Project Title: CNN Implementation for MNIST Digit Recognition

Importing Libraries

NumPy or Numerical Python is the main library for performing calculations in Python. It provides support for multidimensional arrays and a wide range of arithmetic operations to improve performance on these arrays. The NumPy ndarray object simplifies high-performance   
calculations and allows easy integration with other scientific libraries. Lots of calculations for linear results, numpy procedures, statistical analysis, etc. are needed, as well as moving, scientifically sound calculations. They can be easily prevented

TensorFlow is an open source machine learning framework developed by Google. It provides a comprehensive ecosystem for building, training, and efficiently deploying machine learning models. A key feature of TensorFlow is its computational graph abstraction, which allows users to define complex mathematical functions as a graph of nodes. It supports traditional machine learning algorithms and deep learning models, with built-in support for neural networks and distributed computing tools TensorFlow High-level like Keras for building and training simple models with low-level APIs for more fine-grained handling It provides APIs. Due to its scalability, flexibility and extensive community support, TensorFlow is widely used in research and development for various machine learning projects.

Scikit-learn, commonly abbreviated as sklearn, is a versatile machine learning library in Python. It offers a wide array of tools for data preprocessing, model selection, training, evaluation, and deployment. Sklearn provides implementations of various machine learning algorithms, including classification, regression, clustering, dimensionality reduction, and more. It   
emphasizes simplicity and ease of use, making it suitable for both beginners and experts in the field. Sklearn also supports integration with other Python libraries, such as NumPy and Pandas, and follows a consistent API design, facilitating rapid prototyping and experimentation. Its extensive documentation and active community make sklearn a go-to choice for machine learning tasks across industries.

Matplotlib is a powerful plotting library for Python that enables the creation of a wide range of high-quality visualizations. It provides a MATLAB-like interface for generating plots, allowing users to create line plots, scatter plots, histograms, bar charts, and more with ease. Matplotlib offers fine-grained control over every aspect of the plot, including axes, labels, colors, and styles. It supports both interactive and non-interactive plotting environments and can be seamlessly integrated with other Python libraries like NumPy and Pandas. With its versatility and flexibility, Matplotlib is widely used for data exploration, analysis, and presentation across various scientific and engineering disciplines.

About the Dataset:

The MNIST dataset is a classic benchmark dataset in machine learning and computer vision. It contains a large collection of gray images, each representing a handwritten number (from 0 to 9). These images are 28x28 pixels, resulting in 784 elements when flattened into a one-dimensional layout. MNIST is a standard dataset used to train and test machine learning models, especially for tasks such as image classification and digit recognition. It is widely used because of its simplicity, flexibility and relevance to real-world applications. The data set is basically divided into two subsets: training set and test set. Typically, the training set consists of 60,000 images, while the testing set consists of 10,000 images. This division ensures that models can be trained on large enough data sets to identify meaningful patterns and tested on unobserved data to assess their generalizability The corresponding representative digit is assigned to each image in the MNIST dataset. This labeling can learn from input-output pairs of machine learning models during training. For example, a model may learn to associate certain sets of pixel intensities with a particular digit using techniques such as convolutional neural networks (CNNs) or other classification algorithms Researchers and practitioners use the MNIST dataset to compare and contrast different machine learning models, analyze their performance, and demonstrate how different algorithms and methods -10, Fashion-MNIST, and SVHN, reflect real-world image a they find it very difficult tasks. In summary, the MNIST dataset provides the tools to identify and develop machine learning algorithms, especially in image classification and digital recognition and its broad applications have contributed to progress in the field and are a valuable tool researchers and practitioners.

Loading the Dataset: The code loads the dataset and it is divided into two main parts: training data and testing data.(x\_train, y\_train): This tuple contains the training images (x\_train) and their corresponding labels (y\_train). Each image in x\_train is a 28x28 grayscale image of a handwritten digit (from 0 to 9), while each label in y\_train represents the actual value of the digit.

Preprocessing:

This code prepares the MNIST data set for machine learning training. It resizes the images to a 4D array format, ensuring that each is 28x28 pixels with a single grayscale channel. Pixel values are converted to 32-bit floating-point numbers for computation consistency. Normalization scale values from 0 to 1, helping convergence. Labels are single-hot encoded, representing each digit label (0 to 9) as a binary vector with one '1' and the rest '0'. This step standardizes the input data, preparing it specifically for neural networks for training on the MNIST dataset.

The above code describes a CNN using the TensorFlow Keras API for MNIST score classification.

