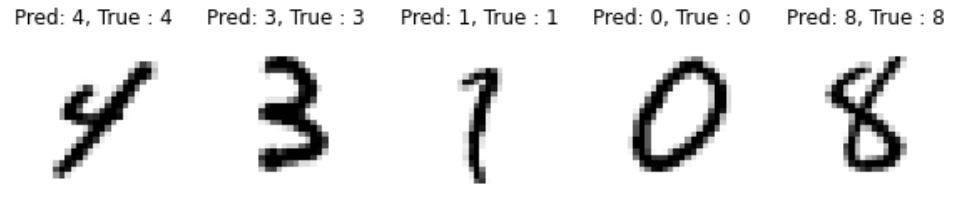
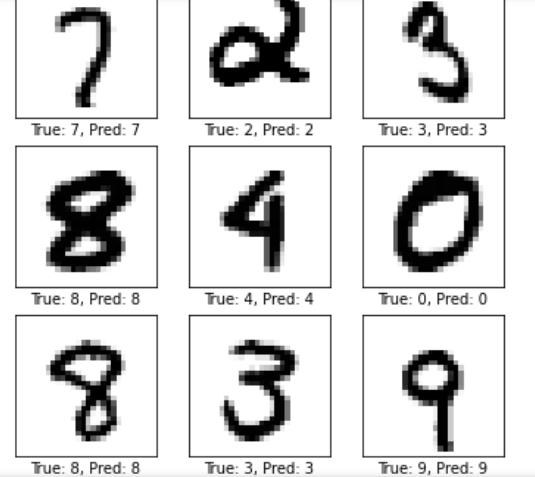
**Overview of the MNIST Dataset Handling  
1.Libraries Used:**  
NumPy: Fundamental package for numerical computation in Python.  
Matplotlib: A plotting library for creating static, interactive, and animated visualizations in Python.  
Keras: A high-level neural networks API, used here to easily access and load the MNIST dataset.  
scikit-learn: Utilized for machine learning tools but specifically mentioned here for potential classification reporting and confusion matrix functionalities, which are common post-model evaluation techniques. **2.Data Loading:**  
The MNIST dataset is loaded from the Keras datasets module. It's a widely-used dataset for benchmarking machine learning models on tasks of handwritten digit recognition.  
The dataset is divided into two tuples: training data (train\_X, train\_y) and testing data (test\_X, test\_y). **3.Data Description:**  
Training Data: Consists of 60,000 grayscale images of handwritten digits, each of 28x28 pixel resolution.  
Test Data: Comprises 10,000 images with the same resolution as the training data, used to evaluate the performance of trained models.  
**KNN  
Data Preprocessing:**  
Before training the KNN classifier, we reshaped the input data to meet the requirements of the algorithm, converting each image into a one-dimensional array and normalizing pixel intensities to a range between 0 and 1.  
**Parameter Tuning:**  
We employed GridSearchCV to find the optimal number of neighbors (k) for the KNN classifier. We tested values of k ranging from 1 to 11 and used 5-fold cross-validation to evaluate model performance.  
**Results:**  
The best-performing model achieved an average cross-validation accuracy of approximately 97.1%.  
The optimal number of neighbors was determined to be 3.  
When tested on the separate test set, the model achieved an accuracy of 97.05%.  
**Model Evaluation:**  
A plot depicting the relationship between the number of neighbors (k) and cross-validation accuracy was generated, showing a gradual decline in accuracy as k increased.  
  
