Textract

Converting Handwritten Mathematical Expressions to LaTeX Using Deep Learning

Austin Jackson, Elijah Sandler, Ayush Sinha, Owen Sharpe

Final Project for CS4100: Foundations of Artificial Intelligence

Khoury College of Computer Sciences

Northeastern University

June 2025

Abstract

Textract is an end-to-end deep-learning system for converting handwritten mathematical expressions into LaTeX code. Our approach addresses the challenge of digitizing handwritten mathematics by combining a CNN-based encoder with an LSTM-based decoder enhanced with attention mechanisms. Using the CROHME 2023 dataset containing over 150,000 mathematical expressions, we execute a three-stage pipeline: convert the InkML stroke data from the CROHME dataset into images, extract visual features from the images with positional encoding, and generate LaTeX sequences through attention-assisted decoding. Our model achieves performance on the test set, demonstrating the feasibility of making an offline, open-source mathematical OCR. This work provides an accessible alternative to paywalled, commercial solutions. Additionally, this work could save researchers and students hours of manual LaTeX transcription.

1. Introduction

Mathematical expression recognition represents a notable challenge in computer vision, as it requires systems to understand individual symbols, their complex spatial relationships, and hierarchical structures. Unlike standard optical character recognition (OCR), mathematical notation employs two-dimensional layouts where symbols' positions convey semantic meaning. For example, superscripts denote exponentiation, subscripts indicate indices, and fractions create vertical arrangements that the model must preserve in the output.

The problem with converting handwritten mathematics to LaTeX extends beyond simple character recognition. Mathematical notation envelops hundreds of operators, characters, and distinct symbols from different alphabets (e.g., Latin, Greek, Hebrew). Moreover, identical symbols can differ based on context and position. Mathematics' spatial grammar allows for nested structures with arbitrary depths, as the structures previously mentioned can encompass one another. This compositional nature requires models to maintain structural coherence throughout the generation process.

Our implementation includes the following: a preprocessing pipeline that handles the conversion from provided InkML stroke data to normalized images, a position-aware feature extraction mechanism that maintains spatial coherence throughout the encoding process, and an attention-assisted decoding strategy that can generate syntactically valid LaTeX sequences of variable length. In its current state, our system operates offline. It requires no external dependencies beyond the standard deep learning frameworks. It provides an open-source alternative to commercial mathematical OCR solutions.

2. Problem Statement and Methods

2.1 Problem Statement

Converting handwritten mathematical expressions into LaTeX code represents a significant challenge in AI pattern recognition. Unlike standard OCR, mathematical expression recognition must simultaneously solve the following problems: segmenting individual symbols from overlapping strokes, classifying each symbol from a diverse mathematical vocabulary, and understanding the two-dimensional spatial grammar within mathematical notation.

Consider an expression like $x^2 + \frac{1}{2}$. A human intuitively understands that '2' is a superscript of 'x' and the second '2' is a denominator. However, computers may see this differently unless trained correctly to detect and interpret these relationships. The symbol's position relative to others determines its role.

This problem becomes particularly relevant in educational and research contexts where users need to digitize handwritten mathematics. Current solutions require manual LaTeX typing or expensive commercial software. An effective offline, open-source solution would provide an easy alternative to mathematical typesetting tools.

2.2 Methods Overview

Our group has developed a three-stage pipeline that transforms raw handwritten strokes into structured LaTeX expressions:

1. **Data Preprocessing:** Converting stroke sequences into normalized images
2. **Encoding and Feature Extraction:** Using CNNs to identify features in images
3. **Decoding and Expression Reconstruction:** Identifying and assembling symbols into valid LaTeX expressions

2.3 Data Preprocessing and Representation

2.3.1 InkML to Image Conversion

The CROHME dataset provides handwritten expressions in InkML format. These files have an XML-based representation where each expression consists of multiple strokes. Each stroke is a sequence of two-dimensional coordinates (x, y) representing the pen positions over time. This information must be transformed into images suitable for convolutional neural network processing. Our conversion process extracts stroke coordinates from the XML structure and renders them into an image as connected line segments. Each stroke represents a continuous pen movement, preserving the handwriting aspect of the picture.

