Recursive CNNs for ImageToLatex problem

Michał Tyrolski, Szymon Tworkowski

Student @ Faculty of Mathematics, Informatics and Mechanics - University of Warsaw

Abstract

While recurrent neural networks became the most accurate solution for the ImageToLatex problem, they are used along with lots of tricky architecture improvements. Commonly adopted approach is to connect CNN encoders and RNN decoders with attention mechanisms attached. We show a more human-like solution based on recursive splitting the expression. According to the fact that we propose a divide-and-conquer algorithm, we also prepared a dataset with unambiguous expression splits, representing some LaTeX subset. We compared the state-of-the-art model vs our structure model combined with that SOTA used for leaf recognition. Finally, we show that our approach performed better on our dataset with a particular number of splits.

1. Introduction

ImageToLatex is quite an old problem. Before the deep learning era, it was commonly approached with different ideas related to optical character recognition (OCR) methods [1]. Recently using encoder + decoder architectures with various modifications became the most popular approach to produce latex expressions from image. In such models, encoder often has convolutional neural networks (CNN) as a backbone for producing feature maps [2,3,4], sometimes using custom modifications like row-encoding [2]. For decoding, there are usually recurrent neural networks (RNN) [2,3,4].

In this work we modify the parsing expression idea presented by Zanibbi et al. [5]. Instead of OCR methods, we use CNNs to predict bounding boxes for operators and operands. Recursively repeating CNN calls gives us knowledge about expression tree structure with bounding boxes for each subtree. Then any model can be used to recognize leaves, especially those with ability to properly recognize expressions with length bounded by a constant C for example like stacked CNNs [6].

2. Model

2.1 Structural Model $M_{\mathcal{S}}$

The model responsible for building the tree structure is a CNN with ResNext18 [10] as a backbone with a small custom head. Bounding boxes are predicted along with categories simultaneously. Category is represented as 1-hot encoding and each bounding box (x_1, y_1, x_2, y_2) as

$$(x_1, y_1, \sqrt{x_2 - x_1}, \sqrt{y_2 - y_1})$$

Similarly to Redmon et al. [9], we define loss function $\mathcal{L}$ for a batch as follows:

$$\mathcal{L}(\{B_i\}, \{C_i\}) = \frac{1}{N} \sum_i (\mathcal{L}_1(C_i, C_i^*) + \lambda \mathcal{H}(C_i^*) \mathcal{L}_2(B_i, B_i^*))$$

where $\mathcal{L}_1 = \text{LOGLOSS}$, $\mathcal{L}_2 = \text{MSE}$ and $(B_i, B_i^*), (C_i, C_i^*)$ are predicted, ground-truth values for boxes and categories respectively, $\mathcal{H}(C) = \mathbb{1}_{C \neq \text{leaf}}$. The purpose of λ is to increase the effect of box loss and it is empirically chosen. In our case $\lambda = 50$. Accuracy of structural model is measured as pair $\mathcal{A} = (A_c, I)$ where A_c is a standard accuracy and $I = \text{IOU}$. On our dataset we achieved $\mathcal{A} = (99.9\%, 95.4\%)$ validation accuracy for the first split.

$M_{\mathcal{S}}$ proceeds with recursive calls to CNN (Figure 1) as long as it doesn't detect a leaf or achieve a hyperparam depth limit. Moreover, training is done only on the first split for each entry from the dataset.

