# Translating handwritten mathematical expressions to LaTeX

**August Regnell** 970712-9491 aregnell@kth.se

**Joar Haraldsson** 951111-1230 joarha@kth.se

**Emil Kerakos** 960720-7777 ekerakos@kth.se

**Carl Höggren** 970514-5036 choggren@kth.se

26 May, 2021

**Abstract**

Converting images of handwritten mathematical expressions to LaTeX-code is among the most difficult problems in Optical Character recognition. Building on the recent trend of using end-to-end Deep Learning models similar to those in Image Captioning, we build an Encoder-Decoder model for the task. The Encoder consists of a CNN with 6 convolutional layers (3 with batch normalization) and 4 max-pooling layers; the Decoder consists of an LSTM with a soft attention mechanism. We train and test the model on the largest publicly available dataset for handwritten mathematical expression recognition: the CROHME dataset. Using beam search during testing we obtain the reasonable performance of 0.57 on Normalized Levenshtein, 0.20 on BLEU, 0.53 on Jaccard Similarity and 0.20 on LMS. Despite inferior results compared to other model implementations on the same dataset, our model managed to accurately translate images of the authors own handwriting, proving the model's ability to generalize to a broad set of handwriting styles.

## 1 Introduction

The problem of optical character recognition (OCR) - converting images of printed, typewritten or handwritten text into its digital counterpart [10] - has strong ties to Pattern Recognition and Deep Learning historically: from Geoffrey Hinton's work on Deep Belief Networks for classifying handwritten digits [11] which helped spark the modern interest in Deep Learning [12], to today's systems for text recognition in images [13].

Within OCR, handwritten mathematical expression recognition (HMER) is one of the most difficult problems [6] and has been studied since the 1960s [3]. There are two different categories of HMER: offline and online [7]. Offline deals with static handwriting on a physical piece of paper whereas online typically deals with the recorded point series produced by a pen or a finger writing on a tablet. Motivated by the authors' own trials and tribulations of learning the language of LaTeXand having to translate pages of handwritten solutions to LaTeX-code in a short period of time, this paper deals with offline HMER. Further developing this kind of technology may help students and researches entering handwritten expressions into LaTeX, increasing productivity.

Traditional approaches to this problem use a three-step (not always sequential) process of symbol segmentation, symbol recognition and structural analysis [6, 3] using pre-defined grammars [3]. Recently, methods based on deep neural networks have been proposed where HMER is treated as a special case of the more general image-to-sequence problem (see for instance [1, 8, 9]), using the Encoder-Decoder framework [3]. These networks have increased recognition performance significantly but due to the complexity of the problem, HMER methods are still limited in accuracy [3].

In this paper we take the same approach: we use a CNN as the encoder while the decoder consist of an LSTM. A soft attention mechanism and a beam-search accompany the decoder to enhance its performance. Performance was measured with 4 metrics; Normalized Levenshtein distance, BLEU, Jaccard Similarity and LMS. With a fully trained network, the average scores on hitherto unseen test data were: Normalized Levenshtein Distance = 0.568, BLEU Score = 0.195, Jaccard Similarity = 0.532 and LMS-score = 0.199. These results are inferior to the state-of-the-art but can, to some extent, be attributed to *difference in input-data*, *shorter training-time*, *different preprocessing techniques* and *less hyperparameter tuning*.

## 2 Related Work

### What You Get Is What You See: A Visual Markup Decompiler[1]

This paper, written by Deng et al, is the main inspiration for the model used in this project. Their model consists of a CNN and RNN encoder, followed by an LSTM decoder with a soft attention mechanism. The model is first pre-trained on HTML-images before it is trained on the `IM2LATEX-100K` dataset, consisting of rendered LaTeX-equations with their LaTeX-code as labels. Furthermore, beam search is used during testing as it was shown to increase performance.

### Image to Latex[2]

This student project, improved upon the paper by Deng et al. The main changes were: the removal of the RNN in the encoder, a reduction in the number of max-pooling layers in the CNN, using the encoded images to initialize the hidden states of the LSTM and using a learning rate schedule with a warm-up. These changes yielded a small improvement in the BLEU and edit distance metric but a reduction in the exact match (EM).

