From Handwritten Math to LaTeX: A Deep Learning Approach with Error Correction

Anthony Brogni
Department of Computer Science & Engineering
University of Minnesota - Twin Cities
Minneapolis, MN, USA
brogn002@umn.edu

Evan Pochtar
Department of Computer Science & Engineering
University of Minnesota - Twin Cities
Minneapolis, MN, USA
pocht004@umn.edu

Victor Hofstetter
Department of Computer Science & Engineering
University of Minnesota - Twin Cities
Minneapolis, MN, USA
hofst127@umn.edu

Ajitesh Parthasarathy
Department of Computer Science & Engineering
University of Minnesota - Twin Cities
Minneapolis, MN, USA
parth057@umn.edu

Abstract

This document outlines our final report for the CSCI 5527 Final Project. It includes an introduction to the goal of our project and the corresponding motivation, the related work used to develop our methods, a proposed approach to the implementation of the solution, and our final results and analysis.

1 Introduction

This project aims to address the problem of converting handwritten mathematical expressions into accurate LaTeX code, a crucial task for digitizing mathematical content. The primary objective is to design a system that can efficiently recognize handwritten math expressions and generate LaTeX code while ensuring minimal syntax errors. Our project combines computer vision, sequence modeling, and error correction techniques to improve accuracy beyond existing methods.

2 Motivation

Converting handwritten mathematical expressions into machine-readable formats, such as LaTeX, presents challenges due to the complexity of symbols, multi-line expressions, and varying handwriting styles. While several models exist for handwriting recognition, they often generate incorrect or incomplete LaTeX syntax. This problem is interesting because solving it would greatly benefit fields that require digitizing mathematical notes, such as education, scientific research, and online learning platforms.

3 Related Work

There is significant research on handwritten math expression recognition, particularly using convolutional neural networks (CNNs) and recurrent neural networks (RNNs). Papers such as IM2LaTeX Kanervisto (2016) and others have achieved reasonable accuracy in handwritten-to-LaTeX conversion. Recent advances like transformers and attention mechanisms have also been used to improve sequence generation tasks. However, these models often suffer from LaTeX syntax errors in their output, which require manual post-processing.

One promising direction is MathWriting Gervais et al. (2024), a dataset for handwritten mathematical expression recognition, which provides a larger and more diverse set of handwritten math expressions than previously available datasets. We plan to leverage this comprehensive dataset for our project.

4 PROPOSED APPROACH

In our proposal, we proposed a hybrid approach to this problem involving two major components:

1. Handwritten Image to LaTeX Conversion: A convolutional neural network (CNN) for image feature extraction, followed by a transformer-based sequence generation model for LaTeX output. We will incorporate attention mechanisms to capture spatial relationships between symbols better.
2. LaTeX Error Correction using a Fine-Tuned Language Model: We will employ a fine-tuned large language model (LLM) to post-process the generated LaTeX and correct any syntax or semantic errors. The model will refine the output iteratively, ensuring valid LaTeX code that adheres to grammar and syntax rules.

5 IMPLEMENTATION

Data Acquisition and Preprocessing (Milestone 1): We have successfully set up our development environment using Google Colab and integrated it with Google Drive for data storage. Initial exploration involved accessing the synthetic portion of the MathWriting dataset provided (‘synthetic_images’ and ‘synthetic_labels.txt’). We developed Python scripts utilizing ‘xml.etree.ElementTree’ to parse the InkML file format, extracting both the stroke coordinates and the ground truth LaTeX annotations.

To prepare the data for image-based deep learning models, we implemented a visualization function (‘visualize_inkml’) using ‘matplotlib’ and ‘Pillow’ (PIL) to convert the InkML stroke data into grayscale PNG images. This function handles the necessary coordinate transformations and saves the resulting images. Building upon this, we created a comprehensive preprocessing pipeline (‘process_inkml_folder’) that iterates through a specified directory of InkML files, generates corresponding PNG images (saved to an ‘images’ directory), and compiles a labels file (‘labels.txt’) mapping each image filename to its respective LaTeX ground truth string. This pipeline was successfully tested on a subset of the synthetic dataset (processing 3000 samples initially).

For efficient model training, we implemented a custom PyTorch ‘Dataset’ class (‘HandwrittenMathDataset’) capable of loading the generated images and their corresponding LaTeX labels. This class incorporates standard image transformations (converting images to PyTorch tensors). We also implemented functionality (‘create_train_test_split’) to split the dataset into training and testing sets using PyTorch’s ‘random_split’ and created ‘DataLoader’ instances (‘train_loader’, ‘test_loader’) to handle batching and shuffling during training.

