Neural Style Transfer Based on Fine Tuning Vision Transformer

Yisi Liu, Yujie Zhao, Yuanteng Chen, Xunzhe Zhou

1. Introduction

Neural style transfer (NST) is a technique that blends the content of one image (named content image) with the style of another image (named style image) using neural networks. The goal of the neural style transfer algorithm is to take a content image and a style image, then produce a new image that retains the content of the original image but has the artistic style of the style image.

Neural style transfer primarily involved the use of Convolutional Neural Networks (CNNs). During this period, Gatys et al[1] proposed to model the content of a photo as the feature responses from a pre-trained CNN, and then model the style of an artwork as the summary feature statistics. The main idea of the paper *A Neural Algorithm of Artistic Style* is to use the feature space provided by a deep neural network (VGG19 pretrained model) to separate style and content, and then recombine them to produce images that are a blend of the content of one image and the style of another.

Due to the inherent localized nature of CNNs, it's difficult to capture and preserve the comprehensive, global details of input images. As a result, traditional neural style transfer techniques often struggle with skewed or partial content representation. To decrease the biased representation issue of CNN-based style transfer methods, ArtFlow[2] comprised reversible neural flows and an unbiased feature transfer to provide a more balanced and unbiased approach to transferring style from one image to another. IEST[3] proposed an internal-external style transfer method with two contrastive losses.

However, all these methods cannot eliminate the biased representation issue of CNN-based models fundamentally. With the popularity of transformer-based architectures used in vision tasks[4], transformer-based models have become the mainstream framework in NST tasks. Benefiting from the ability of learning the global information of the input with the help of self-attention mechanism, the transformer has a strong representation capability to capture precise content representation and avoid fine detail missing. SANet[5] introduced a novel style-attentional network that efficiently and flexibly integrated the local style patterns according to the semantic spatial distribution of the content image. MCC[6] adopted self-attention structures in video style transfer tasks which can also be used in image style transfer. But these Transformer-based methods still face some issues, such as the inability of a single encoder-decoder structure to fully capture the content and the style.

In conclusion, traditional CNNs methods cannot capture long-range dependencies owing to the limited receptive field of convolution operation, while simply adopting single encoder-decoder transformer architecture in NST tasks will face the issue of inability of fully capturing the content and the style of images. In contrast, StyTr^2[7] *adopted two different transformer encoders to generate domain-specific* *sequences for content and style respectively and a multi-layer transformer decoder to stylize the content sequence according to the style sequence*, acquiring better results than other transformer-based methods.

Inspired by StyTr^2, we adopt the main architecture and replace the content and style encoders with the pretrained ViT model. We propose a two-stage training strategy, where we first freeze the pretrained encoders, and just train the decoder; then we wrap the encoders with LoRA and do joint training.

In summary, our main contributions include:

- **Better image quality**: through the incorporation of pretrained models and LoRA, our model performs better than other baselines in terms of both content loss and style loss, with the style loss nearly half of the baselines.
- **Less training computation**: the two-stage training strategy reduces the required training computation significantly, only less than half of the training computation required by the original StyTr^2 model.
- **Comprehensive ablation study**: we have tried different sets of hyperparameters to explore how LoRA and the pretrained model configuration affect the model performance.

Code and model checkpoint are available at
https://github.com/Zhouxunzhe/NST-Project

2. Method

The overall architecture of our model is shown in Figure 1.

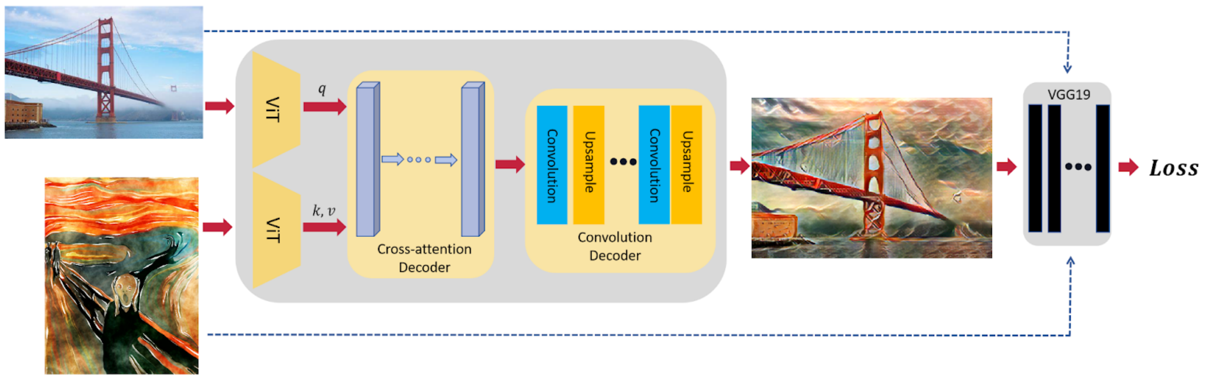

Figure 1. Model Architecture

2.1 Encoder

The encoder in our model is the pretrained Vision Transformer (ViT)[4], whose structure is illustrated in Figure 2. To align with our computational resources, we introduce a parameter to adjust the number of layers used in the pretrained model. Additionally, we leverage LoRA[8] to reduce the number of parameters and enhance computation efficiency.

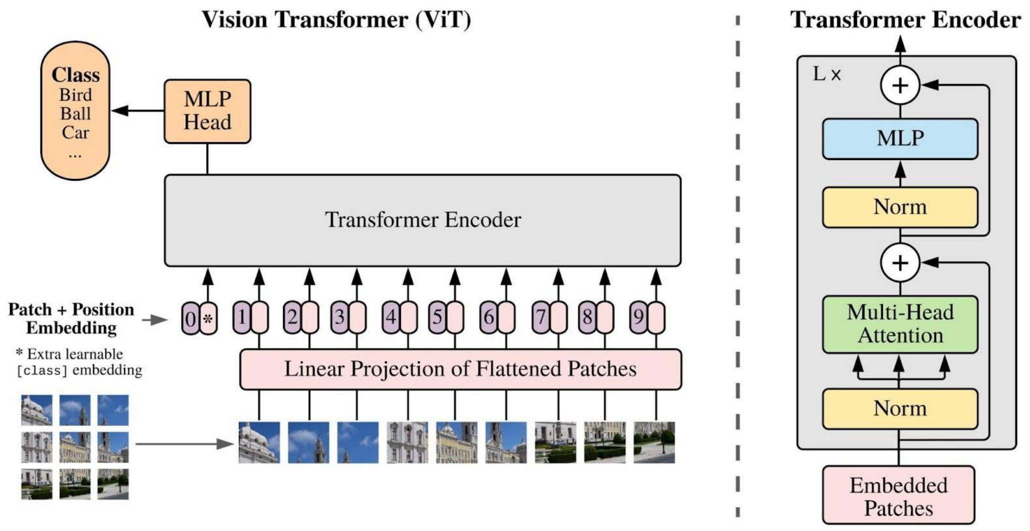

Figure 2. ViT architecture and Transformer Encoder, borrowed from the original ViT paper

The pretrained model used is the Google ViT base patch16-224-in21k model [9], i.e. ViTModel. We set the model's final classification head, the Pooler layer, to an identity layer, as the Pooler layer is unnecessary for our non-classification purpose.

We also adopt the method of LoRA to better fine tune the pretrained encoder. LoRA reduces the number of trainable parameters by reparameterizing the pretrained weight matrix $W \in R^{d \times k}$ into two low-rank matrices $B \in R^{d \times r}$ and $A \in R^{r \times k}$, and freezing the original pretrained weights, only training A and B, as illustrated in Figure 3.

