Department of Computer Science

BSCCS Final Year Project Report
2020-2021

20CS101

Style-Attention Void-Aware Style Transfer

(Volume 1 of 1)

Student Name	:	**ZHANG, Deheng**
Student No.	:	**55199998**
Programme Code	:	**BSCEGU4**

For Official Use Only

Supervisor	:	**Dr LIAO, Jing**
1st Reader	:	**Ms MONG, Yu**
2nd Reader	:	**Dr LAU, Rynson W H**

Student Final Year Project Declaration

I have read the project guidelines and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I hereby declare that the work I submitted for my final year project, entitled:

Style-Attention Void-Aware Style Transfer

does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Project Guidelines.

Student Name:	ZHANG, Deheng	Signature:	ZHANG Deheng.
Student ID:	55199998	Date:	April 13, 2021

Abstract

Arbitrary-Style-Per-Model fast neural style transfer has shown great potential in the academic field. Although state-of-the-art algorithms have great visual effect and efficiency, they are unable to address the blank-leaving (or void) information in specific artworks (e.g. traditional Chinese artworks). The available algorithms always try to maintain the similarity of details in the images before and after transformation, but certain details are often left blank in the artworks.

Therefore, in this project, we solve these problems by developing a style-attention void-aware style transfer model. We first propose a novel attention mechanism based on self-attention with satisfactory statistical meaning and property, from which the attention map of content and style image can be derived. A hard constraint based on the attention map and the SA-Net [1] module is incorporated to build the Style-Attention Void-Aware network (SAVA-net), which is able to learn the style and voidness information in the style image at the same time. The model is trained on MS-COCO [2] and WikiArt [3] dataset in two phases and compared with state-of-the-art methods. Our experiment demonstrates our attention mask and style transfer algorithm can outperform state-of-the-art algorithm.

Acknowledgement

I would like to express my sincere appreciation to individuals who give generous support to complete my final year project. The deepest thanks to my final year project supervisor, Dr. Jing LIAO, for her helpful information, enlightening guidance. She has provided many insightful suggestions, such as the use of style attention, soft correlation mask, and attention loss to match the voidness statistics. With her support, I implemented a novel algorithm from scratch and understood more knowledge about the field of style transfer. I would like to express special gratitude to my previous supervisor, Dr. Kede MA, who has generously guided me on the previous topic about video quality assessment. When I encountered problems, he encouraged me and allowed me to switch my final year project topic to style transfer.

I would like to express my sincere thanks to my friend Kaiwen Xue, who has provided many intelligent ideas on this project. His suggestion to use K-means to generate a hard attention mask solves the overflow problem efficiently. He has also helped me with the implementation of the deep learning functions. I also want to thank Dr. Nanxuan Zhao, who has provided recommendations on baseline selection, model training, experimentation, and GPU memory problems. Special thanks to artists Mr. Xiao Zhang and Mrs. Deling Yang, who provides suggestions about the artwork voidness and part of the test examples.

I would like to thank the staffs in the Department of Computer Science and FYP Lab, who provides the computational resource for me to conduct experiments. I also want to express my appreciation to the lecturers in the City University of Hong Kong, who provide me with computer science knowledge and programming skill to finish the project.

Moreover, I am grateful that my friends, Chengyu Sun, Yang Lou, Leqian Zheng, Ruikang Li, Wenbo Yang, Yifan Xiong, and Zelin Ning have provided many valuable suggestions and encouragement in my daily life and my project.

Last but not least, I want to express my appreciation to my families, who have always supported me in my life. They provide many recommendations when I face difficulties in my daily life and the project.

Table of Contents

1 Introduction

1.1 Background

With the fast development of mobile phones and the Internet, most people can access images readily. Images can be produced by either professional industry or people who capture photos to memorize and share meaningful experiences. With the development of computing resource and data explosion, deep learning can be applied to computer vision tasks such as image classification, 3D model reconstruction, computational photography, object detection, etc. As an effective image editing technique, style transfer enables the creation of new artworks based on the content image in the real world and the style image produced by artists. Without style transfer, image re-drawing in a specific style requires a talented artist and time. However, the development of computer science enables computers to learn the style of each artist and apply it to new images. Therefore, this is the research direction that can broaden the potential of AI in the direction of computer vision.

