VTex: From Vision to LaTeXwith Deep Learning

Codey Sun Kerry Tu Daniel Yang Hubert Yang Jeffrey Liu
The University of Texas at Austin
{codey.sun, hubert.yang, daniel.yang.2000, kerry.jw.tu, jltexas}@utexas.edu

Abstract

Optical character recognition (OCR) is a rapidly developing area of research with many practical applications in the digital era. One such lucrative application is converting handwritten mathematical work into LaTeX code. In this paper, we implement a neural network model to convert images to LaTeX code, comparing the results with other state of the art models. In addition, a full framework is proposed for an app that allows the user to draw equations in the air using an RGB webcam and have those drawings converted to LaTeX code automatically.

1. Introduction

Even in the modern era, mathematical work is often done by hand. However, modern technology requires digital typesetting for ease of reading. LaTeX is a typesetting system used to display mathematical equations, but is tedious to work with and impractical for jotting down rapid thoughts. Despite these drawbacks, all substantial math research must be translated to LaTeX before publishing. Due to the prevalence of LaTeX in the scientific world, it would be beneficial to have a system for recognizing LaTeX from images and text. As such, the goal of this project is to utilize ML techniques to convert hand drawn images of mathematical expressions directly into its LaTeX markup sequence. In addition, this project eliminates the need for pen and paper, allowing the user to hand-write equations "virtually" by using a webcam. While others have tackled subsets of this problem with various results [6, 10, 16, 21], to our knowledge, none have implemented a full app pipeline that works with handwritten equations.

Thanks to major advances in computer vision techniques within the past decade, numerous models have been developed that are highly accurate in detecting text within images. However, successfully translating an entire hand written mathematical expression into its corresponding LaTeX code sequence remains a formidable challenge. This is due to the fact that in addition to correctly determining each symbol, one must also consider the spatial arrangement of each symbol relative to the others. To this end, our model utilizes and encoder-decoder transformer model equipped with positional encoding in order to store spatial relationships between symbols.

2. Related Work

2.1. Air Drawing

With the progression of hand-tracking technology, many researchers have expressed interest in hand tracking as a means of computer interaction, as hand tracking is a natural way to express one's ideas. Studies show that visual cues can improve the remote work experience greatly [17, 19]. Kim et al. have experimented with 3D sketching using hand tracking by having the user build up "air scaffolds" to constrain planes to draw on [11]. Chen, et al. developed an application to input Chinese characters using a smart glasses's camera [2]. Gulati et al. created a real-time drawing application using only a single RGB webcam, MediaPipe Hands, and OpenCV [7]. Combining real-time air drawing and optical character recognition to write mathematical expressions will be the main technical gap we're trying to address.

2.2. MediaPipe

MediaPipe Hands is a finger-tracking solution utilizing MediaPipe - an open-source API by Google that provides numerous ML solutions - that achieves remarkable real-time performance using the processing power of only a mobile phone [22]. In this study, we utilize MediaPipe Hands in order to track the hand strokes by sampling landmarks on the user's hands once per frame. MediaPipe Hands was chosen due to its robustness as well as its lightweight cost, allowing our pipeline to perform in a live environment.

2.3. Optical Character Recognition

OCR is the process of classifying and encoding characters within digital images. Due to its importance in numerous fields, OCR is a well-matured topic computer vision where various successful approaches have been developed. Neural Networks (NN) are one of the techniques among these approaches, and is the strategy that we employ in the VTex

pipeline. Previous works demonstrate the success that NNs have found in OCR. A study by Sabourin [15] shows that a multilayer perceptron network (MLP) is able to outperform other techniques in OCR such as dynamic contour warping classifiers. In recent years, convolutional neural networks such as ResNet [8] and DenseNet [9] have dominated this field as well as many other image classification problems.