### Handwritten Mathematical Expression Recognition via Paired Adversarial Learning[3]

In this paper the authors explored the possibilities of generating LaTeX-code from images of handwritten mathematical equations. The model used consists of an attentional encoder-decoder (a CNN-based feature extractor and a RNN-based feature extractor) as a recognizer and a discriminator. The core idea of this paper is to handle the problem of writing-style variation by mapping handwritten mathematical expressions to their printed templates in features space, making the model learn semantic-invariant features. Specifically the model is trained in an adversarial manner: the discriminator is optimized to distinguish a handwritten mathematical expression with it's printed template whereas the recognizer is optimized to fool the discriminator. The model reached an expression recognition rate of $\sim$44% on handwritten expressions (from the CROHME dataset) and $\sim$84% on printed expressions.

## 3 Data

The dataset originates from the *Competition on Recognition of Online Handwritten Mathematical Expressions* (CROHME) and consists of 7,170 samples of training data, 2,120 samples of test data and 667 samples of validation data encoded as .inkml files. InkML is an XML-markup language intended to describe recorded point series data written with an electronic pen or stylus. Despite its small size the CROHME dataset is currently the largest public HMER dataset [3].

### 3.1 Pre-processing of images

Since we are interested in offline HMER, the InkML files were converted into PNG-files using the the python package `inkml2img`. Once converted, all images were cast into arrays and their dimensionality reduced to only include the X and Y positions of the binary colors black or white. The images were then padded individually in order to achieve a common height-width ratio of 2:5. After the padding, all images were re-scaled to approximately half the mean of the X and Y dimensions of all images, $100 \times 250$ (H$\times$W). Combining padding and re-scaling ensured that the images kept their characteristics, as solely resizing introduced a great deal of distortion. The shrinkage of the images also reduced the effective number of parameters in the network with a factor of $\sim 4$, which yielded faster training times.

The images were then normalized using the mean and standard deviation of the training images. These three steps can be observed in figure 7 in Appendix A.

### 3.2 Pre-processing of labels

Each InkML file also contains a label of LaTeX-code which we extracted by looping through each file with a file parser using the python package `Beautifulsoup`. The labels were written by many different people and thus contain different LaTeX-expressions for the same mathematical expressions. Therefore we performed a normalization of the labels in order to reduce equivocacy prior to training. This normalization was performed with the help of a LaTeX-parser created by Deng et al., which transforms the markup to an abstract syntax tree and then rewrites the labels to a standardized format. The normalization helps the training of the model by removing some of the inherit ambiguity in LaTeX-code, for instance `x_i^j` and `x^j_i` both being rendered to $x_i^j$. Thereafter, the labels were tokenized, as token-based models have been proven to be more efficient than character-based models[1]. The process of tokenization involved: 1) Creating a token set of 144 symbols, including special characters "START", "END", "PAD"; 2) Adding "START" and "END" token to each mathematical expression and padding each sequence with "PAD" in order to have a standard sequence length of 110 symbols; 3) Tokenizing all symbols in each mathematical expression, using the token set from step 1.

## 4 Methods

Since RNN-based encoder-decoder models with attention have been widely used in image-to-sequence and sequence-to-sequence problems like image captioning, machine translation and speech recognition [3] and since this framework has been the basis of the recent improvements in recognition performance in HMER, we also take such an approach in our study. The model architecture we use is an adaptation of the one suggested in the paper *What You Get Is What You See: A Visual Markup Decompiler.* The model consists of a CNN encoder and and a (soft) attention-based LSTM decoder. During testing beam search is also used. All the code is written by ourselves in python using the package PyTorch.

