Image2Latex

Dylan Paulson
paul1217@umn.edu
University of Minnesota, Twin Cities
Minneapolis, Minnesota, USA

Charlie Dedinsky
dedin005@umn.edu
University of Minnesota, Twin Cities
Minneapolis, Minnesota, USA

Michael Safonov
safon010@umn.edu
University of Minnesota, Twin Cities
Minneapolis, Minnesota, USA

1 Main Problem

The aim of this project is to develop a machine-learning model that can automatically convert images of mathematical formulas into LaTex code. LaTex is a widely used markup language for typesetting mathematical expressions and other scientific documents. However, writing LaTex code can be challenging and time-consuming for many users, especially those who are not familiar with its syntax and commands. A convenient and user-friendly way to generate LaTex code from images of formulas would be beneficial for enhancing the productivity and accessibility of scientific communication.

The main challenge of this project is to design and train a machine learning model that can accurately recognize and translate various symbols, notations, and structures in mathematical formulas from images to LaTex code. This task is difficult because mathematical formulas can have high variability in terms of font size, style, orientation, spacing, alignment, and complexity. Moreover, the model needs to capture the semantic and syntactic relationships between different symbols and components in a formula, such as subscripts, superscripts, fractions, integrals, matrices, etc., and encode them correctly in LaTex format.

2 Technical Machine Learning Problem

One of the main challenges in translation problems is to capture the relationships between all the items in a given sequence, such as mathematical formulas or natural language sentences. Many existing models struggle to preserve the connection between the initial and the final parts of a sequence, especially when the sequence is long. This leads to a loss of information and coherence in the generated output. A common approach to address this problem is to use an encoder-decoder architecture, where the encoder transforms the input sequence into a latent representation, and the decoder generates the output sequence based on the latent representation and the previous output symbols. However, this approach also has some limitations. For instance, when encoding images that contain symbols, such as patches or individual character recognition, the encoder may not capture all the relevant features or may introduce noise or distortion. Moreover, the decoder only has access to the information up to the current output symbol, and may not be able to leverage the global context of the sequence. Furthermore, different formulas can be written in various ways using different notations or conventions, which adds another layer of complexity and ambiguity to the problem.

3 Data and Pre-processing

In this study, we used the dataset created by Deng et al. (2016) [1] for their paper "Image-to-Markup Generation with Coarse-to-Fine Attention." This dataset comprises 103,556 pairs of mathematical formulas and their corresponding images in PNG format. The data was sourced by parsing LaTeX sources from academic papers featured in Tasks I and II of the 2003 KDD Cup, which focused on text mining and link analysis. To maintain relevance, only formulas with a length between 39 and 997 LaTeX characters were included, effectively excluding single symbols and text sentences.

Images: We began by pre-processing the images, applying a thresholding technique that converted all pixel values to either 0 or 255, resulting in one-channel binary images. The original images had a resolution of 1654x2339 pixels, but since they primarily contained small equations on large pages, they were inefficient for training. To resolve this issue, we cropped the images to 1254x550 pixels, effectively eliminating unnecessary empty space. Afterward, we divided the dataset into training, testing, and validation sets in the proportions of 0.8, 0.1, and 0.1, respectively, to maintain consistency with previous research on the im2latex 100k dataset.

Formulas: We removed any extra whitespace characters. To ensure consistency between input and output modalities, we also eliminated tokens that weren't visible in the rendered images, such as label, cite, and ref commands. After these adjustments, we tokenized each LaTeX formula using a vocabulary of 533 tokens. This vocabulary included special tokens for the start of sequence <sos>, end of sequence <eos>, padding <pad>, and unknown <unk> elements. Finally, we converted the tokenized formulas into integer values, preparing them for input into the model.

4 Algorithms Used

The primary algorithms utilized in this project are a ResNet encoder and a Transformer decoder. The ResNet encoder, a convolutional neural network, extracts high-level features from the input images. We removed the last two layers of the ResNet encoder (average pooling and fully connected), resulting in an output tensor with a shape of (Batch_size, X,

embed_size), where X is equal to $\lceil \frac{H}{32} \rceil \cdot \lceil \frac{W}{32} \rceil$. Each of the X "patches" is represented by a vector on length embed_size.

