Manipulating Image Style Transformation via Latent-Space SVM

Qiudan Wang
ViaX Online
East District of Jianwai SOHO, Beijing, China
qiudan.wang@outlook.com

Abstract

Deep Neural Networks have been proved as the go-to approach in modeling data distribution in a latent space, especially in Neural Style Transfer (NST), which casts a specific style extracted from a source image to another target image by calibrating the style and content information in a latent space. While existing methods focuses on different ways to extract features that more precisely describe style or content information to improve existing NST pipelines, the latent space of the NST model has not been well-explored. In this paper, we show that different half-spaces in the latent space are actually associated with particular styles of a network's generated images. The corresponding constraints of these half-spaces can be computed by using linear classifiers, e.g. a Support Vector Machines (SVM). Leveraging the understanding of the relation between half-spaces in the latent space and output style, we propose the Linear Modification for Latent Representations (LMLR), a method that effectively increases or decreases the level of stylizing in the output image for any given NST model. We empirically evaluate our method on several state-of-the-art NST models and show that LMLR can manipulate the level of stylizing in the output image.

1. Introduction

Neural Style Transfer (NST) utilizes Convolutional Neural Networks (CNNs) to jointly encode the content and style information of images to generate a new image by mixing desired contents and styles. One well-applied method that separates the content and the style of an image is to consider internal representations from shallower layers of a CNN that encode the content information by directly "watching" localized features in the input whereas deeper layers learn more high-level features, among which one is the style of the input. Recent work has shown that instead of learning a synthesized image for each particular pair of the content and style image [5], one can directly learn a function to realize NST for any given input pairs [11], also known as *universal style transfer*.

The perceived success of *universal style transfer* in efficiently realizing NST, however, is still opaque to humans as the black-box nature of neural networks. Understanding the internal process of an style transfer model is expected to benefit the development and improvement of existing ones. Previous work on explaining discriminative models, i.e. text and image classifiers, attributes a model's output over input features [15]. However, similar to Generative Adversarial Networks (GANs), universal style transfer models usually learn a generative model that produces synthesized images from a latent space, therefore, do not fit into the scope of existing explanation tools. To Pursue better explanations on generative models, InfoGAN [3] demonstrates particular distributions in the latent space corresponding to some certain output behavior. Further, InterfaceGAN [15] illustrates the latent space of a generative model can be partitioned into half-spaces, which are shown to be associated with some interests of the output images, e.g. the brightness of the images and the orientation of the objects.

Given the similarity between generative models and the decoding process of the synthesized images in NST, we peer into the NST pipeline to show *how a style is combined with a content image in the latent space*, leveraging the technique that partitions the latent space from InterfaceGAN [15]. Specifically, we show that, by modifying the latent representation of the NST pipeline in a correct way, we can manipulate different attributes of its output, e.g. the synthesized images. Our method allows us to improve the performance of existing NST pipeline on several widely-used metrics and examples of our approach are shown Fig. 1.

In summary, we firstly demonstrate that different regions of the output space of the encoder, a.k.a. the latent space, of an NST model correspond to different stylized output images. And we secondly show that the decision boundaries between regions that determine the output style can be effectively learned by using a linear model, i.e. Support Vector Machine (SVM) [19]. Thirdly, by leveraging the connection between regions in the latent space and output style, we propose *Linear Modifications for Latent Representa-*

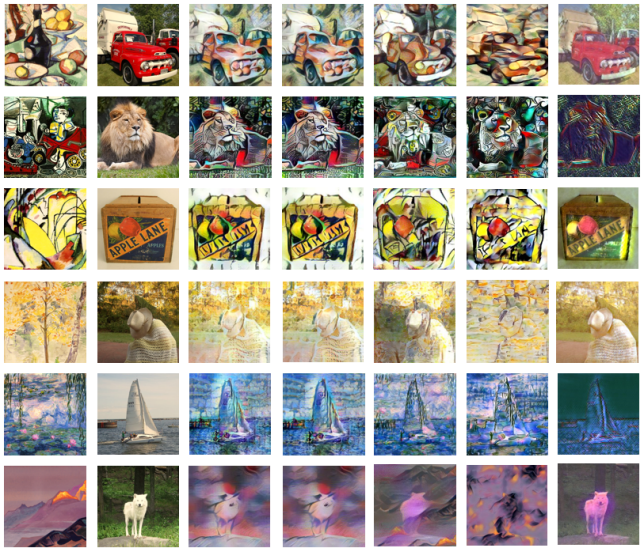

Figure 1. Example style transfer results. The first image of each row is a style image and the second is a content image, followed by the synthesized images of four different methods.

tions (LMLR) as a method to adjust the stylization level of generated images. Our empirical results over WikiArt [10] and ImageNet [13] show the effectiveness of our method.

The rest of the paper is organized as follows: In Sec. 2 we first discuss the notations and the background of style transfer. We introduce the motivation and implementation of the latent-space SVM for manipulating style transfer in Sec. 3. We further evaluate our approach with several NST pipeline and a recent work that can adjust the style in real-time in Sec. 4. We then discuss the recent work that also aims to provide adjustable style transfer later in Sec. 5. We end the paper by discussing an limitation of our work and the conclusion we arrive at Sec. 6 and 7.

