

Image2 \LaTeX Application: Examination of Approaches to Accuracy Assessment

Gu Zhuangcheng (3035827110) - *zcgu@connect.hku.hk*

He Zixuan (3035827861) - *hzx0517@connect.hku.hk*

Deng Jiaqi (3035832490) - *jiaqideng@connect.hku.hk*

Abstract

\LaTeX is a widely used formatted-formula-generating platform, but it takes work to get familiar with it as it requires users to learn about \LaTeX code. Hence, our purpose for this project is to train a model with high accuracy and build a desktop application based on our model to convert images of math formulas into \LaTeX sequences. This problem could be categorized as machine translation. Based on this idea, we use the encoder-decoder structure due to its capability in machine translation applications. We adopt the convolutional neural network (CNN) as the encoder and transformer decoder. To examine the performance of our model quantitatively, we use BLEU and edit distance as our metrics. The desktop application is developed and could be deployed on macOS.

Highlights

- We did four model iterations with experiment data and fine-tuning for our final model
- The model approaches the state-of-art models' performance on the dataset
- MacOS and Command Line Interface (CLI) desktop application developed

1 Introduction

Background

\LaTeX is a software system for formatting and is widely adopted in academics. One common usage is to format mathematical expressions. However, when dealing with a large number of math expressions, it could be a dull job to type in the corresponding \LaTeX expressions manually. Also, in some cases, people may spend a certain amount of time searching online about the proper grammar since not everyone is familiar with all expressions in \LaTeX . The ability to extract math expressions from academic articles and convert them into \LaTeX expressions is critical in certain circumstances.

Project Objectives

Our project aims to solve this conversion problem from image to LaTeX expressions. Since the essence of the problem is a machine translation problem, we propose to use the popular model structure (encoder-decoder structure [1]) that performs well in similar machine translation problems. After several experiments, we finalize our model. And the model is integrated into a desktop application on macOS platforms.

Target Application

We want to build an application for converting formula images to LaTeX code by letting users take a screenshot of the maths formula they want to convert. The expected user input is a formula image containing a typed or computer-generated formula. Since there will be differences in font size, background color, etc. between user-input images and our dataset, our application also provides a series of pre-processing operations on the user-input image before feeding it to the model. After that, our application will generate the LaTeX code for this formula. Two versions of applications with the same functionalities are implemented in our project, a CLI(command line) version and a macOS version. More details of the app design will be introduced in the app development section.

Dataset

We will use the dataset IM2LATEX-100K [1], a prebuilt dataset for OpenAI’s task on Kaggle. The dataset contains around 100K formulas, and images with its formulas were parsed from LaTeX sources provided by KDD Cup 2003. Those images and formulas are split into train, test, and validation sets, with information stored in three CSV files (for train, test, and validation dataset, correspondingly). Each of these three files contains two columns, with the first column storing the LaTeX formula and the second column having the name of the mathematical formula’s image, which provides paths to find the actual image. The following table 1 consists of examples from the training CSV file.

Label	Image
$\tilde{\gamma}_{\text{hopf}} \simeq \sum_{n>0} \tilde{G}_n \frac{(-a)^n}{2^{2n-1}}$	66667cee5b.png
$(\mathcal{L}_a g)_{ij} = 0, \quad (\mathcal{L}_a H)_{ijk} = 0,$	1cbb05a562.png

Table 1: Examples from the CSV file

All formula images are stored together in another file. Each image is a PNG image with the formula in black, and the rest of the image is transparent. The following image 1 is one example from the dataset.

$$\int_0^{r_0} r dr (\tilde{T}_{(0)\theta}^\theta + \tilde{T}_{(0)r}^r) = r_0^2 \tilde{T}_{(0)r}^r(r_0) = r_0^2 \left[\frac{A^2(r_0)}{2e^2} + \frac{\alpha}{2} \Lambda^2(r_0) \right].$$

Figure 1: Sample Image from the IM2LATEX Dataset

2 Model Architecture

Overview

Inspired by the work [2], we adopt the seq2seq model 2. There are mainly two components in our model: encoder and decoder. The encoder is primarily made up of a convolutional neural network (CNN) with positional encoding [3]. And we use the transformer-based decoder as the decoder. The network would input the tensor of images and output the predicted LaTeX sequence.

