DS-UA 301 AI Calligraphic Poet Project Midway Checkpoint

Haoqi (Kevin) Zhang, Millie Chen, Yuanheng Li

hz3223@nyu.edu, mc9362@nyu.edu, yl10337@nyu.edu

I. Methodology

For the *image caption generation* step, we decided to use a pre-trained transformer model and use **transfer learning** to fine tune it for our specific task. The transformer model that we decide to use is **"GIT"**, a Generative Image to Text model [9]. Specifically, the GIT model that is used for our task is the **"git-base-coco"** from Hugging Face, which is the GIT model that is **fine-tuned on the MS COCO dataset** [7]. We decided to use this specific model instead of the more general GIT model that is trained on ImageNet-1K because we have a very small dataset (around 2000 image caption pairs after data preprocessing [2]) available and since our dataset is a subset of the MS COCO dataset, using the pre-trained weights from a model fine tuned on the same dataset will result in a better performance. In addition, the "git-base-coco" model from Hugging Face also has specific pre-trained Image and Caption preprocessors to convert the input data into tensors. We decided to use the preprocessor for the images but choose **a separate pre-processor for the captions** because the model is trained on English captions, so the tokenizer it uses (BERT) is tailored specifically towards the English language [5]. For the purposes of our project, we aim to generate Chinese image captions so we decided to use the **"bert-base-chinese"** tokenizer from Hugging Face, which is a BERT model that is trained specifically for the Chinese language [4]. Furthermore, for fine tuning the model, after experimenting with various configurations, we decided to **freeze the layers that are associated with extracting features from the images and unfreeze layers that are associated with the caption generation** (such as the word embeddings and image feature to caption layers). We decided to use a smaller learning rate of 5e-5 and fine tune the model for 40 epochs.

For the *poetry generation* step, we used a Transformers based pre-trained model and finetuned it on a traditional Chinese poetry dataset. The model, called **SongNet** [6], addresses text generation based on the predefined rigid formats, which is the biggest challenge to special format text generation as it requires strict formatting, in the case of poetry, **rhyme and length**. The backbone of the framework is a Transformer-based auto-regressive language model. Sets of symbols are tailor-designed to improve the modeling performance especially on format, rhyme, and sentence integrity. Inspired by BERT and GPT, a **pretraining and fine-tuning framework** is designed to further improve the generation quality. We first obtain a pretrained model trained on Chinese Wikipedia (1700MCharacters) and a merged Chinese News (9200MCharacters) corpus from the Internet. Then we conduct transfer learning to finetune the model on a domain specific dataset, in this case a **SongCi dataset**, to obtain better performance on SongCi generation. In order to increase the diversity of the poetry generated, **top-k sampling** is chosen as our main decoding strategy. We conducted hyper-parameter tuning on k and chose $k = 32$ as the ideal value.

For the *calligraphy generation* step, we primarily use **GAN-based** and **Diffusion-based** models **from scratch**. Our dataset, **Chinese Calligraphy Styles by Calligraphers** [10], includes 20 target styles by renowned calligraphers, with thousands of samples for each style. To generate the **source style**, we use the **system font "/System/Library/Fonts/PingFang.ttc"** on macOS, which contains a comprehensive set of Chinese characters used in daily typing. **Paired data** 1 is essential for training both GAN and basic diffusion models. When paired data is required for models other than CycleGAN, we use the ***easyocr*** [1] package to automatically detect characters and apply the same method as for source data generation to create paired source characters for each character in the target style. By manually checking 1000 data, the OCR accuracy is 96.7%. The image preprocessing is straightforward, we resize each image to 128 x 128 and We normalize the inputs to $[-1, 1]$ to stabilize training. We then train both GAN and CycleGAN models using this dataset. Each model includes a generator and a discriminator. Based on prior work [3], we incorporate **dense blocks** in the generator to enhance performance. The GAN code defines **a single generator-discriminator network** to generate realistic images. In contrast,