1.2 Problem Definition

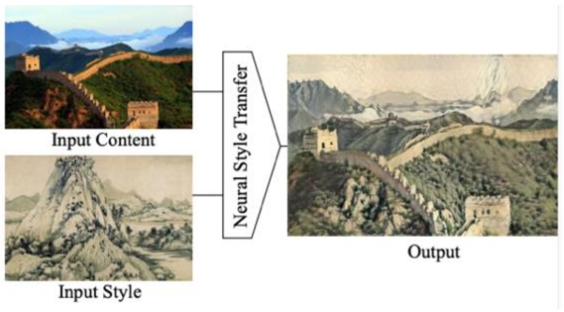

Figure 1: Example of Style Transfer from [1]

According to [4], style transfer algorithms take a content image and a style image as the input, and output an image keeps the content information of the content image and the style information of the style image. Figure 1 shows an example to transfer the style of a Chinese painting onto a given photograph of the Great Wall. We can observe that the structure of the great wall is preserved, while the texture of the style image is learned by the output image.

1.3 Project Goal

Although the state-of-the-art algorithms can learn the texture from the style image, they ignore the blank-leaving information in the style image. Blank-leaving is a common technique used in artworks. For example, the mountain near to the artist is depicted in detail, but the mountain far from the artis is represented by blank texture in Figure 1. To tackle the problem, this project aims to propose a robust Arbitrary-Style-Per-Model Fast Neural Method based on VGG network and self-attention. We will construct a style transfer scheme that utilizes both content attention and style attention to make sure the output image include the "blank-leaving" information of the artwork. Meanwhile, the computation speed is fast enough to generate the output image compared to online image optimization method.

1.4 Organization

The paper is organized as follows. Section 2 includes a review of neural style transfer methods and related works. Section 3 presents the methodology for our style transfer model. Section 4 talks about the design and implementation of the project. Section 5 gives an introduction of experiment setting and result analysis. The last section concludes the project and talks about some possible methods to further improve the performance of the algorithm.

2 Literature Review

This section would first introduce different categories of style transfer and the drawbacks of the previous algorithms. Secondly, the style transfer algorithms related to this project will be introduced.

2.1 Neural Style Transfer

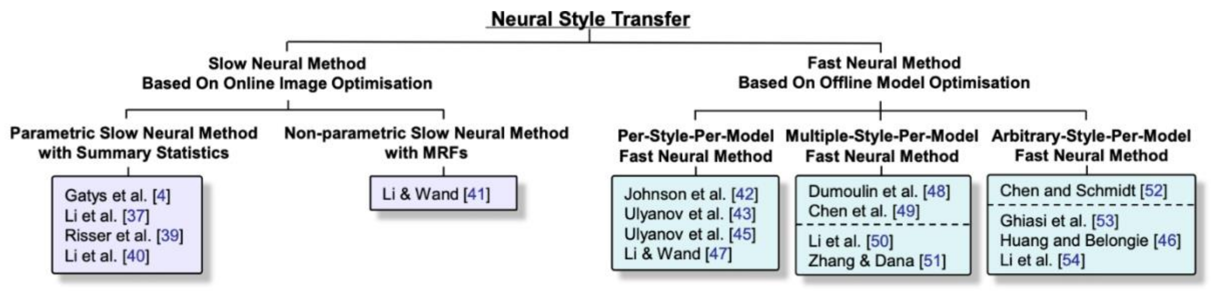
Neural Style Transfer
Slow Neural Method Based On Online Image Optimisation
Fast Neural Method Based On Offline Model Optimisation
Parametric Slow Neural Method with Summary Statistics
Non-parametric Slow Neural Method with MRFs
Per-Style-Per-Model Fast Neural Method
Multiple-Style-Per-Model Fast Neural Method
Arbitrary-Style-Per-Model Fast Neural Method
Gatys et al. [4]
Li et al. [37]
Risser et al. [39]
Li et al. [40]
Li & Wand [41]
Johnson et al. [42]
Ulyanov et al. [43]
Ulyanov et al. [45]
Li & Wand [47]
Dumoulin et al. [48]
Chen et al. [49]
Li et al. [50]
Zhang & Dana [51]
Chen and Schmidt [52]
Ghiasi et al. [53]
Huang and Belongie [46]
Li et al. [54]