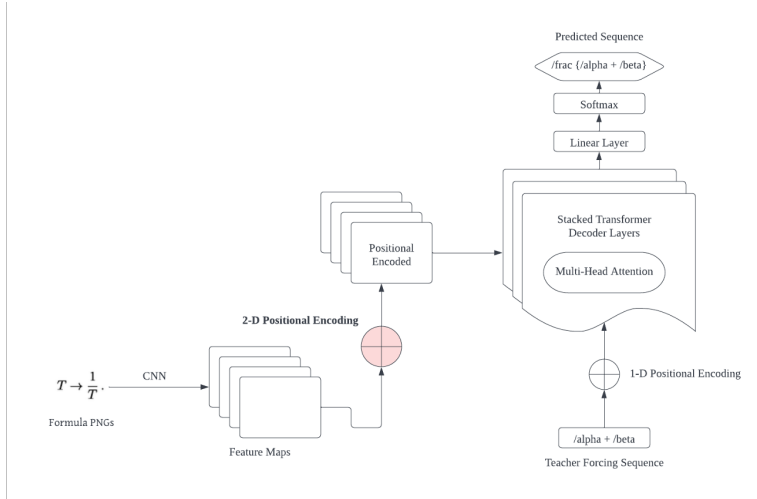


Figure 2: Model Architecture

Encoder

The encoder serves as the feature extractor, which aims to reduce the size of the image through kernels and increase its dimension to extract more information. And the architecture is modified from VGGNet [4]. After each convolution layer, the ReLU activation function is added. For the input, the shape of the image is not limited, and it would be shrunk to 8 times smaller than the original size when it reaches the last layer. The output dimension is set to be 128, which is equal to the embedding dimension in the decoder. You may refer to the table 2 for the details of the CNN.

Considering the image may contain fractions, exponents, or subscripts, the spatial

Layer	Channel	Kernel	Stride	Padding
Conv2d	64	(3,3)	(1,1)	(1,1)
MaxPool2d		(2,2)	(2,2)	(0,0)
Conv2d	128	(3,3)	(1,1)	(1,1)
MaxPool2d		(2,2)	(2,2)	(1,1)
Conv2d	256	(3,3)	(1,1)	(1,1)
BatchNorm2d				
Conv2d	512	(3,3)	(1,1)	(1,1)
BatchNorm2d				
MaxPool2d		(2,1)	(2,1)	(0,0)
Conv2d	128	(3,3)	(1,1)	(1,1)
BatchNorm2d				

Table 2: Configuration of the Convolutional Neural Network

information in the image may be critical for the model prediction. Before feeding the feature map into the decoder, we add spatial information using positional encoding. Instead of 1-D positional encoding introduced in [3], we need to add 2-D positional encoding [2] to the image:

$$\begin{aligned}
\Phi(x, y, 2i) &= \sin(x * c^{\frac{4i}{D}}) \\
\Phi(x, y, 2i + 1) &= \cos(x * c^{\frac{4i}{D}}) \\
\Phi(x, y, 2j + D/2) &= \sin(y * c^{\frac{4j}{D}}) \\
\Phi(x, y, 2j + 1 + D/2) &= \cos(y * c^{\frac{4j}{D}})
\end{aligned}$$

where $c = 10^{-4}$ and $i, j \in [0, D/4)$ specifies the dimension

The encoded feature map would be expanded into a flat sequence, ready to be processed in the decoder.

Decoder

We use the decoder structure 3 of the transformer model with multi-head attention [3]. And we stack multiple decoder layers [5] by feeding the output of one layer to the other's input. Our model uses six layers of transformer decoder and eight heads for the self-attention mechanism. First, The tokenized sequence is embedded [6] into vector space. The embedding dimension is set to 128 (the same as the dimension of the last layer in the encoder). Before being sent into the decoder, the input sequence is encoded with spatial information. The attention is masked [7] according to the input length of the sequence. The output of the decoder is connected with a linear layer with the same dimension as the vocabulary. The probability of each token would be generated using the softmax function. We adopt the greedy method to pick the token with the highest probability to be the model's prediction. The encoded feature map would be expanded into a flat sequence, ready to be processed in the decoder.