Next, we applied a sine-cosine positional encoding to each "patch," as proposed in the "Attention is All You Need" [3] paper. This step encodes the spatial information of the image, providing important context. After the positional encoding was added, the "patches" were fed into the Transformer decoder, which then generated the corresponding LaTeX formula based on the encoded image features. This combination of the ResNet encoder and Transformer decoder allowed our model to effectively process and translate the input images into the target formulas.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

The decoder serves as a sequence-to-sequence model that generates output text based on the input image features and the target sequence. Comprised of several Transformer blocks, the decoder consists of a masked self-attention mechanism, an encoder-decoder attention mechanism, and a position-wise feed-forward network, complete with layer normalization and residual connections. The decoder's design is inspired by Aladdin Persson's implementation [2] of the "Attention is All You Need" architecture [3].

The masked self-attention mechanism enables the decoder to focus on different parts of the target sequence selectively while preventing it from attending to future positions. This is accomplished by dividing the input into multiple heads, calculating the query, key, and value matrices for each head, and then masking future positions using a lower triangular matrix through the softmax operation. The output from all heads is concatenated and passed through a linear layer, which facilitates autoregression. Consequently, the decoder generates one token at a time, conditioned only on the previous tokens.

The encoder-decoder attention mechanism allows the decoder to attend to the input image features (keys and values) generated by the encoder, using its own queries based on the target sequence. This helps the decoder produce context-rich and relevant output text.

By combining the ResNet encoder and Transformer decoder, the model can generate high-quality output text from input images using advanced attention mechanisms and positional encodings.

5 Hyperparameter Tuning

During the hyperparameter tuning process, we leveraged Bayesian optimization with the help of the Scikit-Optimize library's BayesSearchCV. Our search space consisted of four categorical hyperparameters: ResNet type with possible values of 18, 34, and 50; the number of layers with options of 1, 3, or 6; embedding size choices of 256, 512, 1024, or 2048; and the number of heads with values of 4, 8, or 16. The objective function, train_and_evaluate, took these hyperparameters as input, trained a Transformer model using the provided values, and returned the negative BLEU score as the optimization target. We used a Gaussian Process model to guide the search, and the optimization converged after 15 iterations, resulting in the sparse tables seen below.

For each iteration, we trained the model for 3 epochs with a learning rate of 1e-4. Ideally, we would have liked to run each iteration to convergence, which typically takes 5-7 epochs based on our testing. However, due to the model's complexity, we decided to limit the training to 3 epochs per iteration.

Table 1. ResNet 18 results

	1 layer			3 layers			6 layers		
Embed size	256	512	1024	256	512	1024	256	512	1024
4 heads	-	-	-	-	-	0.506	-	-	-
8 heads	0.112	-	-	0.178	-	-	-	0.325	-
16 heads	-	-	-	-	0.204	-	-	-	-

Table 2. ResNet 34 results

	1 layer			3 layers			6 layers		
Embed size	256	512	1024	256	512	1024	256	512	1024
4 heads	-	-	0.447	-	-	0.531	-	-	0.599
8 heads	0.127	273	-	-	-	-	-	0.384	-
16 heads	-	-	-	-	-	0.373	-	-	-

Table 3. ResNet 50 results

	1 layer			3 layers			6 layers			
Embed size	256	512	1024	256	512	1024	256	512	1024	2048
4 heads	-	-	0.669	-	-	-	-	-	0.669	-
8 heads	-	-	-	-	-	-	-	-	-	0.667
16 heads	-	-	-	-	-	-	-	-	-	-

5.1 Resnet Architecture

The Resnet-50 architecture outperformed its 18 and 34-layer counterparts, delivering superior results. Due to the constraints in computing resources, training larger models wasn't a viable option.

5.2 Number of Layers

Our experiments revealed that the optimal number of layers is 6, which is higher than any of the other values we tested. We also attempted to use 12 layers, but this led to batch sizes that were too small to produce meaningful results given our available computing resources.

5.3 Embedding Size

We found that larger embedding sizes generally led to better performance. However, the difference in BLEU scores between the 2048 and 1024 embedding sizes was a mere 0.003, so we chose the smaller size to conserve resources.

5.4 Number of Heads

Our results indicate that the optimal number of heads is contingent on the embedding size. Since the embedding is divided by the number of heads, a 1024 embedding size yielded the best results with 4 heads, while a 2048 embedding size performed optimally with 8 heads. Our tests also demonstrated that a minimum embedding size of 256 per head is necessary for satisfactory performance.

