Fine-tuning Vision Language Model for OCR Task Project Report - ECE 285

Huaijing Hong
Mechanical and Aerospace Engineering
A59024765

Abstract

In this project, we are trying to fine-tune a light weight vision language model (VLM) Florence-2 on the task of translating math equations in images to LaTeX code, which is counted as an optical character recognition (OCR) task. Instead of using full model fine-tuning, we chose to use low-rank adaptation (LoRA) fine-tuning to save the computational resources. We ran training and testing on a dataset of 60,000 pairs of math equation images and their corresponding LaTeX code. Results demostrate high improvement in the performance of the model compared to the zero-shot inference. However, throughout the project, we also found out there are some more things about the model that can be investigated and improved. All the code and weights can be found on github: `https://github.com/JingHHj/VLMim2latex`.

1 Introduction

Vision language models (VLMs) have gone through a tramendous development in the past few years, while showing amazing abilities in understanding images and text. However, most of the VLMs are heavy and require a large amount of computational resources even just for inference. For example, models like Qwen-VL [1] and LLaVA [2], even just the smallest model has 7 billions parmaters. It makes them so powerful, but also limits their applications in real world scenarios. Therefore, in this project we are trying to investigate the ability of a light weight VLM, Florence-2 [3], on a certain task.

The task we chose to do is to translate math equations in images to LaTeX code, which is counted as an optical character recognition (OCR) task. We chose this task is for its harder than simply image captioning or object detection which can already be done be zero-shot inference. Also we chose to use LoRA [4] fine-tune instead of full model fine-tuning to save the computation resources and time. This result the model have a solid understanding of the task, but still did not reach its full potential.

2 Related Work

2.1 Vision Language Models

The development of vision language models can be traced all the way back to the vision question answering (VQA) task, where the model is required to answer questions based on the given image. Fast forward to 2017, the introduction of Transformer [5] has completely changed the game for both natural language processing (NLP) and computer vision (CV). It can not only process each of them separately, but can also fuse the both modalities, which is the key to the success of VLMs. Contrastive language-image pre-training (CLIP) [6] is the best proven example of this, where the model is trained to align the image and text representations in a shared embedding space. It lays the foundation for may later VLMs, such as Flamingo [7], BLIP [8], and Qwen-VL [1].

However, most of the VLMs are heavy and require a large amount of computational resources. Therefore, people start to investigate in how to make the VLMs more lightweight and efficient. Recently, there's light wieght VLMs like SmolVLM [9], nanoVLM [10], ,and Florence-2 [3], the model we are trying to fine-tune in this project.

2.2 VLM Fine-tuning

The reason for fine-tuning VLMs is that VLM were first trained on large-scale datasets and can be used for a wide range of tasks, but might perform poorly on specific tasks. We need to fine-tune the model to release its full potential. Basically, VLM fine-tuning techniques can be divided into two categories: full model fine-tuning and parmaters efficient fine-tuning (PEFT). Full model fine-tuning is the most straightforward way, which simply train the whole model for the given task.However, it not only requires a large amount of computational resources, but also can lead to overfitting. Therefore, people introduced parameters efficient fine-tuning (PEFT) methods, which only fine-tune part of the models. The most well known method is low-rank adaptation (LoRA) [4], which injects trainable low-rank matrices into the model. Other than the above two methods, people also use reinforcement learning (RL) to fine-tune VLMs, mostly reinforcement learning from human feedback (RLHF) [11] [12].

3 Method

In this project we are chosing Florence-2 [3] as the base model which have 0.23 billion parameters. As for the fine-tuning method, we chose low-rank adaptation (LoRA) [4] (LoRA) with rank of $r = 8$ and $\alpha = 16$. The hardware we are using 2 Tesla-V100-SXM2 GPU which have 32GB RAM each. Here are some more details about it.

3.1 Florence-2

Florence-2 is a light weight VLM developed by Microsoft, which is able to perform a wide range of tasks, such as image captioning, object detection, and visual question answering. In this project, we are mainly focusing on its ability to do the optical character recognition (OCR) task. Fundamentally, Florence-2 is a encoder-decoder Transformer model, which use DaViT [13] as the image encoder and BERT [14] as the text encoder. The reason why I chose Florence-2 is that it is a extremely light weight VLM, with only 0.23 billion parameters and easily run inference on a Nvidia RTX-2080-Ti GPU.

