Adaptation of OCR Models for LaTeX Vision

Nikash Chhadia
Stanford University
chhadia@stanford.edu

Sambhav Gupta
Stanford University
samgupta@stanford.edu

Abstract

This research focuses on OCR for LaTeX mathematical expressions, which is the problem of determining the LaTeX code used to generate a compiled expression in image form. We apply two approaches to this problem, namely (1) fine-tuning a general OCR model instead of training a model from scratch, and (2) utilizing a synthetically generated LaTeX dataset to enable training on a large set of data without manual data collection. Both of these approaches are applied with the intention of drastically reducing the quantity of resources and compute required to train an effective model for the task. We find that, after fine-tuning, although the model shows some success classifying simple expressions, performance rapidly breaks down for more complex expressions. We present potential explanations for this performance for a model which succeeds in OCR tasks, which include an oversized token set, undertraining of the model, and lack of CNNs for feature extraction. We also present an interesting visualization of the attention mechanism of the model, which shows bias towards the beginnings of expressions.

1. Introduction

Optical Character Recognition (OCR) is typically defined as the process of converting an image with text into a machine-readable text format. Techniques and strategies for OCR have significantly evolved over the past decades, transforming how we digitize and interact with textual data. It allows for the conversion of different types of documents, such as scanned PDFs or images, into editable and searchable data, which has found applications across numerous fields including data entry automation, content archiving, and digital accessibility.

This research focuses on OCR for mathematical expressions and language, particularly that of the LaTeX markup language, which incorporates deterministic semantics for more complicated mathematical and scientific notation including sub and superscripts, fractions, and special symbols. With this research, we aim to determine the feasibility of using an off-the-shelf general OCR model such as TrOCR [8] to solve the LaTeX to text problem. The input to the TrOCR model is an image of text, and the model uses a vision transformer to output a sequence of predicted tokens which state the content of the text in the image. After fine-tuning, our version of the TrOCR model will take an image of compiled LaTeX as input, using the vision transformer to output a series of tokens relaying the code which would generate that image. We also use a randomized algorithm to generate a large set of LaTeX strings, which can be compiled to form a large dataset of image-text pairs for this problem.

For our results, we find that the general OCR model is relatively poor at learning this task. In particular, we find that the model is able to learn to recognize short LaTeX expressions and is often able to recognize the symbols in longer expressions, but generally has trouble conforming to the strict syntactic structure of LaTeX expressions.

2. Related Work

As a basis of our work, we rely on the following papers, several of which represent attempts at solving the LaTeX OCR problem, one of which presents the general OCR model (TrOCR) which we fine-tune to obtain our results, and one of which presents inspiration for our dataset geneation algorithm.

[3] uses a neural encoder-decoder model for LaTeX OCR, which is aligned with our desired task. It introduces an RNN (in practice, LSTM [7])-based architecture for encoding which recursively encodes the rows and columns of the input image to generate features, followed by a fairly standard RNN-based decoder. It also introduces some interesting and fairly complex alternative attention schemes including sparsemax attention [12] and hard attention. Finally, and most importantly for our purposes, it introduces a large dataset of LaTeX-image pairs which we hope to use to compare our model's performance on synthetic data and real-world data.

[14] is a more recent paper, also on LaTeX OCR, which applies more modern techniques to the problem and achieves much better results, including an increase in BLEU

score from 66.65 to 89.72. The model they use consists of three main modules: a global feature extractor, a transformer-based [19] encoder, and a mask attention-based decoder. The feature extractor is a fairly standard convnet with some custom residual layers. The encoder is standard, using sine and cosine positional embeddings rather than learned embeddings. The decoder is custom-proposed for the purpose of mitigating over-parsing, when some parts of the image are parsed multiple times, and uses a Gate Recurrent Unit (GRU) function [1] — which can be utilized if we notice over-parsing in our model's results.

