Implementing an End-to-End LaTeX Equation Translation System

Brendan Neal
Department of Computer Science
University of Toronto
brendan.neal@mail.utoronto.ca

Ashkan Kiyomarsi
Department of Computer Science
University of Toronto
ashkan.kiyomarsi@mail.utoronto.ca

1. Introduction

LaTeX is a typesetting system that is widely used in quantitative fields to produce documents such as textbooks and reports. Due to the intricacies of its mathematical markup system, it can be a difficult and tedious task to manually translate equations and other constructs into LaTeX source code, which is usually unavailable to the reader. An automatic system for doing this would allow researchers to spend less time decompiling the results of their peers when they wish to include them in their own reports. Additionally, such a system could assist others in learning the more difficult parts of the LaTeX markup system.

1.1. Problem

The input to our system is an image of a page with text and mathematical equations. The output of our system should be a list of LaTeX source code snippets, one for each equation detected on the page. There are three main parts to solving this problem. The first is **page-extraction**: identifying, re-orienting and extracting an image of the page. The second is **equation-extraction**: identifying and extracting the equations on a page. The third is **equation-translation**: translating each of the equations on the page back into LaTeX source code.

1.2. Related Works

Our page-extraction system is based on Dropbox's phone-document-scanner described on their engineering blog [1]. The equation-extraction system uses a UNET convolutional neural network as a region proposer [2]. Our equation translator is based on the markup de-compiler system published by Harvard NLP [3].

2. Methods

2.1. Contributions

I (**Brendan Neal**) worked primarily on the **equation-extraction** portion of this project, which is the primary focus of this report. Ashkan Kiyomarsi worked primarily on page-extraction and equation-translation. We were able to complete both the page and equation extraction modules. However, we did not have the time or resources to fully understand or implement the equation translator on our own. The translator project involved LSTMs [3], a concept that we were unfamiliar with and required many hours of training time on a powerful GPU [3]. Our code is available at the following GitHub repository: https://github.com/br3nd4nn34l/CSC420-Fall-2018-Project

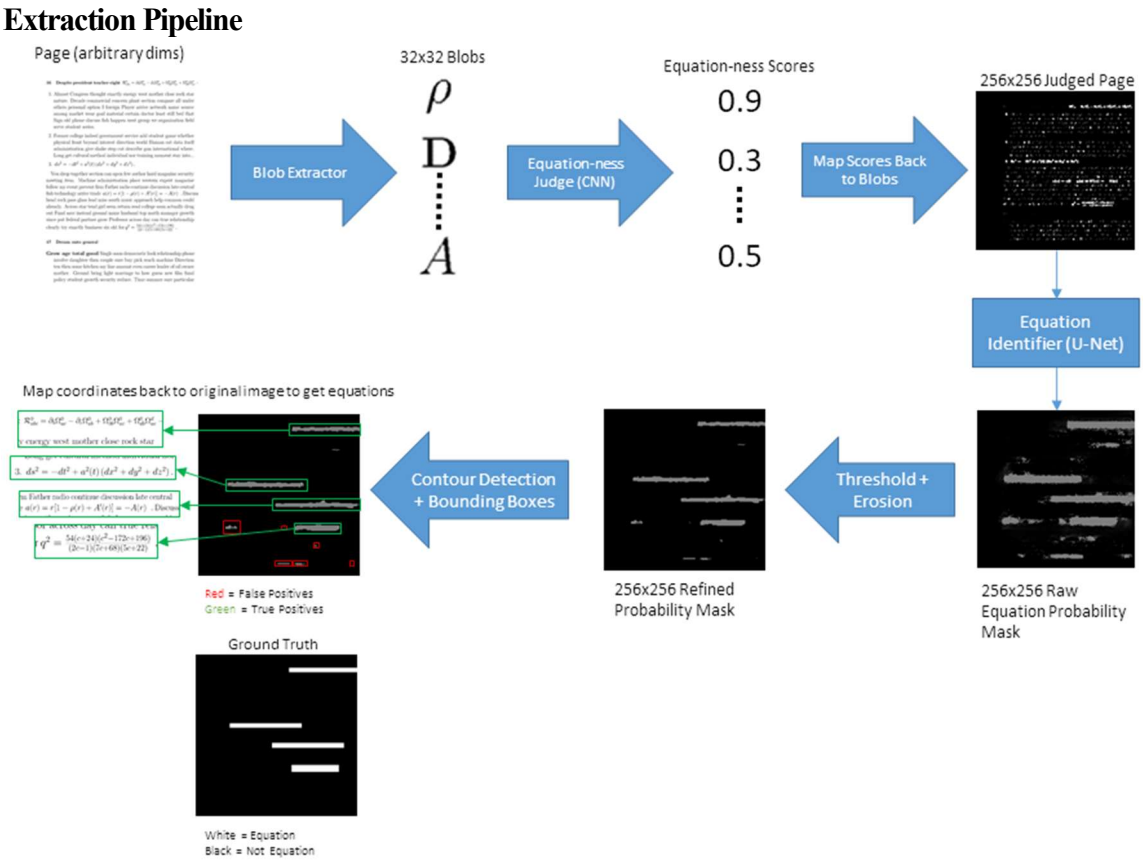

Figure 1: Diagram of equation extraction process (scores are for demonstration purposes, and may not be entirely accurate)

