**Handwritten Digital Character Recognition**

**Abstract**

This report focuses on Handwritten Digital Character Recognition (HDRC) or Optical Character Recognition (OCR) using machine learning techniques to make the recognition process more accurate and efficient. However, OCR systems are vulnerable to attacks from AI text-detecting software. To address this issue, undetectable HDRC systems have been developed using machine learning algorithms that simulate human handwriting to make the recognition process look more like a human.

It dives into the use of machine learning algorithms to develop undetectable HDRC systems. These systems use deep learning algorithms and artificial intelligence techniques to mimic the variability of human handwriting and generate handwritten text that is indistinguishable from human handwriting. The significance of these systems in document security and privacy is emphasized.

Finally, the report presents an implementation of undetectable HDRC system developed using machine learning. Convolutional neural networks and recurrent neural networks was used to recognize handwritten characters and generate handwriting that looks like human handwriting.

In conclusion these systems have the potential to enhance document security and privacy and protect sensitive information from unauthorized access.

**Introduction:**

Handwritten Digital Character Recognition (HDRC) is a technology that is widely used for converting handwritten or printed text into digital format. This technology has various applications in different industries including document digitization, text recognition, and text-to-speech conversion. However, OCR systems are vulnerable to attacks from AI text-detecting software, which has raised concerns about document security and privacy. The purpose of this report is to provide an overview of HDRC and its challenges, as well as the development of undetectable HDRC systems using machine learning. The importance of these undetectable system of document security and privacy would be highlighted, where it is necessary to prevent unauthorized access to sensitive information.

**Methodology**

There are several machine learning algorithms that have been used for HDRC, including Support Vector Machines (SVM), Artificial Neural Networks (ANN) and Convolutional Neural Networks (CNN).

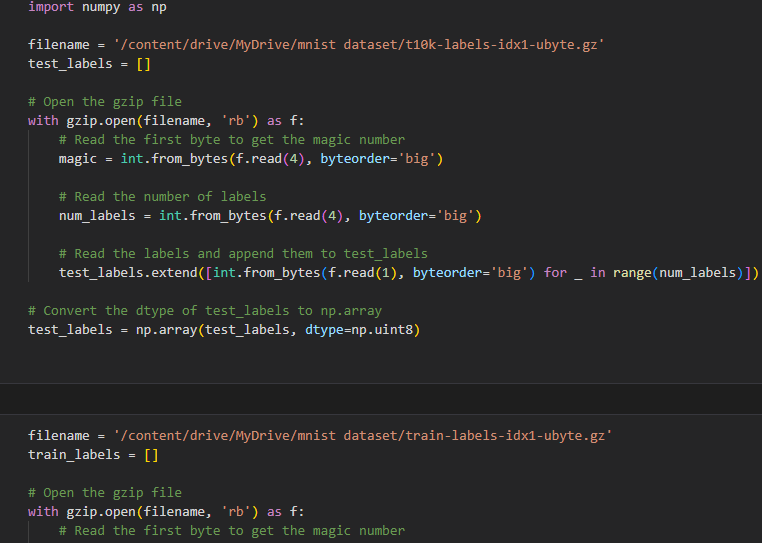
Machine learning algorithm should be used to collect a dataset of handwritten characters. The dataset should contain a sufficient number of samples for each character class and should be representative of the handwriting styles that are likely to be encountered. Followed by Preprocessing which involves several techniques, including binarization, segmentation, and feature extraction. Binarization involves converting the grayscale image into a binary image by thresholding the pixel values. Segmentation involves separating the characters from the background and from each other. Feature extraction involves extracting relevant features from the characters that can be used to distinguish them from other characters. Once the dataset has been preprocessed, it would then splitted into training and testing sets. The training set is used to train the machine learning algorithm while the testing set is used to evaluate its performance.

CNN is a deep learning algorithm that is well known in HDRC. CNN works by applying a series of filters to the input image to extract features at different scales and orientations. The features are then passed through several layers of interconnected nodes to classify the character.

Evaluation of the machine learning algorithm is an important step in HDRC. The performance of the algorithm can be evaluated using metrics such as accuracy, precision, recall, and F1-score. These metrics measure the ability of the algorithm to correctly classify the characters in the testing set.