2.3.2 Stroke Normalization

Stroke coordinates can vary in scale and position depending on the writer and input device. We ensure consistent inputs by normalizing the strokes through a multi-step process. First, we compute a bounding box encompassing all strokes in an expression. This method constitutes finding the minimum and maximum x and y coordinates across the stroke data in an image. Next, we calculate a scaling factor that fits the strokes within the target image size (224x224) while providing a padding buffer of 16 pixels. This process centers the strokes within the target image and applies a uniform scaling factor to preserve aspect ratios. We perform centering and scaling because mathematical symbols often rely on proportions for disambiguation.

2.4 Encoding and Feature Extraction

2.4.1 CNN Architecture Design

Our symbol classification model starts with a convolutional neural network that extracts visual features through multiple processing stages. The architecture utilizes transfer learning by implementing a pre-trained ResNet-34 backbone. This model has learned low-level visual features from millions of images. The structure consists of several components:

Feature Extraction Backbone: The ResNet-34 backbone provides 34 layers of convolutional processing. We omit the final average pooling and classification layers designed for ImageNet because we only need the convolutional feature extractor.

Dimensional Projection: The backbone outputs a 512-dimensional feature map, which projects to a 256-dimensional space using 1×1 convolutions.

Positional Encoding: We add two-dimensional positional encodings to the feature maps. This implementation provides spatial coordinates that help distinguish between similar symbols at different positions.

Feature Refinement: We add two additional convolutional layers to refine the features. These layers transform the generic visual features into specific mathematical symbol characteristics.

2.4.2 Positional Encoding Mechanism

The implemented positional encoding mechanism addresses the limitation of convolutional networks. Their translation can be problematic when spatial relationships convey semantic meaning. We implement two-dimensional sinusoidal position encodings, where each spatial location receives a unique encoding based on sine and cosine functions at multiple frequencies. For a position (x, y) in the feature map, we compute encodings using:

- Horizontal encoding: $\sin\left(\frac{x}{10000^{\frac{2i}{d}}}\right)$ and $\cos\left(\frac{x}{10000^{\frac{2i}{d}}}\right)$
- Vertical encoding: $\sin\left(\frac{y}{10000^{\frac{2i}{d}}}\right)$ and $\cos\left(\frac{y}{10000^{\frac{2i}{d}}}\right)$

'i' represents different frequency components, and d is the encoding dimension. This scheme ensures that each position has a unique signature between adjacent positions [5].

2.5 Expression-Level Understanding and Decoding

2.5.1 Sequence Generation with Attention

We employ a sequence-to-sequence model with attention mechanisms to generate LaTeX

expressions token by token. The Long Short-Term Memory (LSTM) model drives the decoding process. We chose this model for its ability to maintain hidden states that capture sequential dependencies. At each token generation step, the LSTM receives three inputs:

1. **Previous Token Embedding**: The last generated LaTeX token, embedded into a continuous vector space
2. **Visual Context**: Features from the encoder weighted by attention
3. **Hidden State**: The LSTM's memory of previously generated tokens

The attention mechanism dynamically focuses on relevant image regions when generating each token. For instance, the model first attends to the numerator region if producing a fraction. It generates "\frac{" and then shifts attention to the denominator. This awareness is crucial for maintaining the hierarchical structure of mathematical expressions [6].

2.5.2 Attention Mechanism Details

The attention mechanism computes scores between the decoder's current state and each location within the encoded image. These scores determine which image regions contribute the most to the current generation decision. This process follows as such:

1. **Query Generation**: The decoder's hidden state transforms into a query vector
2. **Similarity Computation**: We compare the query with features at each spatial location using dot product similarity
3. **Weighted Aggregation**: Similarity scores are normalized through Softmax to create attention weights, which then help compute a weighted average of encoder features

This method improves the model's understanding of what it has previously generated and influences where it looks next in the image [6].

2.5.3 Beam Search Decoding

Instead of greedily selecting the most probable token at each step, we leverage beam search to explore the most likely 'k' sequences simultaneously. This approach maintains k parallel sequences (usually k=5), expanding each with its most likely continuations [7].