2. Background

In this section, we firstly introduce the notations we use in Sec. 2.1. And secondly we make a brief introduction of NST, then focus on a specific method AdaIN [7], with which our method cooperates in Sec. 2.2.

2.1. Notations

Throughout the paper we will use x to denote a scalar and the bold font $\mathbf{x}$ to denote a vector. We use $||\cdot||$ to denote the ℓ_2 norm of a vector and for a set A, we write its cardinality as $|A|$. Furthermore, We describe different parts of a NST pipeline. One standard architecture that has been used in the literature [8, 7, 14, 6, 12, 1, 4] is a composed function:

$$\mathbf{y} = g \circ (s \circ f)(\mathbf{x}_c, \mathbf{x}_s) \tag{1}$$

where $(\mathbf{x}_c, \mathbf{x}_s, \mathbf{y}) \in \mathbb{R}^a \times \mathbb{R}^a \times \mathbb{R}^a$ are the input content image, the input style image and the output stylized image respectively, and we call the function f, s, g as the encoder, the style transfer function and the decoder respectively. The encoder function is a mapping $f : \mathbb{R}^a \times \mathbb{R}^a \to \mathbb{R}^d \times \mathbb{R}^d$ (usually we let $d < a$) that "compresses" a content image and a style image into low-dimensional representations, while the style transfer function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ integrates the compressed representations of the content and style images into a latent representation $\mathbf{z}$. One of the most well-known style transfer function is AdaIN [7] which will be discussed in Sec. 2.2. Finally, the decoder function $g : \mathbb{R}^d \to \mathbb{R}^a$ maps the latent representation $\mathbf{z}$ into the target stylized image $\mathbf{y}$. The state-of-the-art designs of a NST pipeline usually learn an encoder f and a decoder g with deep neural networks where the design of the style transfer function s usually relies on heuristics.

2.2. Learning Image Style Transfer Networks

Image style transformation aims to transfer the style of a source image $\mathbf{x}_s$ into another source image $\mathbf{x}_c$. To this end, it generates a new image $\mathbf{y}$ with the content from $\mathbf{x}_c$ and the style from $\mathbf{x}_s$. Though the way to differentiate the content and the style information in an image from the other is not rigorously defined, empirical evidence has shown that when feeding an image into a deep model, the internal output of the shallow layers aligns with the content information while the deep layers encode the style information [5]. For example, Gatys et al. [5] are considered as the first individuals to combine internal outputs from different layers to transfer the style of one image to another while keeping the content of the source image aside. Several approaches [8, 17] aim to solve an obvious limitation in Gatys et al. [5] method that each run of the image transfer starts from scratch; therefore, it usually requires thousands of forward passes before convergences happen. Motivated by previous work, Huang et al. [7] propose AdaIN as an end-to-end NST model such that the style transfer can be complete in one run of forward propagation. We refer to the NST model as the architecture used in Huang et al. [7] in the rest of the paper unless noted otherwise. Formally, a NST network is composed of an encoder network f, a decoder network g and a style transfer function s. The encoder network f extracts informative representations from input content (or style) images and are learned via pre-trained classification tasks usually on large-scale data sets, e.g. ImageNet [13]. The weights of such pre-trained networks can be found publicly from Tensorflow or Pytorch with the dense layers removed from the top. The style transformation s is usually designed by heuristics; for example, AdaIN uses the following function:

Definition 1 (AdaIN [7]) *Given a content image $\mathbf{x}_c$, a style image $\mathbf{x}_s$ and an encoder $f(\mathbf{x})$, the style transfer function s for AdaIN is defined as:*

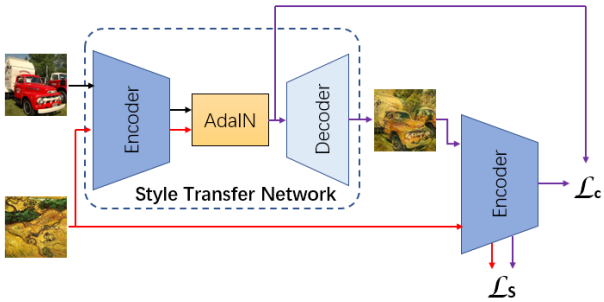

Figure 2. An overview of AdaIN style transfer algorithm.

$$\sigma(f(\mathbf{x_c}))(\frac{f(\mathbf{x_c}) - \mu(f(\mathbf{x_c}))}{\sigma(f(\mathbf{x_s}))}) + \mu(f(\mathbf{x_s})) \quad (2)$$

where $\sigma(\cdot)$ and $\mu(\cdot)$ denote the element-wise standard deviation and the element-wise mean.

Finally, the decoder g is trained to generate a stylized image that combines the content from $\mathbf{x}_c$ and the style from $\mathbf{x}_s$. A visual illustration of the NST pipeline and the training loss for the decoder is shown in Fig. 2.

