Deep Learning for Recognizing and Converting Handwritten Scientific Equations into LaTeX

Individual Report for Haya M. (Group 04)

1. Introduction

This report presents the work on Handwritten Mathematical Expression Recognition (HMER) using deep learning techniques. The initial intention was to explore Convolutional Recurrent Neural Networks (CRNN) and Transformer-based models for this task, particularly due to their effectiveness in structured recognition tasks such as OCR. However, due to time limitations and the complexity of the models, we shifted our focus to building a baseline CRNN based model and an improved version for comparative analysis. Both models were trained on the Competition on Recognition of Handwritten Mathematical Expressions (CROHME) dataset, a collection of handwritten mathematical expressions. In this project, my partner was Sungjoo Chung. The workload was divided equally at every stage of the project. However, in this report I will detail the parts that I either came up with completely (project idea, baseline model development – data pre-processing, creation, training and evaluation) or contributed to theoretically (post-processing steps). Github's project management features were used to keep track of completed and pending tasks for this project. The list of tasks can be found at: https://github.com/users/hayamonawwar/projects/2.

2. Problem Definition and Dataset

Since I am an engineer at the basic level, I tend to think of project problems that either closely relate to dilemmas that all engineers face and/or are closely related to my PhD research field. One major problem that I face when taking down handwritten notes is that it is a task and a half to convert them into LaTex format (a step which most researchers need to perform to type their research publications). Hence, the task at hand is the recognition of mathematical expressions from handwritten data, which involves recognizing symbols and understanding their spatial relationships in a two-dimensional layout. The dataset we used, the CROHME dataset, consists of more than 12,000 examples in image and symbolic format, with each image corresponding to a labeled mathematical expression [1]. Validation was done of the 2014 and 2019 versions of the dataset whereas testing was done with the 2016 version of the dataset only.

To process these images, we employed a custom preprocessing pipeline (code shown in preprocessing.py) that imports image files from the dataset. It then normalizes and renders

the ink strokes into a canvas suitable for feeding into a neural network, using the image rendering equation below:

$$scale = \frac{Image\ size - (2 * padding)}{\max\left(max_x - min_x,\ max_y - min_y\right)}$$

This equation scales the coordinates of each stroke to fit within the image dimensions, ensuring that all expressions are normalized for training. $\boldsymbol{Min_x}$ and $\boldsymbol{min_y}$ represents the minimum values of the x and y coordinates across all strokes, essentially identifying the leftmost and bottommost points in the raw data; whereas, $\boldsymbol{max_x}$ and $\boldsymbol{max_y}$ represent the maximum values of the x and y coordinates, identifying the rightmost and topmost points. The aim of scaling the coordinates was to make sure that they fit within the canvas while preserving the aspect ratio of the formula. Padding was also added to ensure the strokes did not touch the image edges.

The next step was tokenization of these labelled equations. The labels for the training data were tokenized, meaning that each symbol in a formula was assigned an index in the vocabulary. This is crucial for converting the text labels into a format that the neural network can understand. The output of this step was processed images and their corresponding labels which were saved in directories for training, validation, and testing. The script also generated a vocabulary file containing the index mappings for each symbol.

3. Model Development

After preprocessing, we moved on to the development of recognition models. Initially, there were plans to experiment with Transformer-based models (TrOCR), but due to the time constraints and complexity of the models, this idea was discarded in favor of a more tractable approach using CRNNs. The intermediate step here was the creation of a custom dataset class (CROHMEDataset) that was implemented to load images and their corresponding labels from the dataset. The labels were tokenized using the Vocab class, converting each symbol into an integer index. Inspiration for this part of the implementation came from [2]. Next, data augmentation was performed:

- First, the image is resized to a fixed size of 100x100, ensuring consistent dimensions across all images.
- A small random rotation is applied, allowing the model to become more robust to minor rotational variations in the input data.
- I then introduced slight variations in lighting conditions to randomly adjust the brightness and contrast of the image, enhancing the model's ability to generalize to different lighting scenarios.
- Next, the image is converted to a PyTorch tensor using *transforms.ToTensor(),* which scales the pixel values from the range [0, 255] to [0, 1].

- Finally, the image is normalized by subtracting the mean (0.5) and dividing by the standard deviation (0.5), helping to stabilize and speed up the training process by ensuring the data is centered around zero.