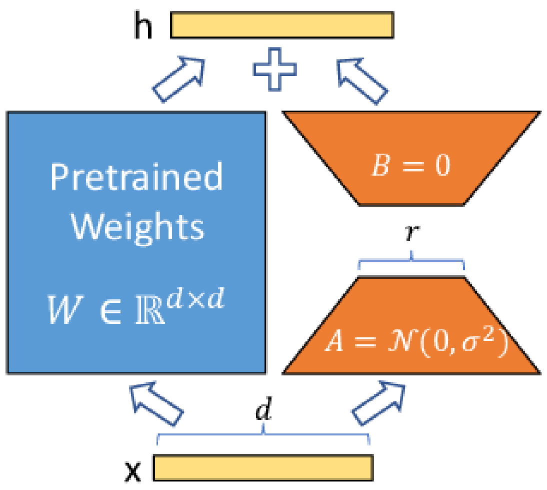
h
Pretrained Weights
$W \in \mathbb{R}^{d \times d}$
$B = 0$
r
$A = \mathcal{N}(0, \sigma^2)$
d
x

Figure 3. LoRA architecture, borrowed from the original LoRA paper

Specifically, for the pretrained weight matrix W_0, LoRA constrains its update by expressing it through a low-rank decomposition $W_0+\Delta W = W_0+BA$, where $B\in \mathrm{R}^{d\times r}$, $A\in \mathrm{R}^{r\times k}$, $r \ll \min(d, k)$. Throughout training, W_0 remains frozen, exempt from gradient updates. We only need to train low-rank weight matrices A and B. For $h = W_0x$, the modified forward pass is now given by:

$$h = W_0x+\Delta Wx = W_0x+BAx$$

For initialization, LoRA employs a random Gaussian initialization for A and zero initialization for B, ensuring $\Delta W=BA$ is initially zero. Subsequently, LoRA scales ΔWx by α/r, where α is a constant relative to r. When utilizing the Adam optimizer, tuning α is similar to tuning the learning rate. Consequently, we set α to the first $r=16$ without further tuning.[10]

2.2 Decoder

Our cross-attention decoder adopts the transformer decoder structure shown in Figure 4.[7]

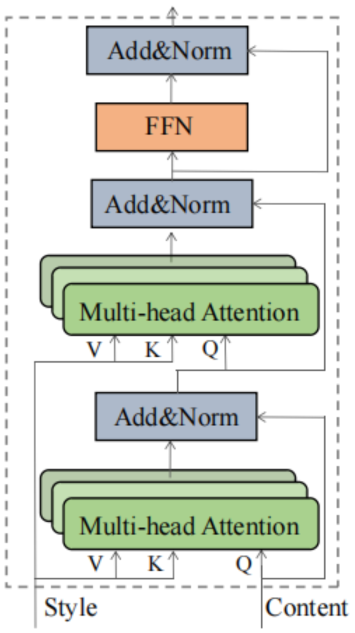

Figure 4. Transformer decoder, borrowed from the original StyTr^2 paper

Each cross-attention decoder has two multi-head cross-attention layers and one feed forward network. It accepts the style sequence and content sequence output by the two encoders respectively. We use the style sequence to generate the key and value of cross-attention, and use the content sequence to generate the query as shown in Figure 1. Cross-attention acts as a way to fuse the features of content images and style images.

Our convolution decoder is similar to the one in the original StyTr^2. The convolution decoder restores the number of channels through the convolutional layers, restores the image size through upsample layers and reconstructs the image to obtain the output image. We used 4 upsample layers with the scale being 2, which is to upsample the image 16 times, since the patch size of the pretrained ViT model is 16*16. For the number of convolution layers, the reconstruction of images will be bad if the number is too small, and the learnable parameters will be too many if it is too large, harming the gradient flow and the fine tuning of the encoder. In this work, we set the number of convolution layers to be 9.

2.3 Network Optimization

Similar to [2, 7], we use the pretrained VGG-19 as the feature extractor and use the feature vectors of certain layers' output to construct the loss function. We use features of the content image I_c and the generated image I_o to construct the content loss L_c as

$$L_c = \frac{1}{N_l}\sum_{i=1}^{N_l} ||\phi_i(I_o) - \phi_i(I_c)||_2$$

where ϕ_i denotes the features extracted from the i-th layer of VGG-19 and N_l is the number of layers. Similarly, we use features of the content image I_s and the generated image I_o to construct the content loss L_s as

$$L_s = \frac{1}{N_l}\sum_{i=1}^{N_l} (||\mu(\phi_i(I_o)) - \mu(\phi_i(I_c))||_2 + ||\sigma(\phi_i(I_o)) - \sigma(\phi_i(I_c))||_2$$

where μ denotes the mean and σ denotes the standard deviation. Then the total loss L is the weighted sum of L_c and I_s as

$$L = \lambda_c L_c + \lambda_s L_s$$

In practice, the normalization coefficient in L_c and L_s can be incorporated into λ_c and λ_s. In addition, we also tried the identity loss mentioned in [7], but with no improvement, so we abandon those loss functions.

Unlike the origin Neural Style Transfer model by Gatys et al.[1], which used the pixel values of the image as learnable parameters, so the network needs to be trained every time a new image is required, we use the above loss functions and gradient descent to train the encoders and decoders, so that we can just do one forward pass to get the stylized image during test time.This can greatly improve the efficiency of image generation.

3. Experiments

3.1 Datasets

For content images, we use COCO [11]. And WikiArt [12] is used as the style images. We split the datasets into (content, style) pairs, and choose 56462 pairs as the training set, 4000 pairs as the validation set. After training and validation, we choose 352 unseen pairs as the test set, and report the test set results.

During training, to make the input compatible with the pretrained ViT model, we first resize each input (content, style) pairs into the same size of 3*224*224, then use the standard ImageNet mean and standard deviation to normalize the resized images. During test time, content images and style images of any size are supported.

3.2 Implementation Details

The training pipeline is divided into 2 stages. The first stage is to train the decoder with the two encoders frozen. The second stage is to unfreeze the two pretrained encoders, wrap them with LoRA, and then do joint training with the decoder.

For the first stage, since the transformer decoder and cnn decoder are trained from scratch, the gradients coming back from them are mostly garbage gradients in the early stages of the training process. Therefore, we find that the model converges faster if we freeze the encoders first, train the decoders for 2 epochs so that their parameters are in a more proper place, then move on to stage 2. Besides, since in stage 1 we do not update any parameters in the encoders, it is trained much faster compared to joint training from the very beginning.

In the second stage, we wrap the encoders with LoRA, and fine tune the pretrained ViT as well as continuing training the decoder.

During training, we set the batch size to be 16, with 2 epochs for stage 1 and 9 epochs for stage 2. We choose Adam as the optimizer, with initial learning rate 1e-4, and the learning rate is multiplied by 0.3 in stage 2 at epoch 4, 6, 8. The coefficients of the loss function is set to be $\lambda_c = 1, \lambda_s = 7$ without identity loss. The training loss is saved every 20 batches, and validation is done every 400 batches. We also explore different model architecture hyperparameters. This includes changing the LoRA rank, LoRA target modules, and the number of layers of each encoder and decoder. The two-stage training and validation loss curves are shown in Figure 7 in the Appendix.

It is worth mentioning that the two-stage training that we propose, which contains 11 epochs in total, with 56462 pairs of images in one epoch, requires going through 56462 * (2 + 9) = 621,082 image pairs to converge. Compared to the original StyTr^2 paper, where it requires going through 160,000 iterations of batch size 8, i.e. 160, 000 * 8 = 1,280,000 image pairs, our training strategy saves more than half of the computation!

3.3 Evaluation

3.3.1 Comparison with Baselines

Our best model is given by the config of 6 encoder layers, 3 decoder layers, and LoRA rank is 16, LoRA target modules are “query”, “value”, “key” and “dense”. Based on the loss equations mentioned in section 2.3, we compare the content loss and the style loss on the held-out unseen image pairs using our best model and the baselines. The results are shown in Table 1.

Method / Metric	Ours	Artflow	IEST	MCC	SANet
Content loss	**9.097**	9.200	9.206	9.204	9.687
Style loss	**1.514**	2.745	2.805	2.945	2.535

Table 1. Test results comparison of our method and baselines

It is clear that our model is the best in terms of both the content and the style, with the style loss at least 40% lower than the baselines.