Definition 2 (Neural Style Transfer) *Given an encoder network f, its internal output f^l at the l-th layer and a style transfer function s, we denote the output of NST as $\mathbf{y} = g \circ (s \circ f)(\mathbf{x}_c, \mathbf{x}_s)$. We learn the parameter θ of the corresponding decoder network g by optimizing the following objective:*

$$\min_{\theta} \mathbb{E}_{((\mathbf{x}_c,\mathbf{x}_s)\sim\mathcal{D})} \left[\mathcal{L}_c(\mathbf{x}_c, \mathbf{x}_s) + \lambda \mathcal{L}_s(\mathbf{x}_c, \mathbf{x}_s)\right] \quad (3)$$

$$\mathcal{L}_c(\mathbf{x}_c, \mathbf{x}_s) = ||f(\mathbf{y}) - (s \circ f)(\mathbf{x}_c, \mathbf{x}_s)||_2 \quad (4)$$

$$\mathcal{L}_s(\mathbf{x}_c, \mathbf{x}_s) = \sum_{l=0}^{L} ||\mu(f^l(\mathbf{y})) - \mu(f^l(\mathbf{x}_s))||_2 + \sum_{l=0}^{L} ||\sigma(f^l(\mathbf{y})) - \sigma(f^l(\mathbf{x}_s))||_2 \quad (5)$$

where $\mathcal{L}_c$ and $\mathcal{L}_s$ denote the content loss and the style loss, respectively; $\sigma(\cdot)$ and $\mu(\cdot)$ denote the element-wise standard deviation and the element-wise mean; and λ is a hyper-parameter[1].

3. Latent Space Manipulation

When a stylized image $\mathbf{y}$ is generated by an NST model, the existing methods cannot manipulate the level of stylization in $\mathbf{y}$. Namely, when the parameters of the decoder network are learned, the output of the decoder is fixed, so that we cannot generate a similar output $\mathbf{y}'$ which is less or more stylized compared to $\mathbf{y}$. Manipulating the level of stylization helps to apply NST to real-world applications where human users can provide feedback to the model to increase or decrease the stylization in $\mathbf{y}$ so that the final output satisfies different user purposes without retraining the decoder.

In this section, we firstly introduce our motivation of the proposed method in Sec. 3.1 followed by how it works empirically in Sec. 3.2.

3.1. Motivation

A related line of work in manipulating the output behavior of a generative model like NST is InterfaceGAN [15], where the authors show that the latent space of the generative model can be partitioned into half-spaces and each half-space is correlated to a particular attribute of the generated image, e.g. brightness, orientation, masked or not, etc. By projecting an arbitrary input vector sampled in the latent space onto the linear constraint that defines the half-spaces above, InterfaceGAN manipulates a particular attribute of the generated image while maintaining other attributes unchanged, e.g. a man with blond hair into a woman with blond hair. Motivated by InterfaceGAN, we are looking for a similar way to explore and separate the latent space of the encoded image in an NST model to find the steepest direction to increase or decrease the stylization of the generated image $\mathbf{y}$. This method, which we introduce in the rest of this section, will provide the most effective way to manipulate the output stylization.

3.2. Method

When a content image $\mathbf{x}_c$ and a style image $\mathbf{x}_s$ are fed into the encoder and the style transfer function, a latent vector $\mathbf{z} = s \circ f(\mathbf{x}_c, \mathbf{x}_s)$ is returned. Motivated by the latent space manipulation in InterfaceGAN [15], we aim to find a boundary H in the latent space such that whenever $H(\mathbf{z}) > 0$ for an arbitrary vector $\mathbf{z}'$ in the latent space, the generated image by decoding $g(\mathbf{z}')$ has a higher level of stylization compared to $g(\mathbf{z})$ and otherwise if $H(\mathbf{z}') < 0$. As Shen et al. [15] use Support Vector Machine (SVM) [19] to learn a boundary H, we follow the idea and formally introduce *Linear Modifications for Latent Representations* (LMLR) for an NST model.

Definition 3 (LMLR) *Given the target class of style t, a SVM classifier $H_t(\mathbf{z}) = sign(\mathbf{w}^\top \mathbf{z} + b)$ is learned by minimizing the following objective:*

$$\mathbb{E}_{((\mathbf{x}_s,\mathbf{x}_c)\sim\mathcal{D})} \max(0, 1 - \mathbb{I}[S = t] * [\mathbf{w}^\top \mathbf{z} + b]) \quad (6)$$

where S is the label for $\mathbf{x}_s$ and $\mathbf{z} = s \circ f(\mathbf{x}_c, \mathbf{x}_s)$.

For each style t we train a corresponding LMLR H_t to partition the latent space into half-spaces by maximizing

[1] Huang et al. [7] use only a few selected layers instead of the output of all layers in $\mathcal{L}_s$.

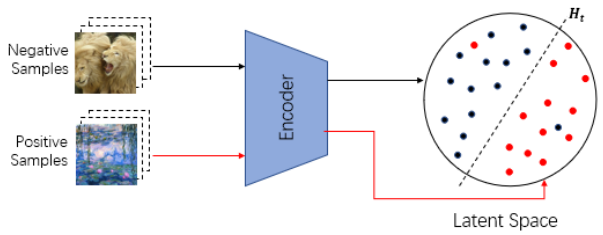

Figure 3. The process of style classification in the latent space. In the first step, we take paintings from a particular style t as positive samples and content images of real world as negative samples. In the second step, we feed them into an encoder to getting their latent representations. Finally, we train an optimal boundary H_t to classify these latent representations.

