Detypstify: OCR for formula generation

Justin Restivo
Yale
justin.restivo@yale.edu

Jachym Putta
Yale
jachym.putta@yale.edu

Abstract

Optical Character Recognition (OCR) has seen widespread use in the past few years. It has been used for data entry automation, document management, and even in the medical field for digitizing medical records. OCR makes the tasks both faster and less error prone. In this paper, we present Detypstify, a tool which uses state-of-the-art OCR to generate math formulas from images. The problem of generating LaTeX formulas from images is not new, in fact it forms the basis of the OpenAI Im2Latex problem which was posted as part of the first request for research [1]. Detypstify tackles this problem in a new way by using a transformer based model. We deploy this model using Web Assembly and WGPU to allow for client side computation.

1 Introduction

In recent years, Optical Character Recognition (OCR) technology has revolutionized various domains by streamlining tasks, reducing errors, and enhancing efficiency. From automating data entry to facilitating document management, OCR has proven indispensable across diverse industries. Moreover, its applications extend into the realm of healthcare, where it aids in the digitization of medical records, contributing to the modernization of healthcare systems. Amidst this landscape, the demand for innovative OCR solutions continues to grow.

This paper introduces Detypstify, a novel tool designed to address the challenge of generating mathematical formulas from images using state-of-the-art OCR technology. While the task of converting images to LaTeX formulas is not new, Detypstify distinguishes itself through its utilization of a transformer-based model. This approach marks a departure from traditional methods and builds upon the foundation laid by initiatives such as the OpenAI Im2Latex problem, which has stimulated research in this domain.

Detypstify leverages Web Assembly and WGPU for deployment, enabling client-side computation and enhancing accessibility. By harnessing the power of these technologies, Detypstify not only offers a sophisticated solution but also ensures seamless integration into existing workflows. This paper provides an overview of the design, implementation, and performance of Detypstify, underscoring its potential to advance the field of OCR and mathematical representation.

2 Background and Related Work

2.1 Model

Fine tuning of large models has proven to be an effective way of achieving state-of-the-art performance on a wide range of tasks [2]. This performance is often better than smaller models dedicated to this particular task [3]. As a result, we decided to fine-tune a large model for our task as well.

Spring 2024 Applied ML Final Project (CPCS 582).

Text recognition is usually done either with Convolutional Neural Networks (CNN) or with Recurrent Neural Networks (RNN). We decided to diverge from the conventional approach and use a transformer based model. TrOCR [4], transformer based optical character recognition, is a model which outperforms the state-of-the-art in OCR tasks, both handwritten and printed. As such we decided to use this model as our base model for fine tuning.

2.2 Other formula generation tools

While there are several other tools which implement the same functionality as Detypstify, Detyptify has several features which distinguish it from the competition.

1. **Support for Typst**, there are no tools which generate Typst formulas from images.
2. **Web Assembly**, Detypstify is deployed using Web Assembly which allows for deployment as a static website without a backend. All computation is performed by the client which we haven't seen in other tools.
3. **Transformer based OCR**, most tools use CNN or RNN based OCR, Detypstify uses a transformer based OCR.

2.3 Method description

2.4 Model

2.4.1 Dataset

At the time of writing, there were no available datasets of images of formulas and their corresponding Typst encoding. We created our own dataset by converting the formulas from the Im2Latex dataset [5] which has been widely used for similar tasks when converting to LaTeX. The bulk of the conversion was done using Pandoc, however, the conversion was not perfect and required several correction passes to ensure valid Typst formulas. This was because, to our knowledge, none of the datasets were generated post-Latex2e which led to several incompatibilities with Pandoc. The final dataset is available on Kaggle [6].

2.4.2 Training

We fine tuned the TrOCR model on our dataset using native PyTorch with the VisionEncoderDecorerModel class. The model was trained on a single Nvidia GeForce RTX 4090 GPU for 7 epochs with a batch size of 1 because of VRAM memory constraints. We could not fit the model and more than one image into VRAM at a time. To evalutate the model we use the Character Error Rate (CER) which is the number of incorrect characters divided by the total number of characters. This turned out to be a poor metric for our task.

