SAPIENZA
UNIVERSITÀ DI ROMA

Teaching machines to read handwritten words

Faculty of Information Engineering, Informatics, and Statistics
Department of Informatics
Bachelor's degree in Applied Computer Science and Artificial Intelligence

Attilio Vanacore, 2022820
Oriana Deliallisi, 1984082

A.A. 2022-2023

Abstract

Handwritten text recognition is a challenging problem that plays a crucial role in document digitization, text preservation, and automatic transcription.

The main challenges for this task were the changes in handwriting styles, size and alignment of the characters, and their respective mapping to the labels.

To tackle these issues, our proposed model combines the strengths of CNNs and LSTM networks. CNNs are employed to extract local image features, capturing important patterns and structural information in the handwriting. LSTM networks, on the other hand, model the sequential nature of the text and capture long-term dependencies, enabling a deeper understanding of the context and relationships between characters.

The CTC framework allows end-to-end training and decoding, eliminating the need for explicit order annotations. This enables our model to handle variable character alignments and accurately map them to their corresponding labels.

We achieved notable results through hyperparameter tuning: we tried different optimizers, such as AdamW, Adagrad and RMSProp, employed early stopping and weight decay as regularization methods to prevent overfitting, and experimented with changing batch sizes and numbers of epochs.

Furthermore, issues with the current implementation, alongside possible fixes and improvements, will be addressed.

Problem Definition and Model Architecture

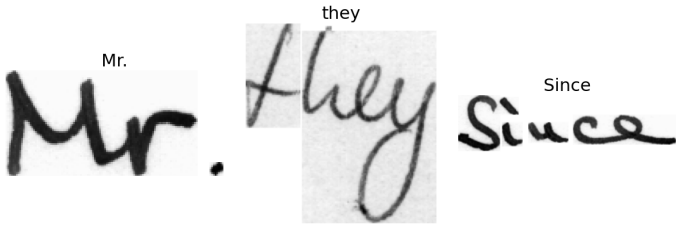
Problem Definition

Handwritten text recognition refers to the ability of computer applications to identify and interpret human-written characters. While OCR technology has made notable progress in recent years, there remain numerous obstacles that need to be addressed for further enhancements in accuracy and efficiency.

Let's explore some of the key challenges faced in this field:

- **the variability in handwriting styles**; cursive writing, capitalized letters, different stroke pressure, and also the size of the characters complicate the recognition task
- **the noise and distortions in the images**; properly scanning the documents, deslanting the image, and filtering the noise can potentially fix the issue but having a large dataset complying with these requirements is hard to find
- **the lack of labels for classification**; for digital texts, it's far easier to extract the labels to associate with the words, while for handwritten text it's much more difficult

For dataset selection, we carefully considered various options, such as the Bentham and Ricordi datasets, and ultimately chose the IAM dataset for its comprehensive coverage of diverse writing styles, including cursive, stylized, and normal writing. The dataset also stood out for its meticulous labeling of words and their respective labels, providing us with precise and reliable data for training and evaluation purposes. [1] [2]

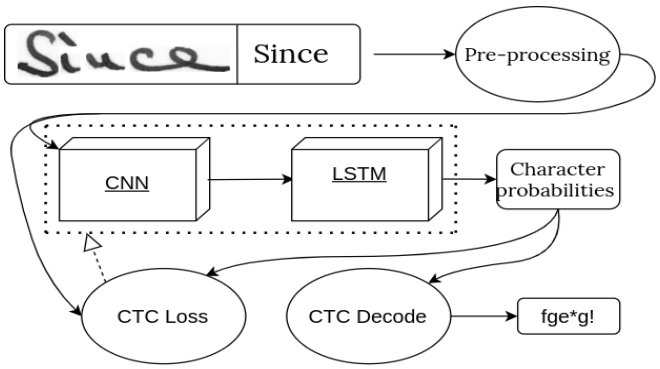


Some images from the IAM handwritten dataset

Regarding our approach, we initially tried reducing the problem to a character classification problem, However, this approach proved to be much more complex than we thought, mainly because the model would have to detect each character, recognize it and reassign it in the exact position on the page where it was located. Instead, we opted for a word recognition approach, where we extracted the word image along with its corresponding label and applied appropriate transformations to facilitate accurate analysis within the neural network layers.

