# The Write Way: Accurate Handwriting Transcription Using Deep Learning

Ibrahim Dharhan
*Stanford University*

Jakob Nordhagen
*Stanford University*

Raymond Zhen
*Stanford University*

## 1 Introduction

The transcription of handwriting into digital text is a valuable and interesting mechanism, with broad applications in a variety of modern industries. For instance, transcription is useful to insurance companies for parsing handwritten claim documents, hospitals for processing large numbers of patient forms, university students for digitizing course notes, or any initiative that aims to preserve old handwritten material. There are existing transcription services that perform this task by hand. However, manual handwriting transcription is far too time-consuming and labor-intensive to be scalable. To that end, we examined and built upon various methods of automatic, software-based handwriting transcription that shared the basic goal of converting images of handwritten text into digital text.

The task of handwriting recognition, to a computer, is more difficult than it might seem at first. The sheer variability of handwriting poses a significant challenge, as the handwriting of different individuals, even writing the same words, can look quite different. Even within one individual's writing, there is considerable human variability in terms of character size, angle, and shape. Moreover, a large portion of the world's existing handwritten data is in cursive, in which words are often slanted and contain flourishes. Boundaries between characters are often unclear and hard to automatically infer. This makes the recognition problem an interesting one, and one that lends itself well to deep learning methods.

This task is well-suited to deep learning methods that have recently been developed for use in computer vision, including Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Recurrent Neural Networks (RNNs). Our research found that to date, the most accurate method of recognizing handwritten text recognition involves the use of a combined Convolutional Recurrent Neural Network (CRNN) architecture and a Connectionist Temporal Classification (CTC) loss function.

The input to our algorithm is an image of a handwritten English word. We used a trained CRNN model to output the predicted transcribed text of the word. We implemented the following to improve the accuracy of the CRNN model: post-processed the model's predictions to correct errors, applied a de-slanting transform to the input images, increased the number of CNN layers, fine-tuned the batch size, and upgraded the decoder to use a word-based beam search algorithm. The metric we used to evaluate the improvements to the model was the Character Error Rate and the Word Accuracy.
We defined Character Error Rate as:

$$CER = \frac{S+D+I}{N}$$

where, with respect to the number of actions to achieve ground truth, S = Substitutions, D = Deletions, and I = Insertions. N = number of characters in ground truth text. The Word Accuracy Rate was defined as the percentage of words we accurately predicted.

## 2 Related Work

**Offline and Online Transcription:** The scope of this examination was focused on offline transcription rather than online transcription: to be clear, offline transcription acts on images of handwriting that have been pre-scanned, whereas online handwriting transcription acts on live or animated handwriting, analyzing movement from handwriting strokes in real time [15]. While online methods generally achieve a higher recognition rate than offline methods, offline transcription is far more practical and has a wider range of applications; for that reason, we chose to focus efforts on offline transcription.

**Preprocessing:** It is a nontrivial task to preprocess images of handwriting into a usable format for training. Images may have noise, uneven illumination, artifacts, or low resolution. For this reason, scanned documents achieve a much higher word recognition rate [20]. The most common preprocessing method is stepwise: a localization step classifies components of an image and groups them into candidate text regions and segments them into exclusive outlines of image blocks [7].

The end result is lines of text with minimal white space and noise.

**The Development of Transcription Techniques:** Early techniques to classify aspects of handwriting used divide-and-conquer methods. Bozinovic et al, for instance, developed a system to remove slants of the strokes, smooth out discontinuities or roughness, and segment parts that could be potential letters [4]. However, with the advent of computational resources, it became possible to recognize handwriting without the need for segmentation. One notable algorithm was backpropagation, which allowed the training of neural networks with nonlinear activation functions [16]. The MNIST dataset, which provided plenty of training data for the recognition of isolated handwritten characters, empowered even more powerful learning methods [10], such as CNNs. CNNs do not require previous knowledge of features and have been shown to be very effective in complex pattern recognition—one model was able to achieve over 99.8% accuracy for single digits [1], and a Gated-CNN-BGRU optical model achieved word precision of 71.54% [6]. Then, in combination with Markov Models [12], interpretive processing of letters and sentences was enabled. Current state-of-the-art models use LSTM and RNNs for line recognition [2].

