ELL409 Machine Learning Project

Handwriting Recognition

Ashi Veerman[1], Rishabh Barola[2], and Vasu Sharma[2]

[1]Department of Mathematics, IIT Delhi
[2]Department of Electrical Engineering, IIT Delhi

November 24, 2024

Abstract

This document explores the use of machine learning for handwriting recognition. We tested various models, including SVM, MLP, and a CNN-RNN combination, which provided the best results. Training on the Kaggle handwritten names dataset, we achieved decent accuracy in recognizing handwritten names on test data.

Keywords
handwritten character recognition, CNN, neural network, RNN

1 Introduction

Handwriting recognition refers to the ability of computers to interpret handwritten input from sources like paper documents, photographs, or touchscreens. It can be classified as offline or online. Online recognition tracks pen movements on devices, capturing dynamic clues like motion and pressure. Offline recognition involves converting handwritten text in images into character codes for text-processing applications. It is more challenging due to the diversity of writing styles and typically requires handling tasks like character segmentation and word prediction. While most OCR systems focus on machine-printed text, intelligent character recognition (ICR) targets handwritten text.

2 Dataset Acquisition

The dataset, obtained from Kaggle, contains 330,961 training images and 41,370 validation images, each representing handwritten names.

3 Image Preprocessing

The dataset preprocessing involved the following steps:

- Handling Missing Data: The training set contained 565 missing values, while the validation set had 78. All rows with the label `UNREADABLE` were removed from both the training and validation sets.
- Image Preprocessing:
  - Resizing and Padding: For each image, the width and height were adjusted to a fixed size of 256 pixels (width) and 64 pixels (height). If the image was wider than 256 pixels, it was cropped to 256 pixels. Similarly, if the image was taller than 64 pixels, it was cropped to 64 pixels.
  - Padding: The images were padded with white space (255) to ensure consistent dimensions of 64x256 pixels.
  - Rotation: After resizing and padding, the image was rotated 90 degrees clockwise to align it with the desired orientation for further processing.

4 Model Training

For handwritten character recognition, we initially tested Support Vector Machines (SVMs), but they were not suitable for handling the complexities of multiple characters. While SVMs work well in binary classification, they struggle with the high dimensionality and sequential nature of handwriting data. We then experimented with Multi-Layer Perceptrons (MLPs) and Convolutional Neural Networks (CNNs). CNNs performed significantly better, as they are adept at identifying patterns in visual data through hierarchical feature learning. However, CNNs alone struggled to capture the sequential relationships between characters. To address this, we combined CNNs with Recurrent Neural Networks

(RNNs), which are effective at modeling sequential dependencies. The CNN handled feature extraction, while the RNN captured the temporal relationships between characters. This hybrid approach resulted in the best performance for recognizing multiple characters and words in handwritten images.

Model Architecture Description

The proposed model architecture is a hybrid deep learning network that combines Convolutional Neural Networks (CNNs) for feature extraction and Recurrent Neural Networks (RNNs), specifically Bidirectional Long Short-Term Memory (Bi-LSTM) layers, for sequence modeling.

Input Layer

The input layer accepts images of size 256x64 pixels in grayscale. The input shape is represented as $(None, 256, 64, 1)$, where "None" indicates that the batch size can vary.

Convolutional Layers

- Conv1: The first convolutional layer uses 32 filters of size 3x3. The output feature map is of size $(256, 64, 32)$. This layer is followed by Batch Normalization and a ReLU Activation function. These operations help stabilize the learning process and introduce non-linearity.
- MaxPooling1: A max pooling layer is applied to reduce the spatial dimensions by half, resulting in an output size of $(128, 32, 32)$.
- Conv2: The second convolutional layer applies 64 filters of size 3x3, producing a feature map of size $(128, 32, 64)$. Batch Normalization and ReLU Activation are applied again.
- MaxPooling2: Another max pooling layer reduces the dimensions to $(64, 16, 64)$.
- Conv3: The third convolutional layer uses 128 filters of size 3x3, with an output size of $(64, 16, 128)$. Batch Normalization and ReLU Activation are applied.
- MaxPooling3: The final max pooling operation reduces the dimensions to $(64, 8, 128)$.