2.4. Encoder-Decoder Models

Encoder-decoder models are by far the most popular architecture for natural language processing (NLP) applications, and math expression conversion is no exception. This architecture consists of two parts: an encoder and a decoder. As the names suggest, the encoder "encodes" the input (e.g. an image) to a hidden layer, and the decoder "decodes" the hidden layer to a sequence (e.g. a sentence). Genthial proposed an encoder-decoder model to tackle the `IM2LATEX-100K` dataset, using a standard CNN for the encoder and a long short-term memory (LSTM) model for the decoder [6]. A beam search is then performed to predict a LaTeX sequence given an input image. Wang improves the encoder-decoder by using a DenseNet encoder and adding spatial and channel-wise attention to the LSTM decoder [10]. Peng utilizes a recurrent neural network (RNN) for the decoder and augments the encoder by using a graph neural network (GNN) to model spatial relationships between image symbols [16]. The model proposed by Wang attempts to improve on previous models such as seq2seq by introducing a two-step training process, separating token-level and sequence-level training [21]. Yan turns the encoder-decoder into a fully convolutional network by using both a convolutional encoder and convolutional decoder, exploiting parallel computation for increased efficiency [20]. Perhaps closest to VTex, the BTTR model utilizes a transformer decoder, vastly improving efficiency and accuracy compared to RNNs [23].

While this paper does not aim to improve upon current state of the art models, it does apply the current models to consumer-driven settings. Rather than document analysis (as is the goal by many competitions hosted by ICDAR), this paper targets document creation.

3. Method

3.1. Airdrawing

For hand tracking and gesture recognition, MediaPipe is an open source API that provides many real-time features. The team will use MediaPipe Hands, which uses "machine learning to track 21 3D landmarks [on the hand] from just a single frame" [1]. With these 21 3d landmarks, different gestures such as raising a specific finger can be detected. The airdrawing can then be programmed based on the user's hand landmarks shown in Figure 1. For example, if we want to detect whether index finger is raised, then we only need to check whether landmark 8 is higher than landmark 6 or 5.

The team will follow similar approaches taken from Gulati et al's drawing application [7] for airdrawing. We used OpenCV to capture videos from the user's camera. Each frame of the video is then processed with MediaPipe Hands to track 21 3d landmarks. Given the relative location of each landmark, our code either draws more points on the white canvas, erase all drawings, or screenshot the current white canvas and feed it into the transformer to output the corresponding LaTeX . Our airdrawing runs indefinitely until the user press the "Esc" key.

3.1.1 Airdrawing Modes

The airdrawing program has 3 modes: draw, clear, and feed.

The user enables drawing mode if the only finger that the user raises is the right index finger. If 2 or 3 right fingers (can include right index finger) are raised up, then drawing would not happen. This way, the user can have the option to pause drawing and move his or her index finger to the next letter location. Then, the user could continue drawing by lifting only the index finger up.

The implementation of the pixel drawing is inspired by this GeeksForGeeks post [5]. Each time the drawing mode is activated, the position of the right index fingertip, which is MediaPipe Hand's landmark 8 in 1, is stored in a deque. At the end of each frame processing, we used a for loop that draws a line between consecutive stored points. This allows the strokes to look continuous even though the index fingertip is sampled at each frame. Figure 2 shows what drawing looks like.

The clear mode is activated when the user raises 4 left-hand fingers (no thumb). This will clear all strokes on the tracking image and the white canvas, as shown in Figure 4.

The feed mode saves the white canvas as a .bmp file and feed it into the transformer model for LaTeX conversion, as shown in Figure 3.

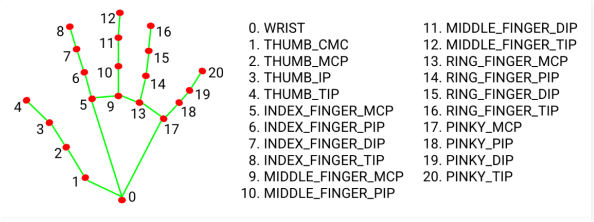


Figure 1. MediaPipe Hand's 21 hand landmarks

3.2. Image to Latex

Given images captured via airdrawing, a transformer encoder-decoder network is used to translate the images to LaTeX . The VTex network borrows much of its ideas

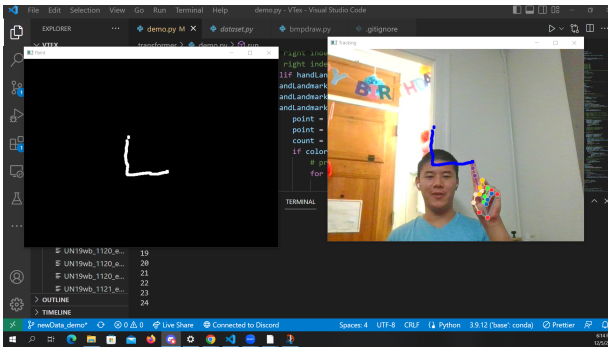


Figure 2. When the right index finger is the only right finger raised, then the drawing will happen at its tip