6 Results

6.1 Computational Resources

We utilized a combination of hardware and platforms to conduct the training, testing, and initial prototyping processes. We employed an NVIDIA GeForce RTX 4090 for a portion of the training while relying on Google Colab for much of the testing and preliminary prototype development. Throughout the training phase, the model and data loader worked in tandem to maximize the use of the available 24GB of VRAM on the 4090. When training the model using a batch size of 16 the process took approximately 45 minutes to iterate over the entire training set of 82,828 images.

6.2 Training

We trained our model employing a learning rate of 1e-4 and continued the process until convergence, which was determined by a validation BLEU score increase of less than 0.01. To maximize efficiency, we adjusted the batch size for each model based on hyperparameters and available computational resources. Cross-entropy loss and the Adam optimizer were utilized for training all models. The final model on the smaller dataset was trained with the following hyperparameters:

Embedding size: 1024
ResNet type: 50
Number of layers: 6
Forward expansion: 4
Heads: 4
Dropout rate: 0

After 5 epochs, our model converged, achieving a test set accuracy of 0.7745 as measured by the BLEU score.

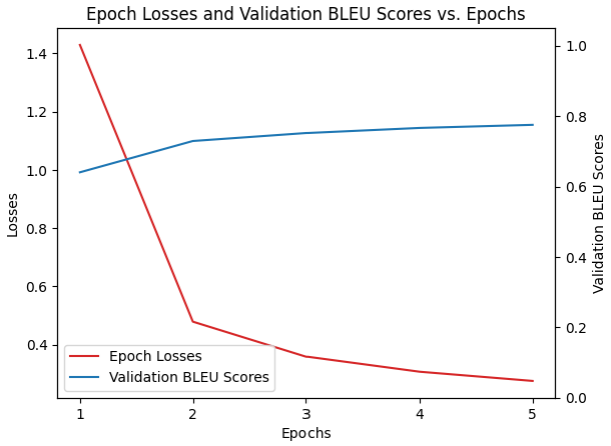

Figure 1. Training Losses and Validation BLEU Scores

Because the input images were quite large, we had to use a smaller batch size of 16 when training the model. This meant that each epoch took longer to complete, as it had to process more than 5000 batches. However, even with this slower training process, we were still able to achieve satisfactory validation accuracy in just 5 epochs as shown in Figure 1.

6.3 Evaluation Metric

When evaluating the im2latex model, we apply the well-known BLEU (Bilingual Evaluation Understudy) score as a primary metric to assess the similarity between the generated LaTeX code and the ground truth. The BLEU score ranges from 0 to 1, with higher values denoting better correspondence between predicted output and reference data. Our BLEU score implementation uses a 4-gram model to consider sequences of up to four tokens. This enables us to evaluate the model's ability to generate not only single tokens but also coherent phrases and structures in LaTeX code, providing a comprehensive assessment of the model's performance.

6.4 Single Image Evaluation

Table 4 (found at the end of the document) shows the model's performance successfully recreate the LaTeX from the images, despite having dissimilar LaTeX code.

6.5 Related Work

The performance of the model developed in this paper was evaluated using the BLEU score metric and compared with other state-of-the-art research using the same dataset. In our research, the best BLEU score achieved was 0.7745 on the test set, indicating a high level of translation quality and demonstrating the effectiveness of our proposed approach in comparison to existing methods.

In the state-of-the-art studies listed in the table below, all of them utilize normalized formula datasets to train their

