Image to Latex

Guillaume Genthial
Stanford University
ICME
genthial@stanford.edu

Romain Sauvestre
Stanford University
MS&E
romains@stanford.edu

Abstract

Converting images of mathematical formulas to LaTeX code is a problem that combines challenges both from computer vision and natural language processing, close to the recent breakthroughs in image captioning. Such a system would be very useful for people using LaTeX in their everyday life such as students or researchers. We addressed this task with a dual setting encoder-decoder, combined with an attention mechanism. This model is trained on a $100,000$ *LaTeX formulas scraped from ArXiv. We evaluate our model against several metrics, both at the code level (perplexity, exact match, BLEU score ...) and at the image level (Levenshtein distance, exact match) by compiling the predicted code to generate new formulas and comparing them to the ground truth. We achieve very good performance with a BLEU score of* 78% *and an image edit-distance of* 62%.

1. Introduction

Recently, 20,000+ handwritten notes of Alexander Grothendiek, one of the greatest mathematician of the 20th century, were scanned and released by the University of Montpellier, making it available to every researcher in the world as it is believed it could advance research in many fields of mathematics. However, reading these notes is not only a challenge due to their very high theoretical complexity, but also because they are not written in a proper format. We can imagine that they would get more attention and research interest if they were nicely computerized in a clean LaTeX document, which would incentivize more researchers to peak into the work of one of the greatest geniuses of the past century. However, it would take years for a human to type all these notes on LaTeX, the most challenging part being to understand and type all the mathematical equations.

Recent breakthroughs in computer vision and Natural Language Processing (NLP) could address this challenge. Computers are already able to achieve almost perfect accuracy to understand hand written text and render it from images. Hence, the next step towards rendering hand-written mathematical notes into a LaTeX document is to teach computers to reconstruct the LaTeX code that generates a given formula. This task differ from standard Optical Character Recognition (OCR) techniques by the complexity of the underlying syntax and is closer to image captioning than standard OCR. To this purpose, we can leverage image captioning techniques to address this task. Our system would take as input a formula (the *image*) and would generate a list of LaTeX tokens (the *caption*).

Image captioning is a surging field of deep learning that has gotten more and more attention in the past few years, being at the crossroads between computer vision and NLP. This is one of the main attempts to combine these two fields of machine learning, to achieve more general artificial intelligence with machines being able to both see and speak at the same time. The current state of the art in image captioning has a similar approach as sequence to sequence models described by [Sutskever et al., 2014]: first we encode the input image in a fixed size vector and then decode this vector by generating tokens from the caption one after the other. Adding an attention mechanism was also proved to greatly enhance the performance of these models [Luong et al., 2015] (cf Figure 1).

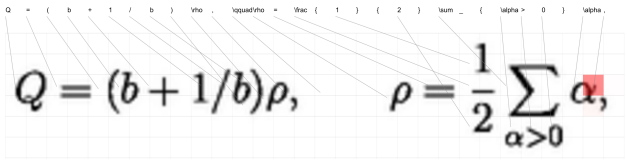

Figure 1. Generating LaTeXfrom images, figure from [Deng et al., 2016]

Little attention has been given to its application to reconstructing LaTeX code from mathematical formulas. OpenAi listed this problem in their blog as a request for research (https://openai.com/

requests-for-research/#im2latex) and earlier last year, a team from Harvard University [Deng et al., 2016] proposed an attention-based model to address this issue. Note that due to the fact that different formulas can produce similar or same images add a complexity to the task, compared to standard image captioning. It also raises issues about the evaluation of such models.

In this project, we combine the approach of [Deng et al., 2016] and the image captioning system of [Xu et al., 2015] and build on their models to improve the results. Our main contributions are:

- a fully end-to-end LaTeX rendering system that could also be applied to image captioning, and more globally to any image-to-text problem.
- a comparison between different encoders and decoders as well as an analysis of the performance of an image-captioning model on this particular problem.

2. Related Work

In the past few years, lots of breakthroughs have taken place in Computer Vision, originated by the performance of [Lecun et al., 1998]'s system to recognize digits. Optical Character Recognition has since gained interest, with highly-accurate systems like the one of [Ciresan et al., 2010]. On the task of recognizing mathematical expressions that introduces new challenges of grammar understanding, some attempts have managed to build systems by exploiting character segmentation and grammar reconstruction, like [Miller and Viola, 1998], or other syntactic-based approaches like [Chan and Yeung, 2000] and [Belaid and Haton, 1984]. Techniques combining neural network and standard NLP approaches like Conditional Random Field have also proven to be very effective to recognize words in images like in [Jaderberg et al., 2014], or using convolutional neural networks like in [Wang et al., 2012].