In addition to the above two papers, we take inspiration from the following papers, which are mentioned in comparisons by the above paper and which we summarize briefly. [17] uses a fairly involved and early approach to LaTeX recognition, which involves connecting an OCR model for non-LaTeX characters with a LaTeX character recognition model. This is somewhat similar to our approach of fine-tuning a non-LaTeX OCR model. [4] uses a standard convolutional network approach for feature extraction, then uses an RNN as a row encoder before passing the marked up tokens into the decoder, alongside some other tricks. This specialized scheme, which uses a hand-tuned architecture specifically based on insights into LaTeX, has a similar design philosophy to [14]. We see similar features in [23]; features are created with a CNN, then passed into an LSTM-based encoder. In this case, they are decoded with two separate attention mechanisms (hence the name), then the output is passed in series to another series of LSTM modules, which keep hidden state and produce the final series of tokens. We also reference [5], which focuses on parsing of handwritten mathematical expressions. This paper utilizes a CNN feature extractor along with GRU units and single headed attention in the decoder, which shares many similarities component-wise with other architectures, but also rearranges them in an interesting order.

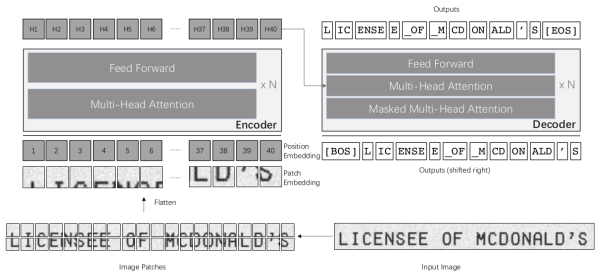

Figure 1. The architecture of the TrOCR [8] encoder-decoder model, with a pre-trained image Transformer as the encoder and a pre-trained text Transformer as the decoder.

[8] releases a pre-trained model called TrOCR, which is a transformer-based OCR model as displayed in Figure 1. The framework is a standard transformer-based encoder-decoder model, where the encoder applies attention over patches of the input that are combined with position embeddings. With this model, we are primarily interested in the fact that this model is a good candidate for fine-tuning into a capable LaTeX OCR model; indeed, the LaTeX OCR task shares many subtasks with the standard OCR task, most notably identification of characters in the input image, so we hypothesize that fine-tuning a general OCR model on synthetic LaTeX expressions will yield interesting results with less computation than training from scratch.

For data generation, we take inspiration from methods presented in [6], which is primarily focused on generating synthetic handwritten examples. We take interest in chapter 2.3 specifically, which presents LaTeX expressions as symbol layout trees, and represents a form of our approach to generating synthetic LaTeX expressions.

For the analysis of attention scores in our model, we reference [20], which presents some visual illustrations of attention that we take inspiration from when illustrating the attention our model gives to patches of input. We also reference [22], though we do not directly use it, as it provides insights like finding attention heads which order image tokens based on brightness and hue—something we find similar behavior to in our model.

3. Data

We synthesize our own dataset by generating random LaTeX expressions. To do this, we write an algorithm which is initialized with lists of expressions of various types: 'singletons' (the alphabet, numbers, and Greek letters such as α), 'one ops' (such as $\sqrt{\#}$ and trigonometric functions), 'two ops' (such as $\# + \#$, $\# \cdot \#$, $\frac{\#}{\#}$) and 'three ops' (such as integrals and summations). We then essentially build up a syntax tree as follows: we start an expression with a single $\#$. While the expression still contains $\#$ characters, we roll a random number to replace the next existing $\#$ with either a singleton, one op, two op, or three op, which themselves are randomly chosen from their respective lists. The weights given to these four possibilities are chosen such that the process terminates in finite time in expectation, and are roughly hand-tuned; we add singletons with probability 0.6, one ops with probability 0.1, two ops with probability 0.25, and three ops with probability 0.05. An illustration of this procedure as a kind of Markov chain is provided in Figure 2, where p_0, p_1, p_2, p_3 are the probabilities of generating a singleton, one op, two op, and three op respectively.