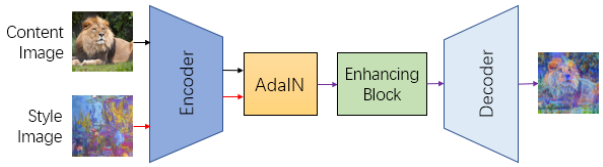

Figure 4. The overall framework of our style transfer model. According to the assumption which is introduced in Sec. 3.2, we add an Enhancing Block to project outputs of the AdaIN layer onto the decision boundary. This manipulation process is defined in Def. 4.

the distance between the latent vectors and the boundary, as shown in Fig. 3. We show that LMLR can achieve up to 100% accuracy in Sec. 4.2.

Manipulating Stylization. By learning a linear boundary with SVM, we now discuss how to manipulate the level of stylization of the generated image $g(\mathbf{z})$. Given a latent vector $\mathbf{z}$, in order to increase the level of stylization for a style t, we move $\mathbf{z}$ away from the linear boundary which is defined by LMLR H_t if $H_t(\mathbf{z}) > 0$; otherwise we move it closer and cross H_t. We refer the process of manipulating the output stylization as an *Enhancing Block* and we define it formally as follows:

Definition 4 (Enhancing Block) *Given a latent vector* $\mathbf{z}$*, and a LMLR* $H_t(\mathbf{z}) = sign(\mathbf{w}_t^\top \mathbf{z} + b_t)$ *for the target style* t*, a more stylized output can be generated by a latent vector* $\mathbf{z}'$ *such that:*

$$\mathbf{z}' = \mathbf{z} + \lambda * d \frac{\mathbf{w}_t}{||\mathbf{w}_t||_2} \tag{7}$$

where $d = |\mathbf{w}_t^\top \mathbf{z} + b_t| / ||\mathbf{w}_t||_2$ *and* λ *is a hyper-parameter to control the distance between* $\mathbf{z}'$ *and its original latent representation* $\mathbf{z}$*.*

In our method, we manipulate the point $\mathbf{z}$ where $H_t(\mathbf{z}) < 0$. By setting $\lambda = 1$, we simply get a point $\mathbf{z}'$ exactly on the boundary defined by H_t by projection. Similarly, to increase or decrease the level of output stylization, we just need to set λ to a different value. In this way, we complete the manipulation of the images in the style latent space, and the style transfer effect of the synthesized image has been enhanced. Furthermore, our method can also be applied to any other models with an encoder and decoder by manipulating the output latent representation with an Enhancing Block shown in Fig. 4.

4. Evaluation

In this section, we provide empirical evaluations of our method and compare it with several baseline approaches. Firstly, we discuss the setup of our experiments in Sec. 4.1. Secondly we show that the latent space can be efficiently separated by one linear boundary in Sec. 4.2. Finally, in Sec. 4.3, we compare our method with three other types of style transfer approaches and one type of the other adjustable method.

4.1. Experiment Setup

Architecture. As described in Sec. 3, our architecture contains two processes. The first process is to train a linear decision boundary for each style and the second is to use the enhancing block ((Def. 4)) to manipulate the style and content preservation in output images. In generating the original latent representation $\mathbf{z}$, we choose the pre-trained AdaIN as our base model. And we add the enhancing block with hyper-parameter $\lambda = 1$ (the justification of the choice of λ to follow) after AdaIN's transfer function s to produce the target latent representation $\mathbf{z}'$:

$$\mathbf{z}' = EnhancingBlock \circ s \circ f(\mathbf{x}_c, \mathbf{x}_s) \tag{8}$$

Where, the encoder f is fixed to the first few layers (up to $relu4\_1$) of a VGG-19 network [16]. The EnhancingBlock is described in Def. 4, and s is defined in Def. 1. The decoder shown in Fig. 4 mirrors the encoder to transfer the representation $\mathbf{z}'$ into the stylized output, with all pooling layers replaced by the nearest up-sampling to reduce checkerboard effects. Both the encoder and the decoder use reflection padding to avoid border artifacts.

Training. Training linear decision boundaries, we use 3000 paintings of 11 artists from WikiArt [10] as positive samples, and the same quantity of photos from ImageNet [13] as negative samples. For all images, we randomly crop regions to be the size 256 × 256, and adopt a pre-trained VGG-19 network [16] to encode them to latent representations. According to Def. 3 and 4, we train a linear decision boundary for every style and manipulate latent representations along the direction of the normal vector.

Testing. In evaluation, we generate 200 stylized images, by the cartesian product of 20 paintings from WikiArt [10] and

Artist	Accuracy for validation set	Accuracy for whole set
berthe-morisot	0.987	0.991
claude-monet	1.000	1.000
el-greco	0.992	0.998
ernst-ludwig-kirchner	0.971	0.988
jackson-pollock	0.971	0.988
nicholas-roerich	0.865	0.946
pablo-picasso	0.965	0.982
paul-gauguin	1.000	1.000
samuel-peploe	1.000	1.000
vincent-van-gogh	0.988	0.991
wassily-kandinsky	0.983	0.989

Table 1. Classification accuracies on linear boundaries in latent space.