Beam search is particularly valuable for mathematical expressions where early decisions strongly constrain later possibilities. For example, starting "\frac" commits to a two-argument structure that the model must complete correctly. Holding multiple sequences, the decoder can recover from initially likely but ultimately incorrect paths.

2.6 Vocabulary and Tokenization

LaTeX presents unique tokenization challenges due to mathematical commands being multi-character sequences. Our tokenization strategy identifies and preserves these units while creating a manageable vocabulary. The tokenizer recognizes several types of tokens:

- **LaTeX Commands**: Multi-character sequences like \frac, \sqrt, \alpha
- **Structural Symbols**: Single characters with special meaning: ^, _, {, }
- **Alphanumeric**: Letters and digits
- **Operators**: +, -, =, and other mathematical operators

We also employ special tokens that mark sequence boundaries (<SOS>, <EOS>), handle padding (<PAD>), and represent unknown symbols (<UNK>). The vocabulary built from the training set contains approximately 100 unique tokens.

2.7 Evaluation Metrics

We evaluate the model's performance through multiple metrics:

Symbol-Level Accuracy: Measures the percentage of correctly classified individual symbols, providing an understanding of the classifier's performance independent of sequence generation.

BLEU Score: The Bilingual Evaluation Understudy (BLEU) score measures n-gram overlap between generated and reference LaTeX. This provides partial credit for expressions with local correctness despite global errors.

3. Related Work

3.1 Synthetic Handwritten Mathematical Expression Generation

(Heska 2021) presents a system for generating handwritten mathematical expressions from markup languages such as LaTeX or MathML. Their approach first operates by parsing input

expressions into Symbol Layout Trees (SLTs), which encode spatial relationships between symbols through six edge types: next, above, below, over, under, and within. Individual handwritten symbols are sampled from the CROHME dataset and positioned according to the SLT structure.

A key contribution from Heska's work is the edge-ordered recursive traversal method that replaces depth-first search for layout construction. This method prevents overlapping conflicts in long expressions. The system uses local and global distortions, increasing variability within the output and maintaining style consistency through metadata-based clustering. Despite achieving an 85% success rate on the National Test Collection for Information Retrieval (NTCIR) dataset expressions, approximately half required bounding box substitutions for missing symbols. This issue highlights the limited symbol coverage in prevalent datasets.

3.2 CROHME Competition and State of the Art

The CROHME 2016 competition report by Mouchère et al. provides vital benchmarks for handwritten mathematical expression recognition. The competition included four tasks: formula recognition from strokes, isolated symbol classification, structure parsing from provided symbols, and matrix recognition. Notably, the highest expression-level recognition rate achieved was only 67.65% by MyScript. The best public-domain system (WIRIS) achieved 49.61% using only the provided training data. The competition revealed several important insights about current limitations. Symbol-level metrics showed that most systems achieve higher precision than recall, suggesting they tend to under-segment symbols when errors occur.

3.3 Deep Learning Methods

(Lu and Mohan 2015) investigate the application of convolutional neural networks (CNNs) to handwritten mathematical expression recognition. This paper marks the first published work to explore CNNs as a solution for this task. Their pipeline consists of five phases: dataset enrichment through distortions, image segmentation using stroke intersection heuristics, data extraction to normalized pixel arrays, character-level CNN classification, and expression-level classification using Hidden Markov Models. Their CNN architecture achieves 90% test accuracy for isolated symbol classification. This result outperforms their baseline SVM classifier's 87% accuracy. However, expression-level accuracy drops dramatically to 39% even with their best CNN. These results highlight the compounding effect of segmentation and classification errors.

4. Experiments

4.1 Metrics, Accuracy, and Evaluation

The experiments were completed using the CROHME dataset, which contains 157,345 training images, 1,699 validation images, and 3,495 test images. Each image was then converted into grayscale and resized to 224x224 pixels. The images were finally normalized with a mean of 0.9 and a standard deviation of 0.1.