We made several opinionated design decisions in the generation of this dataset in order to tailor it closer to the types of expressions we see in the wild; we do not claim that any of these design decisions are optimal, but we expect that in aggregate, they will produce realistic-enough LaTeX expressions that we can expect good results after training a model. We added a probability distribution for the single-

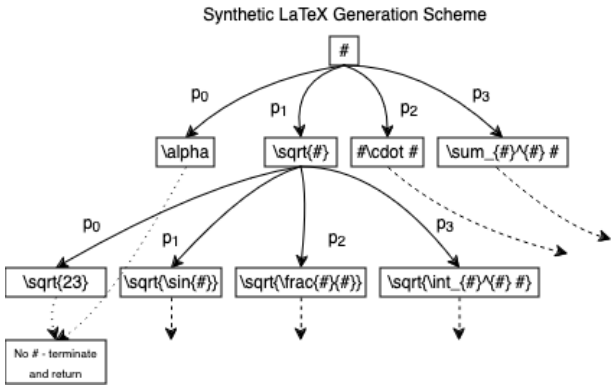

Figure 2. Illustration of synthetic LaTeX generation scheme. Dashed arrows indicate further iterations.

tons and operators in each set to allow us to hand-tune the distribution of operators selected; indeed, we would expect common operators to appear more frequently than niche and special use operators. We also added some three-ops as one-ops with common arguments filled in, like $\sum_{i=0}^{n}$, because this format is unlikely to appear frequently otherwise. We also explicitly include various lengths of expressions in the dataset to avoid the length distribution being arbitrarily derived from our generation scheme; in the current dataset of $48,000$ expressions, we enforce that each block of $8,000$ expressions is constrained to one of the ranges $\{[1,5],[6,10],[11,15],[16,20],[21,25],[26,30]\}$. We find that re-querying our generation algorithm until we find an expression of the desired length does not cause a significant slowdown—as shown in Figure 3, enough sequences of various lengths from 1 to 50 are generated that a brute-force re-querying approach is sufficient. The dataset is quick to synthesize; it takes a few seconds to generate 48,000 expressions.

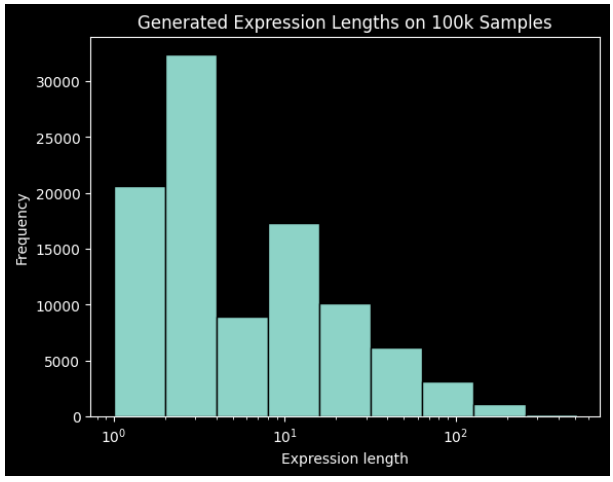

Figure 3. Distribution of expression lengths generated

In figure 4 we show several images from the $[26,30]$ bracket we generated. Notably, most of the expressions are only similar in overall form to real LaTeX expressions, and their specific contents is much more variant.

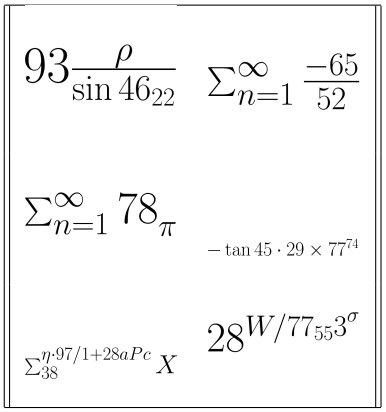

Figure 4. Assortment of compiled LaTeX expressions generated by synthetic dataset generator, lengths 26-30

In addition to this synthetic data, we use the IM2LATEX-100k dataset presented in [3] for evaluation after the model is fine-tuned.