Artist	Original
berthe-morisot	0.7
claude-monet	0.95
el-greco	0.85
ernst-ludwig-kirchner	0.95
jackson-pollock	0.9
nicholas-roerich	1
pablo-picasso	0.9
paul-gauguin	1
samuel-peploe	0.8
vincent-van-gogh	0.95
wassily-kandinsky	1

Table 2. Deception Rates of the original input style images. A Higher score means better stylization effect.

10 photos from ImageNet [13], for every artist. It is to be noted that the training data sets and testing data sets have no intersections and all the tested methods use the same testing data sets. Also, all the paintings and photos are chosen randomly.

4.2. Performance of Latent SVM

For each artist, we use 70% positive and 70% negative samples to train a linear SVM hyperplane and evaluate the performances on the remaining data. Table 1 demonstrates that nearly all the linear boundaries achieve over 95% accuracy on the validation set, which suggests that it is easy to find a linear hyperplane in the latent space that can well separate the data into two groups. We further provide the timing of training the latent-space SVM in Table 4.

4.3. Quantitative Evaluations

In this subsection, we firstly conduct quantitative experiments to examine the levels of style and content preservation in our method and others'. Secondly, we manipulate the distances between output images and decision boundaries to see how style and content preserved in output images are varied. In the end, we demonstrate our method is time-efficient.

4.3.1 Comparisons with Non-manipulable Approaches

We first compare our methods with the following baselines, which are all non-manipulable approaches:

- a flexible but slow optimization-based method (Gatys) [5]
- the fast feed-forward method based on arbitrary instance normalization (AdaIN) [7]
- the universal-style transfer with whitening and coloring transformation (WCT) [11]

We now discuss our numerical metrics used in the previous literature to quantify the level of stylization and content retention.

Deception Rate [14]. we use this metric to evaluate how well the input style is preserved in output images. We train a classifier on WikiArt [10] to categorize styles and use its output score as the probability of the stylized output being classified into the target style. Table 2 demonstrates the average *deception rate* of the original input style images. Also, we compare the *deception rates* of our method with those of AdaIN [7], WCT [11] and Gatys [5] approaches in Fig. 6. The results show that our approach outperforms AdaIN and WCT, even though Gatys has the highest score, the computation speed of Gatys is three orders of magnitude slower than ours.

Content retention [14]. In this evaluation, we train another classifier on ImageNet [13] to categorize contents and use its Top 5 Accuracy to indicate the retention of the content. The average Top 5 Accuracy of the original input content images is 0.3. In Fig. 8, we can see that our approach is better than AdaIN in Top 5 Accuracy, more significantly, the average accuracy of our approach is nearly twice larger than AdaIN's. However, since the average accuracy of the original input content images is not very high, we also use cosine distance to evaluate the similarities between the stylized images and their original input content images. Fig. 9 demonstrates the results of cosine similarity, which also shows that our method works better than AdaIN. Therefore, both metrics signify that our approach can retain the contention as well as AdaIN, meanwhile providing more convincing stylization results.

Distance Effect of Latent Space. As described in Sec. 3.2, the Enhancing Block proposed in our paper can move the latent representation to the classification hyperplane along the