3 Training

We use 75275 samples from the dataset as our training set and 8370 samples as the validation set. The LaTeX sequences are tokenized into index sequences according

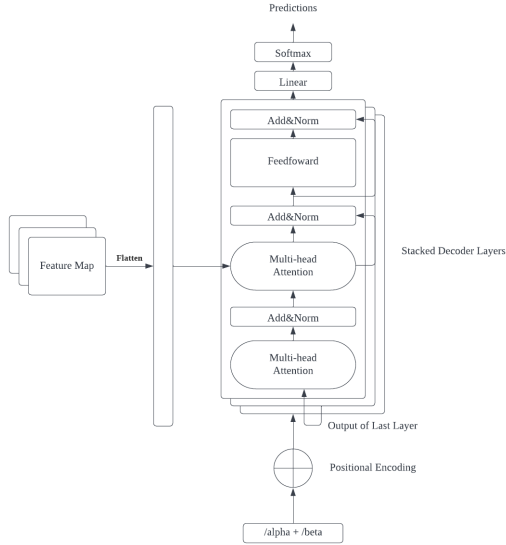


Figure 3: Decoder structure

to the vocabulary dictionary. The dictionary consists of the tokens that appear no less than a particular occurrence. We set the occurrence to be 5. The size of the vocabulary is 435 after pruning. The maximum length of sequence from the training set and validation set is 150 (152 after padding with $\langle \text{SOS} \rangle$ and $\langle \text{EOS} \rangle$). Before training, we processed our data using the binning technique. Binning aims to reduce the number of padding inside each batch. Our strategy is to separate samples into different bins regarding their sequence length. And when forming the batch, we pick samples within the same bin. This approach minimizes the number of padding appearing inside each batch. The number of samples inside each batch is set to eight according to the computational limitation. We use Adam [8] as the optimizer with learning rate 1-e3 and cross-entropy loss as the loss function. The learning rate will decrease by 0.9 times if the loss does not fall by the threshold of 1-e4 after two epochs. To prevent overfitting, we adopt a 0.1 dropout rate [9].

The model is implemented in PyTorch. We use the HKU GPU Farm for computation, which provides a GeForce RTX 2080 Ti 11GB GDDR6 as the GPU resource. After each training epoch, we test the model using the validation set to get the loss and perplexity.

4 Experiments & Result

Data Engineering

We perform the binning technique to reduce the number of paddings during training. We set eight intervals manually, and each training sample would be categorized

into its bucket. Compared with the original dataset 4, the average sequence length after padding is reduced by 36.4%. The decrease in the sequence length would also accelerate the training process.

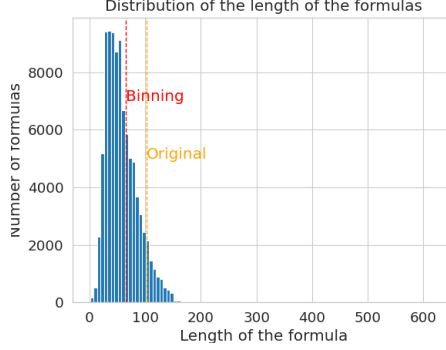


Figure 4: The average length of padded sequence

Model Iteration

For our first model, we use LSTM 5 [10] as the decoder with self-attention. The feature maps from the encoder are fed into the decoder via the attention mechanism. And the LSTM takes the prediction of the last state as the input for its current state. However, we encountered the underfitting problem during training. The training loss of the model fluctuates at a high loss and doesn't appear to converge after training for several epochs. We try to increase the complexity of the decoder (e.g., using stacked LSTM, bi-directional LSTM [11]). Since the LSTM requires heavy computation, as we increase the complexity, we suffer from the slow training process due to the limitation of GPU resources. We decide to modify the decoder in the model.

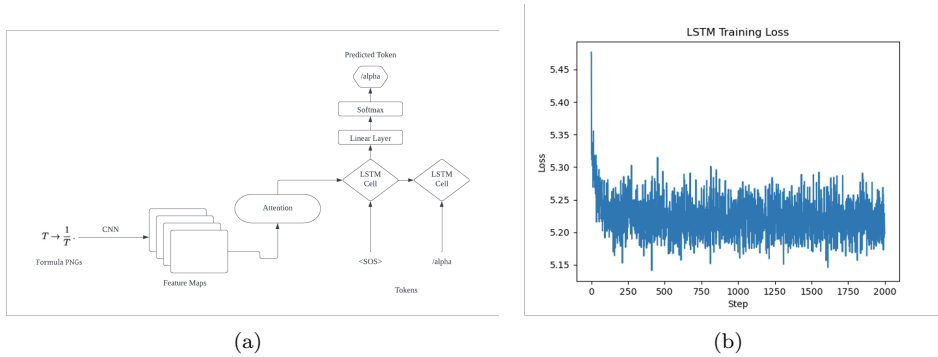


Figure 5: Architecture and Training Loss of the LSTM

As for our second model6 [3], we adopt the idea of the transformer and use the decoder part of the transformer in our model. Instead of one layer, we use stacked decoder layers based on the work of [5]. We also add positional encoding to the input sequence. The new model decreases the training loss continuously in the training steps. More importantly, training the transformer model with 100k data points only takes 3 to 4 hours for 25 epochs, while training the LSTM takes much longer time than that.