Model Architecture

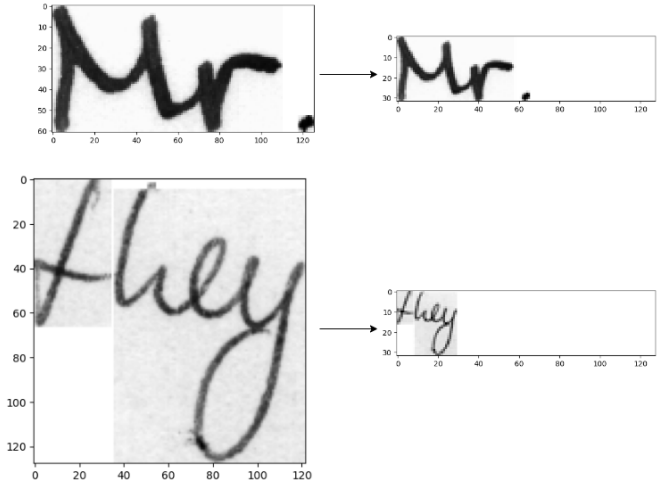
Our final implementation comprises a combination of Convolutional Neural Networks (CNNs), Long Short-Term Memory (LSTM) networks, and Connectionist Temporal Classification (CTC).



An overview of the model architecture

In the CTC-trained neural networks, correctly padding and aligning the images and labels is a crucial step for the model to learn to recognize the proper features.

In our **preprocessing** step, we first resize each image: the aspect ratio is kept constant but the size is changed accordingly to fit in a white image of size (32x128). If the image is larger than the output image, it's shrunk and padded, else it is enlarged and padded to the right (aligning the image to the left). Bicubic interpolation was used for the resizing since it is known for producing smoother results compared to other interpolation methods.



A visualization of the transformations applied to input images before going into the model

Then for the labels, we encoded each character to its corresponding index in the alphabet, which consists of a string containing all the possible digits and punctuation present in the dataset. We padded the resulting labels with blank characters and aligned them to the left, to preserve consistency during processing. The maximum label length is set to 32, which is the number of timesteps outputted by the model.

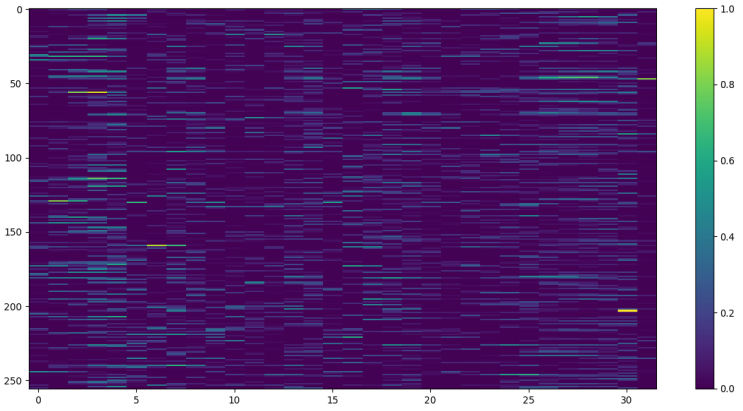
Label	Encoded	Padded
Mr.	[39,70,12]	[39,70,12,0,0,0,0,0,...]
Since	[45,61,66,55,57]	[45,61,66,55,57,0,0,...]
they	[72,60,57,77]	[72,60,57,77,0,0,0,0...]

The **model** consists of a convolutional layer (CNN), a long short-term memory layer (LSTM), and a projection layer, whose output is fed to a Connectionist Temporal Classification decoder (CTC).

The CNN is a sequential model that has 5 stacked blocks, each block is composed of a 2D convolution, batch normalization, Leaky RELU activation function, and 2D max pooling.

The CNN takes as an input grayscale (1 channel) images of size (32x128) and, through the convolutional and max pooling layers, it progressively reduces the spatial dimensions while increasing the number of feature maps, while the batch normalization layers, placed after a convolution, improve model stability and generalization, and the leaky RELU activation functions introduce non-linearity to the model.

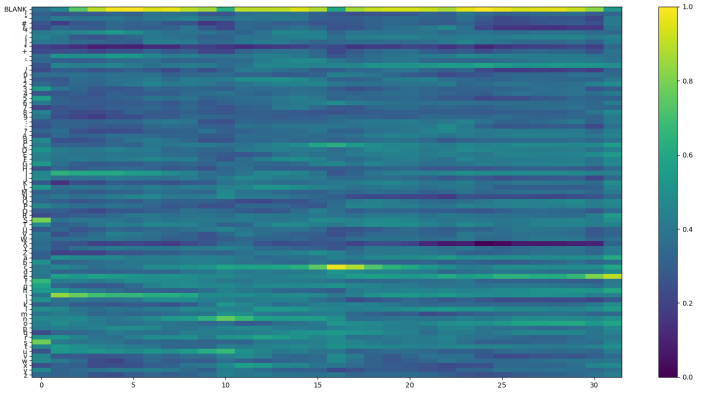
The output of the CNN is a feature map of (256x1x32) representing features through each timestep. The height dimension is dropped from the feature map, which is then fed into the bidirectional LSTM layer.