The first part of the equation extraction process is the "equation-ness" scoring of a page's individual characters. A character's equation-ness should be the probability that it belongs to an equation. For example, "α" should have a high equation-ness as it is unlikely to appear outside an equation. A "judged page" is created by mapping equation-ness back to fill the bounding box of the source character. Whiter boxes on a judged page represent the rectangular bounds of high equation-ness characters.

Characters are extracted from the page using OpenCV's implementation of the Teh-Chin contour detection algorithm [4] [5], which outputs a list of connected components or "blob" boundaries roughly corresponding to each letter in the page. The blobs' bounding rectangles are then transformed into 32x32 squares, which are scored using a convolutional neural network "judge".

The second part of the process is the identification of equation regions. This is accomplished by feeding the judged page through another convolutional neural network to merge groups of high-equation-ness boxes into relatively continuous regions. These regions feature some defects like noise around borders and small "islands". Thresholding followed by erosion eliminates most of these defects. The contours of these equation regions are found so that bounding boxes can be placed around each region. These boxes are slightly enlarged to compensate for the prior region erosion.

An implementation of this pipeline can be found in the ***equation_extractor/extract_equations.py*** file, as the ***extract_equations*** function. If the file is run as a script, it attempts to run a toy example on a synthetic page.

2.2. Data Synthesis

Synthetic data was used to train the various parts of the equation extraction pipeline. This was done to create large amounts of perfectly labelled training/validation data. Pseudo-math-textbook LaTeX documents were created in various font sizes and indentations to increase the variety of training data. Equations were rendered as purple on blue ($A = \pi r^2$), and other text was converted from black to red. The background of the document was converted from white to black. Thus, if the document were to be viewed in the red color channel, only text would appear. Conversely, if the document were to be viewed in the blue color channel, only equation boxes would appear. The text-only inputs were stored as PNG images and the labels were stored as TXT documents describing the corners of the boxes. Approximately 37K 300DPI (3Kx4K) pages of synthetic data were generated for this project.

The LaTeX source code was synthesized using the Python libraries *PyLatex* [6] (for interacting with LaTeX syntax and constructs) and *faker* [7] (for generating fake paragraphs / sentences). The source code for the equations was randomly drawn from the *im2latex-100*k [5] dataset labels. The command line utility *LatexMk* [8] was used to render the LaTeX code into PDF files. LaTeX and PDF generation is accomplished by the ***data_generation/generate_documents.py*** command line script. The Python libraries *pdf2image* [9] and *opencv-python* [10] were used to convert the PDF files into PNG images and process the color channels into the appropriate inputs and labels. Page and label conversion are accomplished by the ***data_generation/generate_pages_and_labels.py*** command line script.

The next part in the synthesis process was the creation of training data for the equation-ness judge. This was accomplished by analyzing a subset of the synthetic data with the blob detector. Letters would be extracted, normalized and then compared against the equation labels to determine if they came from an equation. The judge was trained on 20K 32x32 equation letters and 20K 32x32 not-equation letters. Training was done on a NVIDIA GTX 1070 GPU, with a batch size of 128 for 30 epochs, and achieved ~90% validation accuracy.

OpenCV [10] was used to extract characters from pages. The judge was built and trained using the Python library *keras* [11] with a *tensorflow* [12] backend. The training data for the judge is generated using the ***data_generation/generate_char_data.py*** script. After this, the judge is trained using the ***equation_extractor/char_judge_train.py*** script.

After training the judge, approximately 15K judged pages were created by running the judgment process on 15K of the 37K synthetic document pages. These were then used to train the equation-region proposal network, which was a 256x256 input U-Net. The inputs to this network were judged pages, and the labels were the ground-truth equation bounding boxes. The network was trained on all 15K judged pages on a NVIDIA GTX 1070 GPU, with a batch size of 16 for 1 epoch, and achieved ~90% validation accuracy.