CycleGAN uses **dual generators and discriminators along with a cycle consistency loss**, enabling **unpaired** image-to-image translation by ensuring transformations are reversible between two domains. The GAN loss functions and CycleGAN loss functions are stated in the appendix. For both, the goal is to **learn a mapping function that achieve style translation**. We set a small learning rate, $lr = 0.0001$. For GAN, we set $\lambda = 100$, and for CycleGAN, we use $\lambda_{\text{cycle}} = 25$. The number of DenseBlock is set to 9. Set the noise level to 0.005 to discriminator layers. This noise is essential for preventing the **model collapse** [8] 2. The detailed architecture and implementation are provided at the end 3 4 5. We've also tried **Diffusion model**, but the result is somewhat unpleasant 6.

II. Results

For *image caption generation*, we were able to generate varying results with **different degrees of success**. When manually reviewing the generated image captions for the validation set, there are 3 general categories that the output falls into: generated caption was able to capture all of the main features in the image correctly, the generated caption was able to capture the main features in the image with incorrect grammar or repetitive phrases, and the generated caption was not able to capture all of the main features in the image. In most cases, at least one of the main subjects or features in the image was correctly identified but the model is struggling with images that are very blurry or images that contains a lot of different items.

For *poetry generation*, we were able to generate **well structured, well rhymed, and good quality poetry**. Most of the cases, the generated poetry is very relevant to the input, however, there are also cases where the model under perform **when handling cases where input is more associated with modern objects** and are not a part of the SongCi fine-tuning dataset. For example, when the input is " 一只狗坐在车上", meaning"a dog is sitting in a car", the model didn't handle the "car" token very well and were not able to generate closely related meaningful poetry to this context 9. Although the generated poetry is still well structured, well rhymed, and of quality, it it **not closely related** to the input context sometimes, especially for certain tokens without enough context in the SongCi dataset. Our perception is that the model handles context outside of the finetuning dataset less effectively, and thus generate loosely related poems (instead of closely related) regarding these cases of context. Also, the model requires **additional input information of format**, the input has to be formatted in a certain way which the output is aimed to be formatted in.

For *calligraphy generation*, our results consist primarily of outputs from well-trained GAN and CycleGAN models, showcasing various examples of generated calligraphy. Both successful and less successful examples are presented below in appendix 7, with a detailed analysis provided in the next chapter. The loss plot is also included in the appendix 8. Although the loss does not decrease and converge as expected, empirically, the generated characters become increasingly robust and closer to the target style as the number of epochs increases, which is good enough. Incorporating a **suitable evaluation metric** in the future would provide a clearer visualization of the training process, as the loss alone is not particularly informative here.

By combining all three components, we obtain some intriguing results. Below are three random examples 9 10 11, followed by an analysis based on these findings.

III. Analysis

For *image caption generation*, the main issue that was encountered throughout the process is **overfitting and underfitting**. To resolve this issue, I **experimented with different layers to freeze/unfreeze, different number of epochs, as well as different learning rates** to use throughout the training process. I also tried using different GIT models to find the one that gives the best results for this dataset and task. In addition, another issue that I encountered was that for the same image using the same trained model, the **image caption output will be different each time and the quality of each output is different**. For example, one image caption might be able to capture the main features of the image but running the predict function again might result in a new image caption that was not able to capture all the main features of the image. This variation might be due to the randomness involved in the caption generator but also might be because of the **limited training dataset where the model was not able to consistently capture certain features that are very different from what is in the training set**. Additionally, in some cases, even though the model was able to generate captions that captures the main features in the image, there are some repetitive phrases. For example, in Figure 10, there is an example of flowers in a vase. The generated caption was "一个有一个插着花的花瓶", which translates to "One has a vase with flowers". Even though the main features flower and vase are captured, there is a unnecessary repetition of the phrase "一个" (which translates to "one"). These results were somewhat expected since we are trying to do transfer learning with a very different task than the original model and we are using a

limited dataset.