Figure 2: Different Types of Neural Style Transfer from [4]

Style transfer can be achieved by many different methods, including stroke-based rendering, image analogy, neural network, image filtering, and texture synthesis. In this project, we focus on the style transfer based on neural network (i.e. neural style transfer). As shown in Figure 2, neural style transfer can be divided into *Slow Neural Method Based On Online Image Optimization* and *Fast Neural Method Based On Offline Model Optimization*. Slow neural methods [5, 6, 7, 8] will first extract style and content information from the corresponding images and iteratively optimize the image to match the target representation. As a representative method, Gatys et al. [5] utilize the Convolutional Neural Network to achieve style transfer for the first time.

Compared with the slow neural method, the fast neural method can feedforward the input images and generate the output image immediately with a model trained by the dataset. The fast neural method can be further divided into *Per-Style-Per-Model Fast Neural Method* [9, 10, 11, 12], *Multiple-Style-Per-Model Fast Neural Method* [13, 14, 15], and *Arbitrary-Style-Per-Model Fast Neural Method* [16, 17, 18, 1, 19, 20] based on the number of styles from the style images that the model can learn. This project will focus on the Arbitrary-Style-Per-Model fast neural method (ASPM) since it can learn all kinds of styles in a given style dataset.

2.2 Arbitrary-Style-Per-Model Fast Neural Method

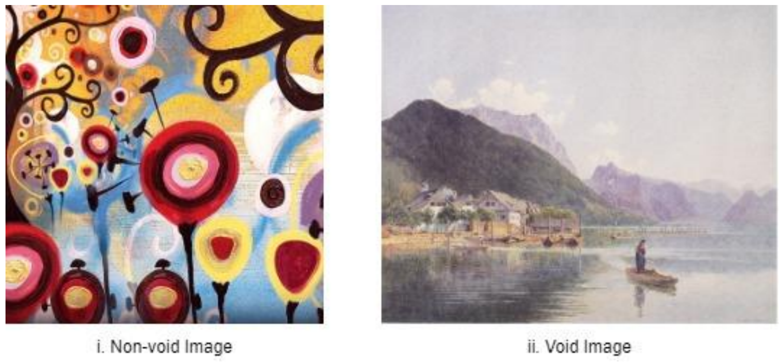

Figure 3: Comparison of non-void and void images

All of the arbitrary-style-per-model method (ASPM) use the VGG network [21] to extract feature from the content and style image. Still, the principle to transfer the content image according to the specific style vary for different algorithms. Based on the particular style transfer module, ASPM fast neural method can be divided into *Parametric ASPM with Summary Statistics*, *Non-parametric ASPM* and *Combined ASPM*. *ASPM with Summary Statistics* transfers the style of the style image by switching the statistic in the content and style image. For example, *Adaptive Instance Normalization (AdaIN)* [16] and *Whitening & Coloring Transform (WCT)* [17] use mean-variance feature statistics and ZCA projection matrix statistics respectively to achieve style transfer. *Non-parametric ASPM* [18, 1] matches content image patches and style image patches based on similarity and swap the content image patches by the corresponding style patches. Combined methods such as [19, 20] utilize both statistics swapping and similar patch switching to achieve style swap.