4.2 Vocabulary Construction and Tokenization

By using a custom vocabulary that includes only tokens that appear twice, all LaTeX expressions in the training split were tokenized. Start-of-sequence (“”) and end-of-sequence (“”) markers were prepended and appended. Sequences were truncated or padded to a maximum length of 256 tokens, and padding was applied per batch to match the longest sequence in that batch. This led to approximately 5,772 distinct tokens in the vocabulary.

4.3 Model Variants

We then continued to develop and compare three different architectures. This included a CNN–GRU baseline, a ResNet-34 encoder with a two-layer LSTM and additive attention, and a ResNet-34 encoder with a three-layer Transformer decoder. The CNN-GRU had three convolutional layers with ReLU activations and pooling, global average pooling, and a single layer GRU decoder. The ResNet-34 model with the LSTM and attention layers projected pre-trained ResNet-34 features 256 dimensions. It also added two-dimensional positional encodings and used a two-layer LSTM that attended over each 7x7 spatial feature to decode. The transformer variant worked by replacing the LSTM decoder with a self-encoder/decoder attention component using eight heads, 1,024 dim-feed forward sublayer, dropout of 0.1 and both image and token positional encodings.

4.4 Evaluation Metrics

We evaluated the performance of the models using cross-entropy validation loss, token-level accuracy(without padding), and BLEU-4 on complete sequences.

4.5 Results and Analysis

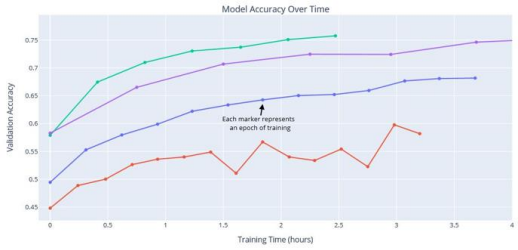

Figure 1: Validation accuracy over training time for Models

	En/Decoder Dimensions	Number of Layers	Batch Size	Pretrained Model?
Small Network	64	1	128	False
Standard Network	256	2	32	False
Large Network	512	3	32	True
Standard Network w. New Transformer	256	3	32	True

Table 1: Configuration of models

As you can see above, the CNN-GRU baseline starts at roughly 0.45 validation accuracy and plateaus around 0.60. Both the ResNet34 with the LSTM and Attention layers and the transformer decoder start at about 0.58 and get to about 0.68, and the Transformer plateaus near 0.75. The larger LSTM variant also begins at 0.58 and peaks around 0.76. Their training losses mirror this ranking, with the baseline ending near 1.4, the standard LSTM near 1.1, the large LSTM dipping to about 0.8, and the Transformer achieving the lowest loss at approximately 0.85.

5. Future Work

5.1 Model Size and Efficiency

The model that achieves 75% expression accuracy contains over 50 million parameters and is represented by a 230MB .pth file. Predicting even a simple expression containing half a dozen tokens takes seconds, and longer expressions can take minutes. Obviously, this isn't sustainable. While we could shorten the inference time by simply throwing a very powerful GPU at the problem, it would be significantly better to lower processing demands of the architecture.

The main culprit of the long inference time and large model size is the ResNet-34 backbone, which uses 34 convolution layers containing over 21.7 million parameters. Any attempt to streamline Textract would invariably start there, but because ResNet is pretrained, our options are to either keep it or scrap it in its entirety. One architecture to consider replacing it with is ResNet-18, a lighter pretrained model with 18 convolutional layers and a more modest 11.8 million parameters.

5.2 Diversifying Training Data and Discrimination

The model performs notably differently depending on the handwriting of the input expression. This is problematic for at least two reasons: first, lower accuracy, and second, ethical concerns.

A model that is sensitive to handwriting is less accurate than one that is insensitive to it. To make the model more accurate and more robust to changes in handwriting, we'd like to source training data from a wider variety of sources.

On the ethical front, a model that discriminates based on handwriting could be problematic. Theoretically, handwriting could encode sensitive information such as socioeconomic status or race that we do not want to influence our model. Resolving this could be relatively difficult, since traditional fairness metrics like calibration or demographic parity don't really apply in this case.