**SVM**  
**Model Construction:**  
**Linear Kernel SVM:**  
We created an SVM classifier with a linear kernel and incorporated it into a pipeline with StandardScaler. This model was trained on the training data.  
**RBF Kernel SVM (Gamma = 0.05):**  
An SVM classifier with an RBF kernel and a gamma value of 0.05 was constructed and integrated into a pipeline with StandardScaler. Subsequently, the model was trained on the training data.  
**RBF Kernel SVM (Gamma = 0.01):**  
Another SVM classifier with an RBF kernel and a gamma value of 0.01 was constructed and included in a pipeline with StandardScaler. The model was then trained on the training data.  
**Results:**  
**Linear Kernel SVM:** Test Accuracy: 94.04%  
  
**RBF Kernel SVM (Gamma = 0.05):** Test Accuracy: 98.26%  
  
**RBF Kernel SVM (Gamma = 0.01):** Test Accuracy: 97.69%  
  
**Evaluation Metrics:**  
**Confusion Matrix:**  
Each classifier's confusion matrix presents the counts of true positives, true negatives, false positives, and false negatives, enabling a detailed evaluation of performance on individual classes.  
**Classification Report:**  
The classification report for each classifier furnishes precision, recall, F1-score, and support metrics for each class, providing a comprehensive overview of performance across all classes.  
**RF  
Random Forest Model Overview::**  
We developed two separate Random Forest classifiers for our analysis. The first classifier included 100 trees (estimators) and a fixed random state of 42 for reproducibility. The second classifier was more robust, encompassing 500 trees, also with the same random state setting. Both models underwent training using the same dataset.  
**Results:  
Random Forest with 100 Estimators:** Achieved a test accuracy of 97.04%.  
  
**Random Forest with 500 Estimators:** Slightly improved test accuracy of 97.12%.  
  
Detailed insights into the classifier's performance across different digit classes were provided through the confusion matrix and classification report.   
**Performance Analysis Tools:  
Confusion Matrix:**For both classifiers, the confusion matrix was utilized to provide precise counts of true positives, true negatives, false positives, and false negatives. This detailed breakdown aids in assessing the model's performance on each individual class.  
**Classification Report:**The classification report provides key metrics such as precision, recall, F1-score, and the number of samples (support) for each class. These metrics help in understanding the detailed performance of both classifiers across all classes.

**CNN  
CNN Model Description:**  
The architecture of our CNN (Convolutional Neural Network) includes several convolutional layers that employ ReLU (Rectified Linear Unit) as the activation function. These layers are interspersed with max-pooling layers that serve to reduce the dimensionality of the feature maps. To help mitigate the risk of overfitting, dropout layers are strategically placed within the network. The network's architecture concludes with fully connected layers that also use ReLU activation, leading up to a softmax activation in the final layer for classifying multiple categories.  
**Results:** The CNN model exhibited exceptional accuracy, achieving 99.55% on the test set after undergoing training for 10 epochs.  
**Training Monitoring:**  
A validation split of 10% was used during the training process to provide continuous feedback on the model’s performance, aiding in adjustments and optimizations as needed.  
  
**Evaluation Metrics:**  
With a test accuracy of 99.55%, the CNN model demonstrates its high efficacy in recognizing digits, confirming its robustness in handling such classification tasks.  
  
  
**Comparison Results:  
KNN**: Achieved a test accuracy of 97.05% with the optimal number of neighbors set to 3.  
**SVM**: The RBF kernel SVM with gamma=0.05 outperformed other SVM variants, achieving a test accuracy of 98.26%.  
**RF**: Both RF models performed comparably well, with test accuracies of around 97%.  
**CNN**: Demonstrated exceptional performance with a test accuracy of 99.55%, showcasing the superiority of deep learning for image classification tasks.

**Conclusion:**

In conclusion, our analysis indicates that while traditional machine learning algorithms like KNN, SVM, and RF can achieve decent performance on the MNIST dataset, CNNs significantly outperform them. The CNN model achieved near-perfect accuracy, highlighting the effectiveness of deep learning techniques, especially for image classification tasks. For future improvements, further exploration into CNN architectures, data augmentation techniques, and ensemble methods could potentially enhance the performance of the models even further. Overall, this study underscores the importance of selecting appropriate algorithms for specific tasks and the continuous exploration of novel methodologies to push the boundaries of performance in machine learning tasks.  
  
Any special instructions that are required to run your code.  
You may need to install the required libraries if you haven't already. Use pip, Python's package manager, to install the necessary libraries:  
pip install numpy matplotlib keras scikit-learn