One more things that we need to mention is the module we are trying to fine-tune. Generally, Florence-2 can be divided into three big sections: the vision encoder, the text encoder, and the decoder. Both vision encoder and text encoder is used to understand the input image and text, which is not suposed to be tuned. Therefore, we only fine-tune the decoder part of the model (query, key, value, and output projection, fully connected layers, and language model head)

3.2 Low-rank Adaptation(LoRA)

Then core idea of LoRA is this equation:

$$W = W_0 + \Delta W = W_0 + \frac{\alpha}{r} B \cdot A \tag{1}$$

where W is the original weight matrix, W_0 is the pre-trained weight matrix, ΔW is the trainable weight matrix, B and A are the low-rank matrices. The reason why it would works, is the following two reaons:

- A huge matrix can always be decomposed into the multiplication of two smaller matrices.
- The effect of fine-tuning on the original matrix can always be approximated by the sum of the original matrix and a new matrix, so as the output.

Meanwhile, we also need to guarantee that the rank of the smaller matrices B and A is much smaller than the original matrix W. For example, if the dimension of the original matrix is 100×100, and the rank is $r = 8$. Then the number of learnable parameters will be reduced from $100^2 = 10,000$

to $100 \times 8 + 8 \times 100 = 16,00$, which is only about 16% of the original number of parameters. In reality, this reduction would be even more significant. In our case, the original model has 0.231 billion parameters, and the LoRA fine-tuning only requires about 2.340 million parameters, which is only about 1% of the original model.

As you can see, another hyperparameter we need to set is the α, which is a scaling factor for the low-rank matrices. The recommended value for α is 2 times the rank r, which is also the value we used in this project.

3.3 Evaluation

There ara basically three metrics to evaluate the performance of VLMs: token level accuracy and sequence level accuracy.

- The token level accuracy calculated by comparing the prediction and the ground truth token by token. It is the most straightforward way to evaluate how good the model generates code compares to the ground truth. However, in code generation tasks, even the token level accuracy reach about 90%, the rendered equation image can looks completely different from the ground truth, due to the fact that the Latex code only works in a very specific grammar.
- Therefore, we introduce the sequence level accuracy, which is calculated by comparing the prediction and the ground truth sequence as a whole. Only the sequence is completely match the ground truth, it is considered as correct. This metric is more strict than the token level accuracy, but it truly reflects how good the model is in generating the LaTeX code.

4 Experiments

4.1 Datasets

In this project, all the training, validation and testing datasets are from "AlFrauch/im2latex" [15], can be found on Hugging Face. This dataset contains about 1.59 million pairs of high quality data, each of them is consist of an math equation image (in PIL.Image format) and its corresponding LaTeX code (string). They were collected from over 100,000 natural science, math and engineering papers.

However, the size of the dataset is too large, only 60,000 pairs of data of it was used throughout the project. 50,000 pairs were used for training, 5,000 pairs were used for validation, and the rest 5,000 pairs were used for testing. The training and validation datasets are shuffled before training. The image was resize to 700x700 pixels, because we found this is the best balance between image size and computation resources. If the image too small, it compresses the information too much, which makes the model harder to extract the features. If the image is too large, it requires more computational resources and slower the training. As for the text input, we always use the prompt '<OCR>' as the text input part, which is a prompt for OCR tasks.

4.2 Results

The best model we have, is trained with following hyperparameters:

- Learning rate: 5e-6
- Batch size: 16
- Epochs: 5
- Optimizer: AdamW
- LoRA Rank: 8
- LoRA Alpha: 16
- LoRA Dropout: 0.1

We result in having token level accuracy for both train, validation and test datasets over 90%, which is quite good. However, the sequence level accuracy is stays pretty low, around 40% for both validation and test datasets and 60% for the training dataset. Here are some examples of the results:

Ground Truth	Prediction	Token Accuracy
$C_F\left[\frac{x^2}{y}\right]$	$C_F\left[\frac{x^2}{y}\right]$	1.0000
$\Delta V^{GG}(x, y)$	$\Delta V^{GG}(x, y)$	1.0000
$f^q(z, \mu)$	$f^q(z, \mu)$	1.0000
$zf^G(z, \mu)$	$zf^G(z, \mu)$	1.0000
$(2x-1)V^{Gq}_{ext}(x, y)$	$(2x-1)V^{Gq}_{ext}(x, y)$	0.9583
$\log(f(\epsilon'_{\lambda n}(\lambda)))$	$\log(f(\epsilon'_{\lambda n}(\lambda))$	0.9200
$\Gamma^{qq}_n(z, z')$	$\Gamma^{qqq}_n(z, z')$	0.9091
$\Gamma^{qG}_n(z, z')$	$\Gamma^{qgG}_n(z, z')$	0.9565
$\Gamma^{Gq}_n(z, z')$	$\Gamma^{Gq}_n(z, z')$	1.0000
$\Gamma^{GG}_n(z, z')$	$\Gamma^{GGG}_n(z, z')$	0.9545