## 3 Dataset

We used the IAM database [14], which consists of 13,353 images of handwritten lines of text (a total of 115,320 words) created by 657 writers, transcribed from the Lancaster-Oslo/Bergen Corpus of British English. The database text was scanned at a resolution of 300dpi and saved as PNG images of varying lengths and widths, with 256 gray levels. The text was separated and labeled at the sentence and word level. Each of the 115,320 words were segmented, labeled, and verified to be ground truth. We split these data into 95% training data and 5% validation data.

industrie," Mr. Brown commented icily. „ let us have a

Figure 1: A line of written text from IAM database

## 4 Methods

**Overview of Baseline Transcription Model:** The model we trained was based on a Convolutional Recurrent Neural Network (CRNN) architecture, as developed by Scheidl [17]. The CNN that we used was built using TensorFlow and consisted of a number of layers, each of which contained a 2D convolution, a batch normalization layer, a ReLU activation function, and a max-pooling layer. The CNN output extracted visual features of the image, which were fed through a bidirectional dynamic RNN implemented as LSTM, which produced an output text that was decoded from a CTC loss function.

### 4.1 Making Text Images a Standard Size

The images from the IAM database did not have a standard size, so we resized the images to a standard format of 128x32 (length by width). We also normalized the gray-value pixels of the image. When the images had a length less than 128 pixels, white space was added.

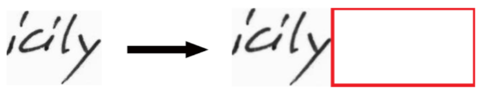

Figure 2: Original image converted to 128x32 by adding white space (highlighted with red box for clarity)

### 4.1 Convolutional Neural Network

A CNN was used to extract visual features from the words. During testing, the best performing network had the following structure:

1. A five-layer convolutional network, where a 5x5 kernel filter was used on the first two layers, and a 3x3 filter was used on the last three layers.
2. A non-linear ReLU function was applied, which had the advantage over other activation functions because it did not require activating all the neurons at the same time.
3. A max-pooling layer was used to reduce the dimensions of the feature maps, diminishing the amount of computation performed in the network. For each layer, the height of the image was reduced by 2. This step diminished the image regions and the output of the feature map had a size of 32 x 256 (time steps x features).

**Results of CNN Layer**: The CNN outputted a sequence of length 32, each with 256 features. At this stage, even before RNN layers were applied, we saw qualitative patterns that corresponded to high-level properties of the input. For instance, one feature had a high correlation of duplicate characters like “mm” in "commented" (Fig. 3), and another feature had a high correlation of the letter “i” and straight vertical characters like “l” (Fig. 4).

### 4.2 LSTM Implementation (Extension of RNN Layers)

Compared to typical NNs, RNNs have loops in them which allow information to persist with future iterations. However, they struggle with learning from long term dependencies, especially when gradient descent is used as the loss optimization function [3]. LSTMs [9] are an extension of RNNs that

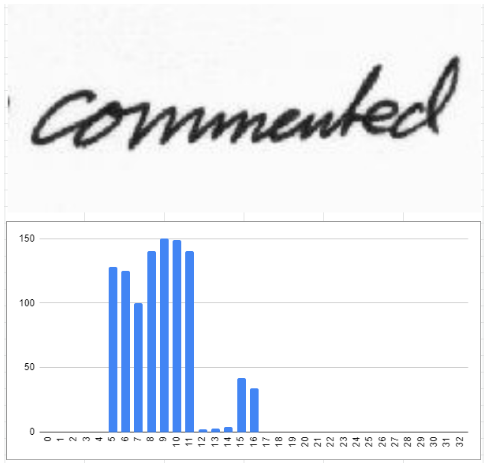

Figure 3: After the CNN layer, the graph of one of the 256 features showed a correlation (y axis value) with double letters in the word "commented".