Dropout

Dropout layers are used after Conv2 and Conv3 to prevent overfitting. These layers randomly set a fraction of input units to 0 during training, which encourages the model to generalize better.

Reshape and Dense Layers

After the convolutional layers, the output is reshaped to combine the height and width dimensions into a single dimension, resulting in a shape of $(64, 1024)$. This reshaped data is then passed through a Dense layer with 64 units, further processing the features extracted by the convolutional layers.

Bidirectional LSTM Layers

To capture the sequential nature of handwriting, two Bidirectional LSTM (Bi-LSTM) layers are used:

- LSTM1: The first Bi-LSTM layer processes the sequence in both forward and backward directions. This layer has 512 units, allowing the model to capture dependencies from both past and future time steps.
- LSTM2: The second Bi-LSTM layer processes the sequence bidirectionally, also with 512 units.

Final Dense Layer and Softmax Activation

After the LSTM layers, a Dense layer with 30 units is applied, which outputs the predicted probabilities for each character in the sequence. A Softmax activation function is applied to ensure that the predicted probabilities sum to 1 for each time step.

Summary of Model Parameters

- Total Parameters: The model has a total of 2,406,878 parameters.
- Trainable Parameters: Of these, 2,406,430 are trainable and will be updated during the training process.
- Non-trainable Parameters: The model has 448 non-trainable parameters, which are fixed during training (e.g., parameters related to batch normalization).

5 Loss Function: CTC Loss

Mathematically, the CTC loss function is defined as follows:

Let the target sequence be $y = (y_1, y_2, ..., y_T)$, where y_t represents the label at time step t (e.g., the character at that time). Let the model output be a probability distribution over all possible labels at each time step. For each time step

t, the model produces a vector of probabilities $\hat{y}_t = (\hat{y}_t^1, \hat{y}_t^2, ..., \hat{y}_t^C)$, where C is the number of classes (e.g., the total number of possible characters plus a special "blank" token to account for no character being present).

The goal of CTC is to compute the probability of the correct label sequence y given the model's predictions, i.e., to find $P(y|X)$, where X represents the input image. The CTC loss is the negative log-likelihood of the target sequence, which is computed by summing over all possible alignments of the input sequence to the target sequence.

The probability of a target sequence y given the input sequence X is computed as:

$$P(y|X) = \sum_{a \in \mathcal{A}(y)} P(a|X)$$

where $\mathcal{A}(y)$ represents all possible alignments of the target sequence y with the input sequence X, and $P(a|X)$ is the probability of an alignment a given the input X. Since there are many possible alignments, this sum is computationally expensive, and the CTC loss is typically optimized using dynamic programming.

The CTC loss function is computed as:

$$\mathcal{L}_{CTC} = -\log P(y|X)$$

In practice, CTC loss enables the model to handle varying lengths of input sequences by aligning them with the output sequence in a flexible manner. The model does not need to explicitly know where each character occurs in the input, which is particularly useful in handwritten text recognition where character spacing and positioning can vary significantly across samples.

Key Features of CTC Loss

- No explicit alignment: CTC does not require exact alignment of input and output sequences, making it robust to handwriting variations.
- Handles variable-length sequences: It works with inputs of different lengths, typical in handwriting tasks.
- Blank token: CTC introduces a "blank" token to model instances where no character is predicted at a given time step.

By using CTC loss, the model is capable of learning to predict sequences in an end-to-end fashion without needing explicit supervision for each character's position in the input image.

6 Results

Training and Validation Loss Plot

The training and validation loss curves are shown in the attached plot, which depicts the convergence of the model over time. The loss values decrease steadily as the model trains, indicating effective learning. The model was trained using the CTC loss function, which is suitable for sequence-to-sequence tasks like handwritten text recognition. The plot shows the performance of the model over the course of training and validates how well it generalizes to unseen data.