In Natural Language Processing, sequence-to-sequence models based on recurrent neural networks have forged ahead in the race to machine translation [Sutskever et al., 2014], improving by a lot the quality of translation from one language to another. The introduction of attention by [Bahdanau et al., 2014] [Luong et al., 2015] eventually established a new standard in Machine Translation systems, allowing impressive performance like zero-shot translation like in [Johnson et al., 2016].

Advances have also been made in Image Captioning, using recurrent neural network for the most part, like in [Karpathy et al., 2015]. [Karpathy and Li, 2014] proposed to learn a joint embedding space for ranking and generation whose model learns to score sentence and image similarity as a function of R-CNN object detections with outputs of a bidirectional RNN.

Combining sequence-to-sequence with Image Captioning techniques, [Xu et al., 2015] encode the image in a fixed size vector with a CNN, and then decoding the vector step by step, generating at each step a new word of the caption and feeding it as an input to the next step. In their work, they also added an attention mechanism, enabling the decoder at each time step to look and attend at the encoded image, and compute a representation of this image with respect to the current state of the decoder (Figure 2).

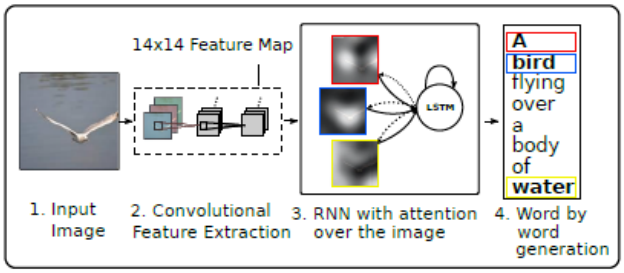

Figure 2. Attention based model for image captioning, figure from [Xu et al., 2015]

Recent work from [Deng et al., 2016] took this same approach and successfully applied it to the generation of LaTeXcode from images of formulas. They used the same mechanism of soft attention as in [Luong et al., 2015], and were able to achieve as much as 77% of exact match score on the test dataset.

3. Model

The architecture of our model is an adaptation Show, Attend and Tell [Xu et al., 2015] for LaTeX code generation that we combine with some details of [Deng et al., 2016] work.

3.1. Encoder-Decoder

The main building block of our model is based on the encoder-decoder framework.

Encoder First, we encode the images with a convolution neural network that consists of 6 convolution layers with filters of size 3x3, stride of 1 and padding of 1. After the convolution layers we also add max pooling layers. The role of these max-pooling is to extract structure from the characters. See details of the architecture of the network on Figure 3.

This CNN encodes the original image of size H x W into a feature map of size H' x W' x C where C is the number of filters of the last convolution layer. The encoder defines one vector $v_i \in \mathbb{R}^C$ for each of the H' x W' regions. Intu-

itively, each v_i captures the information from one region of the image.

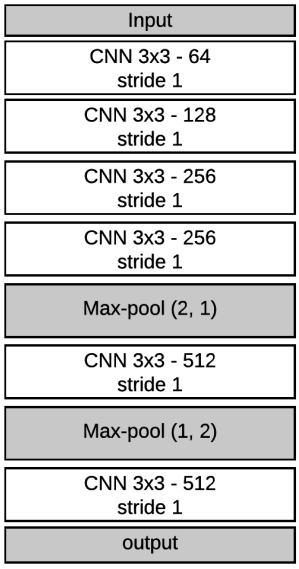

Figure 3. Convolutional Encoder

Decoder Once we have a feature map of size H' x W' x C, (in other words H' x W' vectors $v_i \in \mathbb{R}^C$) we can decode this representation of the image to produce the LaTeX tokens with a recurrent neural network. For this task, LSTMs ([Hochreiter and Schmidhuber, 1997]) have shown to be very efficient to capture long term dependencies and facilitate the back-propagation of gradients.