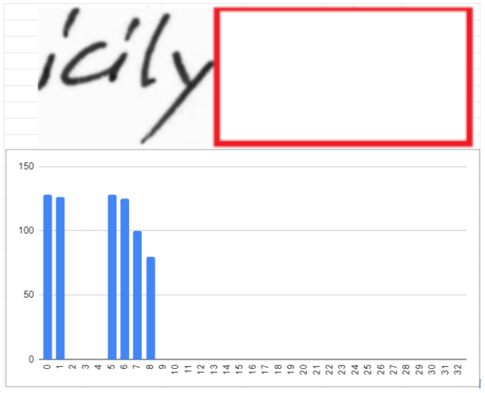

Figure 4: Another feature showed a high correlation with the letter “i” and straight vertical characters like “l”.

use gates to selectively remember or forget information from previous events in the sequence, allowing the network to focus on the most relevant information and prevent previous events from being overwritten by later ones. This allows LSTM networks to make more accurate predictions based on long-term dependencies in the data, and has made them a popular choice for many natural language processing and time series forecasting tasks, including our handwriting transcription task. LSTM networks have multiple layers of neurons, called gates, which process the input data. Each cell has three types of gates that control the flow of information: input gates, output gates, and forget gates. The input gates control which information from the current input and the previous cell state is passed on to the current cell state, and allow the network to selectively remember or forget information from previous events in the sequence. The output gates control which information from the current cell state is passed on to the next cell in the sequence, and allow the network to focus on the most relevant information and prevent previous events from being overwritten by later ones. The forget gates control which information from the previous cell state is forgotten or kept, and allow the network to forget irrelevant or outdated information and avoid "memorizing" the entire sequence. The output of the LSTM network is determined by the final cell state, which contains the most relevant information from the entire sequence. This output can be used to make predictions, classify the input data, or generate new sequences. In our case, a bidirectional RNN with LSTM was used, in which the input sequence was traversed from front to back, and back to front. Each traversal outputted a matrix of size 32x256 (time steps x features), and was concatenated together along the feature axis into a matrix of size 32x512.

## 4.3 CTC Loss Function and Decoding

The next step to outputting the predicted text was applying a loss function to the RNN. The CTC loss function was used because it allowed for insertions, deletions, and substitutions of labels in the predicted sequence. At a high level, CTC is most useful for tasks where alignment between sequences is needed, but that alignment is difficult (for instance, the way syllables in an audio file match up with the timestamp for speech recognition [11]), and is well-suited for handwriting recognition. CTC works by calculating the loss between a continuous series and a target sequence. It first sums over the possibility of possible alignments from input to target. Then it calculates the difference between the aligned sequences to compute a loss value. For given input sequences $X = [x_1, x_2, ..., x_T]$, a sequence from images of text, CTC builds upon an RNN to map to $Y = [y_1, y_2, ..., y_U]$, the final transcribed text. The objective for a single input/output pair is to find the conditional probability, which is equal to computing the probability for a single alignment $a$ at each time step $t$, marginalized over the set of of valid alignments.

$$P(X|Y) = \sum A \in A_{x,y}{}_{t=1}^{T} p_t(a_t|X)$$

Hanun describes the alignment step in [8].