## 4.1 Encoder

As a first step the pre-processed images are encoded using a convolutional neural netowrk consisting of 6 convolutional layers with filters of size $3 \times 3$, stride 1 and padding 0 or 1. In between some of the convolutional layers max-pooling and batch normalization was used. The max-pooling effectively reduced the size of the intermediary images without destroying too much of the characteristics while the batch normalization helped reduce the risk of exploding and attenuated gradients. The full specification can be seen in table 1.

| **Layer** | **Convolutional layer** | **Max-pooling layer** |
|---|---|---|
| 6 (output) | c:512, k:(3,3), s:(1,1), p:(0,0), bn | - |
| 5 | c:512, k:(3,3), s:(1,1), p:(1,1), bn | po:(1,2), s:(1,2), p:(0,0) |
| 4 | c:256, k:(3,3), s:(1,1), p:(1,1) | po:(2,1), s:(2,1), p:(0,0) |
| 3 | c:256, k:(3,3), s:(1,1), p:(0,0), bn | - |
| 2 | c:128, k:(3,3), s:(1,1), p:(1,1) | po:(2,2), s:(2,2), p:(0,0) |
| 1 (input) | c:64, k:(3,3), s:(1,1), p:(1,1) | po:(2,2), s:(2,2), p:(1,1) |

Table 1: Specification of the CNN encoder. c: number of filters, k: filter size, s: stride size, p:padding size, po: pooling size, bn: batch normalization.

The encoder takes an image of size $100 \times 250$ ($H \times W$) and encodes it into a feature map $V$ of size $10 \times 29 \times 512$ ($H' \times W' \times C$). One can intuitively regard every $v_i \in \mathbb{R}^{512}$ in the feature map as a capturing of the information from one region in the original image.

## 4.2 Decoder

We then use a recurrent neural network together with an attention mechanism to generate the LaTeX-tokens from this feature map V (the encoded image). Although fully convolutional Encoder-Decoder have been proposed, they suffer from lack of coverage [3]: some regions of the image are over-attended and some are under-attended in the decoding process. RNN:s don't have this same problem as they can use information on "where" attention has been placed in previous timesteps to inform the attention of the current step [3]. Since LSTMs are very commonly used and can capture long term dependencies well while facilitating back-propagation of gradients [2]. However, we note that other RNN:s like GRUs and bidirectional LSTMs have also been used as Decoders for the task [8, 9].

On a high level the Decoder performs a function $f$ at each timestep, computing a distribution over the vocabulary:

$$p(y_t) = f(h_{t-1}, s_{t-1}, y_{t-1}, o_{t-1}, V)$$

The details of this function are as follows. At each timestep the LSTM cell uses the cell state $s_{t-1}$ and hidden state $h_{t-1}$ from the previous timestep together with the token generated $y_{t-1}$ and the o-layer activations $o_{t-1}$ – also from the previous timestep – to compute the current hidden state $h_t$ and cell state $s_t$ by eq. (1) below. Here, the concatenation $[Ey_{t-1}, o_{t-1}]$ is treated as the vector input to the LSTM, normally denoted $x_t$, and $E$ is an embedding matrix (the standard LSTM equations are found in Appendix D).

The hidden state is emitted to the Attention mechanism and further to the o-layer. The attention mechanism uses $h_t$ together with the feature map $V$ to compute an attention vector $c_t$ (eq. 2 below). Previous studies in image captioning have shown that attention mechanisms help convergence[4] and here we choose to use a soft attention mechanism. At every time step, the attention mechanism computes a weighted average of the vectors $v_i$ in the feature map $V$. The equations describing the attention mechanism can be found in Appendix B.

The attention vector $c_t$ and the hidden state $h_t$ are then fed to the o-layer, which in turn is used to compute the discrete distribution over the vocabulary (equations 3 and 4).

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

$$c_t = Att(h_t, V) \tag{2}$$

$$o_t = tanh(W^c, [h_t, c_t]) \tag{3}$$

$$p(y_{t+1} \mid y_1, \ldots, y_t) = softmax(W^{out} o_t) \tag{4}$$

During training we use this distribution to compute the local cost (cross-entropy) between the generated LaTeX-token and the true token (the label), and then for the next timestep we use the true token as input for the LSTM, together with the output activations. During testing we instead output the token with the highest probability in the distribution and use it for the next timestep (in the greedy approach).

## 4.3 Beam search

Beam search was implemented to optimize the network's performance and lower the risk of finding a sub-optimal solution. It differs from a greedy-approach in that, for each time step, $B$ candidates with the highest probability are selected and used as input for the next time step. $B$ was set to 5 as it yielded good performance given a reasonable time expenditure.