This combination of transformations not only prepares the images for model input but also introduces variability to improve generalization during training.

Baseline CRNN Model

The baseline model consisted of Convolutional Neural Networks (CNNs) for feature extraction followed by a Bi-directional LSTM (BiLSTM) only to capture the sequential dependencies of the mathematical symbols. In the subsequent iteration, I enhanced the CRNN model by:

1. Using a pretrained ResNet18 model for better feature extraction.
2. Freezing the early layers of the ResNet to reduce computation time.
3. Adding Batch Normalization and Dropout layers for regularization and preventing overfitting.
4. Enhancing the LSTM layers to improve temporal sequence handling.

This iterative refinement process aimed to improve the model's ability to accurately recognize complex expressions while ensuring faster convergence during training. The training of both models involved iterative improvements based on performance on the validation set.

5. Model Training and Evaluation

Training Process

The training pipeline was set up using PyTorch, with the following components:

- Dataset and Data Loader: The custom CROHMEDataset class was implemented to load images and their corresponding tokenized labels from the dataset (to convert each symbol into an integer index).
- Model Architecture: The model was built using a ResNet18 backbone (pretrained) for feature extraction, followed by a BiLSTM for sequence modeling. A Connectionist Temporal Classification (CTC) loss function was used to handle the misalignment between input images and output labels.

Iterative Training with Epochs

Training involved multiple epochs, with each epoch computing the CTC loss. CTC loss was found to be particularly suitable for this problem because in handwritten mathematical expressions, the positions of symbols in a sequence are highly variable, and we do not have

direct alignment between the input image and the output sequence of characters or symbols. CTC Loss works by predicting a probability distribution for each frame in the sequence and aligning these predictions with the target sequence of labels using the best possible alignment. Mathematical Expression Recognition is a sequence-to-sequence problem, where the input image is a sequence of pixels (processed by a CNN), and the output is a sequence of symbols (recognized by the LSTM). The sequence is highly variable due to different handwriting styles, sizes of characters, and the complexity of mathematical notation (e.g., exponents, roots, fractions). CTC loss facilitates this alignment in sequence prediction without requiring a one-to-one alignment between input and output [3].

The optimizer used was Adam with a learning rate of 1×10^{-4}, and learning rate scheduling was applied using *ReduceLROnPlateau* to adjust the learning rate based on the validation loss. During each epoch, the model's training and validation performances were evaluated using the Word Error Rate (WER) and Character Error Rate (CER), which were computed using the *Levenshtein* distance. The WER is a common metric used in tasks that involve transcribing text, such as in Optical Character Recognition (OCR). It measures the performance of a system by comparing the predicted sequence of words or symbols against the ground truth. WER is particularly relevant for recognizing mathematical expressions because these expressions are composed of multiple symbols that together form meaningful terms (words in OCR or formulae in this case). The WER is computed using the Levenshtein distance (edit distance) between the predicted and true word sequences using the equation below:

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

Where **S** is the number of substitutions, **D** is the number of deletions, **I** is the number of insertions, and **N** is the number of words in the ground truth [4].

While WER focuses on word-level errors, CER is more granular, calculating the character-level error rate. CER is particularly useful when evaluating the fine-grained accuracy of systems that deal with symbol recognition, such as recognizing individual symbols in mathematical formulas. CER is computed by comparing the predicted sequence of characters with the ground truth at the character level. This is crucial because mathematical expressions often contain symbols that must be identified accurately. Small differences in symbol recognition (e.g., "1" vs. "l" or "x" vs. "X") can result in significant changes to the meaning of the expression. CER is computed by simply obtaining the ratio between the number of character level errors and the total number of characters.

At the end of each epoch, the validation loss and error rates were tracked, and the model's performance was plotted.

6. Training Results

The model's performance was evaluated across several epochs, as shown in the plots below, which depict the training and validation CTC loss and WER (Word Error Rate) over the course of 20 epochs. Due to unsatisfactory results, it was increased to 35 epochs. The results from both the iterations can be seen in Fig. 1 and Fig. 2.