For our case, output matrices from the RNN step were mapped to a matrix of size 32 x 80. This means that for each of the 32 time steps, 80 possible characters were needed. 80 possible characters corresponded to 79 + 1 = 80, because there are 79 total characters in the IAM dataset, plus 1 character for the blank token, ε (using the CTC alignment methodology in [8]). List of characters:
“!”# &’()*+,-./0123456789:;?ABCDEFGHIJKLMNOPQRST UVWXYZabcdefghijklmnopqrstuvwxyz

After training the model, the most likely character based on the pixels in the image was decoded based on the conditional probabilities. This was achieved by solving

$Y^* = \arg\max_Y p(Y|X)$. Sometimes the predictions were fully aligned with the image, but usually they weren't. Since CTC is segmentation free, repeated characters and $\varepsilon$ can collapse. For instance, $\varepsilon$i$\varepsilon$ci$\varepsilon\varepsilon$lyy $\rightarrow$ icily. The accuracy for the baseline model after all of the above steps was 73.1%.

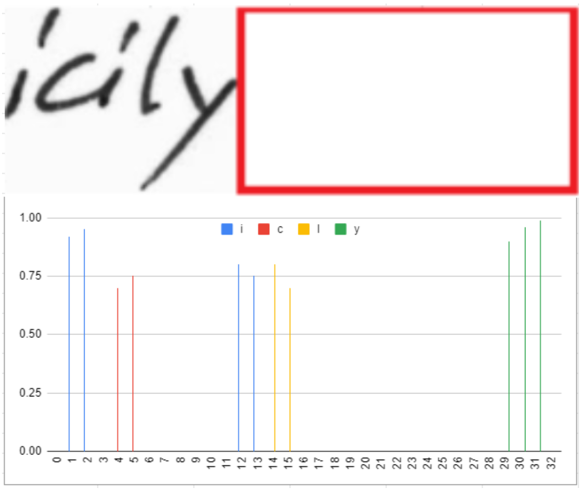

Figure 5: Output from CTC before collapsing characters. The CTC loss function was used to calculate conditional probabilities marginalized over the set of alignments. The graph shows the most likely prediction for a letter at each time step.

## 5 Experiments/Results/Discussion

After researching state-of-the-art methods, it was clear that CNNs and RNNs were the best models to experiment with. We decided to use a basic implementation of a model that utilized CNNs and RNNs and expanded upon it to improve it. Initially we experimented with the hyperparameters looking to improve our model. We decided to use a mini-batch implementation, as our training dataset is relatively large and we wanted our model to converge quickly. We left optimizing the mini-batch size to trial and error and found that a mini-batch size of 1000 was optimal, giving us a word accuracy of 73.638% and a character error rate of 10.81%. We also tried a mini-batch size of 250 and 500, but they both yielded a lower word accuracy and a higher character error rate.

Moving away from hyperparameters, we began looking at other ways to improve the accuracy of the model. An idea we had was to post-process all of our decoded words to verify that they were actual words in the english dictionary and not misspelled. Then, if any of our predictions were misspelled, we used a spell corrector algorithm to select the most probable correct spelling. After implementing the above algorithm, we managed to improve our word accuracy from 73.638% to 76.69%, a significant improvement. However, with the improvement of the word accuracy came an increase in the character error rate, which went from 10.81% to 11.523%. This may seem puzzling at first, as most would think that a significant increase in word accuracy would lead to a decrease in character error rate. However, this observation is reasonable; more words are being correctly identified, but when we misidentify a word, it's more incorrect at the character level. As an example, if our model was given an image of the word "ally" but predicted "aily", we would have a character error rate of 25%. But if we took our prediction of "aily" and ran it through a spell checker that believed that the word closest to "aily" is "daily", then our character error rate would increase to 40%. With our post-processing, we were more likely to predict the correct word, but when our prediction was incorrect we were farther from the ground truth. Additionally, we needed to consider the fact that the input image itself may have contained misspelled words, which was something we may or may not have wanted to reflect in our predictions. Because we were trying to maximize our word accuracy, we decided that using post-processing on the model's predictions provided a net gain.