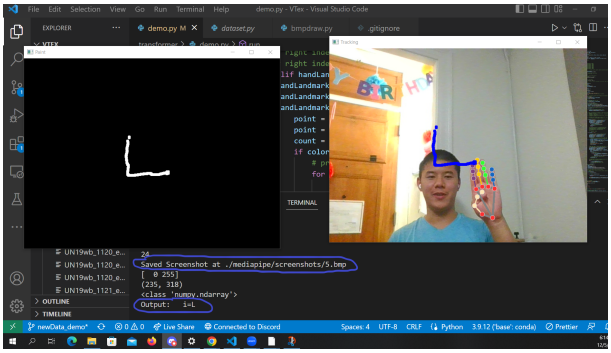


Figure 3. When 4 right fingers are raised (no thumb), then the canvas will be saved as BMP image, and the program will pass the BMP image to the transformer for LaTeX conversion. The program outputted i=L

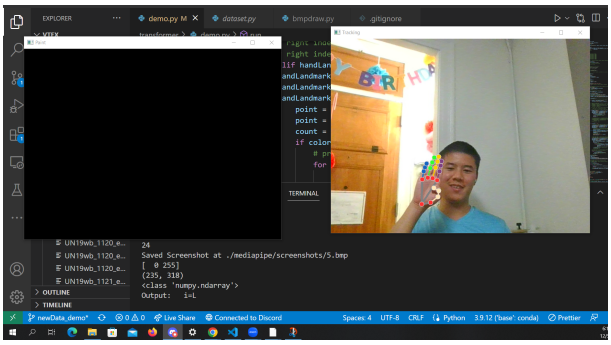


Figure 4. When 4 left fingers are raised (no thumb), then the canvas will be cleared

from the Bidirectionally Trained Transformer (BTTR) without adopting the bidirectional training strategy [23]. The model is implemented from scratch in PyTorch using the architecture shown in Figure 5.

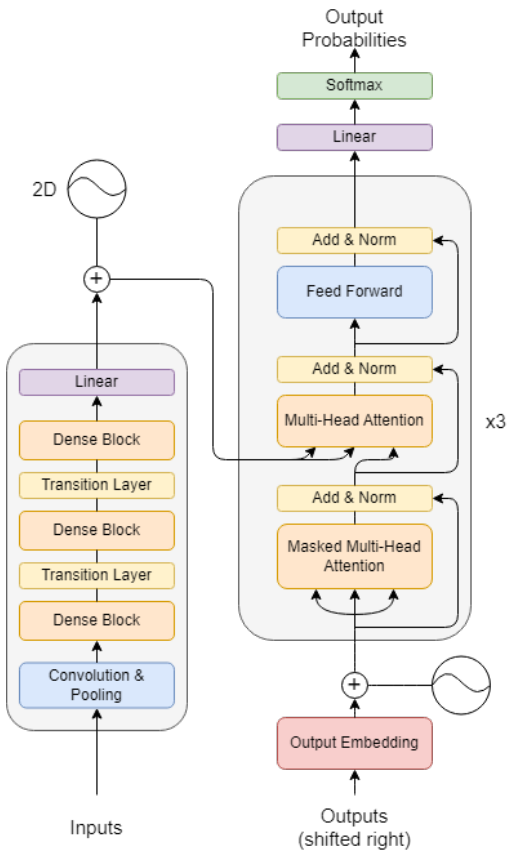


Figure 5. Architecture of the VTex translation network

3.2.1 Decoder

The decoder is constructed in the same manner as the original transformer paper [18]. First, the decoder input is embedded to convert the target sequence to a d_{model} dimension vector. Then, positional encoding is added in the manner described in Section 3.2.3. This is then fed into the decoder layers.

Each decoder layer consists of three sublayers: a masked multi-head attention module that performs self-attention on the decoder input, a multi-head attention module that performs attention on the encoder output, and a feed-forward network. Residual connections and layer norms are appended to each sublayer.

The multi-head attention modules utilize "Scaled Dot-Product Attention" as described in [18]. This attention mechanism matches a set of queries to key-value pairs. Given a set of queries Q, keys K, and values V, the attention can be computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}V) \quad (1)$$

Furthermore, multi-head attention splits Q, K, and V into h different heads with $d_{model}/h = d_k$ dimension vectors,

performing the attention computation shown in Equation 1 on each head in parallel. The heads are then concatenated for the final attention values. Thus,

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2}$$

where W^O, W_i^Q, W_i^K, W_i^V are projection matrices that reshape the vector dimensions to d_{model}, d_k, d_k, d_k dimensions, respectively.