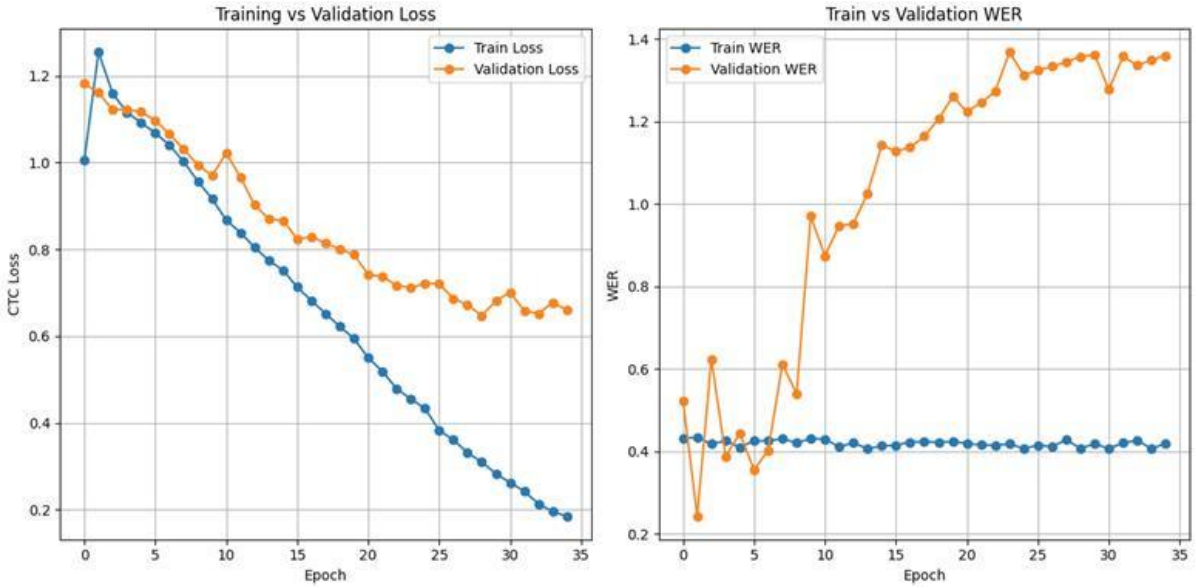

Fig. 1. Training and testing results - iteration 1

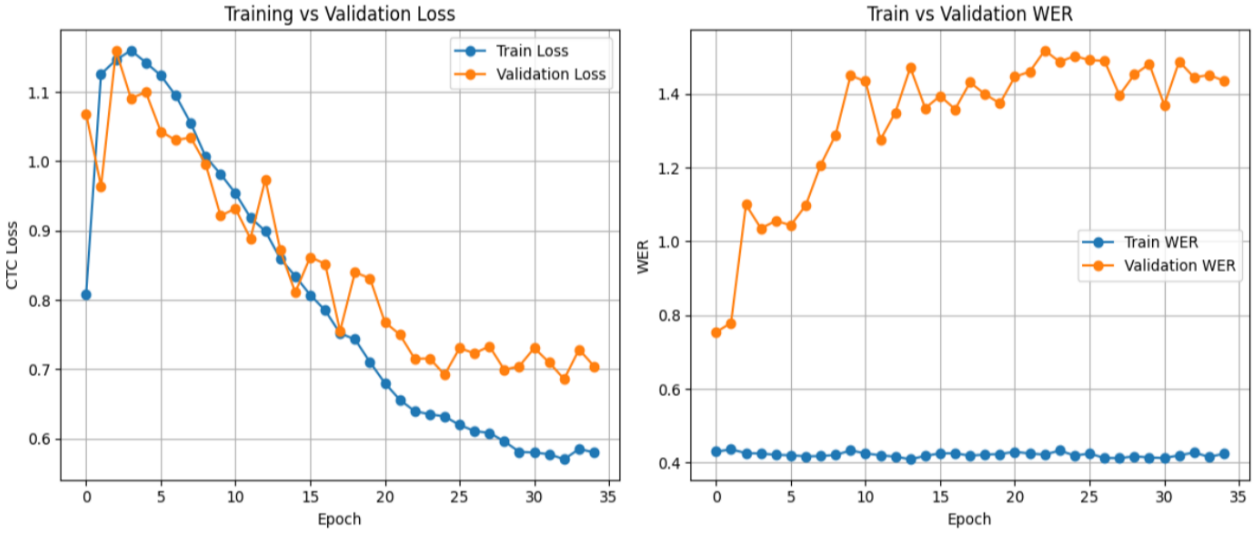

Fig. 2. Training and testing results - iteration 2

Training vs Validation Loss (CTC Loss)

The first plot illustrates the CTC loss during training and validation. Initially, the training loss decreased significantly, indicating that the model was learning to recognize patterns in the mathematical expressions. However, the validation loss began to diverge from the

training loss after a few epochs, suggesting that the model started to overfit to the training data, as evidenced by the increasing gap between the two curves. This is a common issue when the model performs well on the training set but struggles to generalize to unseen data.