A comparison of generated images is shown in Figure 5. To make the illustration easier, all images are resized into the same smaller size after image generation. Note that in the 4th row, where a large content image (3*1600*1064) of the Berkeley campus and a style of the Foxes are chosen, our model performs significantly better than other models, preserving the content details best and with a clear style.

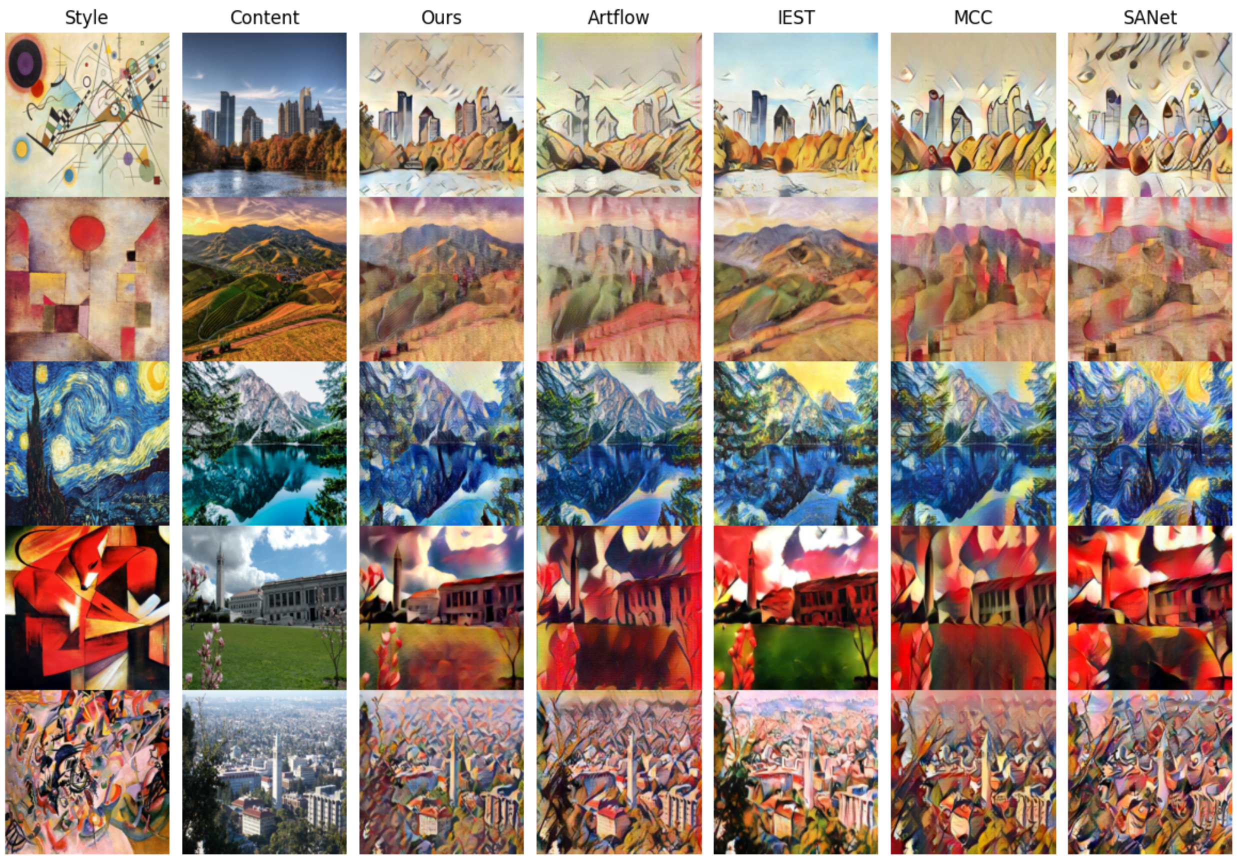

Figure 5. Generated images comparison of our method and baselines

3.3.2 Training Progress

Figure 6 shows a set of images generated in the early, middle, and late stages of training our best model, corresponding to the end of 1st epoch of stage 1, the end of 1st epoch of stage 2, and the end of 9th epoch of stage 2 respectively. The model is already able to capture the style well in the early stage, and as training progresses, the content details are preserved better.

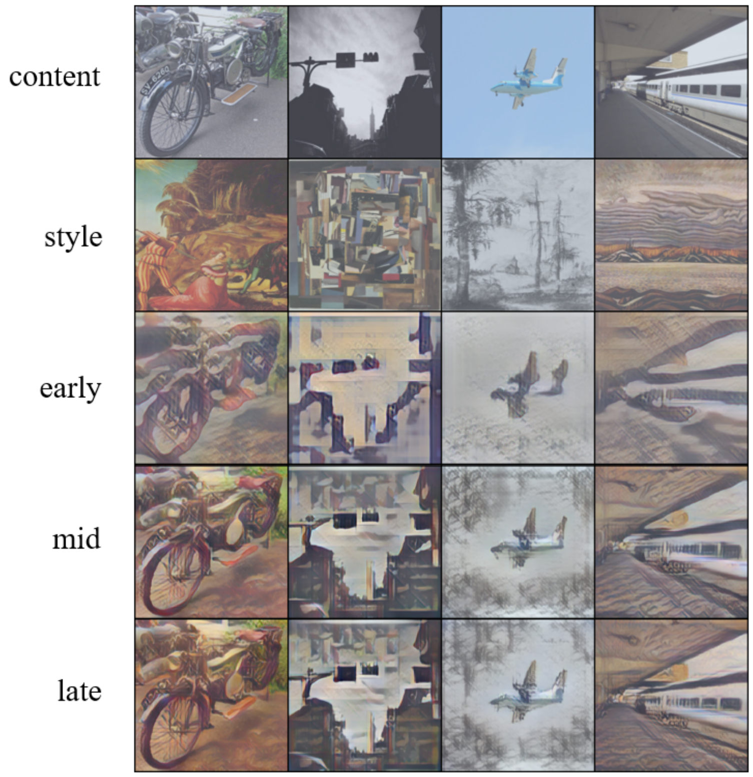

Figure 6. Images generated in the early, mid and late stages of training

3.3.3 Architecture Hyperparameter Search

We have tried different sets of hyperparameters for fine tuning the ViT encoders. This includes the LoRA rank, the LoRA target modules, the number of encoder and decoder layers. After training using the same training strategy in section 3.2, we test the different configurations using the same set of test image pairs mentioned above. The loss table is shown in Table 2.

Exp id:	Lambda_c	Lambda_s	Lambda_id1	Lambda_id2	r	Target_modules	Encoder_nlayer	Decoder_nlayer	Content loss	Style loss
1	1	7	0	0	16	[“query”,”value”]	6	3	9.134	1.533
2	1	7	0	0	32	[“query”,”value”]	6	3	9.113	1.544
3	1	7	0	0	16	[“query”,”value”,“key”]	6	3	9.128	1.480
4	1	7	0	0	16	[“query”,”value”]	6	4	9.192	1.473
5	1	7	0	0	16	[“query”,”value”]	8	3	9.171	1.559
6	1	7	0	0	16	[“query”,”value”]	4	3	9.119	1.548
7 (best)	1	7	0	0	16	[“query”,”value”, “key”, “dense”]	6	3	9.100	1.514
8	1	7	0	0	64	[“query”,”value”]	6	3	9.183	1.555

Table 2. Test results of different model configurations

In general, varying the hyperparameters of the encoders does not affect the performance that much. Since we are only fine tuning the pretrained ViT as the encoders, the number of trainable parameters of LoRA-wrapped encoders is far fewer than the one of the decoder (about 90% fewer). Therefore, it is the decoder that affects the performance the most.