As the decoder input (Output Embedding in Figure 5) contains the full target sequence at each time step, it is imperative for training that tokens do not pay attention to future tokens, as this does not reflect how the model will perform in practice. As such, the decoder input is masked such that tokens can only pay attention to past tokens and the input image.

The second multi-head attention module utilizes the first attention module's output as the queries and the encoder's output as the keys and values. Finally, this output is fed into the feed-forward layer, which expands the d_{model} dimension vector to dimension d_{ff} and back to d_{model}.

The VTex decoder consists of 3 decoder layers, each with $h = 8$ heads, model dimension $d_{model} = 256$, feed-forward inner layer dimension $d_{ff} = 1024$, and dropout rate of 0.3.

3.2.2 Encoder

The original transformer fails when we introduce images as the source input. Applying self attention to every pixel in an image is far too demanding for modern hardware. As such, image analysis is often done by convolutional neural networks (CNN). For the purposes of encoding, a CNN performs well in feature extraction and classification problems. VTex uses the DenseNet architecture to extract information at multiple scales of the mathematical symbols present in an image [9].

DenseNet is based on the concept of "dense connectivity," meaning each layer in the network is connected to every other layer in a feed-forward fashion. Doing so promotes gradient propagation, allowing the network to learn features at all complexity levels. This allows DenseNet to outperform other CNNs when training data is limited, and this is why VTex uses it. Specifically, VTex uses the DenseNet-B model noted in [9], which adds "bottleneck" layers (i.e. 1x1 convolutions) to reduce input features and improve computational efficiency. Finally, a 1x1 convolution is applied to reshape the DenseNet output to fit the decoder.

While a traditional DenseNet is extremely prone to overfitting, VTex greatly simplifies the model by using .bmp binary images. As for the hyperparameters, this paper's implementation uses 3 bottleneck dense blocks with a growth rate of $k = 24$, block depth of 16, and compression factor of $\theta = 0.5$.

3.2.3 Positional Encoding

In a traditional transformer, the decoder must use a positional encoding to embed sequence ordering or tokens. VTex adopts the sinusoidal positional encoding presented in the original transformer for this purpose [18]. For a d dimension encoding, the positional encoding can be defined as:

$$\begin{aligned} PE_{(pos,2i,d)} &= \sin(pos/10000^{2i/d}) \\ PE_{(pos,2i+1,d)} &= \cos(pos/10000^{2i/d}) \end{aligned} \tag{3}$$

where pos is the position of the token and i is the dimension index. This positional encoding is added to the target output before any decoder layers.

In encoding an image, the encoder must account for both a symbol's classification and the symbol's relationship with other symbol (e.g. within the same fraction, in a superscript, etc.). These relationships often cannot be extracted using only a simple CNN. While others have used graph neural networks to learn these relationships [16], VTex's encoder models these relationships as positional encodings much like the decoder does. Specifically, Equation 3 is applied in both the height and width axes of the input image such that for a pixel (x, y):

$$\begin{aligned} PE_{((x,y),2i,d)} &= [PE_{(x,2i,d/2)}; PE_{(y,2i,d/2)}] \\ PE_{((x,y),2i+1,d)} &= [PE_{(x,2i+1,d/2)}; PE_{(y,2i+1,d/2)}] \end{aligned} \tag{4}$$

In the encoder, the positional encoding is added to the CNN output before being fed into the decoder.

3.2.4 Prediction Step

Because the output of a transformer is the next token in a sequence, the search space grows exponentially with the sequence length. Performing an exhaustive search for every image is impractical with modern hardware. Often, inferencing with a transformer is a balance between accuracy and computational efficiency. In this paper, two prediction methods are implemented to infer an output sequence from an image.

A greedy approach starts with a `<SOS>` (start of sequence) token and predicts the most probable next token, iteratively adding the output token to the input sequence until the transformer predicts a `<EOS>` (end of sequence) token. This is the simplest approach to inferencing with a transformer. However, there are cases where a less probable token will become more probable as the transformer uncovers more of the sequence. For example, at time step

1, a transformer will predict the first token as x, but later at time step 5, the transformer will realize that the first token should be y. In this manner, a single wrong classification will disrupt the rest of the output sequence.