It adds layers sequentially: three convolutional layers (32, 64, 64 filters, 3x3 kernel, ReLU activation), two max-pooling layers (2x2), a smoothing layer, and two dense layers (64 neurons, ReLU activation; 10 neurons). , softmax). Convolutional layers extract objects, max-pooling layers down sample, and dense layers distribute. The model summary shows the layer parameters. This configuration works well for 28x28 pixel MNIST images that are commonly used for handwriting digit recognition tasks.

The CNN algorithm consists of convolutional layers that extract features from the input image, followed by a max-pooling layer that reduces spatial dimensions and transforms the feature maps into 1D arrays by smoothing. Condensed layers divide features, and the last layer generates class possibilities. This model contains 93,322 trainable parameters.

Model Compilation:

About the Compilation parameters: Optimizer (Adam): Adam is an adaptive learning rate optimization algorithm that efficiently updates network weights based on the gradients of the loss function, improving convergence speed and performance.

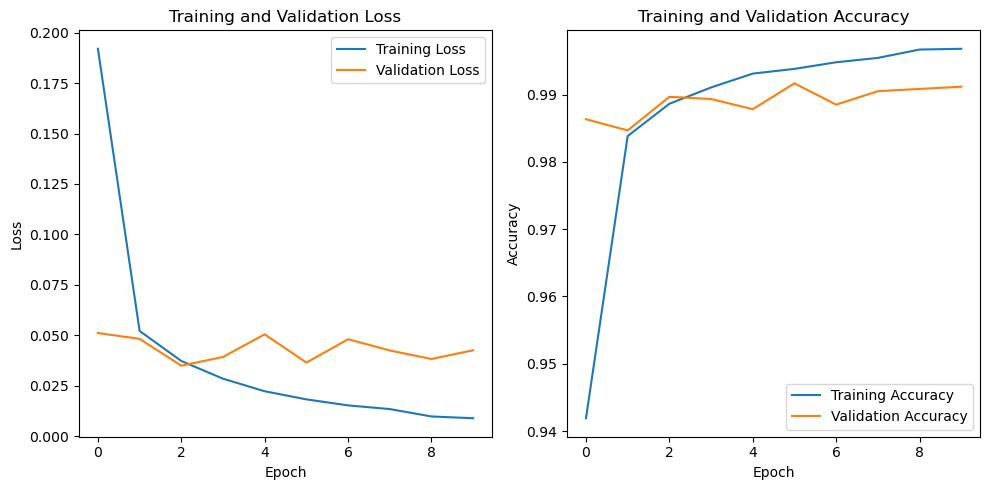
Loss Function (Categorical Cross entropy): Categorical cross entropy measures the difference between predicted probabilities and true class labels for multi-class classification tasks, guiding the optimization process by quantifying the model's performance.

Metrics (Accuracy): Accuracy evaluates the model's performance by measuring the proportion of correctly classified samples among all samples, providing a simple and intuitive measure of classification performance.

The above code organizes the training and evaluation of machine learning models using the TensorFlow Keras API. Initially, the model is prepared for training by data collection by determining the optimizer (Adam), the loss function (categorized cross-entropy), and the evaluation criteria (accuracy) after this process, a named function is defined   
train\_and\_evaluate\_model to simplify the training and evaluation process. In this work, the model is trained on the training data provided in 10 epochs, with subgroups of size 64.

Additionally, 10% of the training data is provided for validation in each epoch for evaluation purposes performance of the model. The accuracy of the test is then printed to the console for testing. Finally, the program returns the training history, which contains the metrics recorded during the training sessions. Overall, this code includes the critical steps for training a machine learning model, making it more efficient and repeatable for different classification tasks

Loss and Accuracy Curves:



Training and Validation Loss: The plot illustrates the training and validation loss of a   
Convolutional Neural Network (CNN) model over multiple epochs. X-Axis represents the number of training epochs and Y-Axis: Represents the loss value. The training Loss tarts at a high loss value (around 0.2) and rapidly decreases as the model learns from the training data. Indicates that the model is fitting well to the training data. The Validation Loss Also starts at a high loss value but decreases more gradually with some fluctuations.

Training and Validation Accuracy: The Training and Validation Accuracy, the X-axis represents Training Accuracy and Y-axis for Validation Accuracy. The x-axis represents the number of training epochs (iterations), ranging from 0 to 8.The y-axis represents the accuracy, with values ranging from 0.94 to 0.99.The Training Accuracy Starts at a relatively low value (around 0.94) and rapidly increases during the initial epochs. It reaches near-perfect accuracy (around 0.99) as the model learns from the training data. It Indicates that the model is fitting well to the training data. The Validation Accuracy Also starts at a low value. Increases but plateaus around epoch 2. The Validation accuracy curve might indicating that the model is slightly overfitting.