4. Methods

We formally define our problem as converting an image of rendered LaTeX to the source LaTeX code. The input $\mathbf{x}$ in this case would be the rendered image, and the output $\mathbf{y}$ would be the sequence of tokens $y_1, y_2, \cdots, y_T$ corresponding to the LaTeX code for the image, so that $\mathbf{x} = \texttt{rendered}(\mathbf{y})$. Our dataset consists of some sample $(\mathbf{x}, \mathbf{y})$ pairs for the model to learn relationships during training, but during inference, only $\mathbf{x}$ is provided, and the goal is for the model to predict a $\hat{\mathbf{y}}$ as close to the ground truth $\mathbf{y}$ as possible.

To address this problem of converting compiled LaTeX images back into their source LaTeX code, we use a comprehensive approach involving multiple methods. Our primary strategy involves fine-tuning a pretrained OCR model on our synthesized LaTeX dataset. Then, in the post-generation stage, we utilize a method to balance brackets/delimiters that often cause compile errors. We additionally propose (though do not implement) two additional methods, one implementing a feedback mechanism for the balancing task during learning, and the other involving a language model (LLM) fine-tuned to correct errors in predicted LaTeX code. Although we do not necessarily aim to compete with the current SOTA for this task, the SOTA such as the one in [14] are suitable points of comparison.

4.1. Primary Method

The core of our approach is a pretrained OCR encoder-decoder model designed to handle the conversion of images

with text present into natural language. The encoder part of the model is a vision transformer (ViT) designed to process the input image and generate meaningful embeddings. Vision transformers have shown exceptional performance in various image processing tasks by treating an image as a sequence of patches, similar to how text sequences are handled in NLP tasks. The decoder part of the model is a text transformer that generates the LaTeX markup sequence from the vectors produced by the ViT encoder.

To adapt the vision transformer model specifically for LaTeX, we fine-tune it on our dataset composed of images and their corresponding LaTeX markup sequences. As our dataset is curated to cover a wide range of LaTeX constructs, we give the model data that should be sufficient to learn to recognize various symbols, structures, and formatting rules inherent in LaTeX formulas.

We summarize the architecture of TrOCR, the ViT-based model we are using. The model resizes an input image to a fixed size, then decomposes the image into square patches that tile to compose the full image. Each patch is flattened and projected to an embedding vector. Once each patch has an embedding, the image can be read as a sequence of embeddings with position embeddings. These embeddings are passed to a normal transformer, which maps all the embeddings to their encoded versions on the encoding step and processes them into a series of text tokens on the decoding step. The TrOCR model was initialized with DeiT [18] and BEiT [2] for the encoder (these are pre-trained transformers for image patches), and RoBERTa [9] and MiniLM [21] for the decoder (these are good quality LLMs).

4.2. Reducing Compile Errors

Compile errors in LaTeX are often caused by mispredicted tokens, especially brackets and delimiters, which are crucial for the syntactical correctness of the markup result. To mitigate this issue, we implement a hard-coded and deterministic method, and propose two additional solutions that may reduce the likelihood of compile errors in the final output:

1. The first strategy is to use an algorithm for balancing brackets in the generated LaTeX code, ensuring that each opening bracket ('{', '[', or '(') has a corresponding closing bracket ('}', ']', or ')') and that they are correctly paired. By appending missing opening brackets before unmatched closing brackets and adding necessary closing brackets at the end, it guarantees properly nested and balanced brackets, which is necessary for compiling LaTeX documents without error. Our implementation of this strategy is displayed in Algorithm 1.

2. A second strategy we propose is a feedback mechanism that can be implemented to penalize the model for generating sequences with unbalanced brackets and delimiters, ensuring better syntactic correctness in the generated LaTeX code. This mechanism introduces an additional loss term $\mathcal{L}_{\text{balance}}$ into the training objective, which penalizes the model based on the degree of imbalance in the brackets and delimiters. The total loss function is a combination of the standard cross-entropy loss $\mathcal{L}$ and the balance loss term:

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \lambda \cdot \mathcal{L}_{\text{balance}}$$

Here, λ is a hyperparameter controlling the weight of the balance loss relative to the standard loss. The balance loss term $\mathcal{L}_{\text{balance}}$ penalizes unbalanced states proportional to the size of the bracket stack at each position in the sequence. This encourages the model to generate LaTeX code with properly balanced brackets and delimiters.