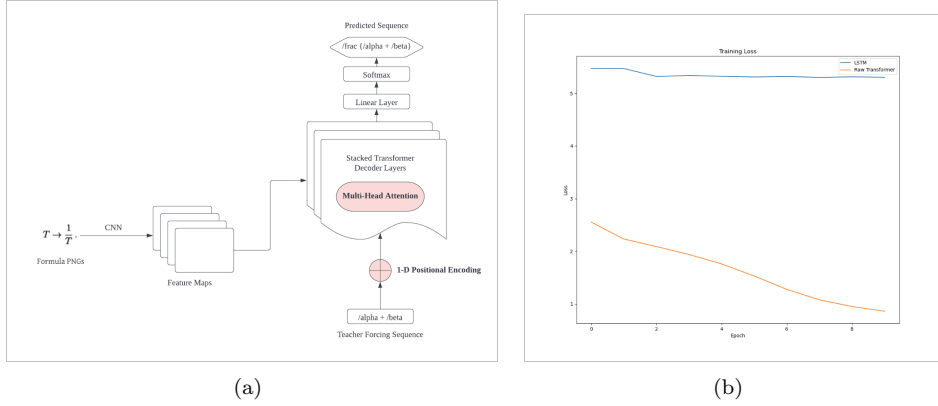


Figure 6: Architecture and Training Loss of the Transformer

To achieve better performance, we include spatial information by using 2-D positional encoding in the feature maps 7. However, we found that the training loss stopped decreasing at an early stage of training. We also noticed that the gradient of the parameters inside the convolutional neural network is close to 0 after several epochs, which means the model stopped to optimize after that.

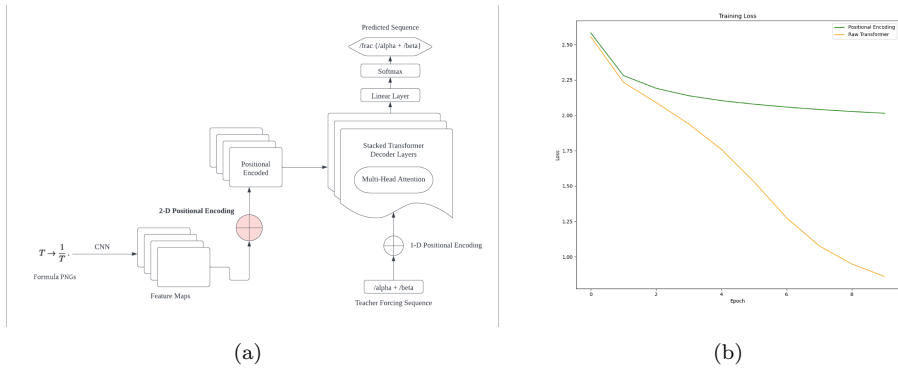


Figure 7: Architecture and Training Loss of the Positional Encoding

We add the batch normalization layer [12] after each convolution layer in the network 8. According to our data, the model improves and meets our performance expectations. We assume that the positional encoding to the feature maps may cause the output to be more dispersed, and batch normalization alleviates the problem. The validation loss also shows the same pattern. Thus, we finalize our model and move to the next stage.