Upon closer inspection, we find that increasing the target modules of LoRA helps the most (exp. 1, 3, 7). It is expected that with more modules fine tuned, more modules will be customized to the downstream task, and the performance will be better. Increasing the LoRA rank also helps a little when comparing exp. 1 and exp. 2, where r increases from 16 to 32. However, when increasing r to 64, the performance is even worse (exp. 1 and exp. 8). There are similar phenomena in the number of encoder layers chosen (exp. 1, 5, 6), where increasing from 6 layers to 8 layers gives worse performance. We think it is probably because r = 16 and nlayer = 6 has somehow reached a performance sweet spot, where the number of parameters for fine tuning is just enough to provide task-specific information, and not too much to contaminate the pretrained model.

4. Conclusion

In this work, we propose to use the pretrained ViT model to be the content and style encoders of the StrTr^2 architecture. We adopt the two-stage training strategy, where we first freeze the pretrained ViT encoders, and just train the decoder; then unfreeze the encoders, wrap them with LoRA and do joint training. Our training strategy only needs less than half of the computation of the original StrTr^2 model, and our model still performs the best compared to other baselines, in terms of both the content and the style of the generated images. The ablation study tells us that increasing the target modules of LoRA helps the most when fine tuning the encoder.

5. References

[1] Gatys, Leon A., Alexander S. Ecker, and Matthias Bethge. "A neural algorithm of artistic style." *arXiv preprint arXiv:1508.06576* (2015).
[2] An, Jie, Siyu Huang, Yibing Song, Dejing Dou, Wei Liu, and Jiebo Luo. "Artflow: Unbiased image style transfer via reversible neural flows." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 862-871. 2021.
[3] Chen, Haibo, Zhizhong Wang, Huiming Zhang, Zhiwen Zuo, Ailin Li, Wei Xing, and Dongming Lu. "Artistic style transfer with internal-external learning and contrastive learning." *Advances in Neural Information Processing Systems* 34 (2021): 26561-26573.
[4] Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).
[5] Park, Dae Young, and Kwang Hee Lee. "Arbitrary style transfer with style-attentional networks." In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5880-5888. 2019.
[6] Deng, Yingying, Fan Tang, Weiming Dong, Haibin Huang, Chongyang Ma, and Changsheng Xu. "Arbitrary video style transfer via multi-channel correlation." In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, pp. 1210-1217. 2021.
[7] Deng, Yingying, Fan Tang, Weiming Dong, Chongyang Ma, Xingjia Pan, Lei Wang, and Changsheng Xu. "Stytr2: Image style transfer with transformers." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11326-11336. 2022.
[8] Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. "Lora: Low-rank adaptation of large language models." *arXiv preprint arXiv:2106.09685* (2021).
[9] "Google/Vit-Base-Patch16-224-In21k · Hugging Face," n.d., https://huggingface.co/google/vit-base-patch16-224-in21k.
[10] Yang, Greg, and Edward J. Hu. "Feature learning in infinite-width neural networks." *arXiv preprint arXiv:2011.14522* (2020).
[11] Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft coco: Common objects in context." In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740-755. Springer International Publishing, 2014.
[12] Saleh, Babak, and Ahmed Elgammal. "Large-scale classification of fine-art paintings: Learning the right metric on the right feature." *arXiv preprint arXiv:1505.00855* (2015).