After getting familiar with the dataset, we realized that a lot of words in our dataset were written in cursive instead of print and that it may be affecting our model and word accuracy. To guard against this, we implemented an algorithm that would deslant our images to make words written in cursive appear as if they were written in print. To do so, we used an algorithm designed by Vinciarelli and Luettin that was based on the hypothesis that a word was deslanted when the number of columns containing a continuous stroke was at its maximum [19]. The deslant algorithm took an image and applied shears with different angles and selected the image that returned the highest "score". The score was calculated by going through each column and calculating the number of foreground pixels per column, then summing over the square of the number of foreground pixels per column only if the column is continuous. The algorithm and its effects are pictured below:

**Algorithm 1** Deslant Algorithm

```
Require: Image $I$, list of shear angles $A$
  scores = {}
  for $\alpha \in A$ do
      $I_\alpha = shear(I, \alpha)$
      for $c \in columns(I_\alpha)$ do
          num_fg = number of foreground pixels in col c of $I_\alpha$
          $\Delta y$ = dist between first and last fg pixel in col c of $I_\alpha$
          if $\frac{\text{num\_fg}}{\Delta y} == 1$ then
              scores($\alpha$)+ = num_fg$^2$
          end if
      end for
  end for
  best_shear = $\arg\max_\alpha$ scores($\alpha$)
  return $shear(I, best\_shear)$
```

After deslanting the images, our word accuracy improved

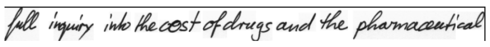

Figure 6: Original image before applying the deslant algorithm.

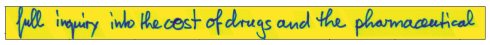

Figure 7: Image after applying the deslant algorithm.

from 73.638% to 74.384% and decreased our character error rate from 10.81% to 10.45%. Although we saw improvements in our accuracies, they were not as significant as we thought they'd be. However, combining the deslant algorithm with the spell correction algorithm allowed us to achieve a word accuracy of 77.835% and a character error rate of 10.589%.

We also experimented with doubling the number of convolutional layers in the CNN architecture. The baseline architecture contained 5 layers, each of which consisted of a 2D convolution, a ReLU activation, and a max-pooling operation; we changed this so that each layer would be two sets of 2D convolution and ReLU, followed by one max-pooling operation. This tweak granted a moderate performance improvement; we saw an increase of about 2% in word accuracy, bringing us up to 79.531%.

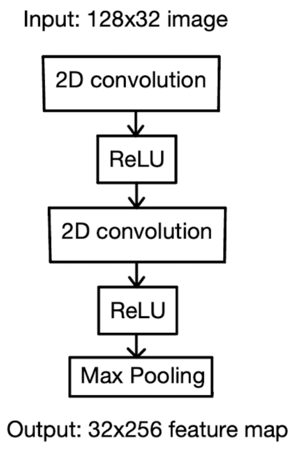

Figure 8: Improved CNN architecture

The final significant area of improvement was examining the CTC decoder. Our baseline model used a naive best-path search in the decoder, which used just the output of the neural network and formed an output by choosing the most likely character at each position. This approach was certainly not ideal; it did not incorporate a language model, and thus was prone to predicting nonsensical words that do not actually exist. To this end, we found and incorporated a word beam search algorithm, as discussed by Scheidl [18] in the decoder. This algorithm had two important advantages: it allowed for non-letter symbols and characters, and it constrained predicted words to a set dictionary, leading to a higher word accuracy. Using the word beam search decoder along with all other improvements, we achieved a CER of 8.178% (more than 2% less compared to the baseline) and a word accuracy of 85.194%—roughly a 12% improvement over the baseline model.

## 6 Conclusion and Future Work

The result of our experiments showed that our path of using CNNs, RNNs, and a CTC loss function was able to achieve a high accuracy in offline handwriting recognition. We achieved a word accuracy level of more than 85%, although only for the IAM dataset.