Finally, a crucial step for the sequence generation task was building a character-level vocabulary (‘build_vocab’) from the LaTeX labels present in the dataset. This resulted in a vocabulary size of 95 unique tokens, including special tokens like `<start>`, `<end>`, and `<pad>`. We developed helper functions (‘string_to_tensor’ and ‘tensor_to_string’) to convert between LaTeX strings and padded tensor representations suitable for input/output with sequence models.

Initial Model Implementation and Training (Milestone 2): Following the proposed approach, we implemented the first major component: the image-to-LaTeX conversion model.

Our initial implementation (‘HandwrittenMathToLatexModel’) utilizes a Convolutional Neural Network (CNN) based encoder (‘CNNEncoder’) for feature extraction from the input images and a standard Transformer decoder (‘TransformerDecoder’) for generating the LaTeX sequence. The CNN encoder consists of basic convolutional and pooling layers, while the decoder uses standard Transformer decoder layers with positional embeddings and causal masking. A linear layer projects the CNN features to match the decoder’s expected embedding dimension. We established a training loop and a testing loop using PyTorch, employing the Cross-Entropy loss function (ignoring the padding index) and the Adam optimizer. Initial training runs over 10 epochs showed promising results, with both training and testing losses steadily decreasing (reaching approximately 1.4 and 1.5, respectively), indicating that the model architecture is capable of learning the task. This is demonstrated in Figure 1 on the next page.

To enhance performance, we developed an improved version of this model (‘ImprovedHandwrittenMathToLatexModel’). This iteration features a more sophisticated CNN encoder (‘ImprovedC-

NNEncoder') incorporating batch normalization, more layers, and adaptive average pooling. The Transformer decoder ('ImprovedTransformerDecoder') was also enhanced with a larger model dimension ('d_model=512'), more layers, increased dropout for regularization, and proper sinusoidal positional encodings ('PositionalEncoding'). We refined the training process by switching to the AdamW Loshchilov & Hutter (2019) optimizer, implementing a OneCycleLR Smith (2018) learning rate scheduler, and incorporating label smoothing into the Cross-Entropy loss function. Training this improved model for 10 epochs also demonstrated successful learning, with losses decreasing consistently (reaching approximately 2.16 and 2.26 for training and testing, respectively). While the final loss was higher than the simpler model in this run, the enhanced architecture and training techniques provide a strong foundation for further tuning and scaling.

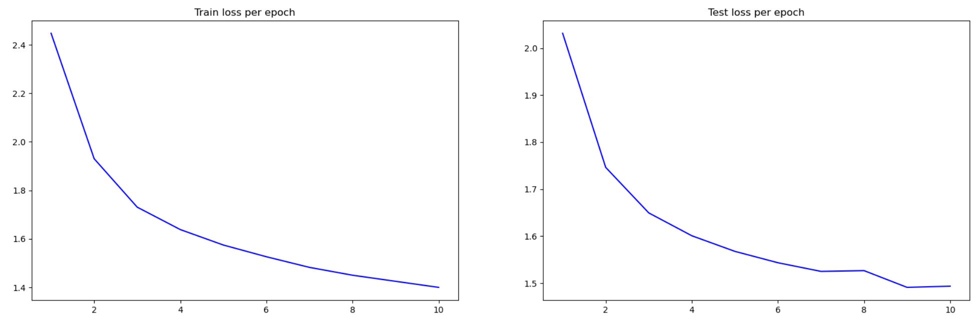

Figure 1: Train Loss and Test Loss per Epoch during Training.

TODO EDIT THIS: Currently, we are exploring alternative encoder architectures. We have begun implementing a Transformer-based encoder ('TransformerEncoder') inspired by Vision Transformer (ViT) models. This approach uses patch embeddings derived from the input image via a convolutional layer, adds positional embeddings, and then processes these patch embeddings through standard Transformer encoder layers. This exploration aims to compare the effectiveness of purely Transformer-based vision models against CNN-based approaches for our task, to see which achieves the best results.

TODO EDIT THIS: The next steps involve completing the implementation and training of the Transformer-encoder variant, conducting more extensive training and hyperparameter tuning for the most promising model architectures, and evaluating their performance using appropriate metrics.

Fine-tune and Integrate the LLM for LaTeX Syntax Correction (Milestone 3): TODO

Fully Functional System with Error Correction (Milestone 4): TODO

6 Results and Analysis

TODO

7 Conclusion and Future Work

TODO

References

Philippe Gervais, Asya Fadeeva, and Andrii Maksai. Mathwriting: A dataset for handwritten mathematical expression recognition, 2024. URL `https://arxiv.org/abs/2404.10690`.

Anssi Kanervisto. im2latex-100k , arxiv:1609.04938, June 2016. URL `https://doi.org/10.5281/zenodo.56198`.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL `https://arxiv.org/abs/1711.05101`.

Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay, 2018. URL `https://arxiv.org/abs/1803.09820`.