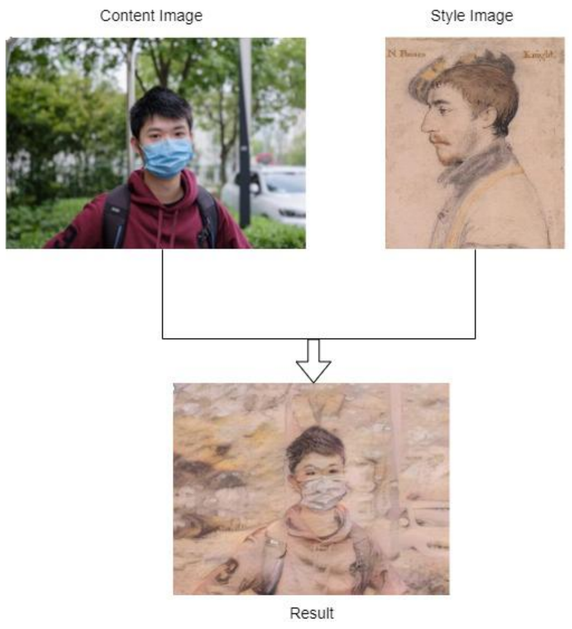

Figure 4: A Failure Case of State-of-the-Art Algorithm

However, those methods are not robust enough to learn the blank-leaving (or void) information. For example, in Figure 3, the left artwork is depicted in detail, while the right artwork left the far view blank. In this example, the voidness of the right artwork is higher than the left one. In state-of-the-art algorithms, the model cannot learn this information appropriately and apply it to the content image. For instance, in Figure 4, a portrait is transferred by a blank-leaving style. The foreground is delineated in detail in the output image correctly, but the background is filled with detailed texture in the output image. The proposed algorithm should learn the blank-leaving style and fill the background with blank texture as our motivation.

2.2.1 AdaIN [16]

AdaIN can swap the statistic of content and style feature map to achieve style transfer since the style information can be represented as mean and variance of the style image. Suppose f_c and f_s are the feature maps of the content and style image in the fourth layer of VGG-19 [21]. The AdaIN module will generate a swapped feature map according to the following equation:

$$AdaIN(f_c, f_s)^{(i)} = \sigma(f_s)\left(\frac{f_c^{(i)} - \mu(f_c)}{\sigma(f_c)}\right) + \mu(f_s),$$

Where μ and σ represent the mean and the variance of the feature map, respectively. Then, the swapped feature map will be fed into the VGG decoder to generate the output image. The loss function includes the content loss and style loss. The content loss is defined as the L_2 distance between the AdaIN result and the encoded feature map of output image, while the style loss is defined as the L_2 distance of mean and standard deviation in each layer of the VGG network.

2.2.2 WCT [17]

Similar to AdaIN, WCT swaps the statistic of the feature maps. But the metric is the result of the ZCA projection. The WCT process includes whitening and coloring stages. In the whitening stage, the algorithm will firstly stretch the content and style feature maps to 2D matrices (each channel is stretched into a row vector) and subtract the mean from the matrices. Then SVD is used to decorrelate the feature map f to $\hat{f}$ along the channel axis:

$$f = U\Sigma V^T, \hat{f} = UV^T.$$

Where the matrices U, Σ and the mean of the feature matrix are statistics used in WCT. In the coloring stage, the swapped feature map is defined as:

$$\widehat{f_{cs}} = U_s \Sigma_s^{\frac{1}{2}} U_s^T \widehat{f_c} + m_s,$$

where the footnote "c" represents content image, "s" represents style image. WCT is applied to each layer of the VGG network.

2.2.3 Style Swap [18]

Different from statistic-based algorithms, style swap selects the most similar patch in the style image for each patch in the content image. The similarity is assessed by the cosine distance between the content and style feature map, the most similar style patch has the largest cosine distance with the content patch:

$$\phi_i^{ss}(f_c, f_s) = argmax_{\phi_j(S), j=1,\ldots,n_s} \frac{\phi_i(f_c) * \phi_j(f_s)}{||\phi_i(f_c)|| * ||\phi_j(f_s)||},$$

where ϕ_i represents the i-th patch of the feature map, ϕ_i^{ss} represents the i-th patch in the swapped feature map, and n_s represents the number of patches in the style feature map. The similarity comparison is simplified by 2D convolution with normalized style patches as filters and the patch swapping is achieved by 2D transposed convolution with style patches as filters.

2.2.4 SANet [1]

Although style swap can generate a good result, it selects only one style patch for each content patch. In SANet, each content patch is replaced by a linear combination (weighted average) of all the style patches. The weight is determined by the similarity between each style patch and the content patch, which is adaptively learned by the covariance map in the self-attention module proposed in [22]. Feature maps from the fourth and fifth layers of the VGG network are used to calculate the swapped feature maps in different scale, which are fused before decoding. SANet is an essential baseline of this project.