A heat map of the feature map computed by the CNN layer of the model

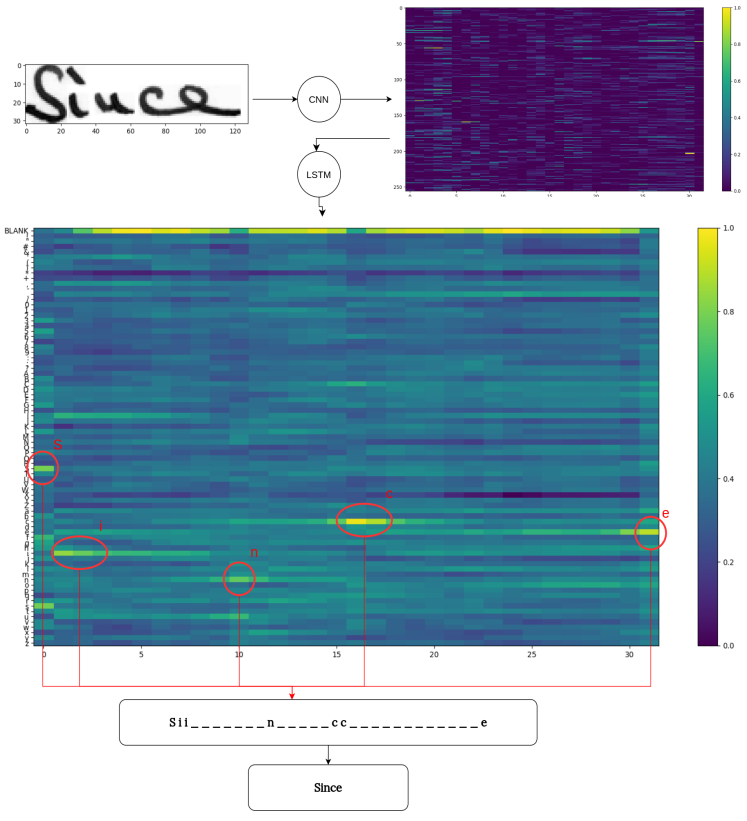
The LSTM propagates relevant information and allows us to extract only the important features of the map. Its output is downsized from (512x32) to (alphabet_length x 32), to map the features to the

characters in the alphabet.



Character probabilities for every timestep

During the **training** process, the CTC loss function is used to compute the loss and to backpropagate the gradient, while, for the actual text prediction, a simple Best Path decoding algorithm is implemented: it decodes the feature map by choosing the most probable character at each time step.



Processing pipeline of the model. Notice how sometimes doubles are detected before being deleted in the CTC decoding step

Evaluation

Methodology

The model performance was evaluated through a systematic hyperparameter tuning process. Several key hyperparameters were modified, including the optimizer, learning rate, weight decay, and batch sizes.

The optimizers we used were:

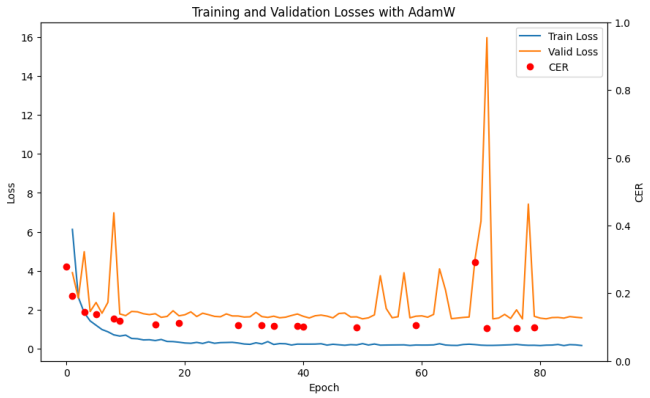
- AdamW with Weight Decay, which gave by far the best performance, taking into consideration validation, training loss, and CER
- Adagrad, which returned a regular training loss and validation loss, but CER rate higher than AdamW
- RMSProp, which gave a steady training loss but a fluctuating validation loss
- RMSProp with Weight Decay, which gave by far the worst results, with validation loss highly irregular

The chosen learning rate for most of the experiments was 0.001, as it consistently delivered the best performance across various tests. This learning rate demonstrated favorable effects on the model's convergence and generalization.
Batch sizes are between 20 and 32, we observed that with higher batch sizes the model would rarely converge and it would terminate by early stopping with undesirable metrics.

In addition to hyperparameter tuning, we used metrics such as CTC loss and CER (Character Error Rate), which measures the dissimilarity between predicted and ground truth labels.