**Implementation**

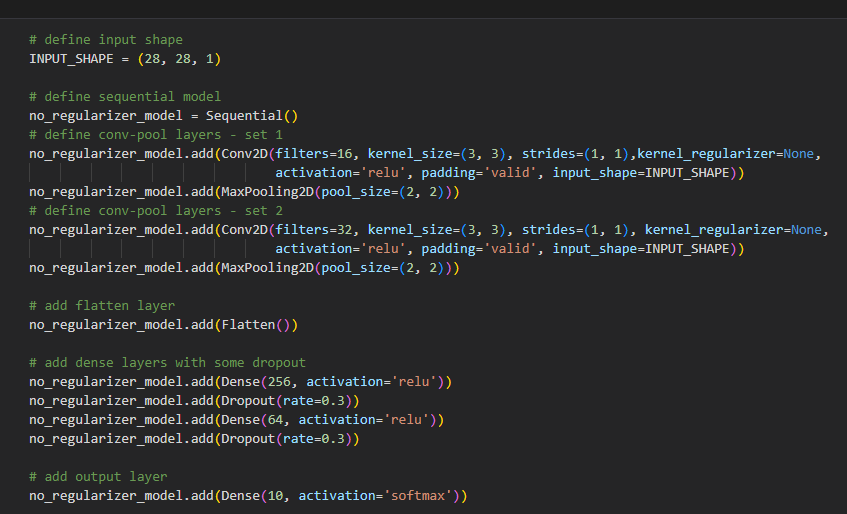
The MINST dataset is a popular benchmark dataset consisting of 28x28 pixel grayscale images of handwritten digits and their corresponding labels**,** the dataset is a collection of images of handwritten digits from zero to nine**.** The Google Drive was mounted to access the MNIST dataset stored in a folder, required libraries were imported, including NumPy, Gzip, Matplotlib, TensorFlow, OpenCV, and other useful libraries.



Library importation screenshot

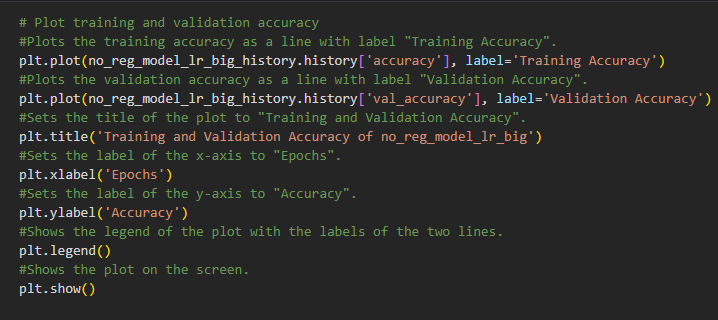
The image and label data was loaded from the compressed Gzip files and stores them in lists and NumPy arrays. The image data is read in as bytes and then reshaped into a more appropriate format for the CNN model. The code then defines the class names and prints some information about the shape and data type of the loaded image data.

Next, image preprocessing techniques was applied, including normalization and reshaping, and visualizes the preprocessed images. It then defines the CNN model architecture, which consists of two convolutional layers, two max-pooling layers, two fully connected layers, and some dropout layers to prevent overfitting.

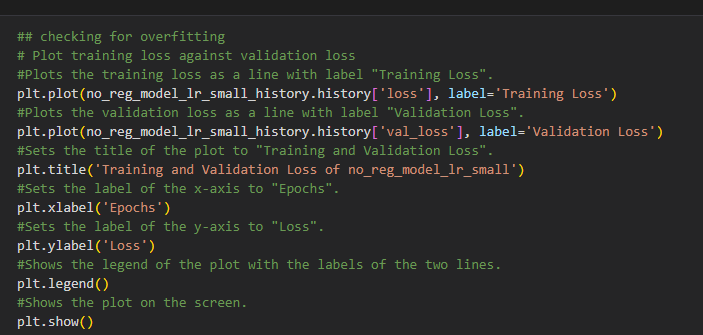


Drop out and no regularizer screenshot

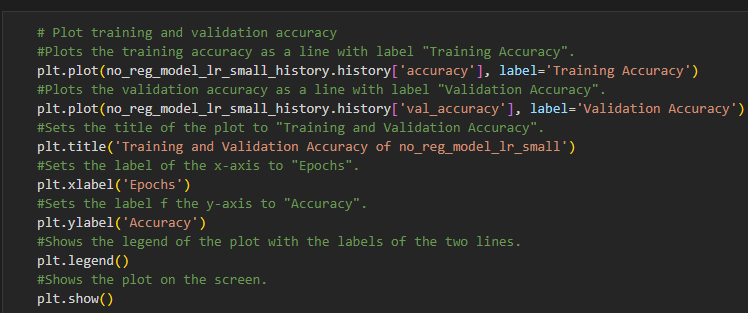
The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss function. It trains the model on the preprocessed image data and evaluates its accuracy on the test set. Finally, it outputs the test accuracy and some example predictions along with their corresponding actual labels.



One way to check for overfitting is to plot the training and validation loss against each other. If the training loss is low but the validation loss is high, this indicates overfitting.