6. Appendix

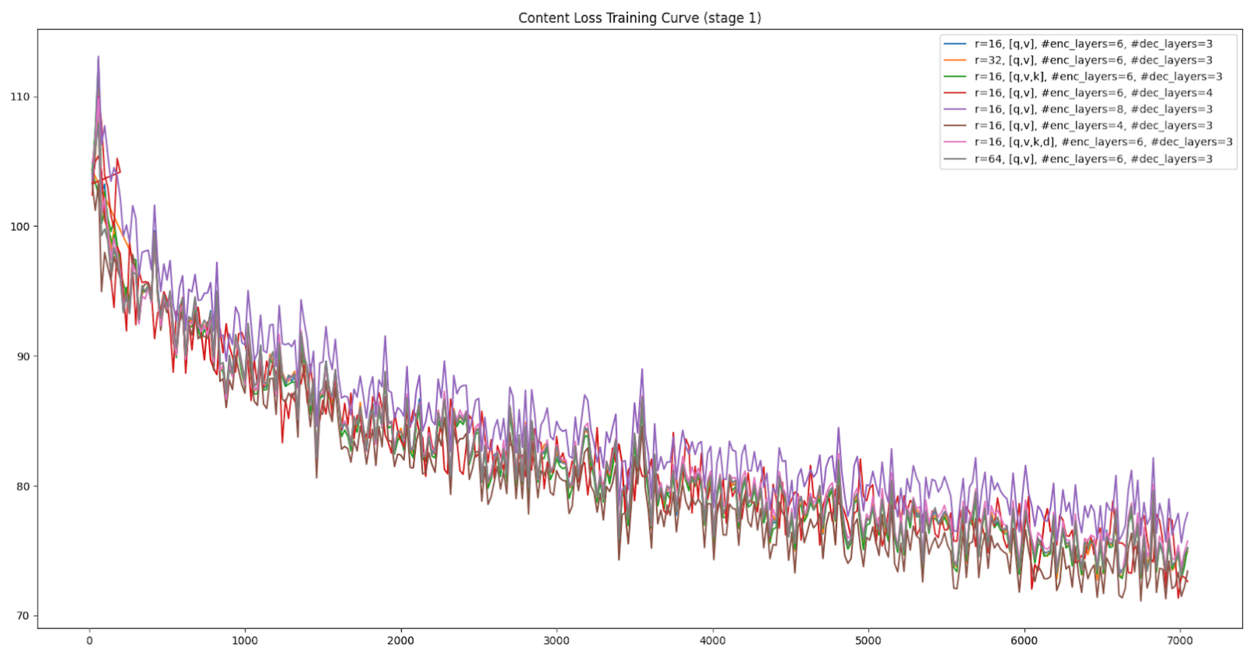

Figure 7.a. Content loss training curve of training stage 1

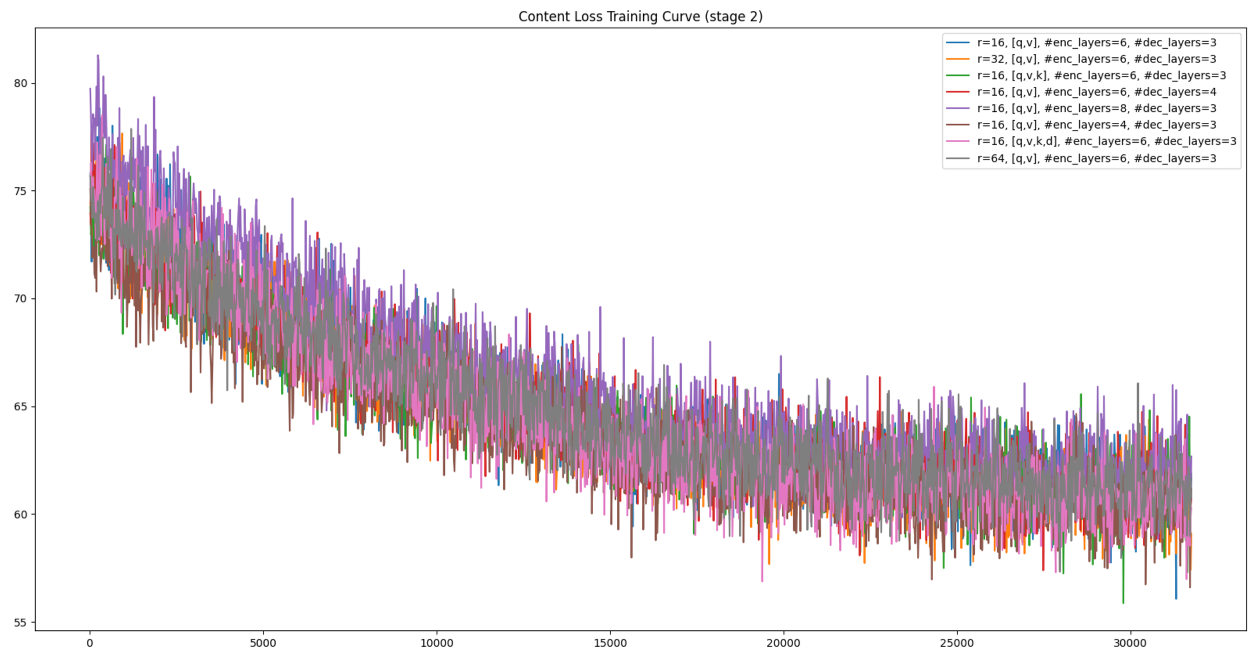

Figure 7.b. Content loss training curve of training stage 2

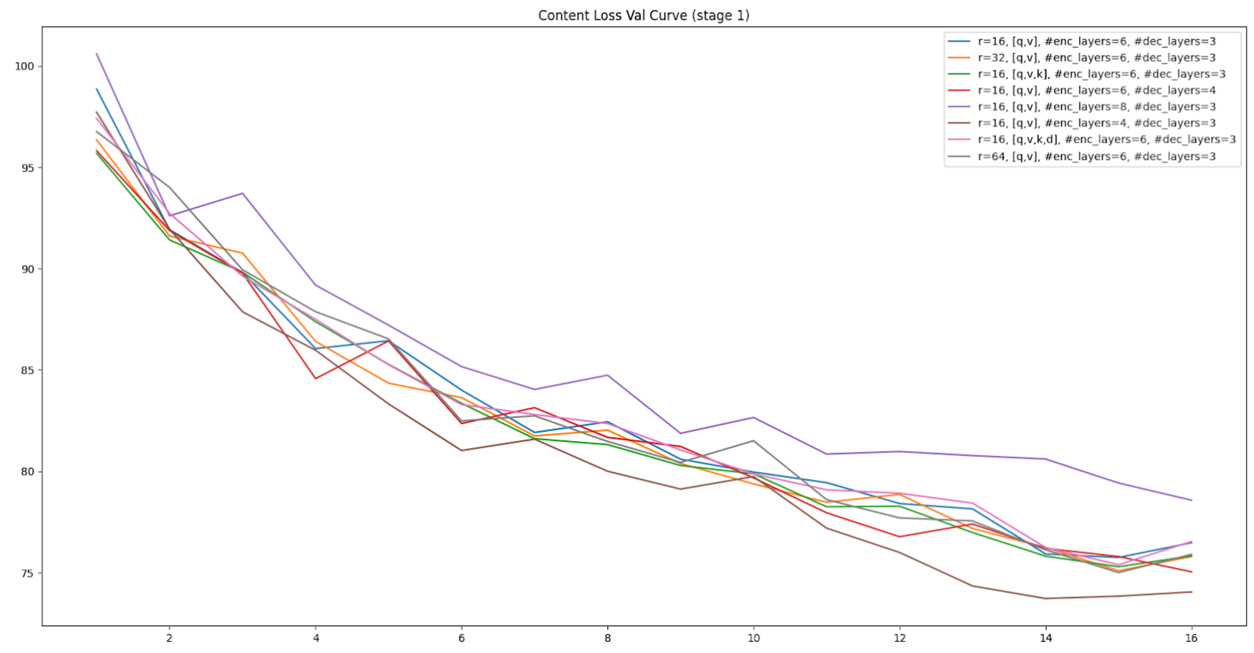

Figure 7.c. Content loss validation curve of training stage 1

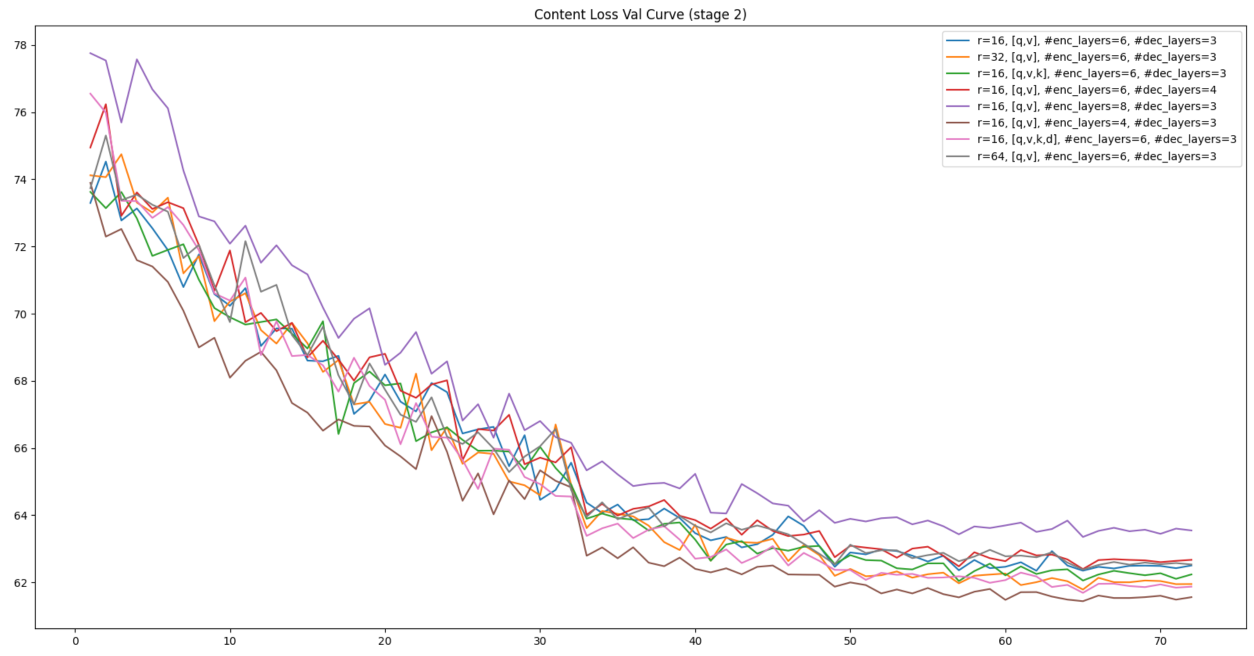

Figure 7.d. Content loss validation curve of training stage 2

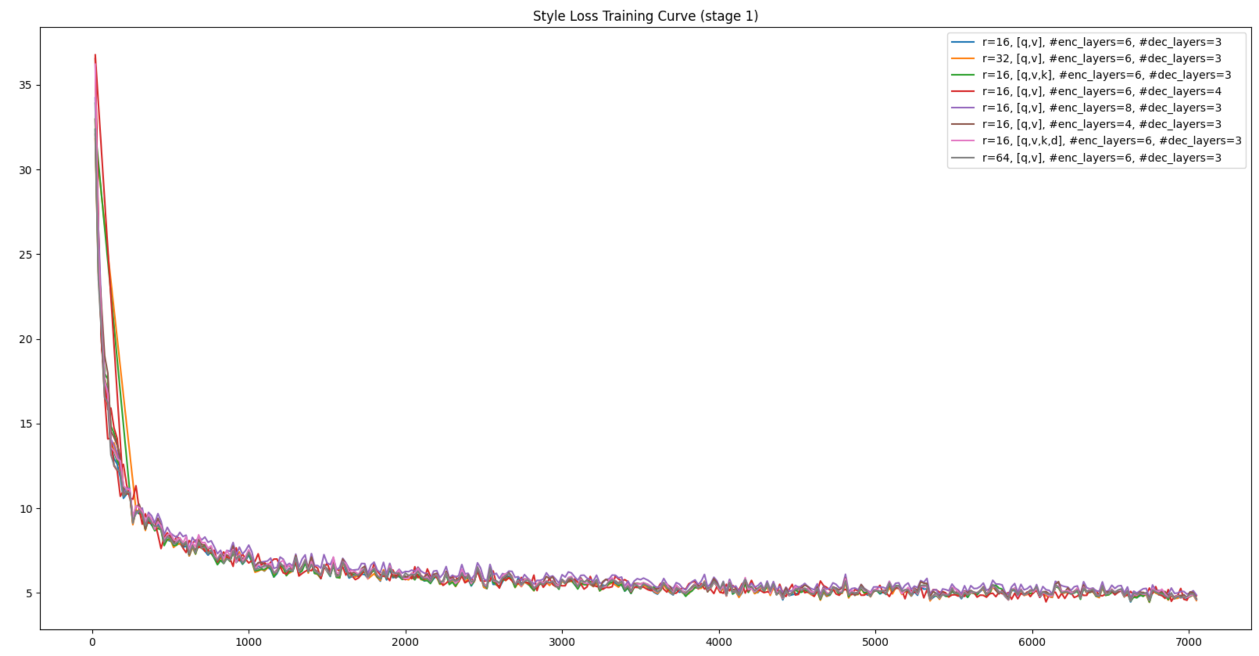

Figure 7.e. Style loss training curve of training stage 1

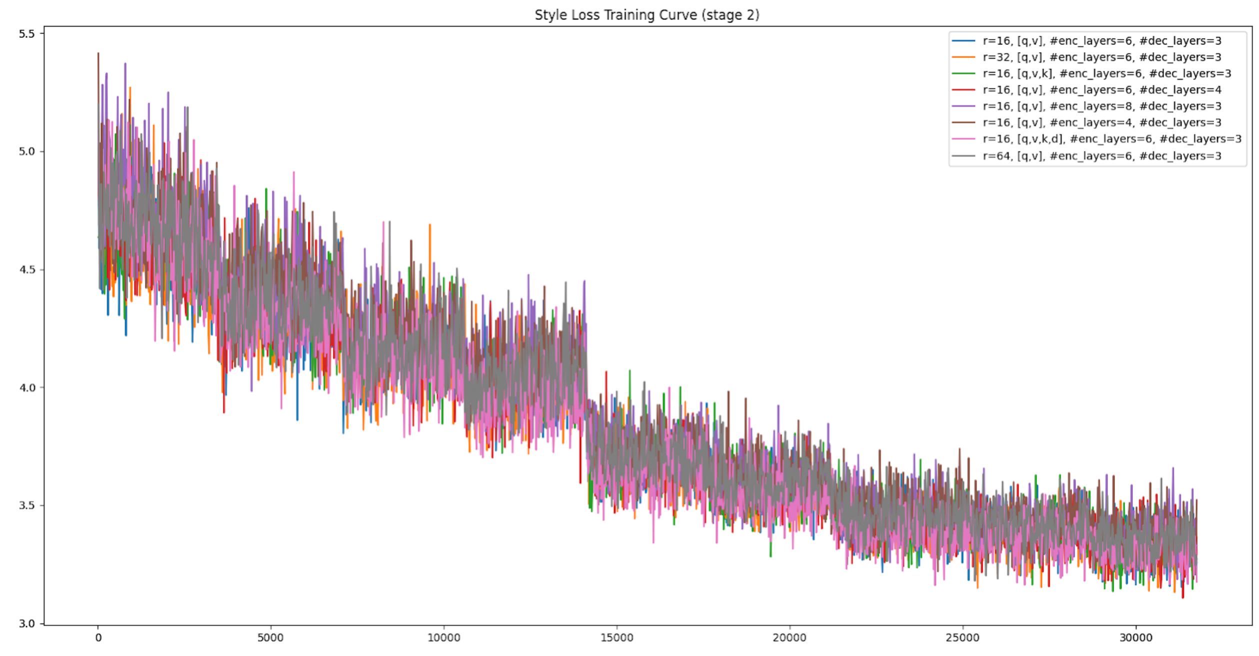

Figure 7.f. Style loss training curve of training stage 2

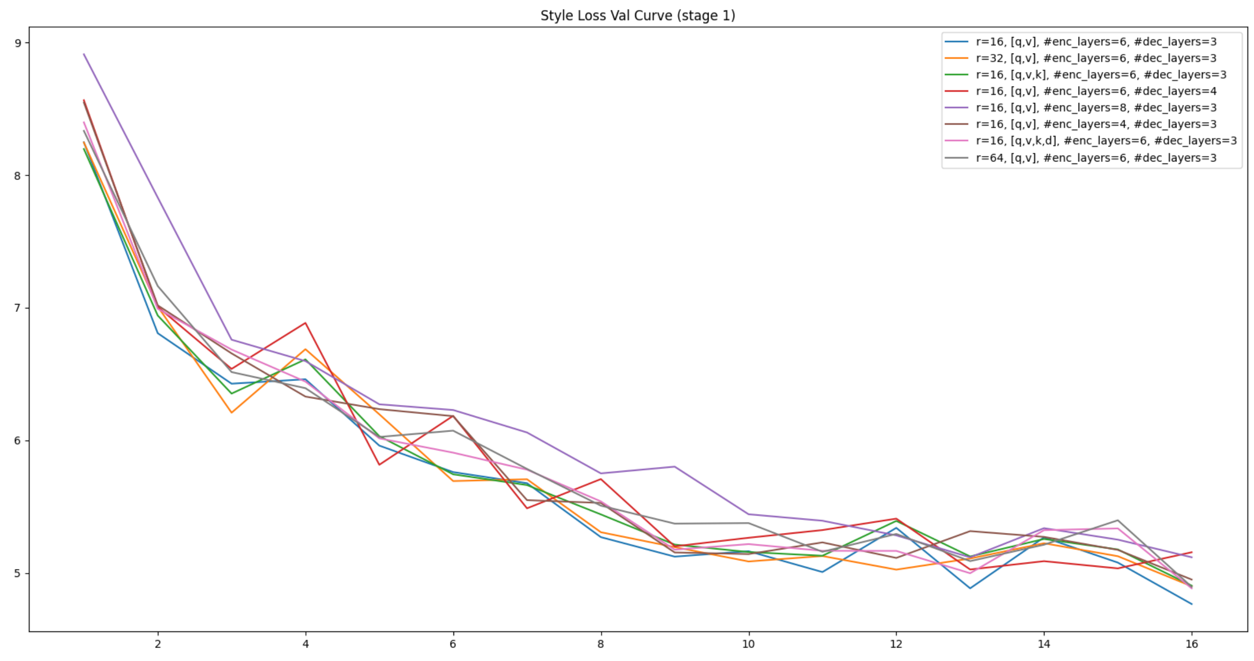

Figure 7.g. Style loss validation curve of training stage 1

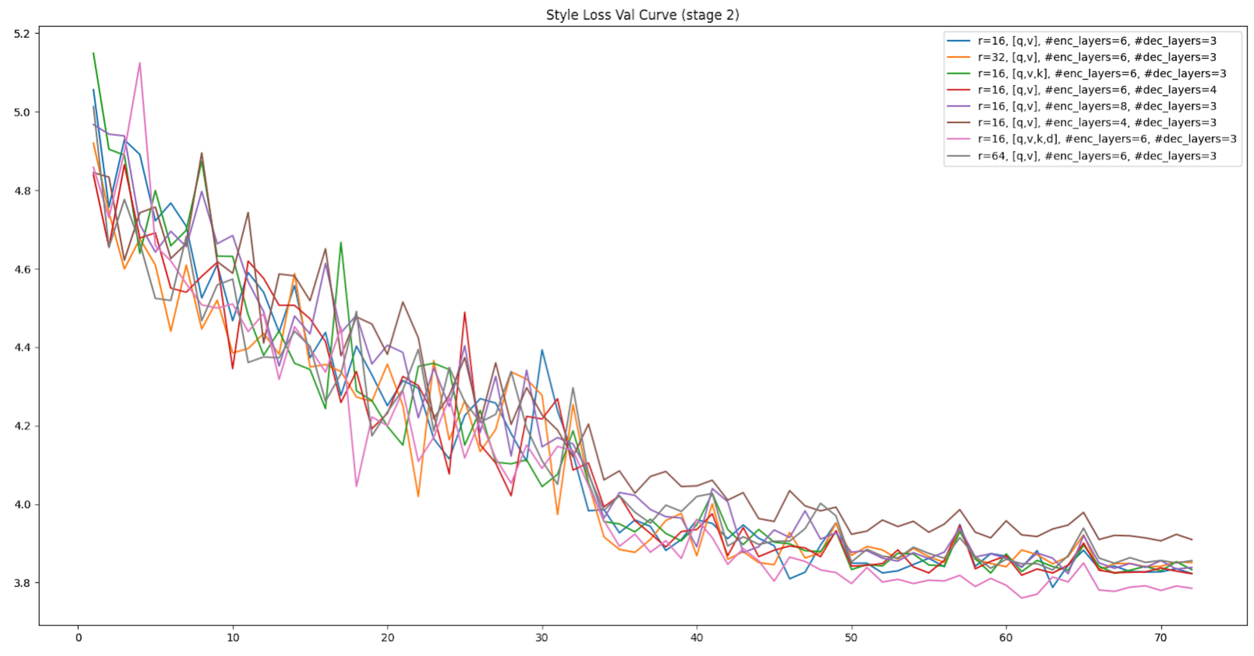

Figure 7.h. Style loss validation curve of training stage 2

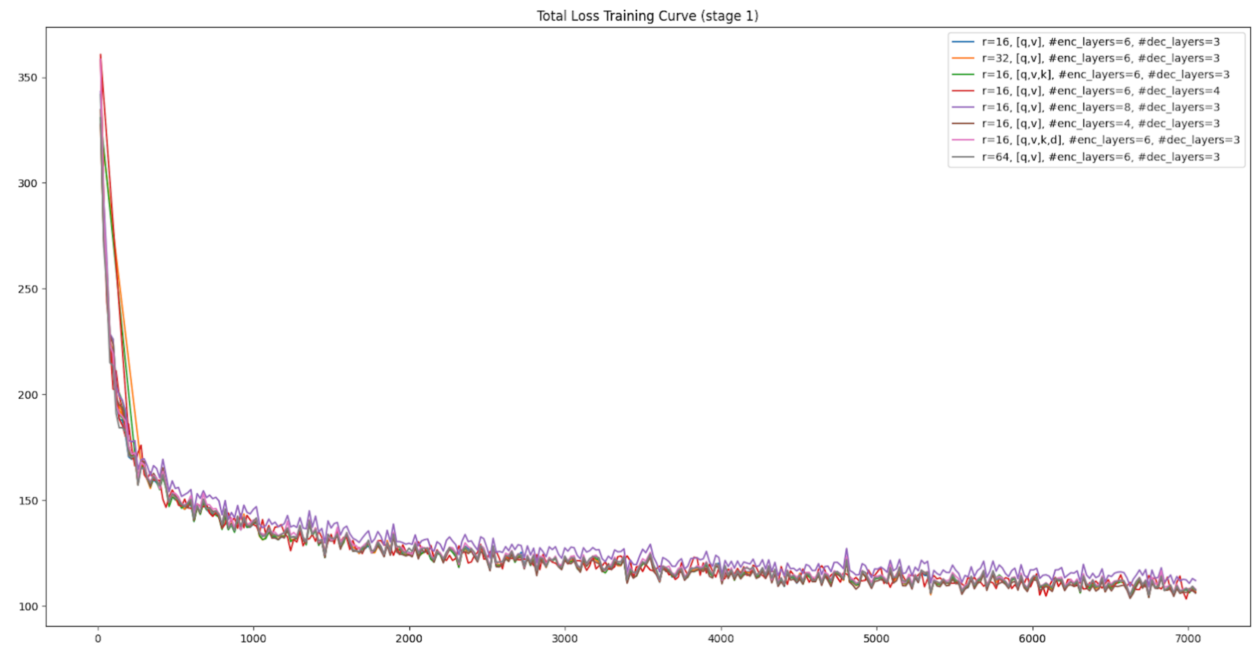

Figure 7.i. Total loss training curve of training stage 1

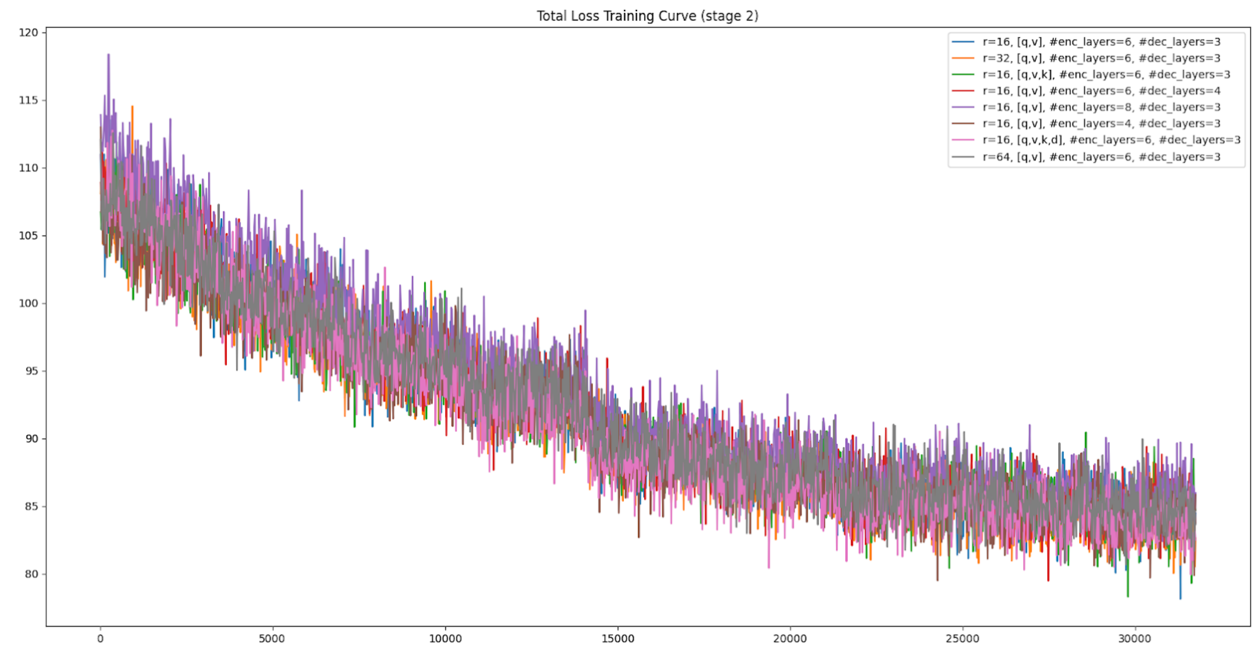

Figure 7.j. Total loss training curve of training stage 2

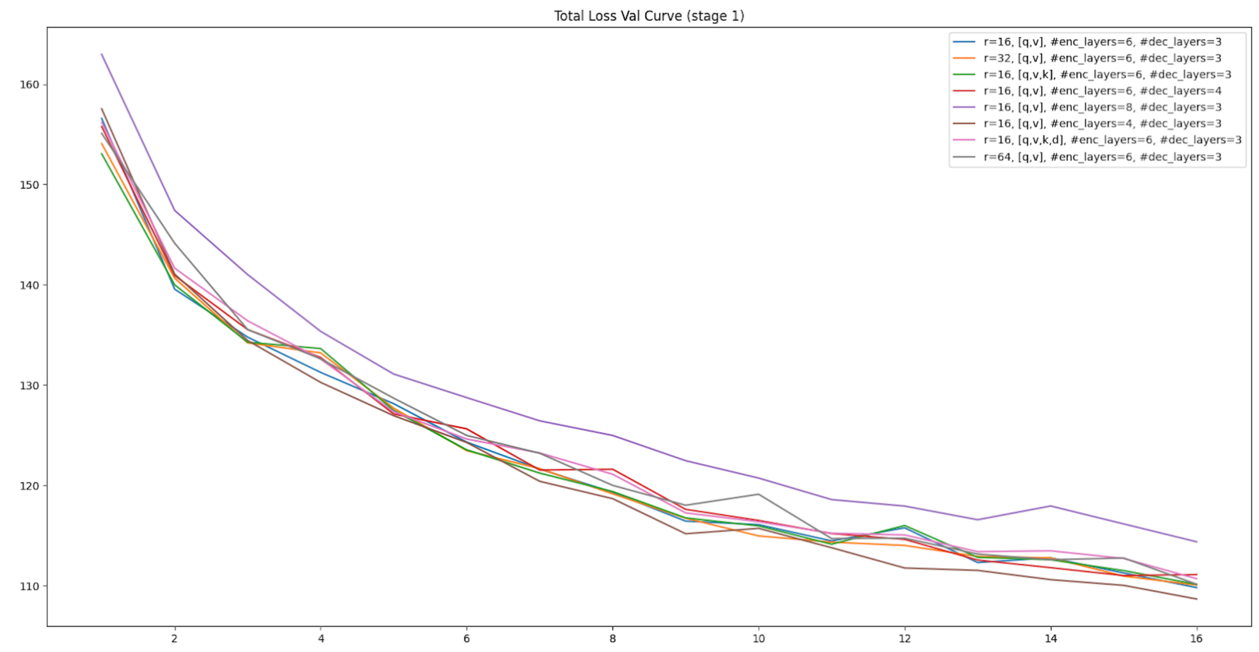

Figure 7.k. Total loss validation curve of training stage 1

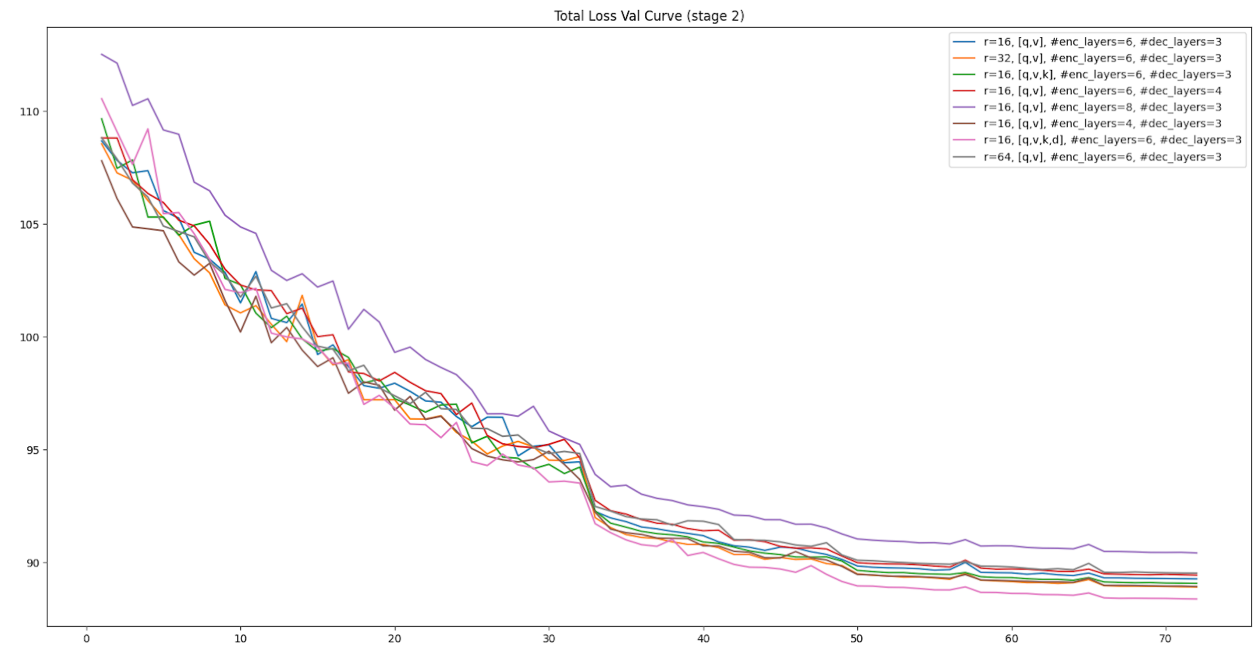

Figure 7.l. Total loss validation curve of training stage 2

7. Self-review

7.1 What is the main goal of the project?

We devise a model based on StyTr^2 that can concurrently enhance runtime efficiency and performance by fine-tuning on a pre-trained model.

7.2 What are the main claims?