Table 4. Comparison of Training and Generated LaTeX Expressions

Trial	Training Formula	Generated Formula
1	$b_{2k} = \frac{y_2(y_1^2+y_2^2)}{2k(y_1^2-3y_2^2)}[R_k + \frac{y_1^2+y_2^2}{\zeta y_2}P_k].$	$b_{2k} = \frac{y_2(y_1^2+y_2^2)}{2k(y_1^2-3y_2^2)}[R_k + \frac{y_1^2+y_2^2}{\zeta y_2}P_k].$
2	$I = \int d^2w dx_0 \frac{1}{2x_0^3}\left[\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + \frac{1}{2}L(L+2)A_\mu A^\mu\right]$	$I = \int d^2w dx_0 \frac{1}{2x_0^3}\left[\frac{1}{4}F_{\mu\nu}F^{\mu\nu} + \frac{1}{2}L(L+2)A_\mu A^\mu\right]$
3	$F_\nu^{ext} =, -\frac{\alpha_f \hbar c}{2\pi}\int d^3r\vec{R}\nu\mu^{ext}\vec{j}^\mu = \int {}^*\vec{F}\nu\mu^{ext}d^3r\vec{j}^\mu.$	$F^{ext}{}_\nu =, -\frac{\alpha f\hbar L}{2\pi}\int d^3\vec{P}\nu\mu^{ext}\vec{J}^\mu = \int F^{ext}{}_{\nu\mu t^3}\vec{J}^\mu.$
4	$Z = \sum \sigma_i = \pm 1 \sum_{colorings} p^b(1-p)^{N_b-b}.$	$Z = \sum_{\sigma_i=1 \circ colinss} p^b(1-p)^{N_b-b}.$

models. Normalizing the dataset of formulas is crucial because there are dozens of ways to represent the same equation in LaTeX. Due to time constraints, our research did not involve extensive normalization of the dataset to adopt a uniform set of conventions for all formulas. However, we tested our model using a formula dataset with all whitespace removed, as the majority of whitespace characters do not impact the resulting equation. With this dataset, our model achieved a BLEU score of 0.8786 on the test set, which is significantly closer to other state-of-the-art research with minimal data normalization.

Although the BLEU score on our dataset without whitespace is substantially higher than that of our original dataset, we opted to use the original model as our final product. This is because it can accurately predict the necessary whitespace characters to render the output, and any additional whitespace characters that it predicts, which negatively impact the BLEU score, ultimately make no difference in the rendered output.

Paper	Model Type	BLEU Score
Our Model	Resnet-Transformer	0.7745
Our Model -ws	Resnet-Transformer	0.8786
Zhang et al. [5]	CNN-RNN	0.8842
Vyaznikov et al. [4]	RNN-LSTM	0.7
Zhou et al. [6]	DenseNet	0.84
Deng et al. [1]	Transformer	0.8773

7 Next Steps

In order to further refine our model and address the limitations we encountered during our research, we have identified two key areas to focus on: further hyperparameter tuning and data normalization.

7.1 Further Hyperparmeter Tuning

Due to time and resource constraints, we could only test a small selection of hyperparameters for our model. Moving forward, we believe that trying out a wider variety of hyperparameters might help us enhance the model's performance. We're particularly interested in exploring larger ResNet architectures like ResNet-101 and ResNet-152, as they might offer greater representational capacity. On top of that, we want to play around with other hyperparameters, like dropout rates and forward expansion factors, which could potentially boost the model's ability to generalize and improve its overall performance. With more time and resources, we're keen to fine-tune our model and achieve even better results in tackling the im2latex challenge.

7.2 Further Data Normalization

As mentioned above, the im2latex 100k dataset is not standardized, as it utilizes a diverse range of conventions for representing the same LaTeX code. Consequently, identical images can be rendered through numerous distinct methods, making it challenging for the model to establish meaningful relationships between the formulas and their corresponding images. This lack of normalization significantly impacts the efficacy of the BLEU score evaluation metric in accurately reflecting the quality of the predicted formula. In many instances, the BLEU score may yield low values, or even zero, when the actual rendered output of the predicted formula precisely matches the input image.

For instance, the following two lines of LaTeX generate the same formula but have a low BLEU score due to unnecessary spacing in the training set. One line is from the training set, one is the output from our model:

Training Formula: `{{g(\tau ,\vec \sigma )}\over {\gamma (\tau ,\vec \sigma )}}=g_{\tau\tau}(\tau ,\vec \sigma )-\gamma^{{\check r}{\check s}}(\tau ,\vec \sigma )g_{\tau {\check r}}(\tau ,\vec \sigma )g_{\tau {\check s}}(\tau ,\vec \sigma ).`

Generated Formula: `\frac{g(\tau,\vec{\sigma})}{\gamma(\tau,\vec{\sigma})}=g_{\tau\tau}(\tau,\vec{\sigma})-\gamma^{\check{r}\check{s}}(\tau,\vec{\sigma})g_{\tau\check{r}}(\tau,\vec{\sigma})g_{\tau\check{s}}(\tau,\vec{\sigma}).`

To address this issue, we propose the development of a robust and standardized preprocessing pipeline for the im2latex dataset. This pipeline should be designed to harmonize the various conventions used, thereby reducing ambiguity and promoting a more consistent representation of LaTeX code. Additionally, we suggest investigating alternative evaluation metrics that account for the semantic similarity between the predicted formula and the ground truth, rather than relying solely on the BLEU score. These metrics could potentially provide a more accurate assessment of the model's performance, leading to better insights and subsequent improvements.