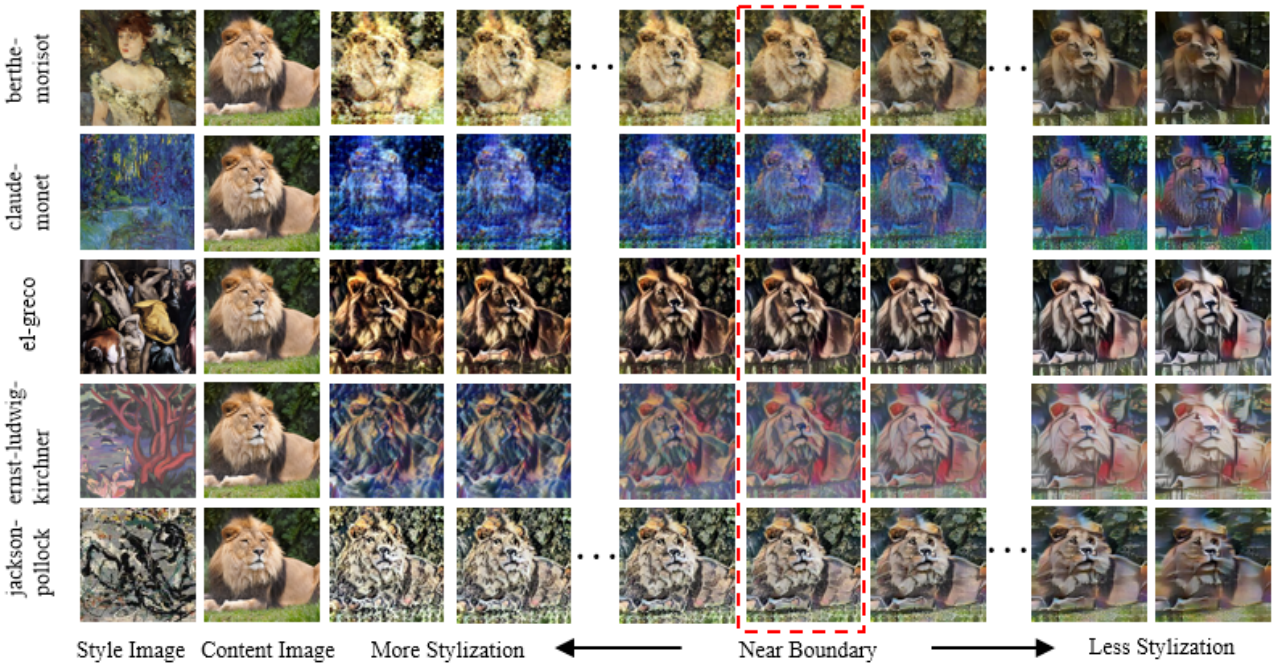

Figure 5. Illustration of the distance effect by latent representation manipulation. The images in the red dashed box stand for synthesized images on the boundary. When the distance keeps increasing in the positive direction, the style of images is more pronounced.

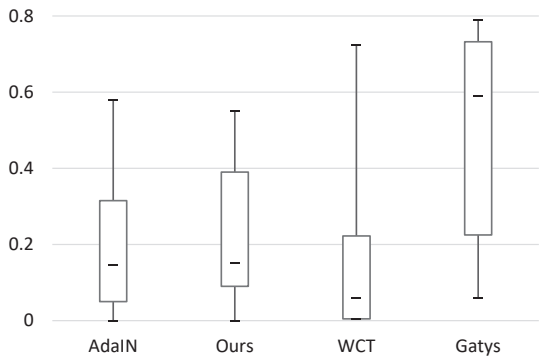

Figure 6. Comparison with Deception Rates of AdaIN, Ours, Gatys and WCT shown in boxplot.

direction of the normal vector. When manipulating the latent representation, we observe, through the distance effect, that moving the latent representation can produce continuous changes in stylization. Fig. 5 clearly shows the output images near the boundary are well balanced for stylization and content retention, while the images in the positive direction far from the boundary have more stylization and the images in the negative direction have less stylization. We also make a qualitative and quantitative analysis of distance effect with another adjustable method from Fig. 10 to 12.

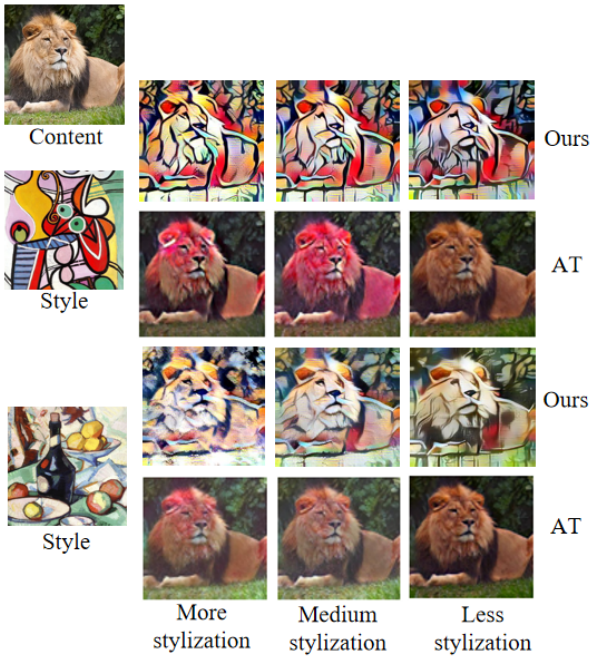

Figure 7. Examples of our method and AT, from left to right, λ of our model decreases from 5 to -5, α_s of AT decreases from 10^{10} to 0.

4.3.2 Comparing with another adjustable approach

A recent work by Babaeizadeh & Ghiasi [2] introduces a real-time Adjustable Transfer (AT) for the NST pipeline. In

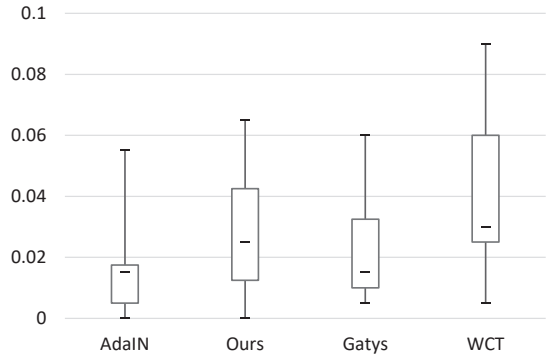

Figure 8. Comparison with Top 5 Accuracies of AdaIN, Ours, Gatys and WCT shown in boxplot.