2.2.5 Avatar-Net [19]

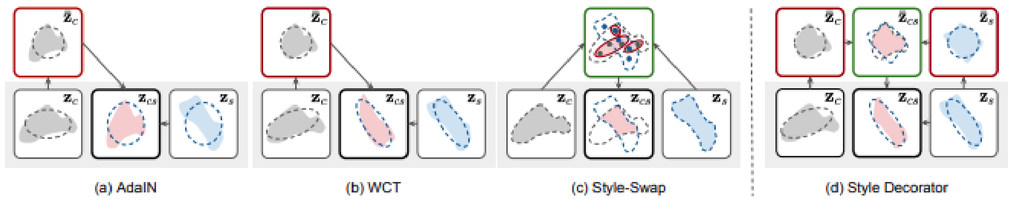

Figure 5: Comparison of distribution transformation by different feature transfer modules from [19]

As shown in Figure 5, Avatar-Net is a combination of style swap and WCT. The feature maps are normalized in the whitening step at each layer of VGG. Then, style swapping is applied in the fourth layer of VGG. After swapping, the swapped feature is colorized during the decoding phase.

2.2.6 AAMS [20]

AAMS is a combination of Avatar-net and self-attention. During the training stage, the decoder and the self-attention module have trainable parameters. Only the content dataset is used to train the reconstruction network. Content feature map in the fourth layer of the VGG network f_c will be fed into the self-attention module to generate an attention map A_C, which assigns a higher attention value to the non-trivial (zero) parts of the image. The reconstructed feature map is defined as:

$$f_{rcon} = A_c \odot f_c + f_c \, ,$$

where $\odot$ represents element-wise multiplication. The style feature map will be sampled into different resolutions in the testing stage to create sets of style patches in different scales. Then, the content image will be transferred in different scales. According to the learned attention, the multi-scale feature maps are fused softly using K-means. AAMS uses averaged residual feature map as the attention, which is not appropriate for some special case. Our attention module modified it to improve the performance.

3 Methodology

3.1 Algorithm Overview

The network architecture of SAVA is shown in Figure 6. The style transfer model proposed is consisted of a VGG encoder-decoder network, a modified self-attention module, and a style-swapping module (SAVA-net) in two scales. The proposed model can generate stylized images based on the style and voidness information. The whole system is designed to provide the following functionalities:

(1) Take parameters as input and train a self-attention network based on the MS-COCO dataset [2]. The baseline of this part refers to [20].

(2) Load pre-trained self-attention module into the Style-Attention Void-Aware Network (SAVA-net). Train the SAVA-net based on the MS-COCO dataset [2] and the WikiArt dataset [3]. The baseline of this part refers to [1].

(3) Be able to transfer the content image according to the style image with the awareness of voidness in the style image.

