Turning equations into LateX

Pipeline vs. End-to-End***

Motivation

There is currently no good solution for converting handwritten notes into LateX. As a consequence, STEM students around the world struggle. In this project we convert images of equations into LateX markup . We compare two approaches:

1) **a multi-stage pipeline approach** where two main steps are:

   segmenting the image into individual symbols

   classifying each individual symbol.

   We adapt the approach proposed by Chang et. al [1]

2) **an end-to-end model** utilizing a similar seq-2-seq architecture found in many prominent translation models. Inspired by the work of Genthial [2].

Data and Features

1. InftyCDB-3 – Mathematical Symbols – 70k
   - High resolution images of math symbols
   - Black and white in different sizes
   - Ground truth math symbol, 275 tokens, (not in LateX)
   - We downsample, normalize and center crop.
   - Split into Train / Val / Test
   - Used in the Pipeline approach
2. InftyMDB-1 – Equations – 5k
   - High resolution images of math equations
   - Black and white
   - Used to case by case assess performance of pipeline model.
3. Im-to-latex-100k – Equations – 100k
   - Downsampled images of math equations sampled from arXiv [3]
   - Gray scale
   - 484 different tokens
   - Token lengths up to 140. We limit to 70, which gets us 50k data,
   - Split into Train / Val / Test
   - Ground truth in LateX (Used in End-to-End Approach)

Discussion

Training the end-to-end model was very difficult, partly because it was so slow and partly because attention models are in general difficult to train. The end-to-end approach do have the advantage of not having to rely on the many hand-engineered rules that the pipeline model uses. We did better with the pipeline approach which offers many advantages: 1) a lot faster to train and iterate; 2) easier switch to handwriting since there exists enough data on individual symbols (the same is not true of full equations which is needed in large quantities for the end-to-end model). The need for data augmentation will likely persist in the pipeline model when switching to handwriting.

IN THE FUTURE getting the pipeline model to work on the im-to-latex-100k would let us quantitatively compare the two approaches. For the pipeline model, the next step is to implement a sophisticated structural analysis algorithm like the one proposed in [4]. For the end-to-end model the next step is to implement beam search and try to get attention working.

References

[1] Chang, J., Gupta, S., & Zhang, A. (2016). Painfree LaTeX with Optical Character Recognition and Machine Learning.
[2] Genthial, G., & Sauvestre, R. (2016) Image to Latex.
[3] Deng, Y., Kanervisto, A., Ling, J., & Rush, A. M. (2017, July). Image-to-Markup Generation with Coarse-to-Fine Attention. In *International Conference on Machine Learning* (pp. 980-989).
[4]Eto, Y., & Suzuki, M. (2001). Mathematical formula recognition using virtual link network. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on* (pp. 762-767). IEEE.

Adam Jensen (ojensen@stanford.edu)
Henrik Marklund (marklund@stanford.edu)

5 Stage Pipeline

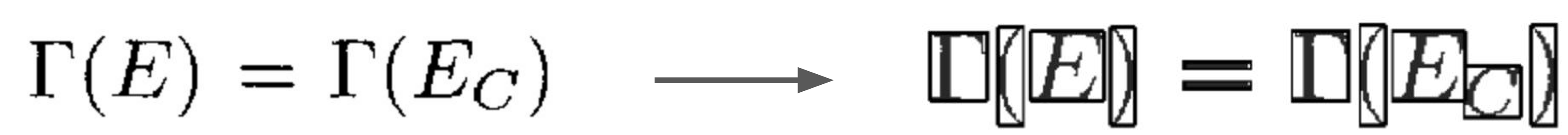
Model overview

1. Segment into individual symbols
2. Classify individual symbols
3. Resegment based on classification (e.g two minus signs on top of each other → equal sign)
4. Structural analysis (how do symbols relate to each other?)
5. Parse to Latex

Between 80k and 1.5 million parameters depending on the specific implementation.

Segmentation

We first use **Recursive Projection Profile Cutting** to segment the images into individual symbols. Since we cut vertically and horizontally this method does not separate out square roots and some characters written in italic for instance. Therefore we complement this approach by **finding the contours** of the image and separating out the characters this way. If a character has two disjoint parts (e.g ≥, = or i), this character will be split into two (as can be gleaned from the figure below). Therefore a resegmentation, based on hand engineered rules, after the individual symbols have been classified is needed.



Classify individual symbols (OCR)

Our main model is a fully connected layer to a softmax layer. We also try two different convolutional networks without improving results. We traom pm tje InftyCDB-3 dataset.