Thw above code describes the function plot\_training\_history used to visualize the training history of a machine learning model. It takes history as input, including metrics (loss and accuracy) collected during training and validation. Two subplots are constructed: one to illustrate training and validation losses, and the other to plot training and validation accuracies between epochs. The program then plots the metrics and displays them using Matplotlib. The annotated section suggests using this functionality after training and evaluation of the model to visualize its performance over several epochs.

|  |
| --- |
| k-Fold and Confusion Matrix:y\_pred = np.argmax(model.predict(x\_test), axis=-1) y\_true = np.argmax(y\_test, axis=-1)  cm = confusion\_matrix(y\_true, y\_pred)  print("Confusion Matrix:")  print(cm)  Fold 1  Epoch 1/5  375/375 [==============================] - 15s 38ms/step - loss: 0.2768 - accuracy: 0.9212 - val\_loss: 0.0945 - val\_accuracy: 0.9709 Epoch 2/5  375/375 [==============================] - 14s 36ms/step - loss: 0.0667 - accuracy: 0.9799 - val\_loss: 0.0698 - val\_accuracy: 0.9778 Epoch 3/5  375/375 [==============================] - 13s 36ms/step - loss: 0.0474 - accuracy: 0.9850 - val\_loss: 0.0572 - val\_accuracy: 0.9835 Epoch 4/5  375/375 [==============================] - 14s 36ms/step - loss: 0.0367 - accuracy: 0.9881 - val\_loss: 0.0461 - val\_accuracy: 0.9852 Epoch 5/5  375/375 [==============================] - 14s 37ms/step - loss: 0.0279 - accuracy: 0.9916 - val\_loss: 0.0425 - val\_accuracy: 0.9872 313/313 [==============================] - 1s 3ms/step - loss: 0.0344 - accuracy: 0.9892  Test accuracy: 0.9891999959945679  Fold 2  Epoch 1/5  375/375 [==============================] - 15s 38ms/step - loss: 0.2530 - accuracy: 0.9245 - val\_loss: 0.0802 - val\_accuracy: 0.9758 Epoch 2/5  375/375 [==============================] - 14s 37ms/step - loss: 0.0654 - accuracy: 0.9796 - val\_loss: 0.0623 - val\_accuracy: 0.9816 Epoch 3/5  375/375 [==============================] - 14s 36ms/step - loss: 0.0456 - accuracy: 0.9862 - val\_loss: 0.0496 - val\_accuracy: 0.9856 Epoch 4/5  375/375 [==============================] - 14s 37ms/step - loss: 0.0367 - accuracy: 0.9884 - val\_loss: 0.0449 - val\_accuracy: 0.9871 Epoch 5/5  375/375 [==============================] - 14s 37ms/step - loss: 0.0279 - accuracy: 0.9915 - val\_loss: 0.0388 - val\_accuracy: 0.9874 313/313 [==============================] - 1s 4ms/step - loss: 0.0300 - accuracy: 0.9901  Test accuracy: 0.9901000261306763  Fold 3  Epoch 1/5  375/375 [==============================] - 14s 37ms/step - loss: 0.2841 - accuracy: 0.9172 - val\_loss: 0.0955 - val\_accuracy: 0.9698 Epoch 2/5  375/375 [==============================] - 17s 45ms/step - loss: |