3. A third strategy we propose is to use a small LLM fine-tuned to correct errors in the predicted sequences. This approach leverages an LLM's capability to understand and generate grammatically and syntactically correct sequences. We would first create a synthetic dataset by introducing common LaTeX errors into correct se-

Algorithm 1 Balance Brackets in LaTeX Code

```
function BALANCE_BRACKETS(latex_code)
    bracket_stack ← []
    bracket_pairs ← {'{': '}', '[': ']', '(': ')'}
    new_code ← []
    for char in latex_code do
        if char in bracket_pairs then
            bracket_stack.append(char)
        else if char in bracket_pairs.values() then
            if bracket_stack and bracket_pairs[bracket_stack[-1]] == char then
                bracket_stack.pop()
            else
                for opening, closing in bracket_pairs.items() do
                    if closing == char then
                        new_code.append(opening)
                        bracket_stack.append(opening)
                        break
                    end if
                end for
            end if
        end if
        new_code.append(char)
    end for
    while bracket_stack do
        new_code.append(bracket_pairs[bracket_stack.pop()])
    end while
    return ''.join(new_code)
end function
```

quences. The LLM is then trained on the synthetic dataset with an objective to transform erroneous LaTeX sequences into their correct forms. The training objective remains the cross-entropy loss, applied to the token sequences.

$$\mathcal{L}_{\text{LLM}} = -\sum_{t=1}^{T} \log P(\hat{y}_t^{\text{corrected}} | \hat{y}_t^{\text{error}})$$

By fine-tuning the LLM in this manner, we would aim to enhance the robustness of the overall system, ensuring that even if the initial predictions contain errors, they can be effectively corrected for the final output.

4.3. Baseline Method

We use the the results in [14] as a baseline for our results, with the caveat that we do not necessarily expect to exceed these results in raw performance. Instead, we aim to investigate how our proposed methods, which include using an existing OCR model and a synthetic dataset along with some tricks to improve output quality, compare in performance to baseline models trained specifically for LaTeX OCR using real data, which requires much more computation and effort to generate training data.

To compare our proposed methods against the existing results, we will use the CER (Character Error Rate) and BLEU (Bilingual Evaluation Understudy) metrics. These metrics provide a quantitative measure of the accuracy and quality of the generated LaTeX code:

1. CER measures the edit distance (insertions, deletions, substitutions) between the generated and reference LaTeX sequences, normalized by the length of the reference sequence. A lower CER indicates fewer errors and higher accuracy.

$$\text{CER} = \frac{\text{Number of Character Errors}}{\text{Total Number of Characters in Reference}}$$

2. The BLEU score is a precision-based metric that evaluates the similarity between the generated LaTeX code and the reference (ground-truth) LaTeX code. It considers n-gram matches between the generated and reference sequences, with a higher BLEU score indicating better alignment with the ground-truth.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

where BP is the brevity penalty, w_n are the weights for different n-grams, and p_n are the precision scores for n-grams of length n. In particular, we utilize The Natural Language Toolkit (NLTK) implementation [10] of BLEU score in our evaluations.

To establish a baseline for comparison, we use results listed in [14]:

Model	BLEU
INFTY [17]	66.65
WYGIWYS [4]	87.73
Coarse-to-Fine Attention [3]	87.07
Double Attention [23]	88.42
Global Context [14]	89.72

5. Experiments

We started by generating a small dataset of 1000 training examples to ensure that our fine-tuning pipeline was functioning properly. After this validation, we ran the fine-tuning operation on a full set of $48,000$ images (split into 80% training images and 20% testing images using Pandas DataFrame [13]). We used Pytorch [15] to run our experiments, which involved loading the data, loading the model (as per the code provided by [8]), and used the AdamW optimizer [11].