## 4.4 Parameters and Optimization

| **Parameter** | **Choice** | **Reason** |
|---|---|---|
| LSTM dimensions | 512 | Same as in [2] |
| # Embeddings | 80 | The embedding size is usually decided to be 50-300 as a rule of thumb and doesn't have a notable effect on performance over a lower bound on many NLP tasks [14]. Thus, using the settings in [2] seemed reasonable |
| Batch size | 5 | Limited by our computers memory |
| Learning rate | Adam optimizer & learning rate schedule suggested by [Goyal et. al.][5] | The schedule made sure the network was able to learn (it had trouble learning when starting with too high learning rates) |
| Number of epochs | 15 | Follows from the learning rate schedule |
| Beam size | 5 | See section 4.3 |
| Activation function in the CNN | ReLu | Standard |
| Loss function | Cross entropy | Same as [1] and [2] |
| Weight initialization | All weights were initialized using normal Xavier initialization. For hidden states, see below. | See *Experiments* |

Table 2: This table summarizes our choices of hyperparameters, initialization, activation and loss function, and other things which were used to train our best performing model.

The hidden states in the LSTM were initialized using equation (5) where the matrix $W_h$ and the bias vector $b_h$ are learnable parameters. Adopted from [Xu et. al.][4].

$$h_0 = \tanh\left(W_h \cdot \left(\frac{1}{H'W'}\sum_{i=1}^{H'W'} v_i\right) + b_h\right) \tag{5}$$

## 4.5 Evaluation

To evaluate the network's performance, four performance metrics were used; Longest Mutual Sequence (LMS), Levenshtein Distance, BLEU Score and Jaccard Similarity.

**Longest Mutual Sequence**
*LMS* compares two input sequences $P = p_1, p_2, p_3, p_4...$ and $T = t_1, t_2, t_3, t_4...$, each with length $P_l$ and $T_l$, having the longest common sequence $L$ for which $t_i = p_i$. LMS-score is then calculated as LMS $= L/\max(P_l, T_l)$.

**Levenshtein Distance**
*Levenshtein Distance* is used to compare item-to-item similarities between a prediction and its ground truth. The metric indicates the number of *insertions*, *deletions* and *substitutions* needed to transform one string into another string, outputting a value in $[0, \infty)$ where 0 indicates that the two strings are identical. To facilitate comparing the results with other papers, the reported levenshtein distance between prediction $P$ and ground truth $T$ distance is given by

$$\text{NormalizedLevenshtein} = 1 - \text{Distance}(P, T)/\operatorname{len}(\max(P, T))$$

**BLEU Score**
*BLEU Score* is an alternative metric to compare item-to-item similarities between two inputs and has been shown to more closely correlate with that of human perception. It compares n-grams and outputs a score between 0 and 1, where 1 indicates a perfect match between two inputs.

**Jaccard Similarity**
*Jaccard Similarity* attempts to capture two aspects; the possibly non-normalized output of the model and the

ambiguity of mathematical expressions. For mathematical ambiguity, consider the two expressions $C = e + 1$ and $D = 1 + e$. It is evident that C = D although e+1 $\neq$ 1+e are dissimilar in their string-edit-distance. It's important to note that Jaccard Similarity, defined as Jaccard$(A, B) = (A \cap B)/(A \cup B)$, merely captures the token similarities between the predicted and ground-truth sequence. As a result, a Jaccard Similarity of 1 only indicates that all tokens present in the predicted sequence are also present in the ground truth sequence. This *could* mean that sequence A is semantically identical to B, but does not guarantee it.

## 5 Experiments

**General performance**: Both Deng et al and the authors of *Image to Latex* train and test their models on a dataset consisting of rendered LaTeXexpressions. Our dataset, CROHME, consists of handwritten expressions which has shown to be significantly more difficult to predict; a handwritten character is (almost) never identical between different writers (and even the same writer depending on the day). Noteworthy is also that our dataset is significantly smaller, the `IM2LATEX-100K` dataset is almost 15 times larger than the CROHME dataset we are using. However, we have still chosen to compare our results to theirs. Note that we have chosen to compare to Deng et al's equivalent model (a simple CNN encoder) since their original model has a CNN-RNN encoder. As expected, our results on the CROHME test set, which are summarized in figure 1, are noticeably worse, particularly for the BLEU score. However, they are quite close in Normalized Levenshtein distance.