In summary, our main claims include: better image quality, less training computation and comprehensive ablation study, as shown in the *Introduction* part.

7.3 What are the experiments?

7.3.1 What is the evaluation protocol?

We calculate content loss L_c and style loss L_s on the test set, the lower the better.

7.3.2 What is the data?

For content images, we use COCO. And WikiArt is used as the style images. We split the datasets into (content, style) pairs, and choose 56462 pairs as the training set, 4000 pairs as the validation set. After training and validation, we choose 352 unseen pairs as the test set, and report the test set results.

7.3.3 What is the task?

Style transfer is a task in computer vision that aims to alter the visual style of an image while preserving its content. Using deep learning techniques, particularly Convolutional Neural Networks (CNNs), this process involves separating and manipulating the content and style of two images. The ultimate goal is to generate a new image that combines the content of one image with the stylistic elements of another, resulting in a visually appealing composition that merges the characteristics of both input images. This task has applications in art, design, and image processing, allowing users to apply different artistic styles to their photos or graphics.

7.4 How do the experiments support the goal/claims of the paper?

As shown in section 3.3.1, when tested on the held-out test image pairs, our model is better than other baselines in terms of both content and style loss, indicating a better generation quality. The comparison of generated images can also be found in Figure 5.

As mentioned in section 3.2, by adopting the proposed two-stage training strategy, it takes less than half of the computation of the original StyTr^2 model for our model to converge. This saves computation significantly.

In section 3.3.3, we conduct comprehensive experiments on model hyperparameter exploration. The ablation study shown in Table 2 tells us that increasing the number of target modules in LoRA helps the most in fine tuning the encoders.

7.5 Are any of the limitations discussed in the paper?

As mentioned in section 3.3.3, changing the hyperparameters of the encoders does not affect the model performance that much, since the decoders are trained from scratch, and the number of trainable parameters of the LoRA-wrapped encoders is overwhelmed by the much larger number of parameters of decoders.

7.6 What are the strengths of the paper?

Our model, which is based on the architecture of StyTr^2, utilizes fine tuning the pretrained ViT model. It achieves better generation quality compared to baselines, and reduces the training computation.

7.7 What are the weaknesses of the paper?

As mentioned earlier, the impact of decoder is larger than fine tuning the encoder, which may be the bottleneck of the whole system.

7.8 Provide a suggestion for improving the paper.