Our fine-tuning runs consisted roughly of the following trials:

1. Verification run to ensure model is set up correctly; $1,000$ training examples.
2. Several different short runs at various learning rates; we find here that $5\text{e}{-5}$ is optimal for our purposes.
3. Several long runs at $5\text{e}{-5}$ learning rate; we experienced issues with GCP SSH pipe breaks, which delayed the project significantly. After several trials we completed a run, which took about 10 hours.
4. Low temperature run at $T = 0.01$ (previous default 1); we ran into numerical instability issues at this temperature.
5. Low temperature run at $T = 0.1$. This run completed without issue; we ended at a loss of 2.27.

Our justification for reducing the temperature is due to the fact that the LaTeX generation process should, in theory, be close to deterministic (in practice there are slight differences with the positions and necessities of brackets). We achieve a BLEU score of 28.78 on our own dataset and a CER score of 0.515 on our own dataset, both fairly low (note that these results are not on the IM2LATEX-100k dataset):

Model	BLEU	CER
Fine-tuned TrOCR	28.78	0.515

Selected generations are shown in Figure 5. We notice informally that some of these results give the impression

15/{\mu}/{\phi}	15///{\//{{\\\{\}}}/{\}}
\cos{l}\times 36	\cos
-{92}^{\sqrt{U}}	-92{\{\{\{\}}}}{}}
W	W
\sec{{\upsilon}}	\sec{{\\\upsilon}}{}}{}}{}}{}}{}}
\csc{{\omega}}-79	\c{{\\\ome}}
5823	5
\frac{33}{{\pi}}	\frac{{\pi}}{}{}}
{\phi}-a\cdot c	{\phi}{}}i{}}
{Y+99}{{49}^{e}}	Y+++{++{49}}{}}{}}{}_{49}}{}}{}}
94	94
\frac{{\alpha}}{h}	\frac{{n}}
90	{90}{}}
\sqrt{62}	\sqrt{{{}}}
\sin{\cot{20}}	\sin{{{{{20}}}}}
\tan{\int_{0}^{1}B}	\tan{{{\int_{0BBB}BB}}}
{{\iota}}{93}	{{\iotaotaota}}otaota{93}{}}{}}{}}
{\eta}/{65/95Z81}{53}1f	{\eta}{}}Z{}}/{}}{65}{}f{}}
53	53

Figure 5. Assortment of generations produced by the model, after a deterministic bracket balancing step. Ground truth is on the left and generations are on the right.

of an LLM which has been paused in the early stages of training, as text generation transformers often learn basic character patterns and high level grammar first before learning lower level concepts like semantics and producing sentences that make sense. The model we are using, TrOCR, is rather large at 334M parameters, so we do not believe this level of performance is entirely due to architectural restrictions.

In Figure 6, we can see a very interesting visualization of the attention operation over the patches of the image. Recall that, to process the image, it is converted into a set of fixed size patches and each patch is converted to an embedding. These embeddings are then processed by the encoder, which applies attention. Since there is a 1 to 1 correspondence between the patches and the embeddings, we can compress the attention scores over the course of the attention operation into a vector of patch attentions, which roughly give the amount of importance each patch of the image contributed to the computation with respect to the attention operation.

From Figure 6, we can extract some very important insights. For one, the attention scores are consistently finding parts of the structure of the underlying image—so much so that in some images, it is possible to see the underlying expression in the heatmap alone. For two, the attention is consistently applied with bias towards the left side of the image, especially for more complex expressions. This is extremely important for interpreting the results in Figure 5, as we can see a pattern of the model attaining high accuracy on the first one or several tokens, but low accuracy on all others.

Figure 6. Visualization of attention given to each patch in the input image

We note several reasons we believe are likely or potential causes of the poor performance.

1. Our fine-training pipeline may be suboptimal for this model and task. Although the model saw all datapoints in the dataset during the fine-tuning process, it only saw datapoints in batches of size 4, so the gradient descent operation was likely to be noisy. Unfortunately, we could not scale above size 4 batches due to time constraints, as the model took about 10 hours to train one epoch (covering the full training set) at this batch size. Training was completed using a T4 GPU on GCP. Factoring in failed training runs and experimental runs, these factors approached the upper bound of our time and compute resources. We believe that a model trained for more passes on the dataset in larger batch sizes has significant potential, as our model began to learn the semantics of LaTeX.