Our second attempt at training a model was using the Vision Transformer (ViT) model from scratch [7]. We chose this implementation as it performed well for generating LaTeX code. The ViT model was trained on the same system as the TrOCR model with the same batch size, but since we are training from scratch we train for 25 epochs.

2.5 Webapp

2.6 ONNX

ONNX is a standardized format for representing machine learning models. We export our trained model to ONNX so that we can integrate it with the rest of the application.

2.7 Burn

Burn is a machine learning framework written in Rust, compatible with the ONNX model format. Burn is in active development and some of the necessary operations of the ONNX framework were not supported (`expand`, `slice`, `squeeze`, `range`, `less`, `constantofshape`). We confirmed with the developers of Burn that these operations were not supported. The developers are working to add support. Once these operations are supported, integration is a simple include.

2.8 Wasm + WGPU

One of the main draws of using Web Assembly is the ability to run machine learning models in the browser on the client side. This means that we are not required to host a backend and can simple host the web assembly binary. The client will then download the binary and run the model locally. To compliment this portability, we use WGPU, a Rust library for interfacing with GPUs. This allows the user to run the model on any hardware that has a GPU (for example, a phone or laptop) without having to install any libraries.

3 Results & Discussion

3.1 TrOCR Fine Tuning

Although the fine-tuning took a week and a half, the results were quite disappointing. The model was unable to predict even single characters like α. This is likely due to the Character Error Rate (CER) which works very well on handwriting and extracting the formulas themselves. Since we are mapping images to code and not text, we do not have a bijection between the input and the output. As a result the model doesn't perform well. Armed with these insights, we decited to try training a model from scratch on the Typst dataset

3.2 ViT Trainig

After training the model for 25 epochs, which corresponds to around 48 GPU hours, the accuracy plateaued at 2% which is nowhere near the level required to make the tool usable. This is likely due to our lack of experience with the ViT model which led to poor hyperparameter choices. We are reaching out the authors of LaTeX OCR to see if they can provide us with the hyperparameters they used to train their model and try again.

4 Conclusion

We present Detypstify, a tool that uses OCR for formula generation. We fine tune a transformer based large model for this task and deploy it statically using Web Assembly and WGPU. Our results are not as good as we hoped, but with the help of the authors of LaTeX OCR we hope to improve the performance and make the tool usable. The deployment framework is ready, but with no model to deploy it is not very useful. After we improve the model we will deploy it and make it available to the public. We hope that this tool will eventually be useful for the Typst community.

References

[1] OpenAI, “Requests for Research.” [Online]. Available: https://github.com/openai/requests-for-research/blob/master/_requests_for_research/im2latex.html

[2] Z. Han, C. Gao, J. Liu, J. Zhang, and S. Q. Zhang, “Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey.” 2024.

[3] J. Howard and S. Ruder, “Universal Language Model Fine-tuning for Text Classification.” 2018.

[4] M. Li *et al.*, “TrOCR: Transformer-Based Optical Character Recognition with Pre-trained Models,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 11, pp. 13094–13102, Jun. 2023, doi: 10.1609/aaai.v37i11.26538.

[5] “IM2LATEX-230k.”

[6] “im2typst-230k.”

[7] L. Blecher, “LaTeX OCR.” [Online]. Available: https://github.com/lukas-blecher/LaTeX-OCR/tree/main

[8] J. Alexander and M. C. Mozer, “Template-based algorithms for connectionist rule extraction,” *Advances in neural information processing systems*, vol. 7, 1994.

[9] J. M. Bower and D. Beeman, *The book of GENESIS: exploring realistic neural models with the GEneral NEural SImulation System*. Springer Science & Business Media, 2012.

[10] M. E. Hasselmo, E. Schnell, and E. Barkai, “Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3,” *Journal of Neuroscience*, vol. 15, no. 7, pp. 5249–5262, 1995.