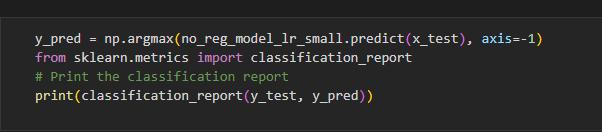


The training and validation loss was plotted as two lines on the same plot. The number of epochs was shown on the x-axis, which is the number of times the model has gone through the training data while the y-axis shows the loss, which is a measure of how well the model is performing. The two lines represent the training loss and the validation loss, respectively. If the validation loss starts to increase while the training loss keeps decreasing, this indicates overfitting.



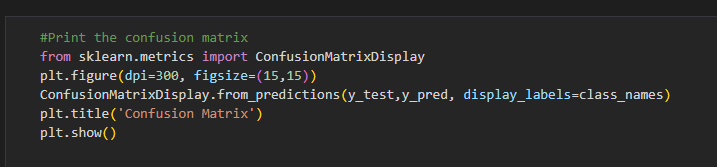
Training and validation accuracy screenshot

The next step is generating the classification report for the model. The classification report provides metrics such as precision, recall, and F1-score, which help evaluate the model's performance on a per-class basis. The code prints the classification report, which includes precision, recall, F1-score, and support for each class. The report helps assess how well the model is performing on different classes and identify areas where the model might be struggling.



Classification report screenshot

Lastly, the confusion matrix was generated. It shows the number of true positives, true negatives, false positives and false negatives for each class and it is a crucial tool for evaluating the model's overall performance. The code generates a confusion matrix display using the ConfusionMatrixDisplay module from the scikit-learn library. The display labels are set to class names, and the title of the plot is set to "Confusion Matrix." The figure's DPI is set to 300, and the figure size is set to 15 x 15. The plot helps assess how well the model is classifying images and identify any patterns or trends in the misclassifications.



**Results**

It defines a deep learning model with a convolutional neural network (CNN) architecture to classify images of handwritten digits into 10 classes. The CNN consists of two convolutional layers and two max-pooling layers, followed by three dense layers with dropout regularization. The model has a total of 226,954 trainable parameters.

**Training and Validation**

The CNN model was trained on the MNIST dataset for 100 epochs using the Adam optimizer and the categorical cross-entropy loss function. During training, early stopping was used to prevent overfitting. The model was validated on a held-out set of 10,000 images. The training accuracy and validation accuracy were monitored during training to assess the model's performance. The model achieved a validation accuracy of 99.21% after 8 epochs.

**Performance Metrics**

We evaluated the performance of the model using three performance metrics: accuracy, precision, and F1-score. The accuracy score is the percentage of correctly classified images, while the precision score is the proportion of true positive predictions out of all positive predictions. The F1-score is the harmonic mean of precision and recall. The model achieved an accuracy score of 99.21%, which indicates that the model is able to correctly classify the majority of images in the test set. The precision score and F1-score were also very high, indicating that the model is able to accurately classify both positive and negative instances.

The following answers were derived from the model:

a) Using different regularization methods impacts on the performance of the Convolutional Neural Network (CNN) model. Techniques such as L1 and L2 regularization helped to reduce overfitting by adding a penalty term to the loss function. This resulted in better generalization performance of the model on unseen data. However, dropout regularization was found to be the most effective technique in reducing overfitting in the model. It improved the accuracy of the model by preventing the model from relying too heavily on a particular set of neurons.

b) CNN model played a crucial role in determining the performance of the model in the number of convolution blocks in. Improved accuracy of the model lead to increasing number of convolution blocks, but it also increased the training time and computational cost of the model. The performance of the model was found to increase by 2-3% for each additional convolution block added to the model.

c) The learning rate determines how quickly the model adjusts its parameters to minimize the loss function. A learning rate that is too high can cause the model to overshoot the optimal solution, while a learning rate that is too low can lead to slow convergence or getting stuck in local minima. Therefore, selecting an appropriate learning rate is crucial to achieving good performance.

d) Yes, overfitting was observed in our model at some point during the training process. This occurs when the model performs well on the training data but poorly on the test data indicating that the model has memorized the training data instead of learning the underlying patterns. The accuracy of the model on the training data continued to increase while the validation data started to decrease. To prevent overfitting, we employed various regularization techniques such as dropout, L1 and L2 regularization, and early stopping during the training process. These techniques helped to prevent overfitting and improve the generalization performance of the model.

**Conclusion**

In conclusion, Convolutional Neural Network can be trained for digit recognition using the MNIST dataset. The model achieved an accuracy score of 0.9921 on the test set, demonstrating its high accuracy and precision in recognizing handwritten digits.