| **Method** | **Levenshtein** | **BLEU** | **Jaccard** | **LMS** |
|---|---|---|---|---|
| Greedy | 0.372 | 0.173 | 0.505 | 0.168 |
| Beam search (B = 3) | 0.502 | 0.182 | 0.512 | 0.186 |
| Beam search (B = 5) | 0.568 | 0.195 | 0.532 | 0.199 |
| Beam search (B = 10) | 0.570 | 0.197 | **0.540** | **0.202** |
| Deng et al, CNNEnc[1] | - | 0.750 | - | - |
| Image to Latex[2] | **0.76** | **0.780** | - | - |

Figure 1: Our results compared to the equivalent model of Deng et al. and *Image to Latex*'s model respectively. **Training of model currently running, these will be updated when it's finished.**

**Beam search**: We did observe a significant increase in the performance of our network when we used beam search instead of the greedy approach, which can be seen in figure 1. However, the improvements were not enough to reach the results of those models we compare to.

**Initialization**: Furthermore, our initialization of the hidden states in the LSTM (see section 4.4) makes the model use the encoded image $V$ not only in the attention mechanism but also in the RNN-part! This led to some improvements, which can be seen in the results of the following small experiment. After a full batch of training on a subset of the total training set (1000 images) the model with an initialization as suggested in section 4.4 achieved an average loss of 1.823 compared to the model with no special initialization (the hidden states were initialized as zero-vectors) achieved an average loss of 1.893. We thus see an improvement of around 4%. This improvement will most likely compound during longer runs since the model makes better use of the encoded images and since this initialization effectively adds more model parameters in $W_h$ and $b_h$.

**Performance by sequence length**: Looking at figure 3 we notice that the model performs worse, in terms of BLEU score, the longer sequence length it tries to predict. This is expected since a longer sequence gives more chances of erroneous prediction and the errors also more easily compound. However, looking at the distribution of the sequence lengths in the data one would also think that a plausible explanation is the lack of longer sequences.

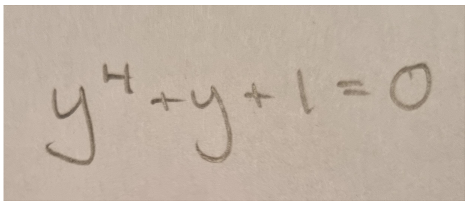


Figure 2: Prediction: `y^{4}+y+1=0`
Rendered code: $y^4 + y + 1 = 0$

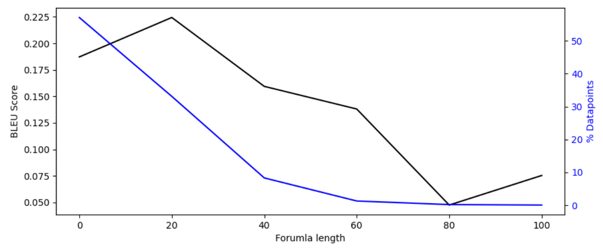


Figure 3: BLEU-score for different sequence lengths on the test data

**Tests on expressions written by the authors**: As another test we wanted to see how well our model performed on expressions written by ourselves. In figure 8 in Appendix C there are four examples of this, the first one is