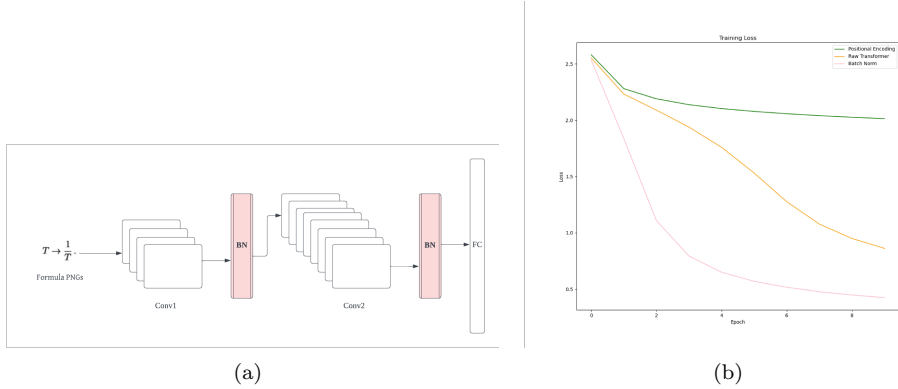


Figure 8: Architecture and Training Loss of the Batch Norm

Fine Tuning

To fine-tune the hyperparameters, we first pick five main hyperparameters from our model: output dimension of the encoder, number of decoder layers, number of heads for attention, dimension of feedforward layer, and dropout rate. We vary these hyperparameters while holding others fixed. And we pick the two hyperparameters that influence the performance of the model the most. Due to the computation resource limitation, we cannot perform fine-tuning for all hyperparameters. We train models for ten epochs with 20k samples from the training set and record their performance on the validation set. According to our data 9, we choose the number of decoder layers and the number of heads as the target hyperparameters for fine-tuning.

We perform the grid search [13] to choose the optimal hyperparameter set of the number of decoder layers and the number of heads. In total, 25 models are trained for 15 epochs with 20k samples from the training set and evaluated using the validation set. The heatmap 10 shows eight heads are the optimal setting in the given interval. And more layers would give the model better performance. We choose the largest value in our bucket, which is six. Therefore, after fine-tuning these two hyperparameters, we finalized the hyperparameters of our final model.

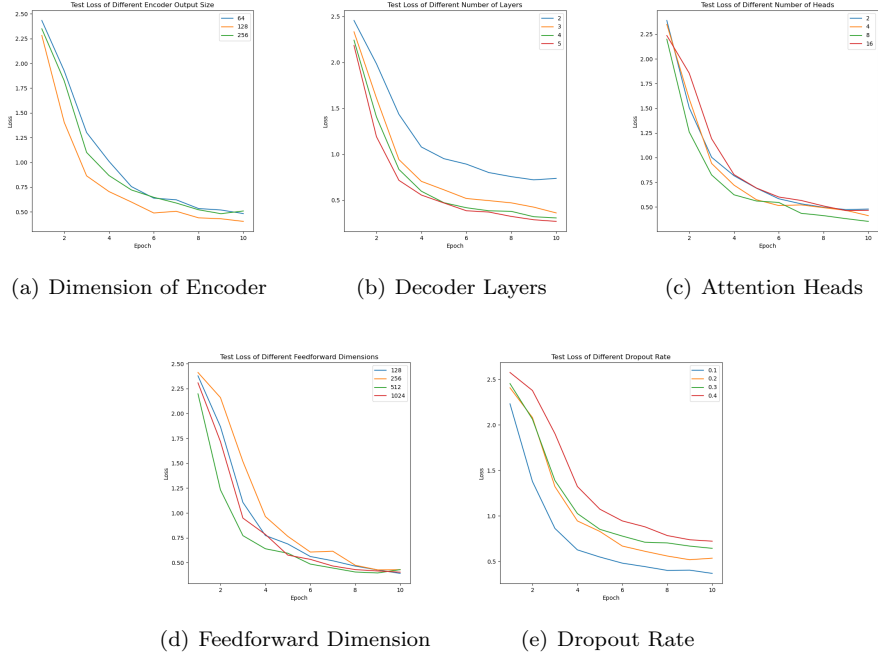


Figure 9: Validation loss when hyperparameter varies

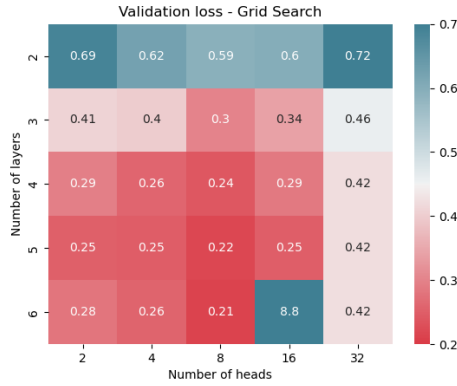


Figure 10: Grid Search for Fine Tuning

5 Model Evaluation

Metrics Definition

The metrics should be able to reflect the similarity of the ground truth sequence and the model prediction sequence in order to provide a fair evaluation of our

model. Since our task is quite similar to the machine translation problem. We adopt Bleu-Score and edit-distance to measure the similarity between the ground truth latex sequence and the prediction produced by our model.