Formally, our decoder takes as input a hidden vector h_{t-1} (the hidden state of the LSTM, that include both the hidden state and the memory) and takes as input the previous token y_{t-1}. At each time step, the decoder computes an **attention** vector (see Section 3.2) c_t that depends on the image and the next hidden vector.

It then produces a distribution over the next token with a recursive formula

$$p(y_t) = f(y_{t-1}, h_{t-1}, c_t)$$

More specifically, we also compute another **output** state o_t used to compute the distribution probability over the vocabulary. This implementation is similar to [Deng et al., 2016] and differs from [Xu et al., 2015]. The details of the recurrent step are as follows

$$h_t = LSTM(h_{t-1}, [Ey_{t-1}, o_{t-1}])$$

$$c_t = Att(h_t, V)$$

$$o_t = tanh(W^c[h_t, c_t])$$

$$p(y_{t+1}|y_1, ..., y_t,) = softmax(W^{out} o_t)$$

where E is an embedding matrix, and W are matrices.

Note that here c_t stands for the attention vector (or *context* vector, see Section 3.2), and not the cell state of the LSTM (the cell state is not represented in these equations but is inherent to the LSTM cell). V is the encoded image.

We use two special tokens `START` and `END`. Once our decoder predicts the `END` token, we stop generating new tokens. The `START` token is used to initialize the decoder. Instead of initializing hidden vectors to zeros like in [Deng et al., 2016], we combine with the more expressive initialization from [Xu et al., 2015] and each hidden state is initialized with the following rule

$$h_0 = \tanh\left(W_h \cdot \left(\frac{1}{H'xW'} \sum_{i=1}^{H'xW'} v_i\right) + b_h\right)$$

where we learn an independent matrix W_h and bias b_h for each of the hidden states (including the memory of the LSTM and the output vector o_t).

3.2. Attention mechanism

To enhance the performance of the decoder, we add a **soft attention mechanism** as it was proved to help convergence in the case of image captioning [Xu et al., 2015]. Hence, at each time step of the decoding process, our RNN attends to the feature map V and compute a weighted average of these vectors. We used the following attention mechanism from *Mannings et. al* [Luong et al., 2015]:

$$e_i^t = \beta^T tanh(W_h h^{t-1} + W v_i)$$

$$\alpha^t = softmax(e^t)$$

$$c^t = \sum_{i=1}^{H'*W'} \alpha_i^t v_i$$

3.3. Beam search

At each time step, the LSTM produces a vector y_t of dimension V that is the size of the vocabulary and that represents the distribution probability over the vocabulary for the next word. From the vector y_t we need to decide which token to output and feed as input to the next step. We tried two different approaches to solve this problem:

- **Greedy approach**: the simplest approach is to simply output the token that has the highest probability at each time step. We refer to this approach as being *greedy*

because it can be suboptimal. For instance when looking for an optimal path in a graph to maximize the rewards on the edges of the graphs, the greedy approach is to take at each node the edge with he highest reward. However this technique can be suboptimal and lead to a lower reward at the end.

- **Beam search**: to try to bridge the gap that can exist between the greedy solution and the optimal solution, we implemented a beam search method to find the optimal output sentence. Now, at each time step we keep in memory the B hypotheses (sequence of tokens) with the highest probability, B being the beam size (e.g. B = 5). At each time step we input the B tokens to the decoder, compute B vector y_t and then update the list of tokens that we keep while keeping track of the "parents" of the B tokens with the highest probability over the B vectors y_t. What we want to maximize over the time steps of decoding is the product of probabilities of each token in a sentence given its parent. Hence, we can reconstruct the optimal sentence at the end of the decoding process and output our "optimal" sentence. Note that B = 1 is equivalent to the greedy approach. This method would be more robust to an mispredicted label at some time step.

4. Dataset and Training

4.1. Dataset and Implementation

Our main dataset is the prebuilt dataset from [Deng et al., 2016], **im2latex-100k**, that includes a total of ∼100k formulas and images splitted into train (∼84k), validation (∼9k) and test (∼10k) sets. Formulas were parsed from LaTeX articles on arXiv (`https://zenodo.org/record/56198#.WP1ksdI182x`). We note that this problem is particularly difficult as roughly half of the formulas contain more than 50 tokens! (See Figure 4). In the seq-to-seq setting, as the generated tokens depends on the beginning of the sentence and the previous tokens, this is a particular challenge!