   To further expand on this, our final loss was around 2.27, which is equivalent to randomly guessing from a set of 190 or so tokens. Given that the set of tokens used to represent the training data is likely much

smaller than the $31,959$ tokens supported by TrOCR, we hypothesize that the model initially learned only basic character frequencies of the fine-tune dataset, and training was paused before proper LaTeX semantics could be learned. This reasoning is further compounded by the fact that the model learned to attend to the first part of the expressions it saw, which correspond with the earliest tokens in each generation.

2. The token set for the TrOCR model is likely too large for the LaTeX prediction task. Indeed, the types and distributions of tokens which appear in LaTeX are very different from those which appear in standard OCR tasks, and OCR tasks make use of many longer tokens of alphabetic characters which would rarely if ever appear in a LaTeX dataset. A refinement to our approach, which we at one point considered, would be to run an algorithm like byte pair encoding [16] on the LaTeX dataset, then prune the TrOCR model's tokens to just those which have the potential to appear. This has the extensions of explicitly adding full-size LaTeX command tokens like $\backslash alpha$ to the token set. This smaller model can improve training speed.

3. The TrOCR model does not use CNN filters for feature extraction. We theorize that this is less detrimental for an OCR model than it is for a LaTeX-reading model. This is because an OCR model generally must read line-by-line for a given image, and so it is often that the patch creation and position embedding process will distort the underlying text very little. However, LaTeX images do not follow this standard line-by-line structure, as expressions like a fraction, integral, and subscripts or superscripts will break the pattern. We theorize that a model which takes advantage of CNN filters like [14] has an advantage in learning the structure of LaTeX, as our fine-tuned TrOCR model often gives the impression of a model that can identify characters but not order them.

4. In conjunction with the previous item, the TrOCR model may simply be too underspecialized for the LaTeX task at an architectural level. Indeed, [14] uses a very specific and hand-tuned architecture for the task, which includes a custom "Global Context" block to extract features (as well as CNNs). It may be the case that the network machinery required to learn LaTeX semantics is not present in this relatively basic OCR model.

5. Our dataset may be too unusual in form due to its randomness. Many other LaTeX-reading models train directly on the IM2LATEX-100k dataset, which is extracted from real-world expressions, so there may be regularity in these expressions which our model is never exposed to. We believe the issues in performance are more likely attributable to architecture, however, as the IM2LATEX-100k dataset also contains much longer and more complex expressions, so the regularity in these expressions from being human-generated is unlikely to provide a great advantage.

6. The characters which are most often recognized by OCR models are very different from those read generated by a LaTeX reader. Furthermore, the LaTeX model must generate many symbols which do not appear in the original image; with respect to these differences, the weights of the pre-trained model may actually prove to be a hindrance, as in the second case the model must unlearn a 1:1 mapping between characters in the image and characters to print.

Although we proposed two additional methods for reducing LaTeX compile errors and are confident that they would be highly effective as part of an already effective model, we found that our own fine-tuned OCR model did not achieve results with sufficiently low noise that this method was likely to be successful; many of the results our model produced did not have sufficient information to recover the original expression, especially for longer expressions. For further experiments, and especially for models with high quality baseline results, we believe that these strategies can be used to refine the model accuracy.

6. Conclusion

We find that the TrOCR vision transformer is ill-suited to learn to solve the problem of generating LaTeX code from a compiled image. Despite the existing ability of the model to recognize characters in images, the model is unable to learn the ability to recognize the structure of LaTeX at a sufficiently noise-less level. Even when restricted to our synthetically generated dataset, the model fails to achieve a BLEU score over 30, which is not approaching the score of existing models on the much more difficult IM2LATEX-100K dataset. However, we were able to visualize the way in which the model applied attention to the input image, which provided valuable insight into the ability of the model to generate accurate outputs.