A beam search approach remedies this problem by exploring multiple possible predictions, i.e. "beams", for each time step. With a beam size of k, the top k sequences across all time steps will be considered for the decoder input. In addition, longer sequences will be penalized to prevent the inferencing from never ending. Inferencing terminates when all beams contain a `<EOS>` token. Then, the most probable beam is chosen as the final output sequence.

This paper implements beam search with size 10. The results using greedy vs. beam search are discussed in Section 4.3.2.

3.3. Data Collection

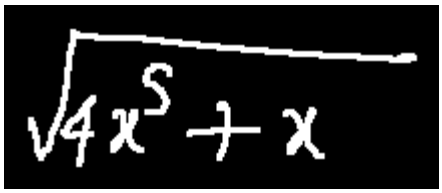
Two datasets were considered for training the model. The public `IM2LATEX-100K` dataset provides 103,556 pairs of LaTeX equations with pictures of their rendered images [3]. The large size of this dataset makes it a prime candidate for training a complex neural network like VTex. However, its rendered images are of printed text–not handwritten equations. As such, it may not perform well when paired with the virtual drawing component of the VTex pipeline. To remedy this, transfer learning can be applied using a handwritten math symbols dataset, fine-tuning the CNN encoder [13].

Alternatively, the Competition on Recognition of Handwritten Mathematical Expressions (CROHME) provides a dataset of 8836 training images of handwritten mathematical expressions [12]. CROHME does not provide data in the format VTex desires. Rather, the input is an InkML or Label Graph (LG) file that represents strokes captured by a tablet, and the output is a tree-based symbolic representation named Symbol Layout Graph (symLG). Fortunately, CROHME provides tools to convert these file types into images (.bmp) and LaTeX code, respectively, shown in Figure 6. As VTex's final goal is to process handwritten equations, the CROHME dataset was chosen despite its smaller size. This would present challenges as documented in Section 4. The CROHME 2014 test set, containing 986 images, is used as the validation set.

4. Results

4.1. App Pipeline

The application will be stored on our GitHub repo. We have a transformer folder that includes all the transformer architecture and data directories for training and testing. The demo.py inside the transformer folder is the application we wanted to demo. It incorporates mediapipe, and when the user uses feed mode, the drawing canvas will be fed into the transformer to output LaTeX code in the terminal.



```
\sqrt { 4 x ^ { 5 } + x }
```

Figure 6. Example converted .bmp image with its ground truth from the CROHME training set

To use the demo, please run demo.sh script in our GitHub, which will create a virtual environment for the user, download all the packages in requirements.txt, and run demo.py in the transformer directory to start the application. Then follow the GitHub instruction or Section 3.1.1 for enabling different MediaPipe modes. The current iteration of the application does not work on M1 Macs.

The application works as follows: the program first loads the transformer model checkpoint in the root directory. Then air drawing function is run that processes each frame of the camera video in real-time to analyze the user's hand. When the user enables feed mode, the drawing canvas will be saved as passed to the checkpoint model for LaTeX conversion.

4.2. App Performance

The application overall performs well. Figure 7 shows one of the results we got and that the transformer correctly convert our drawing of $e = mc^2$.

There are some drawbacks, of course. For example, the whole hand has to be in the camera for MediaPipe to accurately identify the index fingertip. Furthermore, since MediaPipe constantly detects hands in each frame, there is no easy way to detect "pen up" operation. To counter this, the program enables drawing mode when the only finger raised is the right-hand index finger. Lifting the middle finger along with the index finger on the right hand can can pause drawing. This is slightly inconvenient for the user but is the only feasible solution we can come up with.

One inconvenience of this application is that the user has to point the right index finger upward to enable drawing. This is because our implementation enables drawing when landmark 8 is higher than landmark 6 in Figure 1. Therefore, the user can not tilt his hand to the point that makes landmark 8 less than landmark 6 when this index finger is raised.