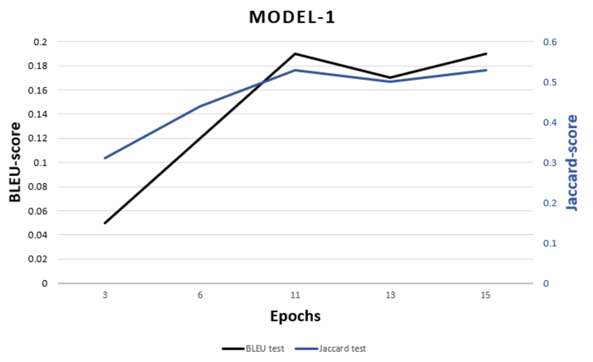


Figure 4: Performance of our learning rate schedule

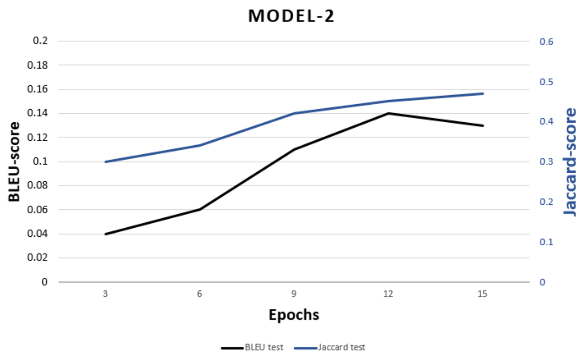


Figure 5: Performance of the constant learning rate

highlighted in figure 2 below. As one can see in figure 2/8a, 8b and 8c the network seems to perfectly predict the labels. This is probably due to that these expressions were chosen from the training set. Looking at the fourth example in figure 8d we see that the model struggles when it is an expression it hasn't trained on, even though it is quite close. This may be an indication of the network being overfitted.

**Hyperparameter/Learning rate search**: We trained the network with two different learning rate schedules. Model-1 used a warm-up phase with learning rate 1e-4 for two epochs, followed by seven epochs of constant learning rate of 1e-3, followed by five epochs of exponentially decaying learning rate until 1e-5 at epoch 15. Meanwhile, Model-2 used a constant learning rate of 1e-3 for all 15 epochs. The warm-up learning rate schedule was proposed by [Goyal et al.][5], and yielded faster training and higher performance, the latter of which can be seen on the test set BLEU and Jaccard-scores in figure 4 and 5.

**Possibility of overfitting**: Comparing our models performance on the test set and the training set (see figure 6) it almost looks like that the network has started to overfit. This is most likely explained by the size of our dataset; having more data to train on helps the network to learn general patterns and thus effectively acts as a regularization.

However, when looking at the performance metrics on the model we still notice an improvement in all of them for every additional epoch the model is trained, as seen in figure 4, indicating that the overfitting probably isn't the case.

| **Dataset** | **Levenshtein** | **BLEU** | **Jaccard** | **LMS** |
|---|---|---|---|---|
| Test | 0.568 | 0.195 | 0.532 | 0.199 |
| Train | 0.957 | 0.876 | 0.957 | 0.908 |

Figure 6: The model's performance on the test and training set using beam search with $B = 5$.

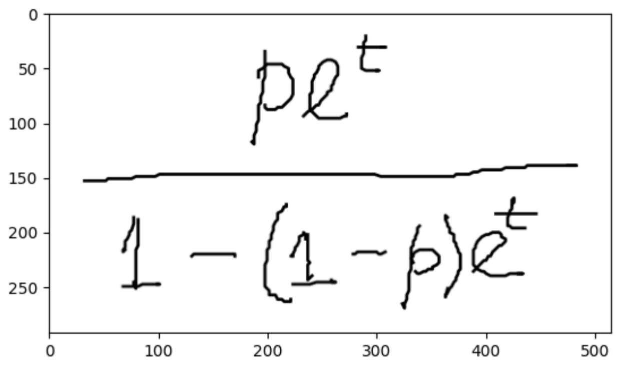
## 6 Conclusion

Studying the results, our implementation gives inferior results to other comparable implementations. Our best performing model achieved a BLEU score of 0.20 on the test data, significantly lower than [Genthial et. al.][2] (0.78) and [Deng et. al.][1] (0.75) achieved on a dataset consisting of LaTeX-rendered code. Comparing to another model structure tested on the same dataset[4] we only achieved a perfect match of 2.1% compared to their 44%. We thus conclude that the model structure of [4] most likely is more fit for this purpose.