|  |
| --- |
| 0.0676 - accuracy: 0.9794 - val\_loss: 0.0680 - val\_accuracy: 0.9788 Epoch 3/5  375/375 [==============================] - 14s 38ms/step - loss: 0.0473 - accuracy: 0.9848 - val\_loss: 0.0550 - val\_accuracy: 0.9831 Epoch 4/5  375/375 [==============================] - 15s 39ms/step - loss: 0.0361 - accuracy: 0.9888 - val\_loss: 0.0403 - val\_accuracy: 0.9881 Epoch 5/5  375/375 [==============================] - 14s 37ms/step - loss: 0.0303 - accuracy: 0.9906 - val\_loss: 0.0357 - val\_accuracy: 0.9891 313/313 [==============================] - 1s 3ms/step - loss: 0.0289 - accuracy: 0.9903  Test accuracy: 0.9902999997138977  Fold 4  Epoch 1/5  375/375 [==============================] - 15s 38ms/step - loss: 0.3027 - accuracy: 0.9057 - val\_loss: 0.0820 - val\_accuracy: 0.9751 Epoch 2/5  375/375 [==============================] - 14s 38ms/step - loss: 0.0715 - accuracy: 0.9779 - val\_loss: 0.0713 - val\_accuracy: 0.9786 Epoch 3/5  375/375 [==============================] - 14s 38ms/step - loss: 0.0514 - accuracy: 0.9846 - val\_loss: 0.0508 - val\_accuracy: 0.9843 Epoch 4/5  375/375 [==============================] - 14s 38ms/step - loss: 0.0384 - accuracy: 0.9884 - val\_loss: 0.0440 - val\_accuracy: 0.9858 Epoch 5/5  375/375 [==============================] - 14s 38ms/step - loss: 0.0331 - accuracy: 0.9899 - val\_loss: 0.0416 - val\_accuracy: 0.9872 313/313 [==============================] - 1s 4ms/step - loss: 0.0321 - accuracy: 0.9901  Test accuracy: 0.9901000261306763  Fold 5  Epoch 1/5  375/375 [==============================] - 16s 40ms/step - loss: 0.2942 - accuracy: 0.9140 - val\_loss: 0.0887 - val\_accuracy: 0.9712 Epoch 2/5  375/375 [==============================] - 15s 40ms/step - loss: 0.0670 - accuracy: 0.9790 - val\_loss: 0.0625 - val\_accuracy: 0.9797 Epoch 3/5  375/375 [==============================] - 16s 41ms/step - loss: 0.0476 - accuracy: 0.9852 - val\_loss: 0.0534 - val\_accuracy: 0.9833 Epoch 4/5  375/375 [==============================] - 16s 42ms/step - loss: 0.0350 - accuracy: 0.9891 - val\_loss: 0.0427 - val\_accuracy: 0.9869 Epoch 5/5  375/375 [==============================] - 16s 42ms/step - loss: 0.0285 - accuracy: 0.9911 - val\_loss: 0.0409 - val\_accuracy: 0.9875 313/313 [==============================] - 1s 4ms/step - loss: 0.0308 |

|  |
| --- |
| - accuracy: 0.9893  Test accuracy: 0.989300012588501  313/313 [==============================] - 2s 5ms/step Confusion Matrix:  [[ 977 0 0 0 0 0 1 1 1 0] [ 0 1132 1 1 0 1 0 0 0 0] [ 0 0 1029 0 1 0 0 2 0 0] [ 1 0 1 1001 0 6 0 0 1 0] [ 0 0 0 0 967 0 1 0 2 12] [ 1 0 0 3 0 886 1 1 0 0] [ 10 2 0 0 3 6 936 0 1 0] [ 0 2 8 3 0 0 0 1011 1 3] [ 3 0 2 1 0 2 0 1 964 1] [ 2 0 0 2 1 6 0 3 5 990]] |

K-Fold: The code uses K-Fold cross-validation for the Convolutional Neural Network (CNN) model trained on the MNIST dataset. K-Fold cross-validation divides the data set into 5 folds, shuffling the data for randomness. At each iteration, one bunch is determined for validation, and the remaining bunch is used for training. The CNN model architecture consists of convolutional layers for feature extraction and dense layers for segmentation. The model is clustered using the ADAM optimizer, the categorical cross-entropy loss function, and the accuracy metric. Training is performed for 5 periods on the training data of each bunch, including the validation data from the validation bunch. After training, the performance of the model is evaluated on the test set to estimate the test loss and accuracy. Additionally, a confusion matrix is constructed using model predictions and true scores from the test set, providing insights into model performance across classes This approach allows for robust and supportive evaluation of model performance identify potential areas for improvement, ensure model reliability and generalizability.

Confusion Matrix: The presented confusion matrix summarizes the performance of the classification model trained on the MNIST dataset, which contains signature digit images (scores 0 to 9.) Each row in the image represents an actual digit class, . while each column represents an instance of the predicted class. The numbers in the matrix indicate the counts of instances in which the model's prediction matches the true label for each class. For example, a value at (0, 0) indicates that the model correctly predicted 977 samples for a score of 0. The diagonal elements of the matrix represent correct predictions, while the non-diagonal elements represents a misclassification. For example, the values in (4, 9) indicate that the model incorrectly classified 12 samples of 9 points as scores Overall, it is a valuable tool for assessing the quality of a classification model work well and maintain its accuracy.