For *poem generation*, the generated poetry is almost always **well structured, well rhymed, and of good quality**. However, one interesting phenomenon that we discovered is the **relevancy of the generated poetry to the input context**. In sometime cases, the model is able to generate very closely related poetry, and in some other cases, the generated poetry is very loosely related. We believe this is associated with the **how much the input context relates to the fine-tuning dataset**. If the input context is well represented in the fine-tuning dataset, the generated poetry is almost always closely related to the input context. Otherwise, if the input is not well represented, the generated poetry is loosely associated. Also, we find that the **input format** is able to control the generated format very well, as the output format is almost always the same with input, meaning the SongNet architecture handles rigid formats exceptionally well.

For *calligraphy generation*, the valuable insights lie in the challenges encountered during training and the techniques applied to overcome them. The first issue encountered is **model collapse**. **Adding noise to the discriminator** helps mitigate model collapse by making it harder for the discriminator to confidently distinguish between real and fake images. This encourages the generator to produce diverse outputs. This regularization prevents the discriminator from becoming overly precise, thus stopping the generator from exploiting specific details to deceive it. Beyond preventing collapse, this approach also helps avoid problematic cases like Bad Cases 2 and 3 7, where the model generates excessively noisy or completely chaotic images. In general, incorporating additional **regularization** techniques can achieve similar improvements. We experimented with various regularization methods, including **adding dropout layers in the generator, introducing regularization terms in the loss function, and applying noise to the original images as data augmentation**. While these techniques did provide regularization, they often resulted in the model becoming **overly constrained**, preventing it from learning meaningful features. Empirically, adding noise to the discriminator has proven to be the most effective regularization technique. Furthermore, comparing the outputs of GAN and CycleGAN reveals some interesting differences. GAN tends to produce results that are more **unstable and varied**, while CycleGAN generates outputs that are more **stable and smooth**. CycleGAN also addresses the **"vanishing stroke"** issue more effectively. Tuning the **cycle consistency loss** coefficient, λ_{cycle}, is crucial here: setting it too low results in an overly volatile model, while setting it too high creates a model that is too constrained and fails to learn stylistic features. The improved stroke preservation seen in CycleGAN outputs is partly due to this consistency loss, which ensures that the generated works are visually complete and more aesthetically pleasing. When integrating with other components, certain characters are still not generated successfully. For instance, in 10, the character "艳"could not be generated correctly. The model requires further tuning to enhance robustness and ensure it can generalize to any input characters, supporting reliable use in the final integration stage.

IV. Plan for additional analysis

For *image caption generation*, we plan to do more research and look for **additional methods** that can help deal with the issue of the model having a limited training dataset and the model not generalizing well. In addition, we plan to implement different evaluation metrics such as **BLEU, METEOR, and CIDEr** to assess the quality of the output captions as these are the standard metrics used in image captioning. Furthermore, I will be exploring **datasets** with images that can provide a more related context for the poem generation.

For *poetry generation*, we would like to investigate further about the **formatting constraint** of the input and the representation of input context in relationship to the fine-tuning dataset. Currently, we have to manually input the context in the way which we want the generated poetry to be formatted in. We would like to give more freedom to the formatting of the input while maintaining the well-structured, well-rhymed, and good quality output. We will tryout **pre-trained tokenizers and constraint decoding methods** that allows us handle the input automatically into the format we want, this way, we give more freedom to the input, while maintaining the structure and quality, also allowing the pipeline of the entire project to be automatic and possible for deployment. We would also like to investigate how to better represent the cases of input that is not well represented in the fine-tuning dataset.

For *calligraphy generation*, there are three main areas I want to explore: 1. **Evaluation Metrics**: Model selection has been subjective, lacking rigorous metrics. I plan to introduce style loss (using feature maps to assess style similarity) and character loss (using OCR accuracy to verify correct character generation) to evaluate models objectively. 2. **Model Comparison**: **Model Comparison**: With proper metrics, I will compare GAN and CycleGAN, as well as architectures like residual and dense blocks, to select the best model for our pipeline. 3. **Cross-Language Inference**: I will

test the best model on other languages (e.g., English, Japanese, Spanish) to see if it maintains style across language domains. 4. **Multi-font**: Untill now, all the characters are mapped towards Baida Shanren font, we will train different model allowing multiple target font.