BLEU Score

The BLEU score [14] indicates how similar the candidate text is to the reference texts by calculating the sum of all occurrences of n-gram from the prediction in the referenced texts divided by the sum of all occurrences of n-gram from our candidate. The calculation of the bleu score can be referred to in the formula below. The result of the bleu score will be a number ranging from 0 to 1. For evaluation of our own model, we will report the bleu score for each n-gram (1-4) and the average score to reflect the continuous accuracy of our model prediction.

$$BLEU\ score = \frac{\sum_{n-grams \in \hat{y}} Count_{clip}(n-gram)}{\sum_{n-gram \in \hat{y}} Count(n-gram)}$$

Edit-distance Score

Besides the Bleu score, the Edit-distance score is also frequently used in related state-of-the-art paper [2]. There are many different standards for edit-distance. Our implementation adopts Levenshtein’s implementation of the edit distance. A pair of sequences with Levenshtein edit-distance 1 means either a deletion, an insertion, or a substitution operation difference. For better compression between models, we will use the following formula to convert it so that it also lies in the 0 to 1 region.

$$Edit\ distance\ score = 1 - \frac{edit\ distance}{max(len(\hat{y}), len(y))}$$

Model Performance

The final model used the architecture 2 with the following hyperparameters:

Hyperparameter	Value
ENCODER_OUT_SIZE	128
EMBEDDING_DIM	128
VOCAB_SIZE	435
NUM_LAYERS	6
NUM_HEADS	8
FEEDFORWARD_DIM	256
DROPOUT	0.1
MAX_LEN	152

Training Parameter	Value
TRAIN_BATCH_SIZE	8
VAL_BATCH_SIZE	8
TEST_BATCH_SIZE	1
NUM_OF_EPOCH	50
LEARNING_RATE	0.001

With these settings, the model achieved the following performance:

Model	BLEU 4	BLEU unigram	BLEU bigram	BLEU trigram	BLEU 4 gram	Edit-distance
Our Model	92.10%	97.22%	94.60%	90.64%	86.31%	88.42%

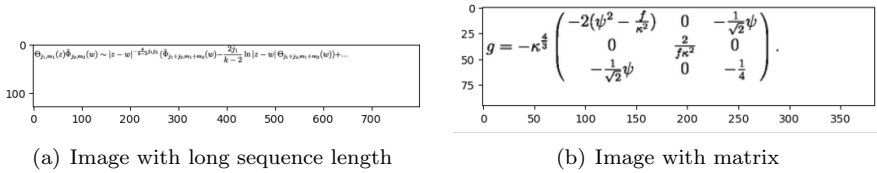
Where the state-of-the-art papers have the following results:

Model	BLEU 4 gram	Visual Match
MI2LS[Wang & Liu] [2]	90.28%	82.13%
IM2TEX[Deng et al.] [1]	87.73%	79.88%

As can be seen from the result above, the model performs well on the test set. With such model configuration, our model has perplexity 1.14 on the development set after 50 epochs. For the BLEU Score, we get 86.31% on 4-gram, where the state-of-the-art result is around 89%. Considering the complexity of the model and the limitation in computational resources, we think the result achieved is worthy of recognition.

Issue Analysis

In this section, we will demonstrate some limitations and issues of the current model based on the test data sets (not user input) and discuss the possible solutions that may help to improve the model.



Sequence Length	BLEU 4 gram	Edit Distance
[0, 50)	88.68%	94.57%
[50, 100)	86.77%	92.48%
[100, 150)	78.89%	81.63%
[150, 200)	66.36%	64.73%
[200, 700)	48.94%	40.44%

Table 3: Performance on different sequence of length

As we can see from the result above 3, we can see that the performance of our model is much better on shorter sequences. The reason for that is that during the inference time, the sequence is generated by the transformer one by one using the greedy search. Not like the training, which have teacher forcing to fix the deviation of the prediction, the prediction output by the model at inference time will have lower and lower confidence as the length of the sequence continue to increase. A possible solution to this problem is to use beam search [15], which can see more further steps during the prediction at the inference time. With an appropriate beam size and parameter k, the prediction should have better performance on long sequences.

Another limitation of the model is that it fails to predict images with complex vertical relationships. This is because of the limited number of samples in our training dataset. To solve this problem, we need to collect more datasets with

vertical relationships or add the portion of datasets with vertical relationships to our training dataset. Another possible solution is to use relative position embedding mentioned in paper [16] to the model instead of fixed positional embeddings, which can help the model to learn the spatial relationship between the tokens. However, due to the limitation of time, we are not able to implement this solution in our project.