We decided to limit the scope of this project to PDF images and not to treat the case of hand-written formulas. An extension of this work could be to run the same model using the CROHME dataset (not a typo!) that consists of + 10,000 handwritten mathematical expressions and their LaTeXcode.

We implemented our models using `Tensorflow` and run our experiment on a Tesla K80. We release the code for our own attention wrapper and Beam Search, as these functionalities are to be part of the next release version (and not yet implemented). [1] We use 15 epochs for a training time on the whole dataset of 12 hours. Our code is roughly twice as fast as the `lua` implementation of [Deng et al., 2016].

[1] The code is available here `https://github.com/guillaumegenthial/image2latex`

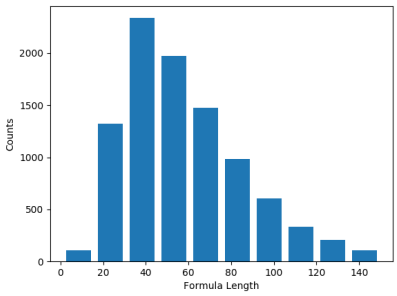

Figure 4. Formula Length

We train our network to maximize the probability of the reference formula.

4.2. Preprocessing steps

First we need to preprocess the images and formulas before we can input them to our model:

- **Images**: Each image is rendered using `pdflatex` and the `magick` tools. We use parameters `-density 200` and downsample images by a factor 2. On top of this, we use a greyscale representation. This way, the input to our network are low resolution images with just enough information to recognize the characters (readable by a human eye, but barely). The typical size is 200 x 160 pixels. We transfer the data on GPU using `uint8` to save on the CUDA `Memcpy`. To make the training go faster, we form buckets of images of the same shape (x2 speedup).

- **Formulas**: We tokenize the formulas as it as been observed that token-based methods are more efficient than character-based. We use the preprocessing scripts of [Deng et al., 2016] that uses Katex, a LaTeX parser written in javascript. Once the formulas are tokenized, we form our vocabulary of size 509 to which we add the special tokens "PAD", "START", "END", and "UNK" that represent the padding tokens, the start/end of sentence token and the unknown token respectively. We then pad the formulas with the "_PAD" token to make sure that they all have the same length (maximum length of 150) when we input them to our recurrent neural network. We build on [Deng et al., 2016] findings and use normalized LaTeX when possible. (This forces the use of conventions like `x_{i}` instead of `x_i`.)

4.3. Parameters and Optimization

We used the following hyper-parameters for our model

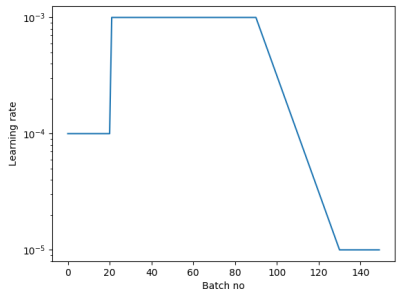

Figure 5. Learning rate schedule with warm-up - log scale

- **LSTM dim**: 512
- **Embeddings**: 80
- **Batch-size**: 20 (limited by the GPU memory)
- **Max length**: we use a default max of 150 tokens unless specified otherwise.
- **Learning-rate**: training attention based seq-to-seq models has been shown to be hard to train. We obtained good results with Adam optimizer ([Kingma and Ba, 2014]). As some runs were unable to learn, we used the warm-up strategy as mentioned by [Goyal et al., 2017]. Due to high variance and high gradients at the beginning of the training, we use a small *warm-up* learning rate of **1e-4** for the two first epochs. Then, we use a constant learning rate of **1e-3** until the 10-th epoch where we start to decay exponentially until **1e-5** at the end of the 15-th epoch. This strategy highly improved both the quality of our results and the speed of training compared to a constant or exponentially decaying schedule. See Figure 5.
- **Beam size**: 5 yielded good performance.
- **Weights initialization**: we use orthogonal initialization for embeddings and recurrent matrices, other wise we use the default Xavier initialization. Orthogonal matrices helped stabilize the training.

4.4. Evaluation

To evaluate the performance of the model, we must first understand that LaTeX is not a normalized language: different codes can still generate the same formula when compiled. For instance, the brackets are not always mandatory and "x^2" or "x^{2}" will still produce the same formulas on a PDF. We study two types of metrics to evaluate our model: one based on how well we reproduce the LaTeX code, one based on how well the reconstructed image is close to the original.