V. Work Plan

So far, each person has completed one part: Millie worked on image captioning, Kevin on poem generation, and Yuanheng on calligraphy generation. Based on the current progress, the work plan has been adjusted as follows:

Week 1: Conduct additional analysis on individual tasks, including dataset exploration, evaluation, further tuning, and model selection. Implement advanced functions as needed for each task.
Week 2: Complete all individual components, finalize additional analyses, and finish individual GitHub code repositories.
Week 3: Integrate the pipeline and address any interface or connection issues. Finalize the GitHub code repository.
Week 4: Finalize the model and code; complete GitHub documentation, including the README. Begin drafting the final report.
Week 5: Complete the final report and prepare presentation slides to showcase the project results and achievements.

VI. Appendix

For GAN, the loss functions are:

$$L_G = L(D(G(x)), y) + \lambda \cdot L_{\text{pixel}}(G(x), y_{\text{target}}) \quad (1)$$

$$L_D = \frac{1}{2}\left(L(D(y_{\text{target}}), y) + L(D(G(x)), \hat{y})\right) \quad (2)$$

For CycleGAN, the loss functions are:

$$\begin{aligned} L_G = {} & L(D_Y(G_{X\to Y}(x)), y) + L(D_X(G_{Y\to X}(y)), y) \\ & + \lambda_{\text{cycle}} \cdot L_{\text{cycle}}(G_{Y\to X}(G_{X\to Y}(x)), x) \\ & + \lambda_{\text{cycle}} \cdot L_{\text{cycle}}(G_{X\to Y}(G_{Y\to X}(y)), y) \end{aligned} \quad (3)$$

$$L_{D_X} = \frac{1}{2}\left(L(D_X(x), y) + L(D_X(G_{Y\to X}(y)), \hat{y})\right) \quad (4)$$

$$L_{D_Y} = \frac{1}{2}\left(L(D_Y(y), y) + L(D_Y(G_{X\to Y}(x)), \hat{y})\right) \quad (5)$$

Figure 1: Paired and unpaired training data

Figure 2: Case of model collapse

DenseBlock

A dense block used in the transfer section of the Generator.

Layer	Parameters	Description
Conv2d	kernel_size=3, stride=1, padding=1	Growth rate of 256 channels
InstanceNorm2d	-	Normalizes each feature map independently
ReLU	inplace=True	Applies ReLU activation
Conv2d	kernel_size=3, stride=1, padding=1	Maintains growth rate
InstanceNorm2d	-	Normalizes each feature map independently
Concatenate	-	Concatenates input with block output

Figure 3: Detailed implementation of DenseBlock

Generator

The generator network structure, which includes an encoder, dense blocks for feature transfer, and a decoder.

Section	Layer	Parameters	Description
Initial	Conv2d	kernel_size=7, stride=1, padding=3	64 channels, InstanceNorm2d, ReLU activation
Downsampling	Conv2d	kernel_size=3, stride=2, padding=1	128 channels, InstanceNorm2d, ReLU activation
	Conv2d	kernel_size=3, stride=2, padding=1	256 channels, InstanceNorm2d, ReLU activation
Transfer	DenseBlock (repeated)	growth_rate=256	Adds channels without changing spatial size
Upsampling	ConvTranspose2d	kernel_size=3, stride=2, padding=1, output_padding=1	128 channels, InstanceNorm2d, ReLU activation
	ConvTranspose2d	kernel_size=3, stride=2, padding=1, output_padding=1	64 channels, InstanceNorm2d, ReLU activation
Final	Conv2d	kernel_size=7, stride=1, padding=3	Tanh activation to output

Figure 4: Detailed implementation of Generator

Discriminator

The discriminator network structure, which is a typical CNN-based classifier.

