ECEN 5060 (Deep Learning) Final Project Individual Report

Sungjoo Chung

May 6

1 Introduction

1.1 Overview

Handwritten mathematical expressions (HMEs) are indispensable in various domains, such as engineering, education, and science. The development of pen-based or touch-based devices has provided a user-friendly interface to input handwritten mathematical expressions, which is more natural and convenient than editors such as Microsoft Equation Editor or LaTeX. Thus, such devices have been widely adopted in various environments, such as offices and educational institutions, especially during the outbreak of COVID-19. This phenomenon has necessitated the development of an accurate model for recognizing HMEs.

The handwritten mathematical expression recognition (HMER) problem differs from the traditional optical character recognition (OCR) problem due to three main reasons:

- The two-dimensional (2D) nature of HMEs adds additional complexity compared with the one-dimensional OCR problem. This can be demonstrated by the following mean squared error (MSE) function: $\frac{1}{N}\sum_{i=1}^{N}(p_i - t_i)^2$.
- The existence of more than 1,500 unique symbols [BFS17], which are often difficult to distinguish from each other (for example, "O", "o", "0", "$\mathbf{O}$", and "$\mathcal{O}$"), especially when considering the variation in handwriting styles. Combined with its structural nature, this leads to an infinite number of combinations of symbols and spatial relationships in MEs.
- The existence of long-term dependencies and correlations among symbols in MEs. For example, "(" and ")" are used to contain subexpressions, and if those subexpressions contain other long subexpressions, these dependencies are challenging to learn.

On a more personal note, as engineering students, we read, write, and share HMEs on a daily basis. Initially suggested by Haya, we thought that applying

the concepts we have learned in the Deep Learning course to tackle the HMER problem would not only be interesting but also very relevant.

1.2 Outline of Shared Work

At the outset of this project, Haya and I divided the work to allow us to work in parallel, accelerating the process. Our collaboration focused on conducting a literature review of the popular methods used to tackle the HMER problem. After developing a solid understanding of the problem, Haya led the project by conducting data preprocessing and building a convolutional recurrent neural network (CRNN)-based baseline model.

While our primary tasks were separated, we collaborated closely throughout the project. Every stage during this final project, we discussed our progress and results, often seeking guidance or advice on how to move forward.

The rest of this report will provide a detailed summary of the individual work I've contributed to this project.

2 Improving the Baseline Model

With the CRNN baseline model provided by Haya, she expressed that the result, in both the word error rate (WER) and the prediction, was not satisfactory. Even after hyperparameter optimization and fine-tuning, the baseline model's WER did not exceed 40%. My efforts on trying to fine-tune the model yielded similar results. This signified that a more fundamental approach had to be taken to exceed the 40% WER threshold for the model to produce meaningful predictions.

It was at this stage that I decided to follow the CRNN architecture proposed in [SBY16]. In the baseline model proposed by Haya, the output of the CNN's final layer is passed through an adaptive average 2D pooling function, which flattens the feature map into a vector. This vector is then used as input to the RNN, which aims to capture the sequential patterns present in mathematical expressions (MEs). However, MEs often contain vertically stacked elements such as fractions or superscripts, and flattening these spatial features can lead to the loss of important structural information.

To address this problem, [SBY16] proposed to extract a sequence of feature vectors from the feature maps produced by the CNN at the final layer, which is the input for the RNN. This means that the i-th feature vector is produced by concatenating the i-th columns of all the feature maps at the final layer. A detailed depiction of the CRNN framework depicted in [SBY16] and adopted in our work can be seen in Fig. 1.

Once the feature sequences are extracted from the CNN, a bi-directional LSTM is implemented to capture the contextual dependencies within the sequence. This architecture is especially effective for the HMER task, as mathematical expressions often contain symbols such as "(" and ")" that exhibit long-term dependencies in both forward and backward directions. Combined