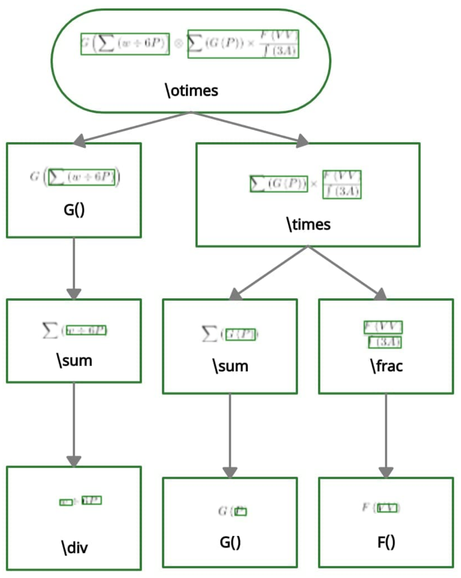

Figure 1: *Sample expression tree built with recursive calls to CNN. Own research.*

During inference, each node which isn't recognized as a leaf is cropped with its bounding box prediction, then centered and padded in order to match the input shape. Then the next CNN call is taken. We do not resize the cropped images, instead we decided to apply a shrinking augmentation in training.

2.2 Leaf Model $M_{\mathcal{L}}$

As a recognizer for leaves and baseline to compare we chose Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, Alexander M. Rush 2016: Image-to-Markup Generation with Coarse-to-Fine Attention [2]. We trained this model for 15 epochs on a full dataset. Expression depth d is defined as the number of nodes in the longest path from root to leaves. If $M_{\mathcal{S}}$ stops before reaching d, $M_{\mathcal{L}}$ is used as a recognizer for the whole subtree.

2.3 Dataset

Dataset is generated from scratch. It contains 100008 images with shape $(224, 224, 3)$. We support 17 operators, 8 of them unary, another 8 binary and 1 leaf. We decided to create our own dataset because $M_{\mathcal{S}}$ need bounding boxes coordinates. Each entry in the dataset consists of the following information: name, operator name, sorted operands bounding boxes in form of (x, y, dx, dy), normalized label and depth. Max supported depth is 4, deeper equations were unreadable in that resolution.

$$\frac{(Og \div I)}{j \times q\Gamma} \div \sqrt{\sum \left(\sum (V\Delta) \right)}$$

Figure 2: *Sample expression of depth 4 along with its labeled version. Own research.*

In order to generate a dataset, we had to create a $M_{\mathcal{S}}$ loss function to take into account various possible splits along with ensuring the convergence of the model or find an unambiguous way of splitting the expressions. We went with the second approach. The split is done on the left most inline operator if the operator on top is binary, otherwise the argument of unary operator is taken.

Let $E(d)$ be a set of all supported expression with depth $\leq d$ and $\text{In} : \mathbb{N}_+ \to P(\text{Expr})$ such that

$$\begin{aligned} \text{In}(d) = (&\{e_1 \triangle e_2 | e_1 \text{ is not an inline expression}\} \\ &\cup \{\Delta(e) | \Delta \text{ is an unary operator}\} \\ &\cup \{\Delta(e_1, e_2) | \Delta \text{ is not an inline operator }\} \\ &\cup \{e | e \text{ is a leaf}\}) \cap \bigcup_{k \leq d} E(k) \end{aligned}$$

In our case inline operators $\in \{+, \times, \star, \div, \otimes, -, \cdot\}$ and an expression is inline if and only if the top operator is inline. Also, for our purpose $\bigcup_d \text{In}(E(d))$ effectively approximates $\bigcup_d E(d)$. That observation allowed us to generate a dataset without need to parse latex expressions. Instead of sampling from E and parsing them, we generated colorful In and deterministically detected boxes as shown in Figure 2. As part of augmentation, we decreased the quality of a small sample of dataset depending on sample depths in order to simulate an inference process.

3. Results and discussions

3.1 Comparison

We compared $M_{\mathcal{S}}$ with $M_{\mathcal{L}}$ for subtree recognizing versus $M_{\mathcal{L}}$ for whole sentence recognizing. Results are shown in Table 1.