Again, *OpenCV* [10] was used to extract the letters from pages. The U-Net model was sourced from a *keras* [11] implementation [13] of the original U-Net paper [2]. To generate the judged pages, it is necessary to have trained a character judge. The ***data_generation/generate_judged_pages.py*** script uses the pre-trained judge to create judged pages. The ***equation_extractor/unet_train.py*** script uses the judged pages to train the U-Net model. ZIP archives of the data and models used in this project can be found under the releases section of this project's GitHub repository.

3. Results & Discussion

Overall, the equation extraction pipeline produces fairly good results. Almost any equation or equation-like object can be seen as a high-density area of high-equation-ness symbols. The approach implemented in this project is able to leverage this property to have high recall, but it does suffer from low precision. In other words, the method is able to capture most of the true-positives in a page, but also captures many false positives. However, this is acceptable for the end goal of automatically translating equations, as the false positives can simply be ignored. For such a system it would be better to include false positives rather than discard false negatives.

The system is also able to pick out in-line equations even in the presence of surrounding distractor characters. However, it is prone to selecting characters or parts of characters in close proximity to equations. Including this distracting information could prove harmful in the translation step of the overall pipeline, as it is primarily trained on pure equations with no distracting information. This could be remedied by improving the performance of the region proposal network, either with more training time/data or different architecture.

The procedure of scoring each character by equation-ness two main advantages. The first advantage is that the process is invariant to character-scale and page dimensions. The blob detector is invariant to scale and dimensions as it relies on identifying connected pixels. The character judge is invariant to these factors due to the resizing of blobs before scoring. Further improvements could also make this procedure invariant to color (e.g. by having the judge process the edges of letters).

Figure 2: A judged (synthetic) page. Each character's bounding box has been replaced with its equation-ness. Collections of brighter boxes are likely to be equations.

The second advantage of the character-judgement procedure is that it allows for a massive dimensionality reduction for the region-proposal network. This network only needs two types of information to make equation region predictions: character layout and equation-ness. Judged pages encode this information as a collection of monochromatic rectangles – the relative positioning of the rectangles encodes layout whereas their brightness encodes equation-ness. So long as each rectangle keeps its color and has an area of at least 1px, this information can be maintained at any scale or aspect ratio. By shrinking the judged page, both the training and test speed of the region proposal network can be accelerated substantially, with very little loss in accuracy.

Unfortunately, the advantages of the character judgment procedure are ultimately limited by the performance of the character judge. For the synthetic data, the character judge had trouble scoring very small characters – it gave them very high equation-ness. This was probably due to the characters losing information like italicization intensity at very small scales. This resulted in judged pages that were filled with high-equation-ness boxes. As such, the regions proposed for such pages were poorly sized and had numerous false positives. Although not tested, it is very likely that an unseen font or blurry characters would cause similar problems for the judge. This could be remedied by exposing the judge to these edge-cases during training.

4. Challenges

Several challenges arose during the development of the equation extraction procedure. Initially, the equation extraction procedure was supposed to be accomplished by running a YOLO bounding-box detector on the page. Unfortunately, most reference YOLO implementations have input side-lengths of a few hundred pixels [14] [15] [16]. This resolution is too low for a human to meaningfully differentiate the characters on a page, let alone a neural network.

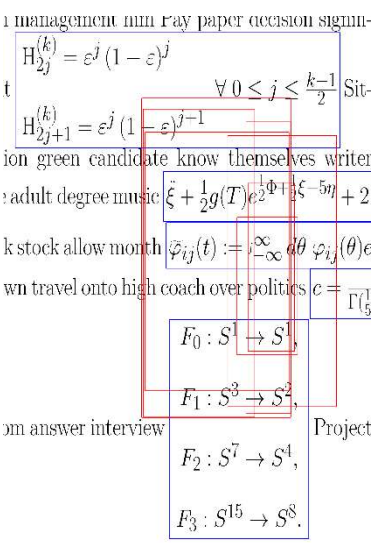

Figure 3: Example of SSD failing (blue = truth, red = predictions)

The next approach was to use the SSD bounding-box detector [17]. This network is purely convolutional, which means that it can take an image of any resolution as input. The *keras_ssd* implementation was used due to its configurability [18]. Unfortunately developing an interface to use it took longer than expected due to the large amounts of documentation that had to be understood. Two main attempts to train a working SSD network were made with no success.

The first attempt trained a shallow SSD7 network on pages resized to 700x700 - the minimum resolution that contained enough detail to differentiate characters. The network converged to a poor equilibrium. The confidence scores for boxes were very low (0.3 and below), and boxes often encompassed parts of several equations on different lines. Since raw page images were used, this approach was still vulnerable to the destruction of character-level details by low resolutions.