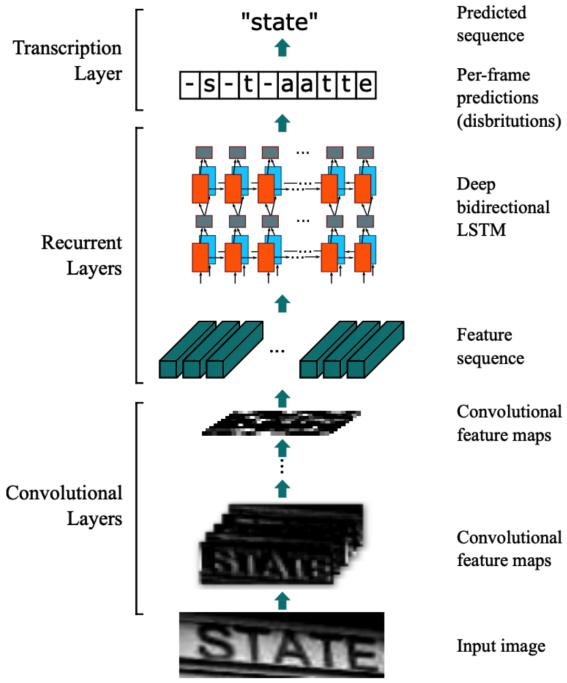


Figure 1: Architecture of CRNN [SBY16]

with the spatial features extracted from the CNN, we expected this architecture to yield significantly improved results than the baseline model. The final transcription layer decodes the output of the LSTM based on the tokens of mathematical symbols contained in the training and validation datasets.

2.1 Initial Results

Apart from adjusting the architecture, an additional change was made regarding the image size. While [SBY16] uses variable widths and only fixes the height of the images to 32, I chose to fix the image sizes to 32×300, in order to accelerate the training process as it enabled faster batch processing by avoiding dynamic padding. The same augmentation made in the baseline model was applied to improve the model's generalizability.

Preliminary results showed very promising results, yielding WER of 19.98% and 21.66% on the training and validation sets, respectively. However, after obtaining the predictions of the model, it was evident that the model's improvement was not due to it making more accurate predictions, but due to it dominantly predicting the most frequently occurring token. As can be seen in Fig. 2, the predictions of the model are dominated by structural tokens such as

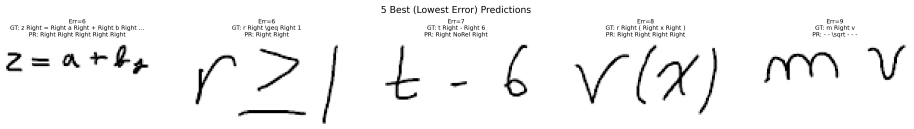


Figure 2: 5 Best Predictions of the Proposed Model in its First Iteration

"Right". Structural tokens were part of the labels for the HME images in order to provide contextual dependencies to the model.

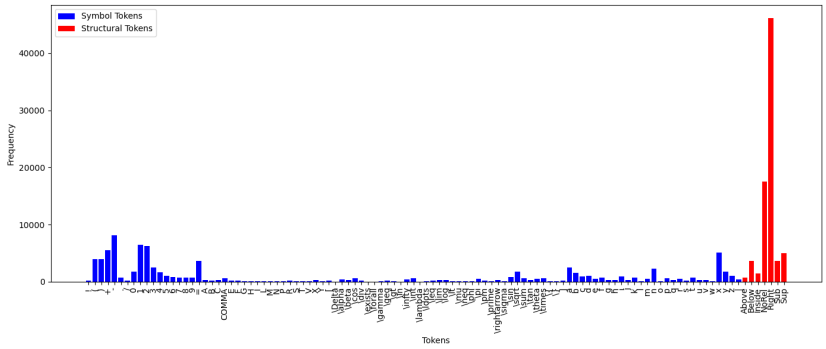
However, it seems like rather than using this contextual dependency to make accurate predictions, the model opted to dominantly output predictions of these structural tokens, thereby inflating the accuracy.

A further investigation into the frequency of structural tokens and symbol tokens revealed that the frequency of structural tokens was similar to the frequency of symbol tokens. However, there are only 7 structural tokens (Right, Above, Below, Inside, Sub, Sup, NoRel), while there are 101 symbol tokens, meaning that the individual frequencies of structural tokens were significantly higher than those of symbol tokens. Fig. 3a illustrates that the structural token "Right" is the most frequently occurring token in the label of the training dataset, appearing a total of $46,147$ times, which constitutes roughly 30% of all token frequency. From Fig. 3b, we can see that even the second most frequent structural token (NoRel) appears roughly twice as many times as the most frequent symbol token ($-$). Thus, the model's behavior can be attributed to the huge imbalance in token frequencies.

