Automatic text transcription (OCR)

Aline SIROT, Mathieu HOUSSART

23 December 2024

All figures are at the end of the documents.

1 Introduction

Automatic text transcription using Optical Character Recognition (OCR) is a technology that converts different types of document, *such as scanned paper documents, PDFs or images*, into editable and searchable data. OCR systems are typically used in digitizing printed text so it can be edited, searched, and stored more compactly. Our project focuses on developing a system that can recognize handwritten digits and letters.

2 Task definition

The task of the project is to develop and train a model for OCR using handwritten characters from images. This model needs to be able to accurately classify characters from a given dataset, which consisted of 62 characters at first (digits 0–9, lowercase letters a–z and uppercase letters A–Z) and only the 26 characters at the end. The OCR model should handle both digit and letter classification, making it applicable for various OCR applications, such as reading handwritten forms and documents.

3 Existing solutions & information sources

Several existing solutions for OCR have been developed, including commercial systems such as Tesseract and Microsoft OCR, as well as academic solutions. Tesseract is an open source OCR engine developed by Google and supports multiple languages, including character recognition for a variety of fonts and handwriting. For machine learning-based approaches, deep learning models like convolutional neural networks (CNNs) have become the dominant technology, outperforming traditional methods in terms of accuracy and adaptability.

4 Our solution

4.1 Definition (Solution 1)

Our solution utilizes a convolutional neural network (CNN) to classify handwritten characters. We used the MobileNetV2 architecture as the base model for transfer learning, which is fine-tuned for our dataset. The model takes grayscale images of 28x28 pixels and outputs probabilities for 62 different classes: 10 digits and 26 lowercase letters and 26 uppercase letters. The preprocessing pipeline normalizes the images and uses data enhancement to improve the generalization of the model.

4.2 Definition (Solution 2)

Our 2nd solution leverages a Residual Network (ResNet) architecture for classifying handwritten characters. ResNet is a deep learning model designed to address the vanishing gradient problem common in deep neural networks by introducing residual connections. These connections allow the network to learn identity mappings, making it easier to train very deep networks effectively.

Data Processing: The ResNet model is designed to process grayscale images of size 28x28 pixels, which are normalized and resized to a 3-channel format for compatibility with the model. The network outputs probabilities for 26 classes, where lowercase and uppercase letters are merged into the same class.

How ResNet Works: The ResNet architecture introduces residual blocks, which consist of one or more convolutional layers. A shortcut connection bypasses these layers, directly adding the input of the block to its output. This approach allows the model to retain important features learned earlier in the network and helps it converge faster during training.

Main Components of ResNet in This Project:

- **Convolutional Layers:** Extract features from the input image by applying filters.
- **Batch Normalization:** Normalizes layer outputs to speed up training and improve stability.
- **ReLU Activation:** Adds non-linearity to the network.
- **Residual Connections:** Directly pass the input of a layer to later layers, improving gradient flow and feature reuse.
- **Global Average Pooling:** Reduces the spatial dimensions of feature maps to a single value per map, simplifying the output for the classifier.
- **Dense Layer with Softmax Activation:** Outputs the probabilities for the 26 classes.

4.3 DataSets

We used two different datasets in our project for experimentation, each influencing the results of our model.

First dataset: For initial experiments, we used the "Handwritten English Characters and Digits" dataset available on Kaggle (`https://www.kaggle.com/datasets/sujaymann/handwritten-english-characters-and-digits`). This dataset contains handwritten digits (0-9) and English letters (a-z / A-Z) in various fonts and styles. The first dataset was utilized to train and validate the model, but it showed some overfitting due to the limited variation in character representation.

Second dataset: For the final set of experiments, we switched to a more diverse dataset that provides a broader range of handwriting styles and additional augmentation, helping improve the model's robustness (`https://www.kaggle.com/datasets/crawford/emnist/data?select=emnist-letters-train.csv`). This dataset included handwritten characters captured in various conditions and from different sources, better simulating real-world handwriting. This dataset only possesses 26 classes. It doesn't contain numbers and grouped the lowercase and uppercase letters together in the same classes which makes it easier to train.

4.4 Firsts experiments

We did encounters some problems, as showed on this confusion matrix. Our biggest problem was the lack of data which caused us a low accuracy : ***Test Accuracy: 27.42%***

figure 1

4.5 Experiments

For the experiments on the second model, we used the other dataset with 26 classes. Data Augmentation was also applied here to make the model perform better with unseen data. This model used Adam optimizer and sparse categorical crossentropy loss.

figure 2

4.6 Results

After training the model for 5 epochs, we achieved an accuracy of 92% on the test set. The model also demonstrated good generalization, with balanced performance across all letters. We used a confusion matrix to evaluate the performance of the model, revealing that most misclassifications occurred between visually similar characters.

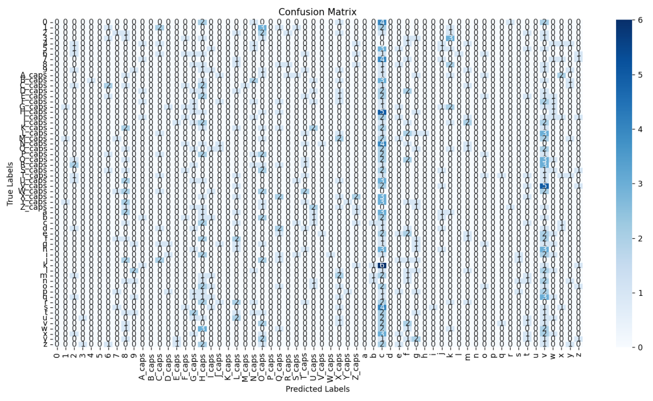

Figure 1: First matrix

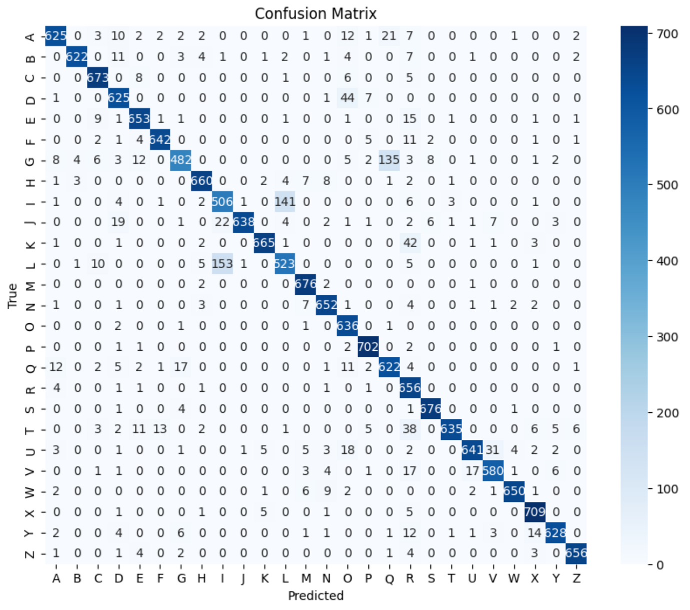

Figure 2: Final matrix