5.3 Metrics, Accuracy, and Evaluation

Mathematical translation makes a strong case for a novel effectiveness metric. Current approaches tend to use accuracy, considering a fully correct translation as a success and everything else as a failure. The CROHME competition also used symbol-wise calculations for precision and recall, but those metrics have their issues when we aren't guaranteed to have the same number of tokens in our output as our ground truth, though they certainly have their place for evaluating the model's ability to pick out individual characters.

Other papers, recognizing the similarity of this project to language translation, have used the Bilingual Evaluation Understudy (BLEU) score, which measures how close a machine translation is to a baseline human translation. While this seems promising at first glance, the underlying calculation seems to have a fatal flaw: BLEU scores punish you for shorter translations, on the logic that a shorter translation contains less information. This doesn't make sense in the case of

mathematical notations, though, because of the following trilemma:

1. Mathematical expressions can't be machine-translated into LaTeX.
2. There is only one correct LaTeX translation for a mathematical expression.
3. There are many correct LaTeX translations for a mathematical expression.

Considering (1) is pointless, we will leave it aside. When we consider (2), we must ask why we need a penalty term for shorter translations: if your translation is too short, then you are necessarily wrong. This case seems to suggest that accuracy is a better metric than BLEU scores. (3) is more contentious, but in a world where there are many ways to type mathematical expressions into LaTeX, wouldn't the optimal one be the shortest, or the most conventional? It seems like what we would really want is some sort of modified BLEU score that actually rewards shortness. Regardless, in any of the above cases, the BLEU score is not effective for evaluating our model or models like it, and further work is required to come up with a suitable metric.

5.4 End User Improvements

Even if we assume a fully functional, accurate, and lightweight Textract, changes will still need to be made to the pipeline to improve the tool's usability. Currently, the inference program takes an image file containing a single handwritten expression. This would get tedious fast. However, there are ways we can consider changing the model wrapper to make use of it more efficient.

A computer vision system to delineate between lines within the same image could allow the system to take in a single image of a piece of paper and split it into many smaller images, each of which contains a single expression that the model can translate, before reassembling the split images into one LaTeX document. That type of system could also be combined with the live stroke data captured by writing tablets such as iPads, which could theoretically allow for near- real- time handwriting-to-LaTeX translation.

6. Link to Code Repository

Visit our GitHub Repository here: Textract GitHub Repository

7. References

[1] Lu, Catherine, and Karanveer Mohan. 2015, *Recognition of Online Handwritten Mathematical Expressions Using Convolutional Neural Networks*, https://cs231n.stanford.edu/reports/2015/pdfs/mohan_lu_cs231n-project-final.pdf. Accessed 2025.

[2] Varotariya, Savan, and Vikas Tulshyan. 2025, *A CNN-Based Approach for Handwritten Mathematical Formula Recognition and LaTeX Generation*, https://www.ijfmr.com/papers/2025/2/41926.pdf . Accessed 2025.

[3] Heska, Vincenzo. 2021, *Generating Synthetic Online Handwritten Mathematical Expressions from Markup Languages*, https://uwaterloo.ca/computational-mathematics/sites/default/files/uploads/documents/vincenzo_heska_-_research_paper.pdf. Accessed 2025.

[4] Mouchère, H., et al. 2016, *ICFHR2016 CROHME:Competition on Recognition of Online Handwritten Mathematical Expressions*, https://www.cs.rit.edu/~rlaz/files/icfhr_crohme2016.pdf. Accessed 2025.

[5] Paleti, Nikhil Chowdary. "Positional Encoding Explained: A Deep Dive into Transformer PE." *Medium*, The Deep Hub, 2024, medium.com/thedeephub/positional-encoding-explained-a-deep-dive-into-transformer-pe-65cfe8cfe10b.

[6] Shevchenko, Eugenii. "Attention Mechanism for LSTM Used in a Sequence-to-Sequence Task." *Medium*, Medium, 2023, medium.com/@eugenesh4work/attention-mechanism-for-lstm-used-in-a-sequence-to-sequence-task-be1d54919876.

[7] Venkatesaramani, R. (2023). Notes on Artificial Intelligence. Notes on AI. https://rajagopalvenkat.com/teaching/resources/AI/