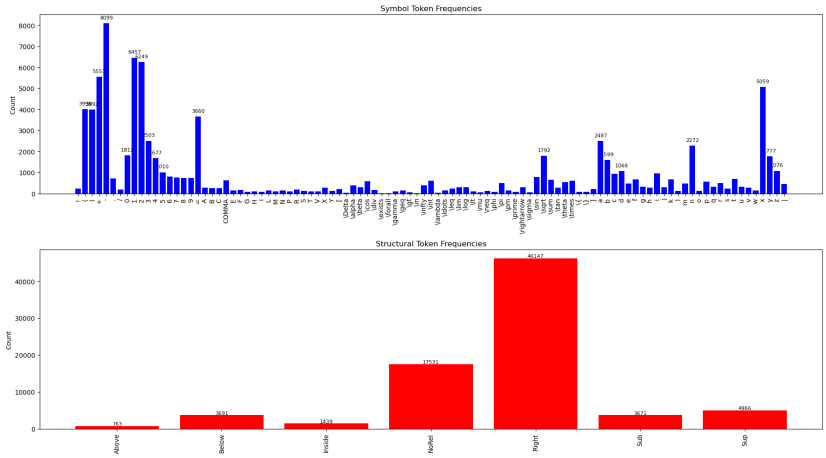
Modifications were made to address this problem, such as penalizing the predictions of structural tokens by incorporating a regularization term in the loss function or implementing a higher class weight for mathematical symbol tokens. However, these efforts proved ineffective. Then, the structural tokens were removed altogether, but this approach yielded a significantly higher WER of 43.29% and 37.14% during training the validation, respectively.

2.2 Changing the Decoding Strategy

After observing that the model's predictions were dominated by structural tokens, it became evident that this behavior stemmed not only from the imbalance in token frequencies but also from the decoding strategy. All previous iterations of the model are trained using the Connectionist Temporal Classification (CTC) loss, which is alignment-free as it enables the model to predict sequences of tokens with intermediate blank labels and repeated symbols. While CTC is powerful for handling variable-length outputs without frame-level alignment, it does not constrain the model to output the most semantically coherent sequence. It only optimizes the probability of the correct label sequence over all valid alignments.



(a) Frequency of Tokens



(b) Comparison of Symbol and Structural Tokens

Figure 3: Frequency analysis of tokens: (a) token-level frequency, (b) category-wise comparison.

As a result, decoding becomes a critical post-processing step for translating the model's raw outputs into a final sequence of tokens. All the previous iterations of the model (including Haya's baseline model) rely on greedy decoding: the most likely token at each timestep is selected independently. However, while simple and efficient, this strategy does not incorporate the contextual dependencies inherent within MEs, thus leading to erroneous predictions, such as the phenomenon shown in Section 2.1, from the previous iterations.

To address this, I replaced greedy decoding with beam search decoding, which considers multiple candidate sequences at each timestep and maintains the top-k most likely sequences (with beam width set to 10 in our experiments). This allows the decoder to explore a broader solution space, shifting its focus from predicting the most likely token at each time step to predicting the most

likely sequences of tokens. This change was made on top of removing the structural tokens in the token vocabulary list.

The improvement yielded from this change was significant, especially in terms of the predictions that the model was generating. We can observe from Fig. 4 that the CTC loss and WER of the model are higher than the previous iterations. While this would generally mean that the model's performance is worse, a closer look at the predictions generated by the model tells a different story.