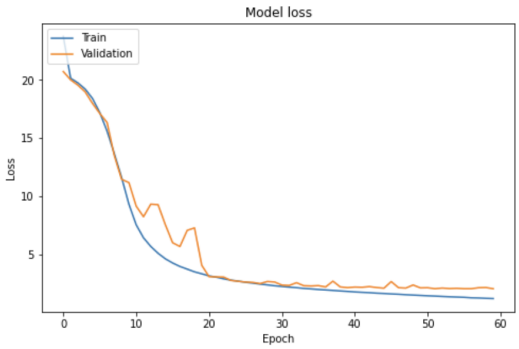

Figure 1: Training and Validation Loss Plot

Performance Metrics

To evaluate the accuracy of the model, two important metrics were used:

- **Character Error Rate (CER):** This measures the percentage of character errors in the predicted sequence compared to the actual labels. A lower CER value indicates better character-level accuracy in recognition.
- **Word Error Rate (WER):** This measures the percentage of word errors, considering both incorrect word recognition and errors in word order. A lower WER indicates better word-level recognition performance.

On the validation set, the model achieved the following results:

- **Correct Characters Predicted:** 88.67%
- **Correct Words Predicted:** 74.93%
- **Character Error Rate (CER):** 7.39%
- **Word Error Rate (WER):** 25.72%

The lower CER and WER values suggest that the model is performing well in recognizing both characters and words, although there is still room for improvement. The accuracy in predicting correct characters (88.67%) and words (74.93%) indicates that the model is capable of handling handwritten text with a reasonable degree of accuracy. The relatively higher WER shows that there may be occasional issues with word recognition or word order, which may be due to variability in handwriting or model limitations.

These metrics confirm that the model is suitable for handwriting recognition tasks, with an emphasis on character-level recognition. Further improvements in training, data augmentation, or model architecture may help reduce the error rates and improve performance.

Confusion Matrix

To further evaluate the model's performance, the confusion matrix was computed for the predicted versus actual labels on the validation set.

The confusion matrix is plotted below:

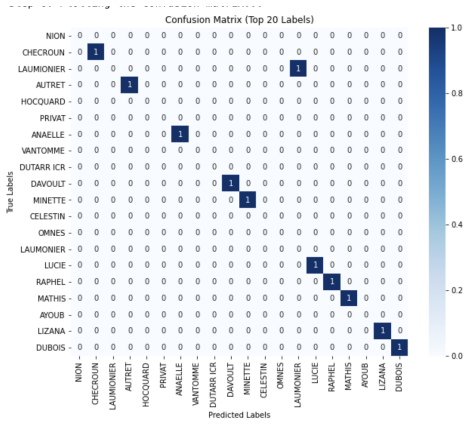

Figure 2: Confusion Matrix for Model Predictions

7 Discussion

Various models, including SVMs, MLPs, and CNNs, were tested, but the combination of CNNs and RNNs proved most effective for handwriting recognition. CNNs extracted spatial features, while RNNs captured the sequential nature of handwriting.

Data preprocessing posed challenges, such as identifying unreadable images and handling rotated characters. We filtered out 'Unreadable' images and standardized orientations to ensure consistent input.

The iterative testing and preprocessing led to strong performance, but there is still potential for improvement in model architecture and dataset diversity.

Conclusion

This study proposed a handwriting recognition model combining CNNs and RNNs, where CNNs extracted spatial features and RNNs captured sequential dependencies, achieving high accuracy on a large dataset.

Future work could explore transformer-based models like Vision Transformers (ViTs) for better handling of long-range dependencies, and augmenting the dataset with diverse handwriting samples. Incorporating attention mechanisms, expanding to multiple languages, and integrating with tasks like document parsing or text-to-speech could further enhance the model's performance and versatility.

In summary, future improvements could focus on transformers, better datasets, attention mechanisms, and multilingual support.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems (NeurIPS), 25:1097–1105, 2012.

[2] Sepp Hochreiter, Jürgen Schmidhuber. *Long Short-Term Memory*, Neural Computation, 9(8):1735–1780, 1997.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, MIT Press, 2016.

Github Link

The codebase for this project is available on GitHub and can be accessed through the following link:

https://github.com/crownCTDM/ELL409_Project