4.3. Neural Network Results

4.3.1 Training

As mentioned in Section 3.3, the model is trained on the CROHME training dataset and validated on the CROHME

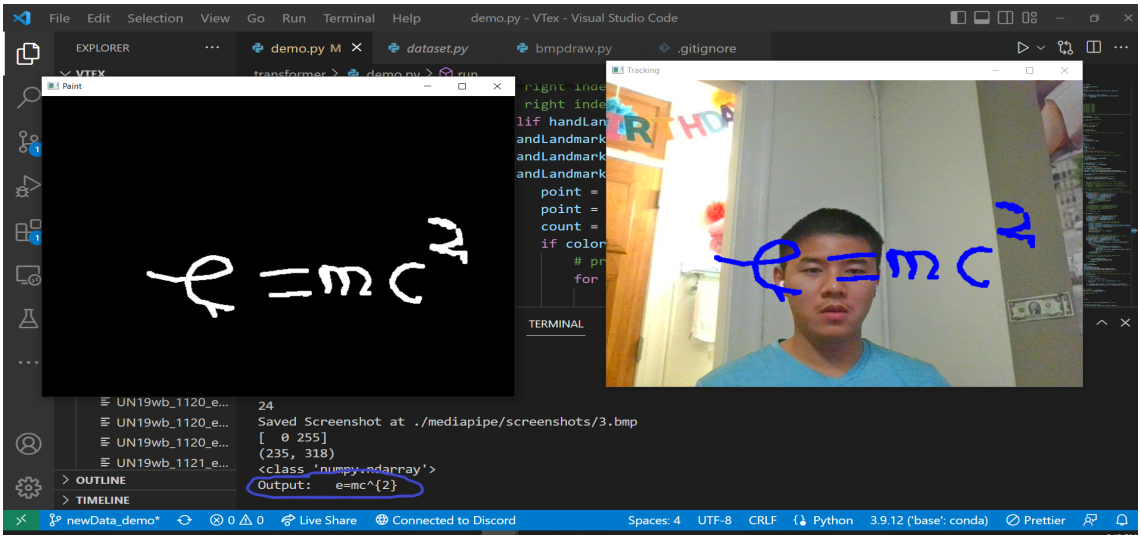


Figure 7. Example showing the correct LaTeX conversion of our airdrawing of Einstein's famous equation. The program outputted `e=mc^{2}`

2014 test set. Figure 8 shows the training progress of the transformer on two NVIDIA GeForce GTX 1070 GPUs using a batch size of 8. The red lines depict the validation loss, while the green lines depict the training loss. The training shows a significant improvement in both the training and validation loss until approximate epoch 20, when the training loss levels out and no longer decreases, and the validation loss becomes very noisy. Training is stopped after 500 epochs when it becomes clear that the total loss will no longer improve significantly.

While the large fluctuations in validation loss is concerning, it was disregarded due to the loss's overall downward trend. Nonetheless, there are several possible reasons for this variance, including large learning rates, small batch sizes, and model complexity. We suspect that in this case, the variance is caused by a small validation set (986 vs. 8836 in the training set). If so, this can be easily remedied by transferring some data from the training set to the validation set. Alternatively, the overall training set can be dramatically increased by means of transfer learning, perhaps with the aforementioned `IM2LATEX-100K` dataset [3]. Furthermore, while not implemented, an ensemble approach can aggregate the best performing epochs to reduce inconsistencies caused by the variance.

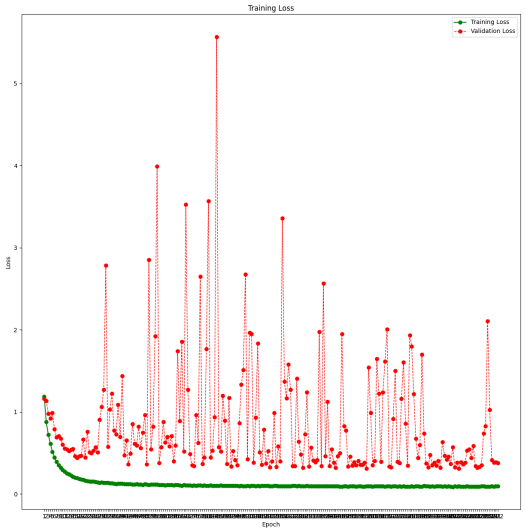


Figure 8. Training Progress from Epochs 1 - 222

4.3.2 Evaluation

As with other natural language processing applications, model evaluation becomes difficult when considering that a single input can be translated into many different correct sequences. For LaTeX possible translations increase dramatically due to optional formatting tokens, e.g. `x_i` vs. `x_{i}` vs. `{x}_i`. This is why CROHME evaluates using the symLG format rather than a LaTeX sequence [12]. For more general applications, evaluation paradigms such as BLEU account for other translation possibilities [14].