Maybe in the future, we could try to modify the existing pretrained transformer decoder model to fit in with our task, such that the decoder is also in a good starting position.

7.9 What is the relevant work?

Gatys et al first proposed the NST task and they used the pre-trained CNNs to model the content and style of images in the paper *A Neural Algorithm of Artistic Style.* ArtFlow, IEST, etc followed the CNNs-based NST and introduced reversible neural flows and internal-external style transfer method to decrease the biased representation issue of CNNs-based style transfer methods. SANet, MCC, etc introduced transformer-based architectures into NST tasks and got better results than CNNs-based architectures.

7.10 Is the paper reproducible?

Yes, our code has been released on Zhouxunzhe/NST-Project: NST Project CS182, Berkeley (github.com).

7.10.1 Can you rerun the experiments?

Yes, please follow the instructions of our repo to rerun the experiments.

7.10.2 Can you reproduce the results in the paper?

Yes, all of the hyperparameters of the experiments can be found in our repo.

7.11 Are all the plots in the paper clearly interpretable with well-defined and explained axes, with methodology clearly explained in the paper text?

Yes, they are.

7.12 Is the English in the paper correct and clear?

Yes, it is.

7.13 Do you have any feedback on any TODOs that the authors have left at this stage?

We have finished the project. (except for the TODOs provided by other reviewers which are coming soon)