Naturally, the next step for this research project would be to apply it to another dataset. Rather than lines of text from English writers, we could train our model on another language. Unlike alphabet-based writing, which usually involves around 100 symbols, the set of Hànzì characters developed by the Institute of Automation of Chinese Academy of Sciences, comprises a total of 7,356 entries [13]. We anticipate that the accuracy would be much lower, given the jumbled nature and top-down structure of Chinese. On a fundamental level, we could perform cross-validations between the databases to better understand if they can be generalized. The model also could be tested at line level, which would allow us to analyze the behavior of the segmentation strategy of IAM. We could also experiment with data augmentation methods to improve how our model generalizes. For example, Ciresan et al. suggests elastic distortion as a technique to apply geometric and perspective transformations [5].

## 7 Contributions

All team members were heavily involved in brainstorming project implementations, as well as research into model architectures and current implementations. Work was completed synchronously and largely occurred as a combined effort rather than in partitions.

## References

[1] Savita Ahlawat, Amit Kumar Choudhary, Anand Nayyar, Saurabh Singh, and Byungun Yoon. Improved handwritten digit recognition using convolutional neural networks (cnn). *Sensors (Basel, Switzerland)*, 20, 2020.

[2] Ahmed Alsaffar, Suryanti Awang, Wafaa AL-Saiagh, Ahmed Salih Al-Khaleefa, and Saad Abed. A sequential handwriting recognition model based on a dynamically configurable crnn. *Sensors*, 21:7306, 11 2021.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[4] R.M. Bozinovic and S.N. Srihari. Off-line cursive script word recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(1):68–83, 1989.

[5] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012.

[6] Arthur Flor de Sousa Neto, Byron Leite Dantas Bezerra, Alejandro Hector Toselli, and Estanislau Baptista Lima. A robust handwritten recognition system for learning on different data restriction scenarios. *Pattern Recognition Letters*, 159:232–238, 2022.

[7] Khaoula Elagouni, Christophe Garcia, and Pascale Sébillot. A comprehensive neural-based approach for text recognition in videos using natural language processing. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, New York, NY, USA, 2011. Association for Computing Machinery.

[8] Awni Hannun. Sequence modeling with ctc. *Distill*, 2017. https://distill.pub/2017/ctc.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[11] Yichong Leng, Xu Tan, Linchen Zhu, Jin Xu, Renqian Luo, Linquan Liu, Tao Qin, Xiang-Yang Li, Ed Lin, and Tie-Yan Liu. Fastcorrect: Fast error correction with edit alignment for automatic speech recognition, 2021.

[12] Hang Li. Language models: Past, present, and future. *Commun. ACM*, 65(7):56–63, jun 2022.

[13] Cheng-Lin Liu, Fei Yin, Qiu-Feng Wang, and Da-Han Wang. Icdar 2011 chinese handwriting recognition competition. In *Proceedings of the 2011 International Conference on Document Analysis and Recognition*, ICDAR '11, page 1464–1469, USA, 2011. IEEE Computer Society.

[14] Urs-Viktor Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5:39–46, 2002.

[15] Anisha Priya, Surbhi Mishra, Saloni Raj, Sudarshan Mandal, and Sujoy Datta. Online and offline character recognition: A survey. In *2016 International Conference on Communication and Signal Processing (ICCSP)*, pages 0967–0970, 2016.

[16] David E Rumelhart, James L McClelland, PDP Research Group, et al. *Parallel distributed processing*, volume 1. IEEE New York, 1988.

[17] Harald Scheidl. Build a handwritten text recognition system using tensorflow, May 2021.

[18] Harald Scheidl. Word beam search: A ctc decoding algorithm, Dec 2021.

[19] Alessandro Vinciarelli. A new normalization technique for cursive handwritten words.

[20] Jerod J. Weinman, Erik G. Learned-Miller, and Allen R. Hanson. Scene text recognition using similarity and a lexicon with sparse belief propagation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:1733–1746, 2009.