As of the writing of this paper, the tools provided by CROHME to translate LaTeX to symLG and thus evaluate a score are unavailable. As an alternative, this paper evaluates the VTex model using an approximation of the expression recognition rate (ExpRate). Specifically, the Levenshtein distance between the output and ground truth sequences is calculated, providing the number of transformations needed

Model	ExpRate	2016	2019
VTex (beam search)	0 error	20.65%	24.14%
	$\leq$1 error	34.69%	39.77%
	$\leq$5 error	64.80%	72.64%
	$\leq$10 error	84.84%	90.46%
VTex (greedy)	0 error	21.67%	24.60%
	$\leq$1 error	35.40%	39.31%
	$\leq$5 error	62.97%	70.80%
	$\leq$10 error	80.47%	87.36%
BTTR	0 error	52.31%	52.96%
	$\leq$1 error	63.90%	65.97%

Table 1. Test Result Accuracy for 2016 and 2019 CROHME Datasets

to convert the output to the ground truth. Table 1 shows test accuracy from the 2016 and 2019 CROHME datasets for different margins of error. Only approximately 20-24% of results had no errors at all, but once a margin of error of 1 is taken into account, the effective accuracy jumps to 35-40%.

The beam search algorithm performs better in the $\leq$5 and $\leq$10 error metrics, but greedy performs better in the 0-1 error metrics. In theory, beam search outperforms greedy in longer sequences, as beam search gives the decoder more chances to recover from an incorrect prior prediction. We believe that the increase in performance in the $\leq$5 and $\leq$10 error metrics is due to this property as test sequences falling into this category are likely to be longer. However, we are unsure why greedy performs better in the 0-1 error metrics. It is possible that the beam search algorithm misleads the decoder into diverging from the correct path, particularly with larger beam sizes.

Unfortunately, our model did not perform well compared to BTTR. BTTR adds several additional features VTex does not, including bidirectional training [23]. In addition, BTTR is built on the PyTorch Lightning framework, which may provide improvements in training.

5. Future Work

As for air drawing, the interface could be made more accurate and user friendly. The current version of the air drawing would often take actions unwanted by the user through misinterpreting gestures. For example, the application would occasionally take screenshots while the user was still in the middle of writing an equation, due to false detecting the screenshot gesture. Also, due to nature of writing in the air, the resulting writing looks somewhat shaky, so we could implement some form of smoothing. Furthermore, there is currently no user-friendly interface for the application, so it would be difficult to navigate the gestures and controls for a new user.

On the neural network side, the accuracy is not sufficient for practical use. While a $\leq$1 error of 35% may be useful to a user unfamiliar with LaTeX, experienced users of LaTeX will likely find VTex largely useless.

One simple way to improve the model is to use more data. Currently, each CROHME competition uses the same training set, only changing the test set each year. As a result, the amount of data available is sparse. This can be remedied by either 1) manually collecting more data or 2) applying transfer learning. The second option is much more feasible as similar datasets with much more images exist, e.g. `IM2LATEX-100K`.

Additionally, one can aim to improve the architecture of the model. The current model is admittedly outdated as more modern methods of combining CNNs and transformers have been developed. Most recently, vision transformers have revolutionized the image recognition field, achieving comparable results with modern CNNs while utilizing fewer resources [4]. The current architecture of combining a DenseNet CNN and transformer decoder results in a complex model that is prone to overfitting. A vision transformer can remedy this as it is a much simpler model. However, as vision transformers require extremely large amounts of data, transfer learning is necessary for good results.

6. Conclusion

This paper presents a novel application of hand tracking and document analysis that provides a way to write LaTeX equations using only a webcam. This app pipeline consists of two steps: air drawing via hand tracking and image-to-LaTeX conversion via a neural network.

Although the current airdrawing is not user-friendly and convenient, it still works really well. The user could still draw any expression that he wants with only some minor inaccurate hand detections. Due to the nature of airdrawing, the resulting strokes are shaky. More time could have spent on smoothing the strokes since this can improve LaTeX prediction as well.

The neural network's performance lags behind its state-of-the-art competitors with only a 20% ExpRate. However, it still proves useful for novice applications with a ≤ 5 error rate of 63%. Training the network on a larger dataset could improve the performance, as could changing up the neural network architecture.

7. Team Work Assignment

Codey:

- Designed neural network architecture
- Implemented neural network via PyTorch framework, including DenseNet and transformer
- Implemented testing code and collected testing results

- Wrote neural network Related Work, Methodology, and Results sections

Kerry:

- Load / Extract CROHME dataset for training
- Implemented training code for neural network
- Trained neural network on CROHME dataset
- Wrote Training, Evaluation, and Future works sections

Daniel:

- Setup MediaPipe Hands and used it to create airdrawing application
- Integrated airdrawing application with our transformer model for LaTeXconversion
- Wrote airdrawing / MediaPipe section in Method and Results section.