Table 1: *Comparison between $M_{\mathcal{S}}$+$M_{\mathcal{L}}$ and $M_{\mathcal{L}}$. Own research.*

Depth limit	BLEU	EDIT
0 Baseline	81.1%	78.1%
1 $M_{\mathcal{S}}$+$M_{\mathcal{L}}$	$\mathbf{89.1\%}$	$\mathbf{85.4\%}$
2 $M_{\mathcal{S}}$+$M_{\mathcal{L}}$	$\mathbf{87.2\%}$	$\mathbf{82.4\%}$
3 $M_{\mathcal{S}}$+$M_{\mathcal{L}}$	$\mathbf{83.3\%}$	75.6%
4 $M_{\mathcal{S}}$+$M_{\mathcal{L}}$	80.3%	68.2%

As shown in table, $M_{\mathcal{S}}$+$M_{\mathcal{L}}$ achieves much better results for 1 and 2 depth steps and better BLEU metric for $d = 3$. As can be expected, for more steps baseline still achieves better scores. We identify two main reasons for decreasing dominance of $M_{\mathcal{S}}$+$M_{\mathcal{L}}$ over $M_{\mathcal{L}}$ standalone. Firstly, while $M_{\mathcal{S}}$ first bounding box detection is almost ideal, error propagates and at a certain depth model is not able to recognize proper boxes. Moreover, after each $M_{\mathcal{S}}$ inference, we rescale box expression which degrades the quality of next input images. Our augmentation mentioned in the dataset section improves $M_{\mathcal{S}}$ but after few rescaling expressions become unreadable, even for humans. On the other hand, many im2seq models have problems recognizing the structure of long expressions, especially without much training. It can be concluded that our approach has potential to improve accuracy using a small number of splits for sota models on real itl100k dataset.

3.2 Future work

That approach is able to improve many models on image to sequence problems. In order to compete on a real dataset, the current one should be extended for more symbols, more unary and binary operators along with operators with much more operands. Also upper and lower indexes and complex structures like matrices should be supported. It should be considered if it is worth detecting forest structure instead of trees only, especially if the target is to generalize. To make models useful on a bigger scale, the architecture should accept bigger image shapes, custom too. Also, in terms of longer sequences, there is a need for the loss and the dataset creation which prefer the middle operator.

4. Implementation

To ensure the reproducibility of this work and to support open science principles, we made the code and dataset available on https://github.com/kakainet/TexNet and https://github.com/kakainet/TexSet. Details are provided in the readme. Code is written in fast.ai [7] and PyTorch [8]. We use ready implementation of model proposed by Deng, Y. et al. [2], with small modifications related to data augmentation as a backbone and train on our dataset.

5. References

1. Suzuki, M., Kanahori, T., Ohtake, N. and Yamaguchi, K., 2004, July. An integrated OCR software for mathematical documents and its output with accessibility. In International Conference on Computers for Handicapped Persons (pp. 648-655). Springer, Berlin, Heidelberg.

2. Deng, Y., Kanervisto, A., Ling, J. and Rush, A.M., 2017, July. Image-to-markup generation with coarse-to-fine attention. In International Conference on Machine Learning (pp. 980-989).

3. Wang, J., Sun, Y. and Wang, S., 2019. Image to latex with densenet encoder and joint attention. Procedia computer science, 147, pp.374-380.

4. Genthial, G. and Sauvestre, R., 2016. Image to Latex.

5. Zanibbi, R., Blostein, D. and Cordy, J.R., 2002. Recognizing mathematical expressions using tree transformation. IEEE Transactions on pattern analysis and machine intelligence, 24(11), pp.1455-1467.

6. More, A., 2018. IMAGE TO LATEX VIA NEURAL NETWORKS.

7. Howard, J. and Gugger, S., 2020. Fastai: A layered API for deep learning. Information, 11(2), p.108.

8. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. and Desmaison, A., 2019. Pytorch: An imperative style, high-performance deep learning library. In Advances in neural information processing systems (pp. 8026-8037).

9. Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788)..

10. Xie, S., Girshick, R., Dollár, P., Tu, Z. and He, K., 2017. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1492-1500).