Text-based First, we relied on widely used metrics in Natural Language Processing such as exact match, perplexity and BLEU score ([Papineni et al., 2002]). The exact match quantity between a predicted formula and ground truth formula is one if the tokens of both formulas are exactly the same (same order, same number and same nature). The perplexity of a language tries to capture how confident (or confused) the model is at predicting one token. The perplexity of a language model is defined as

$$perplexity = e^{-\frac{1}{N}\sum_{i=1}^{N} log(x_i)}$$

where N is the total number of tokens to predict in the whole dataset, and x_i is the predicted probability of the true token i. For instance, when the model is completely random, the perplexity will be the number of words in the vocabulary V, meaning that for every word to predict, the model hesitates between V words. Conversely, a very good model should achieve a perplexity close to 1 meaning that the model is very confident in predicting the true word. We also used the BLEU score that captures the overlap of n-grams, as it has been shown to be the most correlated with human judgment and it is thus a standard metric in translation (1 is perfect). Finally, we can also compute the edit distance on the text (**Levenshtein** distance), between the predicted formula and the ground truth. This distance between two strings tells us how many characters we should add/remove/change in one string to obtain the other one. We report the **edit distance** which is the percentage of the reconstructed text that matches the original. Thus, a perfect match has an edit distance of 1.

Image-based

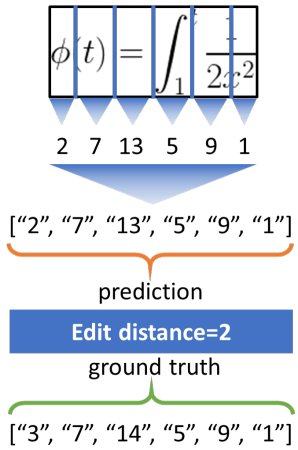

Figure 6. Edit distance on images

However, all these evaluation metrics do not take into account the problem of non normalization of LaTeX, and we

Decoding	EM Img	BLEU	Edit text	Edit Image
Greedy	0.22	0.76	0.76	0.35
Beam search (prediction)	0.32	**0.78**	0.76	0.62
Beam search (best proposal)	0.35	**0.78**	**0.76**	**0.62**
[Deng et al., 2016] CNNEnc	**0.53**	0.75	NR	0.61

Table 1. Comparison of beam search with greedy decoder and the equivalent model from [Deng et al., 2016] (CNNEnc). We obtain similar performance on most of the metrics, except for the Exact Match where CNNEnc obtains a much higher score.

might underestimate our model if we only rely on these metrics. This is why we also implemented an evaluation metric on the images generated from the predicted formulas, called the edit distance for images (or Levenshtein distance for images). Figure 6 depicts the method we apply to compute this distance over images. First we split the image in several columns and we encode each column with an integer. Slicing the images makes sense at the *expression* level and does not dilute distance like a pixel-wise distance would. From the Levenhstein distance between the columns of the reference and the predictions, we can compute a percentage of columns that are the same. We report his **edit distance**. 1 means that we have an exact match.

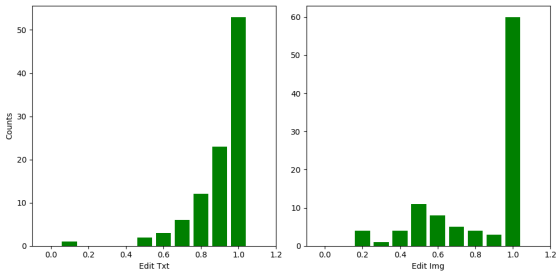

Figure 7. Distribution of Edit distances

5. Experiments

General [Deng et al., 2016] use a more complex architecture for the encoder with an recurrent additional layer to extract temporal dependencies in the image. We first compare the performance of their equivalent model (simple CNN encoder) with ours on the full dataset (all formula lengths). We obtain similar results except for the exact match distance. It may be due to the fact that we have a hard threshold of zero-difference between 2 images to count it as an exact match, but [Deng et al., 2016] may use a softer threshold that would explain the discrepency in the results.