Hubert:

- Proposed hand written mathematical expression to LaTeX conversion idea
- CROHME dataset image/ground truth pair retrieval
- Wrote introduction, MediaPipe, and optical character recognition related works sections

Jeffrey:

- Helped debug and develop transformer neural network architecture
- Cleaned up the repository and added support on running the project
- Wrote technical gaps section

References

[1] Google AI. Mediapipe hands, 2014. Overview and documentation page for MediaPipe Hands application. 2

[2] Yung-Han Chen, Chi-Hsuan Huang, Sin-Wun Syu, Tien-Ying Kuo, and Po-Chyi Su. Egocentric-view fingertip detection for air writing based on convolutional neural networks. *8th Global Conference on Consumer Electronics*, 2021. 1

[3] Yuntian Deng, Anssi Kanervisto, Jeffrey Ling, and Alexander M. Rush. Image-to-markup generation with coarse-to-fine attention, 2016. 5, 6

[4] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2020. 7

[5] geeksforgeeks.org. Create air canvas using python-opencv. 2022. 2

[6] Guillaume Genthial and Romain Sauvestre. Image to latex, 2016. 1, 2

[7] Shaurya Gulati, Ashish Kumar Rastogi, Mayank Virmani, Rahul Jana, Raghav Pradha, and Chetan Gupta. Paint / writing application through webcam using mediapipe and opencv. *International Conference on Innovative practices in Technology and Management*, 2022. 1, 2

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[9] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 4

[10] Shenling Wang Jian Wang, Yunchuan Sun. Image to latex with densenet encoder and joint attention. *Procedia Computer Science*, 147(1):374–380, 2019. 1, 2

[11] Yongkwan Kim, Sang-Gyuu, Jonn Hyub Lee, and Seok-Hyung Bae. Agile 3d sketching with air scaffolding. *CHI conference on Human Factors in Computing Systems*, pages 1–12, 2018. 1

[12] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchère, and Utpal Garain. Icdar 2019 crohme + tfd: Competition on recognition of handwritten mathematical expressions and typeset formula detection, 2019. 5, 6

[13] Xai Nano. Handwritten math symbols dataset, 2016. 5

[14] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics. 6

[15] Michael Sabourin and Amar Mitiche. Optical character recognition by a neural network. *Neural Networks*, 5(5):843–852, 1992. 2

[16] Ke Yuan Zhi Tang Shuai Peng, Liangcai Gao. Image to latex with graph neural network for mathematical formula recognition. *International Conference on Document Analysis and Recognition*, 12822(1):648—-663, 2021. 1, 2, 4

[17] Theophilus Teo, Gun A. Lee, mark Billinghurst, and Matt Adcock. Hand gestures and visual annotation in live 360 panorama-based mixed reality remote collaboration. *International Conference on Computer-Human Interaction*, pages 406–410, 2018. 1

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. 3, 4

[19] Erroll Wood, Jonathan Taylor, John Fogarty, Andrew Fitzgibbon, and Jamie Shotton. Shadowhands: High-fidelity remote hand gesture visualization using a hand tracker. *International Conference on Interactive Surfaces and Spaces*, pages 63–75, 2016. 1

[20] Zuoyu Yan, Xiaode Zhang, Liangcai Gao, Ke Yuan, and Zhi Tang. Convmath: A convolutional sequence network for mathematical expression recognition. In *2020 25th International*

Conference on Pattern Recognition (ICPR), pages 4566–4572, 2021. 2

[21] Jyh-Charn Liu Zelun Wang. Translating math formula images to latex sequences using deep neural networks with sequence-level training. *International Conference on Document Analysis and Recognition*, 24(1):63–75, 2021. 1, 2

[22] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. Mediapipe hands: On-device real-time hand tracking. *CoRR*, abs/2006.10214, 2020. 1

[23] Wenqi Zhao, Liangcai Gao, Zuoyu Yan, Shuai Peng, Lin Du, and Ziyin Zhang. Handwritten mathematical expression recognition with bidirectionally trained transformer. *International Conference on Document Analysis and Recognition*, 12822(1):570—-584, 2021. 2, 3, 7