Layer	Parameters	Description
Conv2d	kernel_size=4, stride=2, padding=1	64 channels, LeakyReLU (negative slope=0.2)
Conv2d	kernel_size=4, stride=2, padding=1	128 channels, InstanceNorm2d, LeakyReLU
Conv2d	kernel_size=4, stride=2, padding=1	256 channels, InstanceNorm2d, LeakyReLU
Conv2d	kernel_size=4, stride=1, padding=1	512 channels, InstanceNorm2d, LeakyReLU
Conv2d	kernel_size=4, stride=1, padding=1	1 channel, Sigmoid activation

Figure 5: Detailed implementation of Discriminator

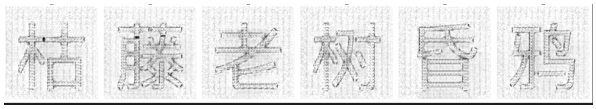

Figure 6: Unsatisfactory result of Diffusion from scratch

Figure 7: The good and bad results gained using GAN and CycleGAN

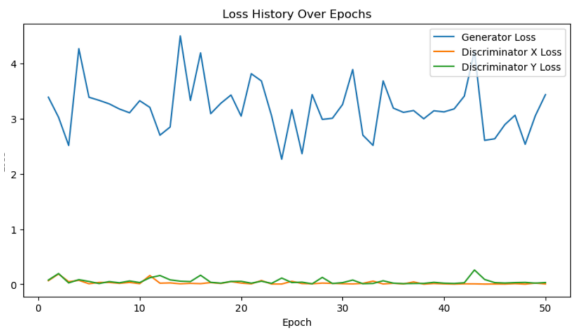

Figure 8: Example loss plot when training CycleGAN

Figure 9: 1st random generated examples

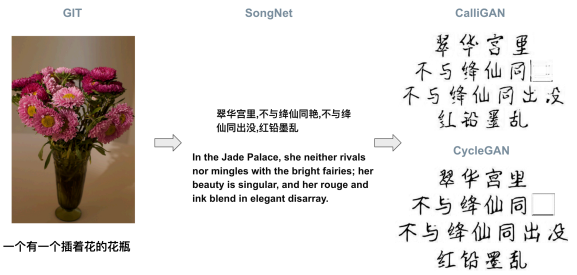

Figure 10: 2nd random generated examples

Figure 11: 3rd random generated examples

References

[1] Jaided AI. Easyocr: Ready-to-use optical character recognition with 80+ supported languages, 2020. Accessed: 2024-11-10.

[2] Cai-Liwei. Github - cai-lw/image-captioning-chinese: Image captioning in chinese using lstm rnn with attention mechanism. Available at `https://github.com/cai-lw/image-captioning-chinese`, 2018.

[3] Bo Chang, Qiong Zhang, Shenyi Pan, and Lili Meng. Generating handwritten chinese characters using cyclegan. In *IEEE Winter Conference on Applications of Computer Vision*, 2018.

[4] Google. Bert base chinese model card. https://huggingface.co/google-bert/bert-base-chinese, 2024.

[5] Hugging Face. Gitforcausallm - forward example documentation. https://huggingface.co/docs/transformers/main/model_doc/git#transformers.GitForCausalLM.forward.example, 2024.

[6] Piji Li, Haisong Zhang, Xiaojiang Liu, and Shuming Shi. Rigid formats controlled text generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 742–751, Online, July 2020. Association for Computational Linguistics.

[7] Microsoft. Git-base-coco model card. https://huggingface.co/microsoft/git-base-coco, 2024.

[8] Reddit User. D: Adding gaussian noise to discriminator layers, 2021. Accessed: 2024-11-10.

[9] Jianfeng Wang, Zhengyuan Yang, Xiaowei Hu, Linjie Li, Kevin Lin, Zhe Gan, Zicheng Liu, Ce Liu, and Lijuan Wang. Git: A generative image-to-text transformer for vision and language. *arXiv preprint arXiv:2205.14100*, 2022.

[10] Yuanhao Wang. Chinese calligraphy styles by calligraphers. https://www.kaggle.com/datasets/yuanhaowang486/chinese-calligraphy-styles-by-calligraphers/data, 2023. Accessed: 2023.