

**HANDWRITTEN DIGIT RECOGNITION**

Report by, Ahmed Pasha

**Dataset:** Built-in MNIST dataset (from keras.datasets)

You can access the code using the link below,

[zawr-assignment-1.ipynb - Colab](https://colab.research.google.com/drive/1NfHHQkRbBLsw5PUw7k727TYeNSxS6C4t" \l "scrollTo=xE0-NWzpmRxK)

**Explanation:**

This script implements a deep learning-based approach for handwritten digit recognition using the MNIST dataset. It follows a structured workflow covering data preprocessing, model training, evaluation, and real-time digit classification. The dataset, which consists of grayscale images of handwritten digits (0-9), is first loaded and normalized to improve model efficiency.

The script employs three classification models: **Convolutional Neural Networks (CNN), K-Nearest Neighbors (KNN), and Support Vector Machines (SVM)**. The CNN model is designed with convolutional layers, pooling layers, batch normalization, dropout regularization, and dense layers to extract hierarchical image features. KNN and SVM are used as baseline models, trained on flattened image data for comparison.

For CNN training, data augmentation techniques such as rotation, zoom, and shifting are applied to enhance model generalization. The model is compiled using the Adam optimizer and trained using the sparse categorical cross-entropy loss function. After training, the model is evaluated on the test set, and accuracy metrics for CNN, KNN, and SVM are compared to determine performance.

Additionally, the script includes a real-time handwritten digit classification module, allowing users to upload an image of a digit for prediction. The uploaded image undergoes preprocessing, including grayscale conversion, resizing, normalization, denoising, and thresholding to enhance clarity. The trained CNN model is then used to predict the digit, displaying both the predicted label and its confidence score. If the confidence is low, an alternative prediction is suggested.

Finally, the results are visualized using **matplotlib**, showing both the uploaded image with its predicted label and a probability distribution of all possible digit predictions. The trained CNN model is saved for future inference, enabling real-time digit recognition for applications like **automated form processing, postal code recognition, and banking security systems**.

**Challenges Faced:**

During the development of the **Handwritten Digit Recognition** project, several challenges arose, particularly due to the limitations of Google Colab and the complexity of implementing an efficient and accurate model.

One of the initial difficulties was **finding a suitable dataset**. While the MNIST dataset is widely available, sourcing it externally proved problematic due to its large size. Google Colab, with its restricted memory and storage limitations, was unable to handle the dataset efficiently, leading to the rejection of the idea of fetching the dataset externally. Instead, I had to rely on alternative methods available within Colab.

Another challenge was the **difference in development environments**. Most online tutorials and GitHub repositories implemented the project in **Visual Studio Code**, whereas I was constrained to use **Google Colab**. This meant that certain functionalities, library installations, and file handling approaches needed to be adjusted to work within the Colab environment. I referred to multiple videos to understand the implementation but had to modify the approach significantly to ensure compatibility.

Additionally, the **complexity of existing GitHub repositories** made it difficult to directly integrate available solutions. Many of the repositories contained multiple interconnected files, making it hard to extract and modify specific portions of the code for my own implementation. Instead of following these complex structures, I decided to make my implementation unique by adding a **user-uploaded image feature**, which was not included in the standard approaches I had come across.

Training the models was another hurdle. I implemented **three different models** to compare performance, but the **training process was highly time-consuming**. Saving the trained model in Colab also required careful handling, as it consumed significant storage space.

The most challenging part was **processing uploaded images and ensuring correct predictions**. Initially, the model was producing **incorrect predictions** due to noise and variations in user-uploaded images. To resolve this, I applied **image denoising techniques** and introduced **confidence level calculations using probability logic** to refine predictions. These enhancements significantly improved accuracy but required multiple iterations to fine-tune the results effectively.

Despite these difficulties, I successfully completed and deployed the project, ensuring that it functioned smoothly within the Colab environment while maintaining uniqueness in its implementation.

**Common Questions:**

1. ***What algorithm is used?***

The code uses multiple algorithms for handwritten digit recognition:

* **Convolutional Neural Networks (CNN)**: A deep learning approach used for image classification.
* **K-Nearest Neighbors (KNN)**: A simple machine learning algorithm that classifies based on the majority class among the k-nearest neighbors.
* **Support Vector Machine (SVM)**: A classification model that finds the optimal hyperplane to separate data points into different classes.

1. ***What models are used?***

* · **CNN Model**: Built using TensorFlow and Keras, consisting of multiple convolutional layers, pooling layers, dense layers, dropout, and batch normalization for improving accuracy.
* · **KNN Model**: Implemented using KNeighborsClassifier from sklearn.neighbors.
* · **SVM Model**: Implemented using SVC(kernel='linear') from sklearn.svm.

1. ***What evaluation metrics are used to evaluate the model?***

· **Accuracy Score**: Used for both KNN and SVM using accuracy\_score(y\_test, knn.predict(x\_test\_flat)) and accuracy\_score(y\_test, svm.predict(x\_test\_flat)).

· **Loss and Accuracy for CNN**: Evaluated using model.evaluate(x\_test, y\_test), which returns the loss and accuracy of the model.

1. ***Why should we drop missing values?***

Dropping missing values is necessary because:

* Missing values can introduce bias and lead to incorrect predictions.
* Many machine learning models cannot handle missing values directly and require imputation or removal.
* It ensures that the dataset remains clean and structured for training models effectively.

1. ***Why do we have to convert categorical variables into dummy/indicator variables?***

· Machine learning models cannot directly process categorical text data; they require numerical input.

· Encoding categorical variables into dummy variables (one-hot encoding) ensures that the model can properly interpret the relationships between different categories.

· It helps prevent the model from misinterpreting categorical variables as ordinal values.

1. ***What is the meaning of standardization of features?***

· Standardization scales features to have **zero mean** and **unit variance**, ensuring that no single feature dominates due to differing scales.

· It helps in faster convergence of gradient descent in neural networks and improves the performance of distance-based models like KNN and SVM.

1. ***What parameters are used?***

· **For CNN Model**:

* · Conv2D(32, (3,3), activation='relu'): 32 filters, 3x3 kernel, ReLU activation.
* MaxPooling2D((2,2)): Reduces spatial dimensions by half.
* Dense(128, activation='relu'): Fully connected layer with 128 neurons.
* Dropout(0.5): Regularization technique to prevent overfitting.
* BatchNormalization(): Normalizes activations to speed up training.
* optimizer='adam': Adaptive optimization technique.
* loss='sparse\_categorical\_crossentropy': Suitable loss function for multi-class classification.

· **For KNN Model**:

* · n\_neighbors=3: The number of nearest neighbors to consider.

· **For SVM Model**:

* · kernel='linear': Uses a linear hyperplane for classification.

1. ***What functions are used?***

· **Data Preprocessing**:

* · mnist.load\_data(): Loads the MNIST dataset.
* reshape(): Converts images to required dimensions.
* to\_categorical(): Converts labels to one-hot encoding for CNN.
* ImageDataGenerator(): Augments image data for better generalization.

· **Model Training & Evaluation**:

* · Sequential(): Initializes a neural network model.
* model.compile(): Configures the model with optimizer, loss function, and metrics.
* model.fit(): Trains the model on input data.
* model.evaluate(): Evaluates the trained model on test data.
* accuracy\_score(): Computes accuracy for KNN and SVM.

· **Image Processing**:

* · Image.open(): Loads an image.
* convert('L'): Converts the image to grayscale.
* resize(): Resizes the image to 28x28 pixels.
* cv2.GaussianBlur(): Applies blurring to reduce noise.
* cv2.threshold(): Applies binary thresholding for better contrast.