Model	Train acc	Val acc	Train loss	Val loss
FC + Softmax	0.996	0.990	0.019	0.079
Conv1	0.993	0.986	0.046	0.112
Conv2	0.996	0.988	0.022	0.089

Cross-Entropy Loss (used for both approaches, for each token prediction)

$$H(p, q) = -\sum_i p_i \log q_i$$

Data Augmentation

When first trying out the OCR on the segmented symbols in the equations it performed poorly. There is a mismatch between the images generated by the segmentation algorithm and the images of individual symbols that we are training on. We augmenting the data by zooming in, and shifting the images in horizontal and vertical directions. We then retrained on a larger set of 140k images. This made all the difference. The algorithm classifies symbols in the InftyMDB-1 equations well.

Putting it all together with structural analysis

We used a simple structural analysis which read the characters from left to right, top to bottom. It introduced a superscript if satisfying a few criteria the most important one being if the center is up to the right of the previous character (whilst checking to see if the previous character was a subscript in which case additional criteria is used. Analogous rule for subscript. The structural analysis is the bottleneck of the algorithm as of right now as it can not handle equations where a symbol have both a superscript and subscript. For all other equations the algorithm seems to perform well (since it classifies almost all of the tokens correct when we manually check performance on a subset of 30 examples).

$\{\alpha, \beta\} = \{\gamma, \delta\}$ ⟶ *\LeftBrace \alpha_{ , } \beta \RightBrace = \LeftBrace \gamma , \delta \RightBrace*

$[a,\ \infty) \subset u(S)$ ⟶ *\LeftBracket a_{ , } \infty) \subset u (S)*

*** The implementation of the end-to-end approach was done as a joint project with CS230. In CS229 we want to highlight the stage pipeline approach and the comparison between the approaches.

End-to-End

Model overview

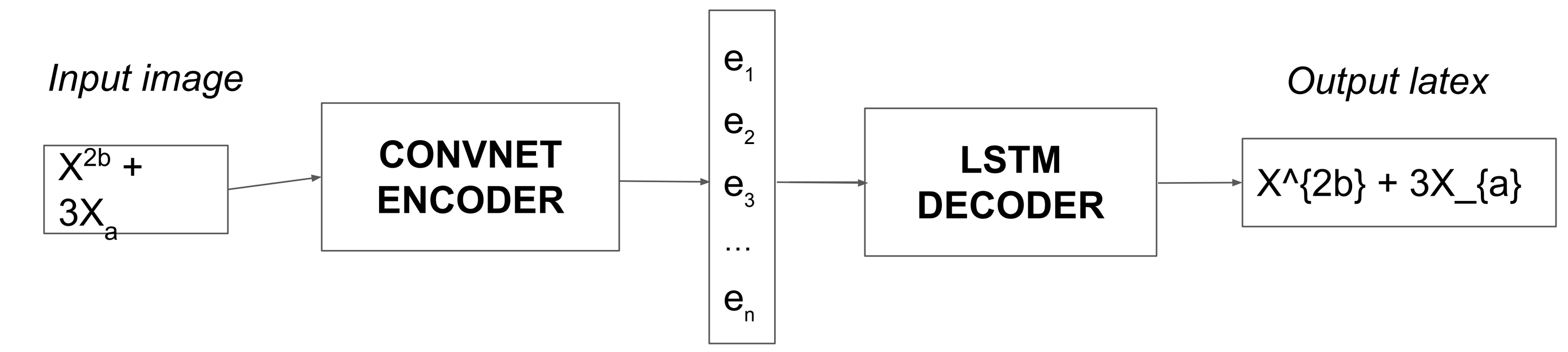
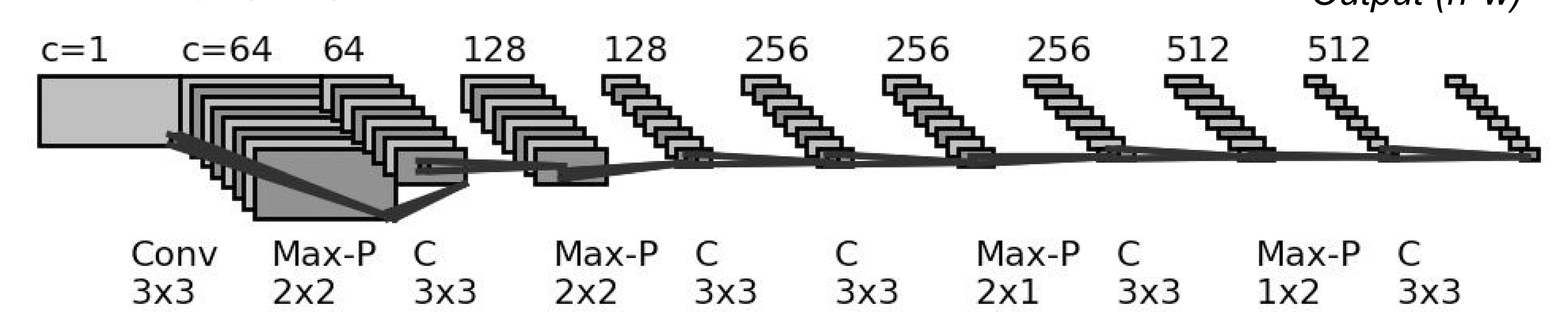


Image-to-seq model. Achieved by switching the encoder to a convnet in a translation model. Between 7 and 10 million parameters depending on the specific implementation.