6 Application Development

User Input Generalization

Because the user input image might be very different from the data from the datasets. For example, they may have different alignment, font, font-size, and resampling methods. If we directly feed the image to the model, the model will not be able to predict the correct latex sequence. Therefore, we need to pre-process the image to mimic the image in our dataset before feeding it to the model. The pre-processing steps we used can refer to the following image 11. This image 12 shows the effect of the generalisation pipeline on the user input image.

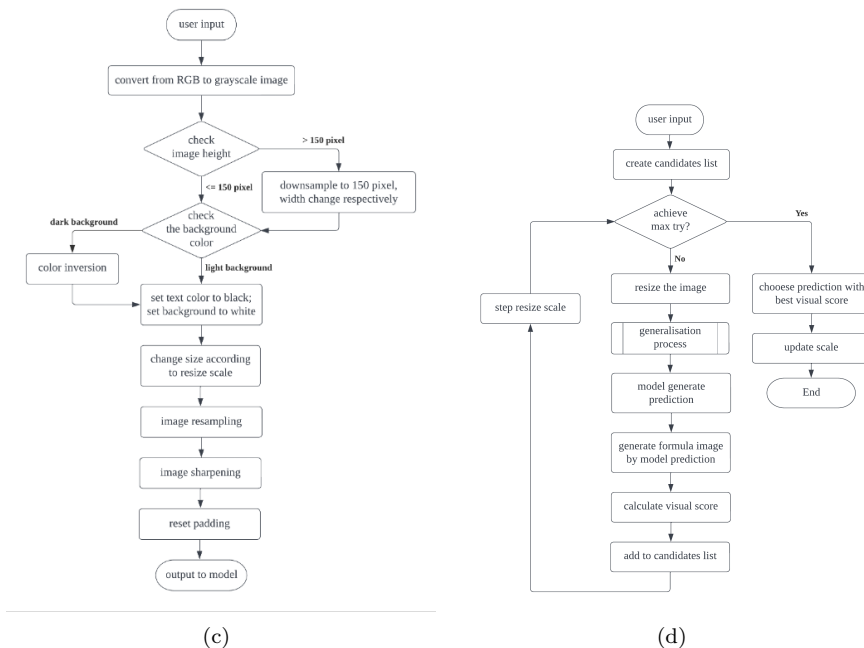


Figure 11: Genralization Pipeline

Until now, we have successfully implemented the generalisation pipeline. However, the model still cannot handle input of different font-size and resolutions. The resizing scaling is a difficult problem to solve (it cannot simply be solved by processing

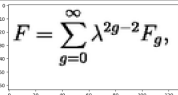
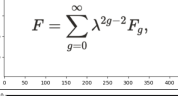
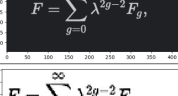
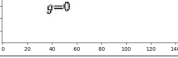
Source	Image	Model Prediction
dataset image		$F = \sum_{g=0}^{\infty} \lambda^{2g-2} F_g,$
user input image 1		$F = \sum_{g=0}^{\infty} \lambda^{2g-2} F_g,$
user input image 2		$F = \sum_{g=0}^{\infty} \lambda^{2g-2} F_g,$
processed input		$F = \sum_{g=0}^{\infty} \lambda^{2g-2} F_g,$

Figure 12: dataset image (top) image, user input (mid), processed user input(bottom) compares

user input since the resize scale is unknown; we have seen some projects using a separate CNN model to identify the scaling ratio).

Our current solution to this problem, besides the data augmentation, is to use a calibration before image input on the user side, which will ask the user to input a sample image to calibrate the scaling ratio. The calibration process is achieved by trying a list of the nearest scaling ratio and finding the one that has the highest confidence score. This solution is not ideal since the time latency for trying multiple scaling ratios is not negligible.

Deliverables

Im2Latex CLI tool

This is a command line interface 13(a) that can be used to predict the latex sequence of an image. The user can simply choose the command from a menu. For image cropping, the tool will call the MacOS in-built screenshot command for image collection. Then, the image will go through the generalisation pipeline and feed to the model for prediction. The result will be shown on the terminal and copied to the clipboard.