Otherwise, we notice that **beam search** is slightly more efficient than **greedy decoding** in term of Exact Matches (**0.35** instead of **0.32**). Due to the clever initialization of the hidden states our model is able to slightly outperform the CNNEnc of [Deng et al., 2016]. It is important to note that their model contains more parameters than ours (they add unspecified parameters in the convolutions). Thus our model is both lighter, faster and equivalently efficient.

Interestingly, Figure 7 shows that while there is a continuous distribution of the formulas (Text) edit distance, most of the formulas are perfectly rendered. This suggests that changes in LaTeX do not necessarily induces big changes in the rendered formula, as expected. We also see that our model does a pretty good job at rendering images! Figure 5 presents some examples of mistakes.

We also point out the usefulness of the beam-search that produces multiple proposals. If we have a look at which of the proposal matches the best the reference, we find the distribution on Figure 8. As expected, the first hypothesis is almost always the best one, but if a user were proposed the few proposals, we could gain some performance.

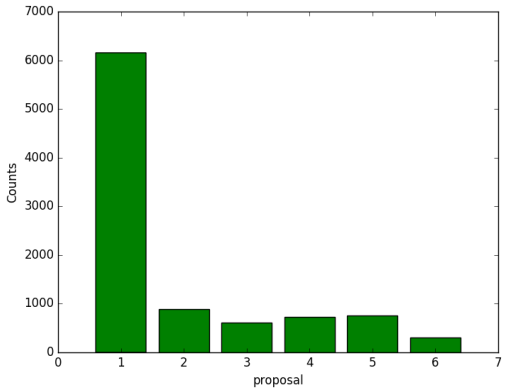

Figure 8. Distribution of Edit distances

Dropout We tried to add dropout to our model to avoid over-fitting but as we can see in the table below the best performing model is still the one with no dropout. However, these results are subject to caution as we evaluated it only on the half of the dataset. We applied the dropout like in [Xu et al., 2015].

Dropout	EM	BLEU	Edit text	Edit Image
0	0.21	0.80	0.83	0.74
0.2	0.2	0.78	0.82	0.66
0.5	0.16	0.75	0.80	0.63

Table 2. Comparison of models with different dropouts on a subset of the dataset - formulas with less than 50 tokens - 36k examples

Embeddings Our model initializes the embeddings of the token randomly. We tried to train embeddings in an unsupervised manner using CBOW ([Mikolov et al., 2013]). However, we would need a lot more data to build interesting representation, that could constitute itself a paper but more focused on the NLP side. We didn't notice any improvement in performance.

Encoders As [Deng et al., 2016] noticed some improvement using a bi-LSTM encoder on top of the CNNEnc, we tried to add more parameters and remove the destructive max-pool layers from the network. Following the intuition of the recent paper [Gehring et al., 2017], we replace these max-pool by an additional convolutional layer with a bigger filter and an equivalent stride (thus the output of the encoder has the same shape as our previous encoding). The architecture of this network is represented on Figure 9.

We also evaluated the impact of adding residual connections between the convolutional layers (see our code for more details).

Encoder	EM	BLEU	Edit text	Edit Image
Vanilla	0.21	0.80	0.83	0.74
Residual	0.20	0.78	0.81	0.66
Variation	**0.31**	**0.84**	–	–

Table 3. Comparison of models with different encoders on the formulas with less than 50 tokens. Our variation seems to achieve much higher results (some metrics are still missing). Residual Connection do not seem to help, on the contrary

Influence of Formula Length As expected, the longer our sentences, the lower the performance. See 10. In long sentence, the model is more prone to attention errors but also is sensible to errors while decoding.

6. Conclusion

This paper presents an efficient implementation of a caption-generation system applied to LaTeX generation from raw images. We obtain good performance, similar to [Deng et al., 2016]. As outlined in the experiments, the encoder might be improved. If [Deng et al., 2016] obtain good results with a bi-LSTM, more recent papers (like

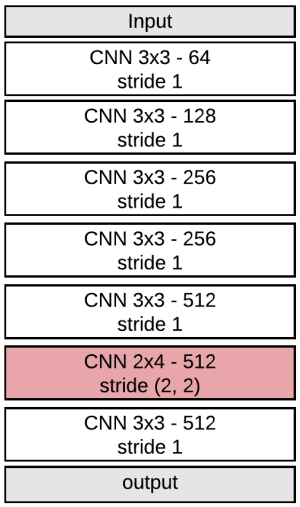

Figure 9. Variation of the Encoder

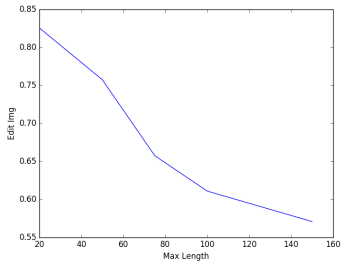

Figure 10. Performance of our model as a function of the formula length

[Gehring et al., 2017]) and our own experiments suggest that it might be a computational waste (bi-LSTM are not as parallelizable as CNN). Future work could even incorporate a CNN-based decoder like in [Gehring et al., 2017], with the advantage of being much faster at training time.