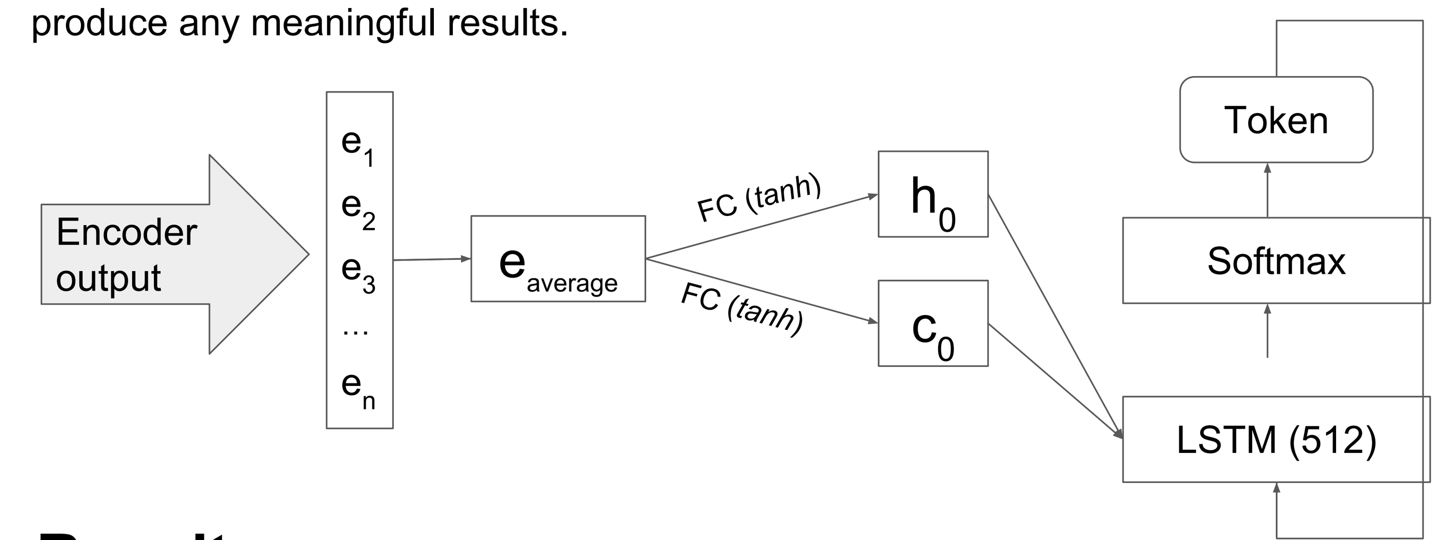
Encoder: Convolutional Net



One of three encoder architectures we tried [2]

Decoder: LSTM + softmax

The average of the output of the convnet is used as the initial state of the LSTM decoder. We also tried implementing attention over the encoder vectors but did not manage to produce any meaningful results.



Results

Hyperparameters			Results			
Encoder	**LSTM Dim**	**Down-sample**	**Token acc.**	**Edit dist.**	**Token acc (15)**	**Edit dist (15)**
Convnet1	512	0.6	19.7	26.4	39.3	7.6
Convnet1	512	0.9	**22.8**	23.7	**43.6**	6.9
Convnet2	512	0.6	17.9	27.7	36.0	7.9

Token acc (15) means the metric only concerns the first 15 tokens in the sequence. The model performs better for early tokens. Convnet2 has the two first max-pool layers removed. Experiments with attention are not included in the table.

Example predictions

Input image: $F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu + [A_\mu, A_\nu] = 0$

Predicted LateX (correct)

F _ { \mu \nu } = \partial _ { \mu } A _ { \nu } - \partial _ { \nu } A _ { \mu } .

Input image: $\Gamma(z+1) = \int_0^\infty dx\, e^{-x} x^z.$

Predicted LateX (mostly wrong)

\Gamma (s + 1) = \int _ { 0 } ^ { \infty } d s \; s ^ { s - 1 } \, .