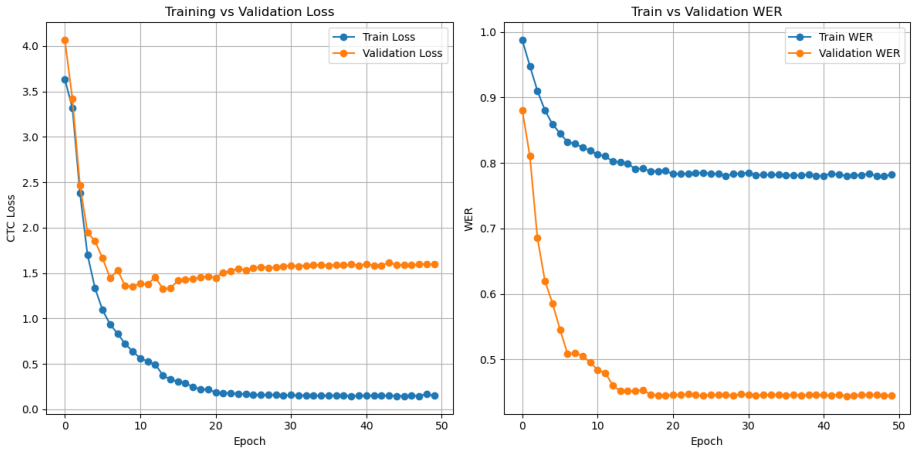


Figure 4: Loss and WER Progression for Training and Validation Dataset

Fig. 5 illustrates the model's predictions across a range of error levels. Compared to the outputs shown in Fig. 2, the improvement is evident. The model achieves high accuracy on shorter mathematical expressions, although its performance tends to degrade as the expressions grow longer and more structurally complex.

Some of the model's errors are, in fact, understandable. For instance, in the 7^{th} image (counting from left to right), the model misclassifies the digit "9" as a "q", incorrectly predicting the label $\sqrt{98}$ as $\sqrt{q8}$. In the 18^{th} image, it misinterprets the symbol "C" as a left parenthesis, resulting in the prediction $(\times($ instead of the correct $C \times C$. While such expressions are clearly invalid to a human reader, the visual similarity between certain handwritten symbols makes these errors plausible from the model's perspective.

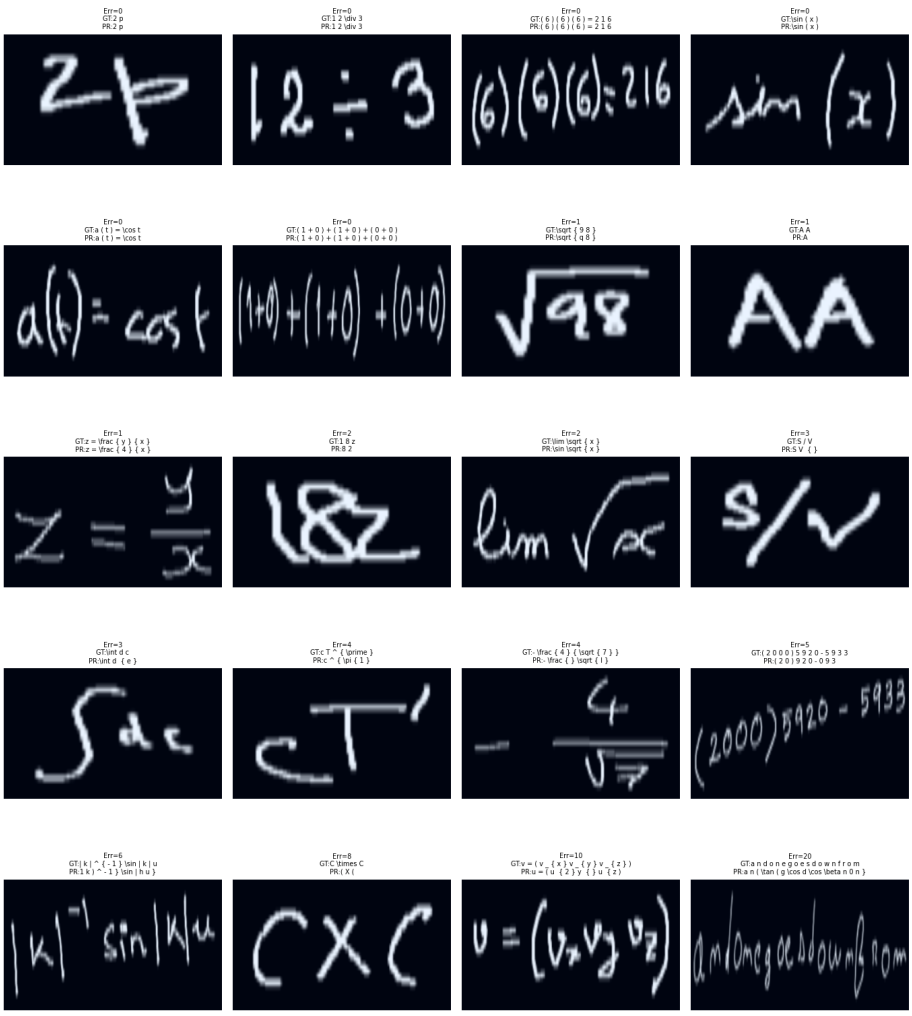


Figure 5: Prediction Samples across a Range of Error Values

3 Post-Processing

During the post-processing stage, both Haya and I noticed that the predictions made by the model contained the token | between most valid tokens (note that the predictions listed in Fig. 5 omits these tokens for better illustration of the improvement of the model, while the CTC loss and WER in Fig. 4 are calculated without omitting these tokens) as illustrated in Table 1. This phenomenon happens because the CTC loss function relies on blank tokens to enable alignment flexibility in its raw output. While greedy decoding collapses blank tokens and consecutive duplicates automatically, beam search decoding does not inherently

remove these blank tokens as it outputs the raw token paths, therefore inflating the WER and consequently underestimating the accuracy of the model.