References

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Belaid and Haton, 1984] Belaid, A. and Haton, J.-P. (1984). A syntactic approach for handwritten mathematical formula recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6(1):105–111.

[Chan and Yeung, 2000] Chan, K.-F. and Yeung, D.-Y. (2000). Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3(1):3–15.

Truth	$\pi^{ij}\gamma_{jl} = \frac{1}{3}\pi\delta_l^i + \tilde{\pi}^{ij}\tilde{\gamma}_{jl} \quad .$	$\frac{d}{ds}\frac{1}{\Gamma(-s)}\Big\|_{s=0} = -1,$	$\|f_L, f_R\rangle \star \|g_L, g_R\rangle = e^{-\int_0^{\frac{\pi}{2}} f(\pi-\sigma)g(\sigma)} \|f_L, g_R\rangle.$
Prediction	$\pi^{ij}\gamma_{jl} = l = \frac{3}{\pi}\delta_l^i + \tilde{\pi}^{ij}\tilde{\gamma}_{jl} \quad .$	$\frac{d}{ds}\frac{1}{\Gamma(-s)}_{s=0} = -1,$	$\Big\\| f_L, f_R\rangle = \|g_L, g_R\rangle = e^{-\int_0^{\frac{\pi}{2}} f(\pi-\sigma)g(\sigma)} \|f_\ell, g_R$
Best	$\pi^{ij}\gamma_{jl} = \frac{1}{3}\pi\delta_l^i + \tilde{\pi}^{ij}\tilde{\gamma}_{jl} \quad .$		$\|f_L, f_R\rangle \star \|g_L, g_R\rangle = e^{-\int_0^{\frac{\pi}{2}} f(\pi-\sigma)g(\sigma)} \|f_\ell, g_R.$

Table 4. Examples of mistakes. We render the predicted formula to compare with the original image using `pdflatex` - The entry **best** shows the best hypothesis among the beam search proposal if different from the first proposal. Among these examples, we notice that some of the mistakes are minor mistakes and that it can happen that the best proposal from the beam search is not the first one.

[Ciresan et al., 2010] Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010). Deep big simple neural nets excel on handwritten digit recognition. *CoRR*, abs/1003.0358.

[Deng et al., 2016] Deng, Y., Kanervisto, A., and Rush, A. M. (2016). What you get is what you see: A visual markup decompiler. *CoRR*, abs/1609.04938.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122.

[Goyal et al., 2017] Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *ArXiv e-prints*.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CVPR*.

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

[Jaderberg et al., 2014] Jaderberg, M., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Deep structured output learning for unconstrained text recognition. *CoRR*, abs/1412.5903.

[Johnson et al., 2016] Johnson, M., Schuster, M., Le, Q. V., Krikun, M., Wu, Y., Chen, Z., Thorat, N., Viégas, F. B., Wattenberg, M., Corrado, G., Hughes, M., and Dean, J. (2016). Google's multilingual neural machine translation system: Enabling zero-shot translation. *CoRR*, abs/1611.04558.

[Karpathy et al., 2015] Karpathy, A., Johnson, J., and Li, F. (2015). Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

[Karpathy and Li, 2014] Karpathy, A. and Li, F. (2014). Deep visual-semantic alignments for generating image descriptions. *CoRR*, abs/1412.2306.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *EMNLP*.

[Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Miller and Viola, 1998] Miller, E. G. and Viola, P. A. (1998). Ambiguity and constraint in mathematical expression recognition. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA.*, pages 784–791.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *NIPS*.

[Wang et al., 2012] Wang, T., Wu, D. J., Coates, A., and Ng, A. Y. (2012). End-to-end text recognition with convolutional neural networks. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3304–3308.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044.