Results

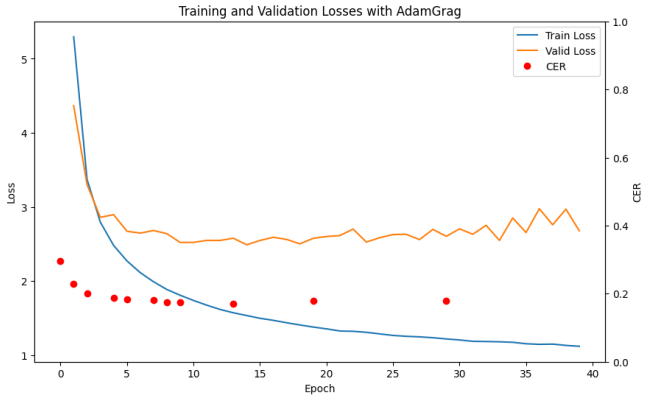
Best results were found using the AdamW optimizer with weight decay. Early stopping is used as a regularization method to prevent overfitting.



The training stopped after 90 epochs by early stopping. CER is sampled every 10 epochs and whenever a new minimum validation loss is observed.

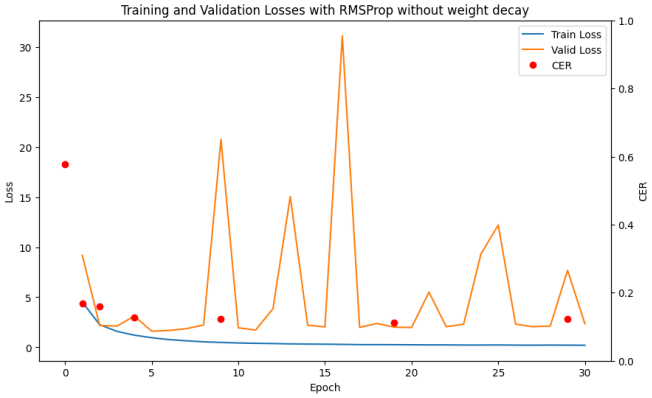
Even though it is our best-performing model, large spikes in the validation loss were observed during training, we were unable to fix this issue despite the attempts such as tuning the learning rate, and the batch size, and enabling the drop_last flag on the data loader to ensure consistent batch size.

The following graphs show the results obtained by using the previously mentioned optimizers.



Training with Adagrad

For the Adagrad optimizer test we used a slightly bigger batch size (from 20 to 32); after a few iterations, it seemed to perform better than AdamW. The training was overall stable and it converged relatively faster compared to AdamW, training loss is similar to AdamW but validation never went as slow as AdamW's. Similarly, sampled CERs were constant throughout the training but were often double the ones obtained with AdamW.



Training with RMSProp with no weight decay

In this training, we could observe how fast the training loss would converge, unlike the validation loss, which was really unstable. The training stopped after just 30 epochs by early stopping, as the validation loss never decreased. Sampled CERs were still acceptable, and better than the ones obtained with Adagrad.

Other attempts were done with different hyperparameters but the results were always highly unstable.

After our many trials, we failed to achieve a validation loss lower than ~1.5, and CER never went under 0.9 (9% of character error rate, obtained with AdamW, after about 60 epochs).
We then concluded that the model might be at capacity and needed to be up-scaled in order to achieve better performance, alongside a few other enhancements that will be discussed in the following section.

Enhancements for Overcoming Method Limitations

To overcome limitations in our approach and improve the performance of our model, we can incorporate the following enhancements:

- A better CTC Decoding algorithm: the best path decoding approach has noticeable flaws, such as deleting doubles and only considering the most probable character per timestep. Instead, Beam Search Decoding could be implemented: it considers multiple possible paths during decoding, capturing more diverse and accurate outputs and leading to better transcription results, or even better, a Lexicon Search Decoder, which constrains to decoded words to a fixed dictionary. [3]

- Generalization to Line-by-Line Text Recognition: Instead of focusing on recognizing individual words, we can upscale our model to perform line-by-line text recognition. This involves training the model to handle complete lines of text, including spaces.

- Data Augmentation: To improve the generalization ability of our model, we can apply data augmentation techniques. By introducing variations in the training data, such as rotation, scaling, and cropping, we can expose the model to a broader range of handwriting styles and variations. This augmentation helps the model generalize better to unseen data during inference.

- Additional Regularization Methods: In addition to Early Stopping, we can add further regularization techniques to prevent overfitting. These methods can include dropout or batch normalization. It is reported that L2 regularizations do not work well with adaptive gradient algorithms like AdamW, hence why we opted for the use of weight decay. [4][5]

References

1. U. Marti and H. Bunke. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition. Int. Journal on Document Analysis and Recognition, Volume 5, pages 39 - 46, 2002.
2. Handwritten Text Recognition in Historical Documents - Thesis by Harald Scheidl
3. Word Beam Search: A Connectionist Temporal Classification Decoding Algorithm
4. Weight Decay == L2 Regularization?
5. Supervised Sequence Labelling with Recurrent Neural Networks Alex Graves