Ground Truth	Prediction	Error
kN	k | N	1
12	1 | 2	1
Pa	p | { | a | }	6
19	7 | 9	2
26	2 | 6	1
1m	1 | m	1
Nm	N	1
Hz	H | { | z | }	5
kg	k	1
mv	m | v	1

Table 1: Examples of predictions containing inserted blank tokens (|) and the corresponding Levenshtein error counts.

A simple post-processing step was conducted by removing all | tokens that were inserted as a blank token, and the model's performance was re-evaluated based on the post-processed outputs. Furthermore, the predictions were categorized into two groups: short sequences ($\leq$ 14 tokens) and long sequences ($\geq$ 14 tokens) to emphasize the limitation of the CTC loss function on long sequences. Table 2 summarizes the true performance of the model. It can be seen

Expression Length	Original WER	Cleaned WER
Short ($\leq$ 14 tokens)	0.7639	0.2385
Long ($>$ 14 tokens)	0.5344	0.3936

Table 2: Comparison of original and cleaned Word Error Rate (WER) for short and long expressions.

that the WER improves by roughly 3 times for the short sequences after post-processing, suggesting that the model's performance has significantly improved after implementing all the aforementioned modifications.

4 Final Model

The final architecture of the proposed CRNN model and its corresponding training setup are summarized in Table 3. These hyperparameters were obtained through iterative experimentation and empirical tuning. Multiple configurations were tested, and the selected settings provided the best trade-off between training stability, accuracy, and generalization to complex handwritten mathematical expressions.

Table 3: Final Model Architecture and Training Parameters

Component	**Hyperparameter**	**Value / Setting**
Input	Image Size	(3, 32, 300)
CNN Backbone	Architecture	VGG16
RNN	Type	2-layer Bidirectional LSTM
	Hidden Size	256
	Dropout (between layers)	0.5
Fully Connected Layer	Input / Output Size	512 (256 × 2) / 110 tokens
	Dropout	0.3
Loss	Loss Function	CTC
Optimizer	Type	Adam
	Learning Rate	5e-4
Scheduler	Type	ReduceLROnPlateau
	Factor and Patience	0.5 (halves LR on plateau), 2
Decoding	Beam Width	10
Training	Epochs	50

5 Summary and Conclusion

My individual contribution to my project focused on improving the baseline model through structural redesign, decoding strategy enhancements, and post-processing. The initial model suffered from overprediction of structural tokens, which were over-represented. Subsequent iterations of the model suffered from the limitations of the greedy decoding strategy. By adopting the feature sequence extraction technique proposed in [SBY16] and replacing the greedy decoding strategy with beam search, the model generated more coherent and semantically valid predictions.

During evaluation of the model's prediction, I noticed that the outputs from the raw beam search resulted in inflated WER due to the inclusion of CTC blank token, specifically the | token. A post-processing step that removed this token significantly improved the performance of the model, decreasing the WER from over 40% to approximately 23.85% for short expressions after post-processing. Not only did the WER improve, but more importantly, the model's predictions became more coherent and semantically meaningful: an aspect that WER alone often fails to capture.

Through this project, I learned the critical importance of decoding strategies and post-processing in CTC-based models, especially when evaluating structural tasks like HMER. Future improvements could include adding explicit alignment supervision, integrating attention mechanisms, or exploring encoder-decoder architectures with structural parsing capabilities to better model long-range dependencies and nested structures in mathematical expressions.

References

[SBY16] Baoguang Shi, Xiang Bai, and Cong Yao. "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.11 (2016), pp. 2298–2304.

[BFS17] Barbara Beeton, Asmus Freytag, and Murray Sargent. *Unicode support for mathematics*. The Unicode Consortium, Mountain View, CA, USA, Tech. Rep. 25. May 2017.