(a) Fine-tuned model inference results

Ground Truth	Prediction	Token Accuracy
$C_F\left[\frac{x^2}{y}\right]$	*unanswerable*	0.0870
$\Delta V^{GG}(x, y)$	*unanswerable*	0.2500
$f^q(z, \mu)$	*unanswerable*	0.2667
$zf^G(z, \mu)$	*unanswerable*	0.3125
$(2x-1)V^{Gq}_{ext}(x, y)$	*coc*	0.2917
$\log(f(\epsilon'_{\lambda n}(\lambda)))$	*unanswerable*	0.2000
$\Gamma^{qq}_n(z, z')$	*unanswerable*	0.1818
$\Gamma^{qG}_n(z, z')$	*unanswerable*	0.1304
$\Gamma^{Gq}_n(z, z')$	*unanswerable*	0.1739
$\Gamma^{GG}_n(z, z')$	*n*	0.1364

(b) Baseline model zero-shot inference results

Figure 1: Comparison between LoRA fine-tune result inference and baseline zero shot inference.

As we can see from the results, the model is able to translate some easy equations, while fail to translate some more complex ones. Also we notice something interesting, that sometimes the complie results is correct, but the token level accuracy is still not 1. The reason why this would happen, its that the model is able to understand the input image and generate the LaTex code that is semantically correct. However, I noticed a lot of the groud truth code in this dataset contains "`\displaystyle`" at the front of the code. This does not really affect the rendering of the equation. If we count this cases as correct, the sequence level accuracy would be much higher, reach at least 50% for the validation and test datasets.

Even the accuracy looke not good enough, especially the sequence level. However, when it compares to some zero shot testing, we can see the huge difference of our finetune model. Most of the time, the model completely not understand the task, and not able to generate any LaTeX code. Even when it is able to under stand the task, it still generates the token level accuracy is only about 20% on average. After the comparison, we can come to the conclusion that the fine-tuning process did hugly improve the performance of the model on this specific task and there are still some more things about the model that can be investigated and improved.

4.3 Ablation Study

During the project, we did some ablation studies to see how different hyperparameters affect the performance of the model. The reason why I did not cover the ablation study on batch size, which is one of the most important hyperparameters, is that I simply did not have enough computational resources to run the experiments with different batch sizes.

1. **Dataset size:** Intuitively, we should use as much data as possible. But in reality, it is unachievable due to the time limitation and the performance does not always increase once the size of the dataset passes a certain threshold. Therefore, we tried a couple different combinations of dataset size and epochs, as shown in Table 1. The reason why we use a combination is because we want to make sure the training step is compared on the same level. As we can see from the table, the performance of 50,000 pairs of data with 5 epochs

Dataset Size (pairs)	Epochs	Token Level accuracy	Sequence Level accuracy
5,000	20	0.92	0.336
50,000	5	0.94	0.421
100,000	2	0.90	0.269

Table 1: Comparison of dataset size and epochs used in the experiments.

is the best.

2. **Learning rate:** As for the learning rate, we tried three different values, 1e-4 and 5e-6 and 1e-6. At first, we were testing high learning rates because the zero shot performance of the model is super bad, so we thought we might need to tune the model more aggressively. However, we did not consider the fact that the model is already pre-trained and its so large. When we were using 1e-4 as the learning rate, we can see quite high accuracy in the first epochs. However, when the second epoch starts, the accuracy just drops all the way down to 0 and never comes back. Meantime, you can also witness this kind of drop in lower learning rates (5e-6), but it is not as severe as the previous one. Lastly, we tried 1e-6 as the learning rate, which is the smallest one we tried. We can see the drop still exist, while the it definitely learns much slower than the previous two. Consiquently, we can come to the conclusion that the best learning rate for this model is 5e-6.