7.3 Other Evaluation Metrics

Since the dataset used in our research was not fully normalized, the BLEU score evaluation metric was insufficient in accurately representing how well the predicted formula corresponds to the original image. One way to address this issue is to fully normalize the dataset; however, doing so is challenging. As an alternative, we recommend developing additional evaluation metrics to compare predictions with the original image. In the past, metrics such as exact match and exact match after removing whitespace have been employed, but it would also be valuable to create a system that compares two rendered images and scores the percentage of pixel match. This approach would eliminate the need for a fully normalized dataset, providing a more accurate assessment of the model's performance.

7.4 Data Augmentation

As we move forward with our research, we also recommend looking into data augmentation as a way to improve our model's flexibility and resilience. Right now, our dataset mostly includes mathematical equations that are similar in both position and font size. However, in everyday situations, users often come across equations that have different font sizes and placements on a page. To make our model more accurate and useful in real life, we suggest expanding the dataset to include equations located in various spots on the page and with a range of sizes. By making these changes, we hope to develop a model that's more adaptable and versatile, able to handle mathematical equations from screenshots regardless of their size or position.

7.5 Handwritten Formulas

In addition to data augmentation, another valuable direction for our research would be to extend our dataset to include handwritten formulas. Currently, our model is only trained on typed mathematical equations, which might limit its effectiveness in real-life scenarios where users frequently encounter handwritten formulas in notes, textbooks, or on whiteboards. By expanding our dataset to encompass a diverse range of handwritten formulas, we can develop a more comprehensive and versatile model capable of accurately interpreting both typed and handwritten equations.

8 Feedback Utilized

At the beginning of the project, we faced challenges in developing meaningful relationships between the input images and output formulas. To address this, we sought feedback from our peers and mentors, who suggested implementing a Transformer architecture with attention to capture the complex relationships between the inputs and outputs. This feedback was invaluable, and we ultimately built our final architecture based on the "Attention is All You Need" Transformer architecture, which significantly improved our model's performance.

We also received feedback on the computational complexity of the input images, which were initially quite large at 1654x2339 pixels. Our peers and mentors suggested resizing the images to reduce computational complexity without losing any information about the equations. We followed this advice and were able to resize the images to 1254x550 pixels, which improved our training speed and model performance.

We received feedback that we needed a way to measure the difference between the predicted output and the ground truth. After reviewing a variety of NLP evaluation metrics, we decided to use BLEU score to measure the similarity between the predicted and ground truth formulas. This feedback was crucial in helping us to improve our model evaluation and demonstrate the effectiveness of our approach.

Finally, we received feedback on the computational resources we were utilizing for our project. To address this, we sought out more powerful computing resources and set up our project on Google Colab, which allowed us to train our models more efficiently and effectively. This feedback was instrumental in helping us to overcome computational limitations and improve our overall performance.

References

[1] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M Rush. 2017. Image-to-markup generation with coarse-to-fine attention. In *International Conference on Machine Learning*. PMLR, 980–989.

[2] Aladdin Persson. 2022. transformer_from_scratch.py. https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/more_advanced/transformer_from_scratch/transformer_from_scratch.py. Accessed: April 23, 2023.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[4] PA Vyaznikov and IGOR D Kotilevets. 2021. Developing a seq2seq neural network using visual attention to transform mathematical expressions from images to LaTeX. 19, 8 (2021), 40–44.

[5] Wei Zhang, Zhiqiang Bai, and Yuesheng Zhu. 2019. An improved approach based on CNN-RNNs for mathematical expression recognition. In *Proceedings of the 2019 4th international conference on multimedia systems and signal processing*. 57–61.

[6] Zhengqing Zhou, Junwen Zheng, and Zhongnan Hu. NA. Image To Latex: A Neural Network Approach. NA, NA (NA), NA.