Im2Latex MacOS app

The functionality of the app 13(b) is the same as the CLI version. This version will stay in the menu bar and can be used to predict the latex sequence of an image. The user can click the icon and choose commands from the menu list. For the installation and operation, please refer to the README.md in the app/ folder and the app demo at the end of our video presentation.

```

~/GitHub/COMP3362-GP/app $ python api.py
Load Resources
Load Model: <All keys matched successfully>
Vocab size: 435

Please enter the command:
1. convert
2. calibrate
3. set scale
4. exit
Command: 1
Current scale: 0.51
Converting...
Screenshot Size: (442, 198)

Prediction:
F = sum - { y = 0 } ^ { \infty } \lambda^{2 y - 2} F_{-} ( g ) .
Feasibility: 0.4444444444444444
Screenshot Size: (442, 198)

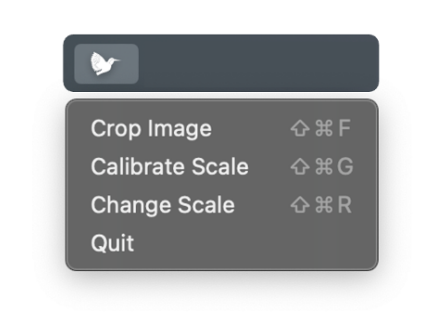
Prediction:
F = sum - { g = 0 } ^ { \infty } \lambda^{2 g - 2} F_{-} ( g ) .
Feasibility: 0.4583333333333333
Screenshot Size: (442, 198)

Prediction:
F = sum - { y = 0 } ^ { \infty } \lambda^{2 y - 2} F_{-} ( g ) .
Feasibility: 0.4874074074074074

Best Prediction:
F = sum - { g = 0 } ^ { \infty } \lambda^{2 g - 2} F_{-} ( g ) .
Result copied to clipboard

```

(a) CLI Version



(b) MacOS App

7 Limitaions & Future Work

Syntax Error

$A_r := k[x, y] / (y^2 - x^{r+1})$	Ground truth:
	$A_{-}(r) := k[x, y] / (y^{^{\wedge}(2)} - x^{^{\wedge}(r+1)})$
	Prediction (not correctly closing tags):
	$A_{-}(r) := k[x, y] / (y^{^{\wedge}(2)} - x^{^{\wedge}(r+1)})$

Figure 13: Prediction with syntax error

The model fails to predict images with syntax error 13 sometimes, though the edit distance is low. This is because the model is trained to maximize the similarity between the ground truth and the prediction using cross-entropy loss. Hence, the model won't penalize syntax error. A trivial solution to solve this problem is to add a syntax error check at the inference time. Also, this problem can be solved by using beam search, as mentioned above.

Data Augmentation

To better generalize the user input, data augmentation could be an option. Currently, the model cannot generate a satisfactory prediction for images with a different font or font size. The model's performance on different sizes of user input images has yet to be optimal. By adding resized, distorted images or images with a different font to the training set, data augmentation may improve the model further.

References

- [1] Y. Deng, A. Kanervisto, and A. M. Rush, "What you get is what you see: A visual markup decompiler," *CoRR*, vol. abs/1609.04938, 2016.

- [2] Z. Wang and J.-C. Liu, “Translating math formula images to latex sequences using deep neural networks with sequence-level training,” Sep 2019.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” Dec 2017.
- [4] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Apr 2015.
- [5] S. S. Singh and S. Karayev, “Full page handwriting recognition via image to sequence extraction,” Jun 2022.
- [6] O. Levy and Y. Goldberg, “Neural word embedding as implicit matrix factorization,” in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS’14. Cambridge, MA, USA: MIT Press, 2014, p. 2177–2185.
- [7] B. Cheng, I. Misra, A. G. Schwing, A. Kirillov, and R. Girdhar, “Masked-attention mask transformer for universal image segmentation,” Jun 2022.
- [8] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” Jan 2017.
- [9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.
- [10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [11] Z. Huang, W. Xu, and K. Yu, “Bidirectional lstm-crf models for sequence tagging,” Aug 2015.
- [12] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” Mar 2015.
- [13] P. Liashchynskyi and P. Liashchynskyi, “Grid search, random search, genetic algorithm: A big comparison for NAS,” *CoRR*, vol. abs/1912.06059, 2019.
- [14] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL ’02. USA: Association for Computational Linguistics, 2002, p. 311–318.
- [15] A. Graves, “Sequence transduction with recurrent neural networks,” *CoRR*, vol. abs/1211.3711, 2012.
- [16] P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” Apr 2018.