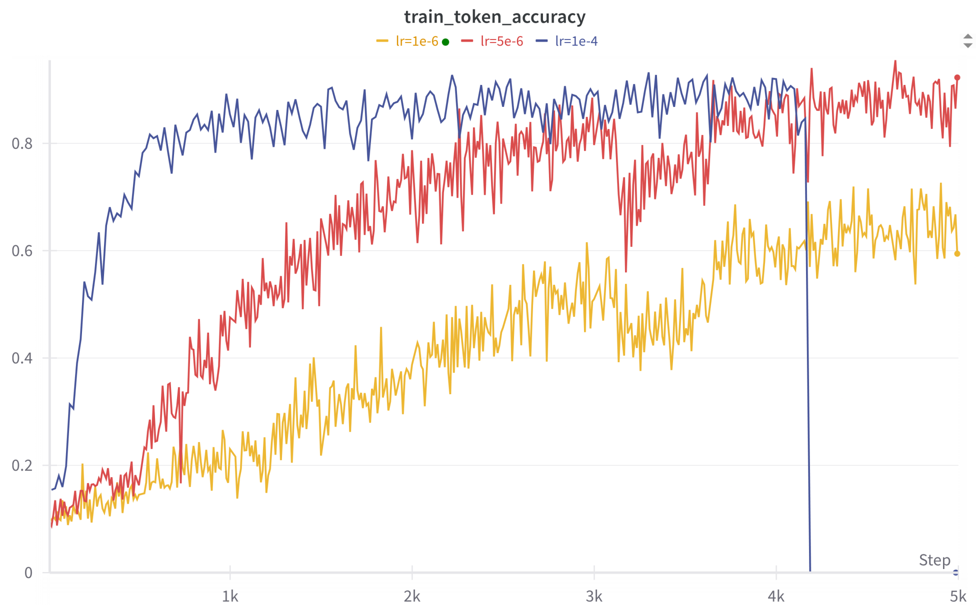

Figure 2: Train token level accuracy with different learning rates.

3. **LoRA rank, alpha, and dropout:** In this project we tried couple different combinations of LoRA rank and alpha:
   - Rank $r = 8$ and alpha $\alpha = 8$
   - Rank $r = 8$ and alpha $\alpha = 16$
   - Rank $r = 16$ and alpha $\alpha = 32$

   Among three of these combinations, I did not see much difference in the performance. However, if we chose rank 16, we have to lower the batch size to 8, since there is not enough memory to run the model. As for the ratio between rank and alpha, it was 2 was a recommanded value, therefore we chose the second combination.

5 Conclusion

In this project, we LoRA fine-tuned a light weight vision language model (VLM) Florence-2 to translate math equations in images to LaTeX code The results was evaluated with token level accuracy and sequence level accuracy. The first one was able to achieve about 94% and the second one was able to achieve about 40%. However, after we compiled the LaTeX code, we found that the model performance was actually underestimaed due to lack of semantic accuracy evaluation. Meanwhile when we compared the results to the zero-shot inference, we can see a huge improvement in both accuracy.

5.1 Future Work

During this project, we have found several key points that can be futher investigated and improved.

1. **Hyperparameters:** In section 4.3, some ablation studies were done, but not thoroghly enough. we believe the best learning rate, dataset size and LoRA hyperparameters have not been found yet. This would definitely further release the potential of the model.
2. **Evaluation and Loss function:** In this project, we simply used the cross entropy loss function to train the model, and evaluated the model with token level accuracy and sequence level accuracy. However, we did not evaluate or trained the model with sementic accuracy. Latex code works in a very specific grammar, sometimes the model can generate the code that is semantically correct, but have low accuracy in token level and sequence level, vice versa. If the compiled LaTeX code looks similar to the ground truth, the model should be encouraged to so instead of simply trying to match the result token by token.

References

[1] Qwen Team. Qwen-vl. `https://huggingface.co/Qwen/Qwen-VL`, 2023.

[2] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023.

[3] Bin Xiao, Haiping Wu, Weijian Xu, Xiyang Dai, Houdong Hu, Yumao Lu, Michael Zeng, Ce Liu, and Lu Yuan. Florence-2: Advancing a unified representation for a variety of vision tasks, 2023.

[4] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.

[6] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *International Conference on Machine Learning*, 2021.

[7] Jean-Baptiste Alayrac, Jeff Donahue, Paul Luc, Antoine Miech, Ian Barr, Ryan Ring, Jacob Menick, Malcolm Reynolds, Alban Chatelain, Alban Desmaison, et al. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.

[8] Junnan Li, Dongxu Li, Caiming Xiong, and Steven CH Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. *International Conference on Machine Learning*, 2022.

[9] Andrés Marafioti, Orr Zohar, Miquel Farré, Merve Noyan, Elie Bakouch, Pedro Cuenca, Cyril Zakka, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Vaibhav Srivastav, Joshua Lochner, Hugo Larcher, Mathieu Morlon, Lewis Tunstall, Leandro von Werra, and Thomas Wolf. Smolvlm: Redefining small and efficient multimodal models, 2025.

[10] Mukund Agarwalla, Himanshu Kumar, Raj Dandekar, Rajat Dandekar, and Sreedath Panat. Nanovlms: How small can we go and still make coherent vision language models?, 2025.

[11] Hardy Chen, Haoqin Tu, Fali Wang, Hui Liu, Xianfeng Tang, Xinya Du, Yuyin Zhou, and Cihang Xie. Sft or rl? an early investigation into training r1-like reasoning large vision-language models, 2025.

[12] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[13] Mingyu Ding, Bin Xiao, Noel Codella, Ping Luo, Jingdong Wang, and Lu Yuan. Davit: Dual attention vision transformers, 2022.

[14] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[15] Aleksandr Frauch (Proshunin). Recognition of mathematical formulas in the latex: Image-text pair dataset. `https://huggingface.co/datasets/AlFrauch/im2latex`, 2023.