We proposed a set of reasons for this suboptimal performance. To summarize, these are (1) an undertrained model, due to time and compute restrictions, (2) an abundance of tokens which will never be used in a LaTeX expression, burdening the model with too many tokens, (3) lack of CNNs for feature extraction, (4) architecture underspecialization for the LaTeX task, (5) issues with the dataset distribution, and (6) differences in distribution between the tokens which OCR models read and those which LaTeX models read, putting the usefulness of the pretrained model into question.

Although one of the ultimate goals of this research was to show that fine-tuning an OCR model is competitive or better resource-wise than training a handmade model from scratch due to the computational savings of using a pre-trained checkpoint, we are unable to make this comparison without a stronger model.

We believe that there is significant reason to expect the model to improve if given more compute, so the application of more compute to this model presents a significant opportunity for future work. We also see opportunities for further experiments and improvement in the previously noted points (2), (3), (5). The use of an OCR model with a lower parameter count for faster training may also help, as this would allow us to determine the actual ceiling of performance when training is run to completion rather than limited by compute resources.

7. Contributions & Acknowledgements

We would like to acknowledge the TrOCR paper [8] for being instrumental in this research. We would also like to acknowledge the CS 231N course staff and GCP for providing cloud computing resources that were vital to the project.

The two team members contributed equally to the final project. Nikash Chhadia contributed the fine-tuning of the TrOCR model, which involved loading and peparing the model as well as selecting hyperparameters over various trials, which were run on GCP over several days. Sambhav Gupta contributed the custom LaTeX generation code and generated the dataset used for this work. He also generated the visualizations of attention hotspots in the final model.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] H. Bao, L. Dong, and F. Wei. Beit: BERT pre-training of image transformers. *CoRR*, abs/2106.08254, 2021.

[3] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush. Image-to-markup generation with coarse-to-fine attention. Sept. 2016.

[4] Y. Deng, A. Kanervisto, and A. M. Rush. What you get is what you see: A visual markup decompiler. *ArXiv*, abs/1609.04938, 2016.

[5] H. Ding, K. Chen, and Q. Huo. An encoder-decoder approach to handwritten mathematical expression recognition with multi-head attention and stacked decoder. In J. Lladós, D. Lopresti, and S. Uchida, editors, *Document Analysis and Recognition – ICDAR 2021*, pages 602–616, Cham, 2021. Springer International Publishing.

[6] V. Heska. Generating synthetic online handwritten mathematical expressions from markup languages, 2021.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] M. Li, T. Lv, J. Chen, L. Cui, Y. Lu, D. Florencio, C. Zhang, Z. Li, and F. Wei. Trocr: Transformer-based optical character recognition with pre-trained models, 2022.

[9] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[10] E. Loper and S. Bird. Nltk: The natural language toolkit, 2002.

[11] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[12] A. F. T. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification, 2016.

[13] W. McKinney et al. Data structures for statistical computing in python. In *SciPy*, volume 445, pages 51–56, 2010.

[14] N. Pang, C. Yang, X. Zhu, J. Li, and X.-C. Yin. Global context-based network with transformer for image2latex. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4650–4656, 2021.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[16] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units, 2016.

[17] M. Suzuki, F. Tamari, R. Fukuda, S. Uchida, and T. Kanahori. Infty: an integrated ocr system for mathematical documents. In *Proceedings of the 2003 ACM Symposium on Document Engineering*, DocEng '03, page 95–104, New York, NY, USA, 2003. Association for Computing Machinery.

[18] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou. Training data-efficient image transformers amp; distillation through attention. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR, 18–24 Jul 2021.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[20] J. Vig. Visualizing attention in transformer-based language representation models, 2019.

[21] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020.

[22] C. Yeh, Y. Chen, A. Wu, C. Chen, F. Viégas, and M. Wattenberg. Attentionviz: A global view of transformer attention, 2023.

[23] W. Zhang, Z. Bai, and Y. Zhu. 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*, ICMSSP '19, page 57–61, New York, NY, USA, 2019. Association for Computing Machinery.