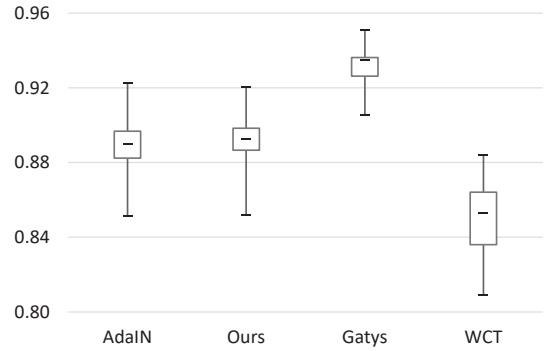

Figure 9. Comparison with Cosine Similarities of AdaIN, Ours, Gatys and WCT shown in boxplot.

this section, we compare our approach with AT on the same metrics mentioned above and discuss our conceptual difference with AT in Sec. 5. Since AT is not an arbitrary transfer, which runs an optimization for every input style image. We use an implementation from github [2] with the same training data sets as ours. For each style, we train it for 3300 iterations. The default style loss weight α_s and content loss weight α_c are set 10^{10} and 10^5, respectively. Unlike our method manipulating the style from both negative and positive directions, AT usually does not have a practical use case for setting its parameters to the negative. Thus, in manipulating the outputs of AT, we modify the style loss weight by multiplying it with a positive ascending sequence that begins with 0.2 and ends with 1.0 with an interval of 0.2, while keeping the content loss weight unchanged. Fig. 7 shows the manipulating results from our method and AT in quality. Our method achieves better results than AT, even though the outputs of AT vary along with the modification of style loss weight α_s, these changes are not very related to the style, it seems like AT only edits luminosity and hues and ignores other key elements like pattern and brushstroke

[2] https://github.com/gnhdnb/adjustable-real-time-style-transfer

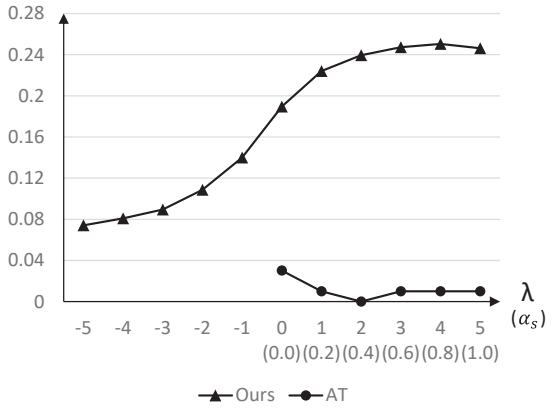

Figure 10. Deception Rates of our method and AT with different distances and style loss weights, respectively.

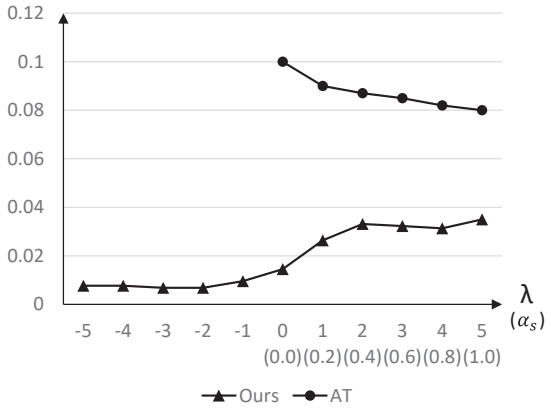

Figure 11. Top 5 Accuracies of our method and AT with different distances and style loss weights, respectively.

in the style inputs. In quantitative analysis, we draw a comparison of distance effect in Fig. 10, 11 and 12. The results represent that *deception rates* of AT are almost zero, but the cosine similarities and Top 5 accuracies of it are within expectation. Although, in Fig. 7, we can tell that AT does not achieve good performance in stylization, the results in paper [2] are quite well in quality even the authors demonstrated no quantitative experiments. Therefore, with more time, we will contact the authors and discuss this issue to see if there is a problem.

Speed Analysis. Evaluating the effect of EnhancingBlock on time consumption, we use AdaIN and our method to generate 1000 256 × 256 images each and calculate the average time per image to Encode, Transfer (AdaIN /EnhancingBlock) and Decode. Table 3 shows the results, from which we can see the vast majority of the time is spent by Encode and Decode, while stage Transfer only accounts for 0.62% of the total time. Compared with AdaIN, even though our method takes 30% more time in Transfer, which is mainly used to calculate distances in EnhancingBlock. However,

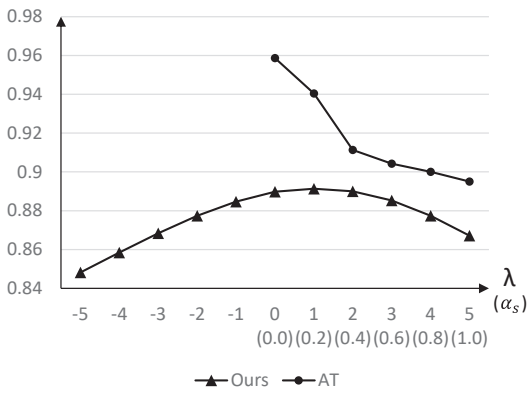

Figure 12. Cosine Similarities of our method and AT method with different distances and style loss weights, respectively.