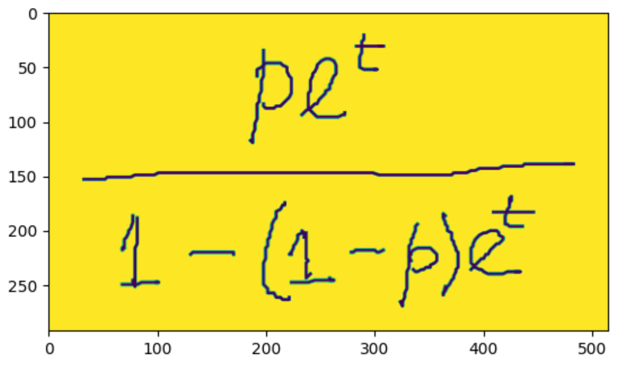
During our experiments, we noted that our model had a tendency to achieve much higher performance on training than test set. Since more training data would probably introduce a regularization-effect and increase generalization performance, one solution could be to artificially increase the size of the dataset by adding noise or other transformations to the existing images. Another solution would simply be to pre-train the network on a "synthetically handwritten" dataset with more samples or combine another dataset with CROHME. Overfitting can also be mitigated by more thorough hyperparameter tuning. While we performed some hyperparameter tuning, more specifically on the learning rate, weight initialization, and beam search, performance could probably be enhanced further by exploring other parameters and methods such as drop out.

Contrary to our initial beliefs, the model managed to transform our own handwritten mathematical expressions into LaTeX-code. Although the best performance was achieved when the expressions came from a subset of the training data, the results display the model's ability to generalize and comprehend a set of different handwriting styles.

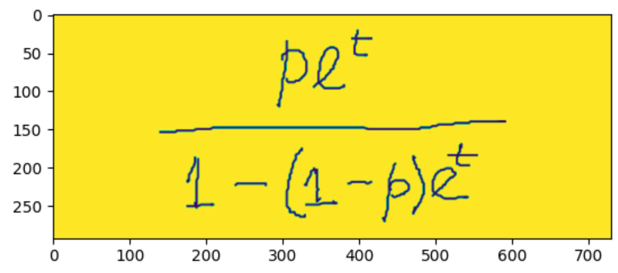
## A A visualization of the steps in the pre-processing of an image



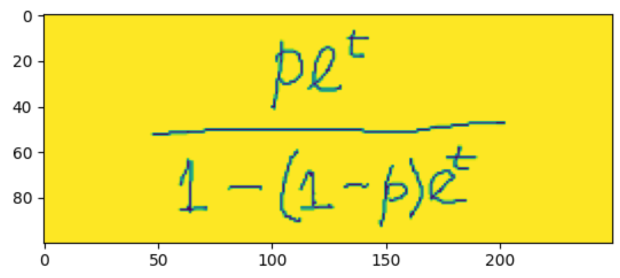
(a) Original image



(b) Removed redundant color channels



(c) Padded image (in width in this case)



(d) Re-scaled image (shrunk in this case)

Figure 7: Example of pre-processing of a single image

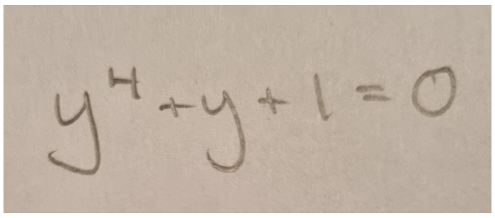
## B The mathematics behind the soft attention mechanism

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

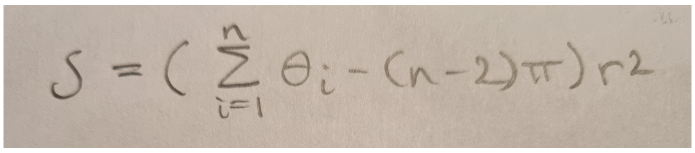
$$\alpha^t = softmax(e^t)$$

$$c^t = \sum_{i:v_i \in V} \alpha_i^t v_i$$

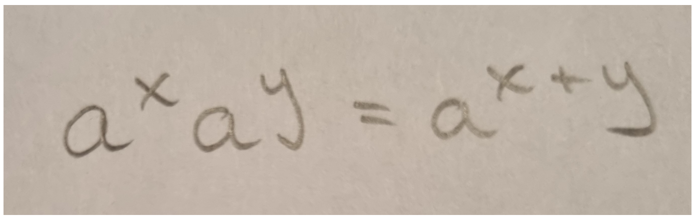
## C Test on expressions written by the authors