Train vs Validation WER (Word Error Rate)

The second plot shows the WER during training and validation. Early in the training process, the training WER was consistently low, indicating that the model was accurately recognizing the mathematical expressions in the training set. However, the validation WER remained relatively high and fluctuated significantly, which suggests that the model was struggling with generalizing its learning to the validation data. This high WER in the validation set further supports the idea that the model is overfitting.

From both the trials, it can be seen that the training process showed a good initial learning phase with quick improvements in training performance, but the validation performance highlighted the need for better generalization. Unfortunately, my results did not improve more than 40% WER even after fine-tuning the model with additional strategies to reduce this gap, such as early stopping and learning rate decay.

Post-processing Steps

After my teammate Sungjoo modified the model architecture, we were able to arrive at a lower WER and CTC loss than my individual trials. Nonetheless, after closely observing the predictions of this model and comparing it with the ground truth, we found that often, there was only the difference of a ' | ' symbol between characters in several cases. This would lead to an increase error score assigned to the predictions – hence, decreasing our testing WER further more. This can be seen in Fig. 3.

GroundTruth	Prediction	Error
k N	k | N	1
1 2	1 | 2	1
P a	p | { | a | }	6
1 9	7 | 9	2
2 6	2 | 6	1
1 m	1 | m	1
N m	N	1
H z	H | { | z | }	5
k g	k	1
m v	m | v	1

Fig. 3. Increased WER due to redundant ' | ' symbols in our model predictions

Owing to this observation, Sungjoo took up the task of implementing a post-processing step in our pipeline so as to refine the final results and reduce the WER. The output of this step would be a highly accurate hand-written equation in LaTex format.

7. Summary

This project aimed to develop a Handwritten Mathematical Expression Recognition (HMER) system using a Convolutional Recurrent Neural Network (CRNN). The methodology involved preprocessing the CROHME 2019 dataset, where image files were rendered and normalized, and labels were tokenized using a custom vocabulary class. A CRNN architecture, consisting of pre-trained ResNet18 for feature extraction and Bi-directional LSTMs (BiLSTM) for sequence modeling, was employed, with CTC loss used to handle the alignment between input images and output symbol sequences. The model was trained and evaluated using CTsC loss, WER (Word Error Rate), and CER (Character Error Rate). The results showed strong performance on the training set, with a significant decrease in training CTC loss. However, validation CTC loss started to increase, and validation WER fluctuated, indicating overfitting and poor generalization. After refining the model more, we incorporated a post-processing step to improve the model's predictions and decrease the WER. The relevant code, reports, and presentation can be found at: https://github.com/hayamonawwar/Final-Project-04.git.

8. Conclusion

This project successfully explored a CRNN-based approach to handwritten mathematical expression recognition, tackling the complex problem of converting ink-based formulas into LaTeX. I led the development of the baseline model, designed the preprocessing pipeline, and implemented key training strategies. Together with my teammate, we refined the model architecture and integrated a post-processing step that significantly improved recognition accuracy. Despite challenges with generalization, our iterative efforts demonstrated significant progress, with reduced WER and enhanced symbol prediction. This work not only validated the potential of deep learning in HMER but also laid the foundation for future enhancements with Transformer-based models. Furthermore, this light-weight model can also be incorporated successfully into online or offline applications to enable equation recognition and output it in LaTex format on the go.

References

[1] Mahdavi, M., et al. 'ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection.' ICDAR, 2019.

[2] "CROHME_CTC_pytorch," *Kaggle.com*, 10-Jul-2024. [Online]. Available: https://www.kaggle.com/code/ntcuong2103/crohme-ctc-pytorch. [Accessed: 05-May-2025].

[3] A. Graves and N. Jaitly, Eds., *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks*. 2014.

[4] F. He, J. Tan, and N. Bi, "Handwritten mathematical expression recognition: A survey," in *Lecture Notes in Computer Science*, Cham: Springer International Publishing, 2020, pp. 55–66.