Stage	AdaIN time-consuming (ms)	Ours time-consuming (ms)
Encode	18.962	18.898
Transfer	0.085	0.111
Decode	12.283	12.289

Table 3. Time-consuming per image of AdaIN and Ours Method at each stage

Stage	AT time-consuming (ms)	Ours time-consuming (ms)
Training	351.526	92.481
Generating	634.717	31.298

Table 4. Time-consuming of AT and Ours Method at training (per iteration) and generating (per image), the training process excludes the time of saving models, and generating processes does not calculate the time of loading models and saving output images onto the disk, respectively.

this increase of time is negligible in terms of the total time consumption. Additionally, we make a comparison of the time spent on training per iteration and generating per image between our method and AT in Table 4. Normally, our method only needs to train 500 iterations to get a qualified result, but AT needs to train 3000 iterations.

5. Related Work

In this section, we compare several recent works in manipulating the stylization of an NST pipeline with our work.

Ulyanov et al. [18] demonstrates a preliminary work on generating multiple stylization for NST task, which only aims to produce more stylized output without enforcing any notation of more stylization or less stylization as used in our paper. Babaeizadeh & Ghiasi [2] introduce a real-time manipulation of the stylization (AT) by learning another network that predicts the scalars in the weighted sum of the target loss used by style transfer (see Eq. 3), which controls the output stylization by finding a better way to aggregate the contribution of each layer in the encoder network. Besides quantitative and qualitative comparisons as shown in Sec. 4.3, another major difference between our method with Babaeizadeh & Ghiasi [2] is our method is resource-efficient given that a linear model (SVM in our case) is sufficient to manipulate the stylization while Babaeizadeh & Ghiasi [2] requires the training of a multi-layer fully-connected network. Leveraging SVM also guarantees the explanation ability of our approach as we show that the stylization is enhanced if the internal representation is far away from the decision boundary compared to any approaches that require additional deep models.

Another line of work either focuses on applying GAN to perform adjustable style transfer [20] or searches for an obvious way to adjust a style-transfer GAN [9]. These methods usually require a new network while our method can be viewed as a plug-in to many existing NST pipelines. Therefore, our method makes a minimal change compared to GAN-based methods. Besides, Babaeizadeh & Ghiasi [2] has shown that the style manipulation in [9] is not highly perceptional to humans.

6. Limitation

An obvious limitation of our work is that a collection of similar style images is required to train a linear SVM in approximating the boundary in the latent space of a NST encoder. As the development of image search, i.e. Google Image Search [3], collecting images with similar styles should not be a huge burden to a service provider who aims to leverage the adjustable stylization as a product for other users.

7. Conclusion

This paper demonstrates that different regions in the latent space of an NST model are related to the different stylization of the generated output image. By learning an SVM in the latent space, we can determine the corresponding boundaries of these regions. By projecting a latent vector into a particular region, we show that the output image can be manipulated in the desired way. Our method is efficient and universal to any NST pipeline that produces a latent space. Given the better performance and simplicity of our method, we believe there are substantial spaces for more improvements in the follow-up work.

References

[1] Jie An, Haoyi Xiong, Jiebo Luo, Jun Huan, and Jinwen Ma. Fast universal style transfer for artistic and photorealistic rendering. *ArXiv*, 2019. 2

[3] https://www.google.com/imghp?hl=en

[2] Mohammad Babaeizadeh and Golnaz Ghiasi. Adjustable real-time style transfer. In *International Conference on Learning Representations*, 2020. 6, 7, 8

[3] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv preprint arXiv:1606.03657*, 2016. 1

[4] Yingying Deng, Fan Tang, Weiming Dong, Wen-Cheng Sun, Feiyue Huang, and C. Xu. Arbitrary style transfer via multi-adaptation network. *Proceedings of the 28th ACM International Conference on Multimedia*, 2020. 2

[5] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 1, 2, 5

[6] Shuyang Gu, Congliang Chen, Jing Liao, and L. Yuan. Arbitrary style transfer with deep feature reshuffle. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. 2

[7] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017. 2, 3, 5

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016. 2

[9] Tero Karras, S. Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405, 2019. 8

[10] K.Nichol. Painter by numbers, wikiart, 2016. 2, 4, 5

[11] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal style transfer via feature transforms. *arXiv preprint arXiv:1705.08086*, 2017. 1, 5

[12] G. Puy and P. Pérez. A flexible convolutional solver for fast style transfers. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 2, 4, 5

[14] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Bjorn Ommer. A style-aware content loss for real-time hd style transfer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 698–714, 2018. 2, 5

[15] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9243–9252, 2020. 1, 3

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 4

[17] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. In *ICML*, 2016. 2

[18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017. 8

[19] Vladimir Vapnik, Isabel Guyon, and Trevor Hastie. Support vector machines. *Mach. Learn*, 20(3):273–297, 1995. 1, 3

[20] Shuai Yang, Zhangyang Wang, Z. Wang, N. Xu, Jiaying Liu, and Zongming Guo. Controllable artistic text style transfer via shape-matching gan. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4441–4450, 2019. 8