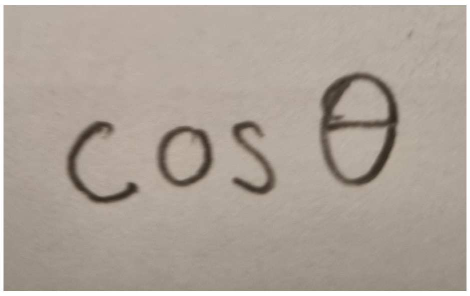
(a) Prediction: `y^{4}+y+1=0`
Rendered code: $y^4 + y + 1 = 0$



(b) Prediction:
`S=\Bigg(\sum_{i=1}^{n}\theta_{i}-(n-2)\pi\Bigg)r^{2}`
Rendered code: $S = \Bigg(\sum_{i=1}^{n} \theta_i - (n-2)\pi\Bigg) r^2$



(c) Prediction: `a^{x}a^{y}=a^{x+y}`
Rendered code: $a^x a^y = a^{x+y}$



(d) Prediction: `e^{os}\theta^{2}`
Rendered code: $e^{os}\theta^2$

Figure 8: Some examples of the model predicting the labels of equations written by ourselves. Not that these are the original images, that is before pre-processing.

## D Standard LSTM Update Equations

At a given timestep t, let $h_t$ denote the hidden state of the LSTM cell, $s_t$ the cell state and $x_t$ the input (in our case $x_t = [Ey_{t-1}, o_{t-1}]$). Further, $i_t$ denotes the input gate, $f_t$ the output gate, $o_t$ the output gate, $g_t$ the candidate values to (potentially) be added to the cell state. Each $W$ is a matrix and each $b$ a bias term. The equations for computing the hidden state and cell state $h_t, s_t = LSTM(h_{t-1}, s_{t-1}, [Ey_{t-1}, o_{t-1}])$ are then given by:

$$i_t = \sigma\left(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}\right)$$

$$f_t = \sigma\left(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}\right)$$

$$g_t = \tanh\left(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}\right)$$

$$o_t = \sigma\left(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}\right)$$

$$\boldsymbol{s_t} = f_t \odot s_{t-1} + i_t \odot g_t$$

$$\boldsymbol{h_t} = o_t \odot \tanh\left(s_t\right)$$

## References

[1] Deng et. al. 2016. *What You Get Is What You See: A Visual Markup Decompiler.* arXiv:1609.04938v1

[2] Genthial et. al. Stanford. *Image to Latex.*

[3] Wu et. al. 2020. *Handwritten Mathematical Expression Recognition via Paired Adversarial Learning.* `https://doi.org/10.1007/s11263-020-01291-5`

[4] Xu et. al. 2015. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.* arXiv:1502.03044v3

[5] Goyal et. al. 2018. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour.* arXiv:1706.02677v2

[6] He et. al. 2020. *Handwritten Mathematical Expression Recognition: A Survey.* `https://doi.org/10.1007/978-3-030-59830-3`$_5$

[7] Simistira et. al. 2015. *Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars* `https://doi.org/10.1016/j.patrec.2014.11.015`

[8] Le and Nakagawa. 2017 *Training an End-to-End System for Handwritten Mathematical Expression Recognition by Generated Patterns* `10.1109/ICDAR.2017.175`

[9] Zhang et. al. 2019 *Track, Attend, and Parse (TAP): An End-to-End Framework for Online Handwritten Mathematical Expression Recognition* `10.1109/TMM.2018.2844689`

[10] Vamvakas et. al. 2010 *Handwritten character recognition through two-stage foreground sub-sampling* `https://doi.org/10.1016/j.patcog.2010.02.018`

[11] Hinton et. al. 2006 *A Fast Learning Algorithm for Deep Belief Nets* `https://doi.org/10.1162/neco.2006.18.7.1527`

[12] Nielsen. 2015. *Neural Networks and Deep Learning*

[13] Ye et. al. 2015. *Text Detection and Recognition in Imagery: A Survey* `10.1109/TPAMI.2014.2366765`

[14] Patel et. al. 2015. *Towards Lower Bounds on Number of Dimensions for Word Embeddings.* Accessed at: https://www.aclweb.org/anthology/I17-2006