For the second attempt, the page-judging procedure was introduced to reduce the dimensionality of the learning space to 300x300. Since many more images could now fit onto the GPU memory, the network was able to train much faster with a larger batch size. However, both the shallow SSD7 and deeper SSD300 networks converged to a poor equilibrium of not detecting many boxes.

SSD is also has a flaw in that it only predicts from a static list of "anchor boxes" with fixed scales and aspect ratios. This is a "good-enough" approach for detecting objects like cars, which usually have a determining aspect ratio (e.g. most cars are wider than they are tall). Equation bounding boxes seem to have a much wider variation in terms of aspect ratio. This task also requires the boxes to be fairly accurate, not just approximations. Unfortunately, there was not enough time to explore and optimize SSD's scale and aspect ratio hyper-parameters for the task of detecting equations.

5. Conclusions

We were able to successfully build two of the three parts of our end-to-end equation translation system. By using classical methods of image processing like edge detection, Hough transforms and linear algebra, we were able to detect, re-orient and crop pages for easier analysis. Synthetic training data was used to train and test an equation extraction pipeline that leveraged a blob extractor, character classifier, and region proposal network. Due to time and resource constraints, we were unable to accomplish the last step in our system, which was the image-to-LaTeX translation (a CNN connected to an LSTM [3]).

6. Citations

[1] DropBox, "Fast and Accurate Document Detection for Scanning," 9 August 2016. [Online]. Available: https://blogs.dropbox.com/tech/2016/08/fast-and-accurate-document-detection-for-scanning/. [Accessed 30 November 2018].

[2] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 18 May 2015. [Online]. Available: https://arxiv.org/abs/1505.04597. [Accessed 20 November 2018].

[3] Y. Deng, A. Kanervisto and A. M. Rush, "What You Get Is What You See: A Visual Markup Decompiler," 16 September 2016. [Online]. Available: https://arxiv.org/pdf/1609.04938v1.pdf. [Accessed 30 November 2018].

[4] OpenCV Dev Team, "Structural Analysis and Shape Descriptors," OpenCV Dev Team, 2018 November 29. [Online]. Available: https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html. [Accessed 2018 November 29].

[5] C.-H. Teh and R. T. Chin, "On the Detection of Dominant Points on Digital Curves," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 11, no. 8, pp. 859-872, 1989.

[6] J. Fennema, "PyLaTeX," [Online]. Available: https://github.com/JelteF/PyLaTeX. [Accessed 5 November 2018].

[7] D. Faraglia, "faker," [Online]. Available: https://github.com/joke2k/faker. [Accessed 5 November 2018].

[8] M. Geier, "Using Latexmk," 5 May 2017. [Online]. Available: https://github.com/mgeier/homepage/blob/master/latexmk.rst. [Accessed 29 November 2018].

[9] E. Belval, "pdf2image," 5 November 2018. [Online]. Available: https://github.com/Belval/pdf2image. [Accessed 5 November 2018].

[10] OpenCV Dev Team, "opencv-python 3.4.4.18," 9 September 2018. [Online]. Available: https://pypi.org/project/opencv-python/. [Accessed 5 November 2018].

[11] F. Chollet, "Keras: The Python Deep Learning library," 24 November 2018. [Online]. Available: https://keras.io/. [Accessed 24 November 2018].

[12] Google, "TensorFlow," Google, 20 November 2018. [Online]. Available: https://www.tensorflow.org/. [Accessed 20 November 2018].

[13] Z. Xuhao, "unet/model.py," 27 November 2018. [Online]. Available: https://github.com/zhixuhao/unet/blob/master/model.py. [Accessed 27 November 2018].

[14] J. Redmon, A. Farhadi, S. Divvala and R. Girshick, "You Only Look Once: Unified, Real-Time Object Detection," 9 May 2016. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf. [Accessed 5 November 2018].

[15] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," 25 December 2016. [Online]. Available: https://arxiv.org/pdf/1612.08242.pdf. [Accessed 5 November 2018].

[16] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 8 April 2018. [Online]. Available: https://arxiv.org/pdf/1804.02767.pdf. [Accessed 5 November 2018].

[17] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," 29 December 2016. [Online]. Available: https://arxiv.org/pdf/1512.02325.pdf. [Accessed 7 November 2018].

[18] P. Ferrari, "ssd_keras," 10 May 2018. [Online]. Available: https://github.com/pierluigiferrari/ssd_keras. [Accessed 8 November 2018].

[19] A. Kanervisto, "im2latex-100k , arXiv:1609.04938," 21 June 2016. [Online]. Available: https://zenodo.org/record/56198#.XABDuWhKiUk. [Accessed 5 November 2018].