3.2 Network Architecture

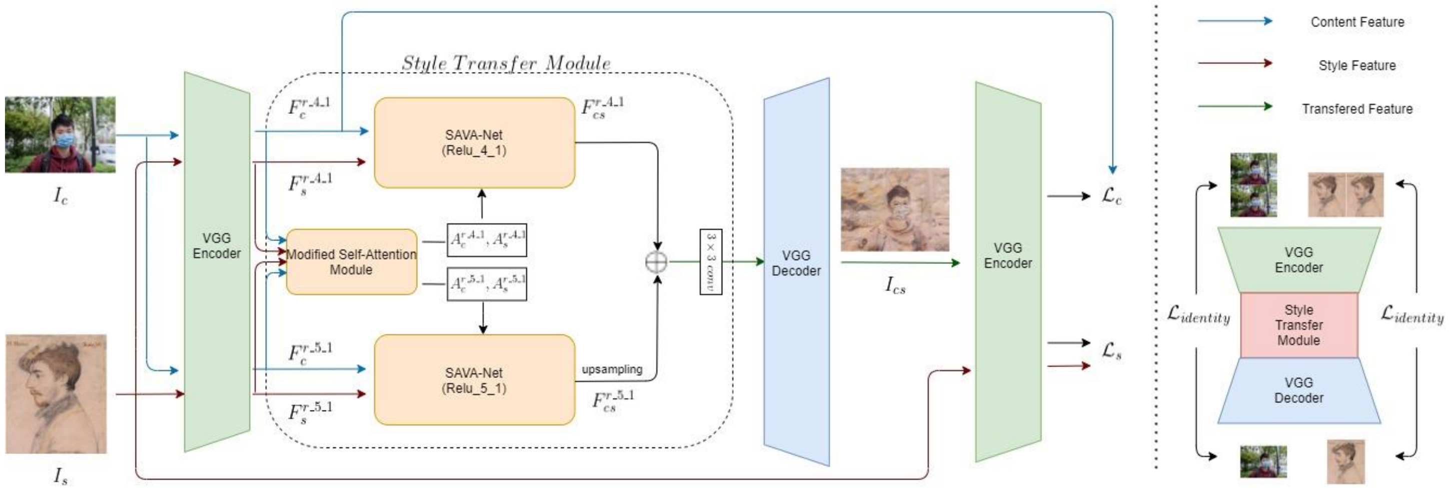

Figure 6: An overview of our proposed network

Our style transfer model takes the content image I_c and the style image I_s as the input and outputs a transferred image I_{cs}, which preserves the structural information in I_c and style-voidness information in I_s. The function can be represented as:

$$Transfer: \mathbb{R}^{h_c \times w_c \times 3} \times \mathbb{R}^{h_s \times w_s \times 3} \mapsto \mathbb{R}^{h_c \times w_c \times 3}$$

$$I_{cs} = Transfer(I_c, I_s).$$

There is a pair of pre-trained VGG-19 encoder and decoder. The self-attention reconstruction network trains the modified self-attention module, and the SAVA-Net modules in different scales are trained jointly with the VGG decoder. The VGG decoder follows the structure in [1].

The VGG encoder extracts the feature maps from the content and style images. We use feature maps in the fourth and fifth layers for style transfer to learn styles in different scales. The function of the VGG encoder can be represented as:

$$VGG_{Encoder}: \mathbb{R}^{h \times w \times 3} \mapsto \mathbb{R}^{h^{r\_4\_1} \times w^{r\_4\_1} \times c^{r\_4\_1}}, \mathbb{R}^{h^{r\_5\_1} \times w^{r\_5\_1} \times c^{r\_5\_1}}$$

$$(F^{r\_4\_1}, F^{r\_5\_1}) = VGG_{Encoder}(I).$$

Then, the modified attention module generates binary attention maps for each of the feature map. The attention map assigns a Boolean value for each vector along the channel dimension, “0” represents the patch is the blank-leaving region in the image, “1” represents the patch is the detailed region in the image. The function of the modified attention module can be represented as:

$$Self\_Attention: \mathbb{R}^{h \times w \times c} \mapsto B^{h \times w} (B \in \{0,1\})$$

$$A = Self\_Attention(F).$$

After extracting the feature maps and generating the attention maps, we feed them to the two SAVA-Net modules to swap the feature under the guidance of attention map and feature correspondence. The function of SAVA-Net can be represented as:

$$SAVA_{Net}: \mathbb{R}^{h_c \times w_c \times c} \times \mathbb{R}^{h_s \times w_s \times c} \times \mathbb{R}^{h_c \times w_c} \times \mathbb{R}^{h_s \times w_s} \mapsto \mathbb{R}^{h_c \times w_c \times c}$$

$$F_{cs} = SAVA\_Net(F_c, F_s, A_c, A_s).$$

Similar to [1], the combined feature maps in different scales $F_{cs}^{r\_4\_1}, F_{cs}^{r\_5\_1}$ are combined by the following function:

$$Combine: \mathbb{R}^{h^{r\_4\_1} \times w^{r\_4\_1} \times c^{r\_4\_1}} \times \mathbb{R}^{h^{r\_5\_1} \times w^{r\_5\_1} \times c^{r\_5\_1}} \mapsto \mathbb{R}^{h^{r\_4\_1} \times w^{r\_4\_1} \times c^{r\_4\_1}}$$

$$F_{cs} = Combine(F_{cs}^{r\_4\_1}, F_{cs}^{r\_5\_1}) = conv_{3\times3}(F_{cs}^{r\_4\_1} + upsampling(F_{cs}^{r\_5\_1}))$$

Finally, the VGG-19 decoder is used to recover the dimension of the combined feature map to RGB color space:

$$VGG_{decoder}: \mathbb{R}^{h^{r\_4\_1} \times w^{r\_4\_1} \times c^{r\_4\_1}} \mapsto \mathbb{R}^{h_c \times w_c \times 3}$$

$$I_{cs} = VGG_{decoder}(F_{cs}).$$

3.3 The Restructuring Module

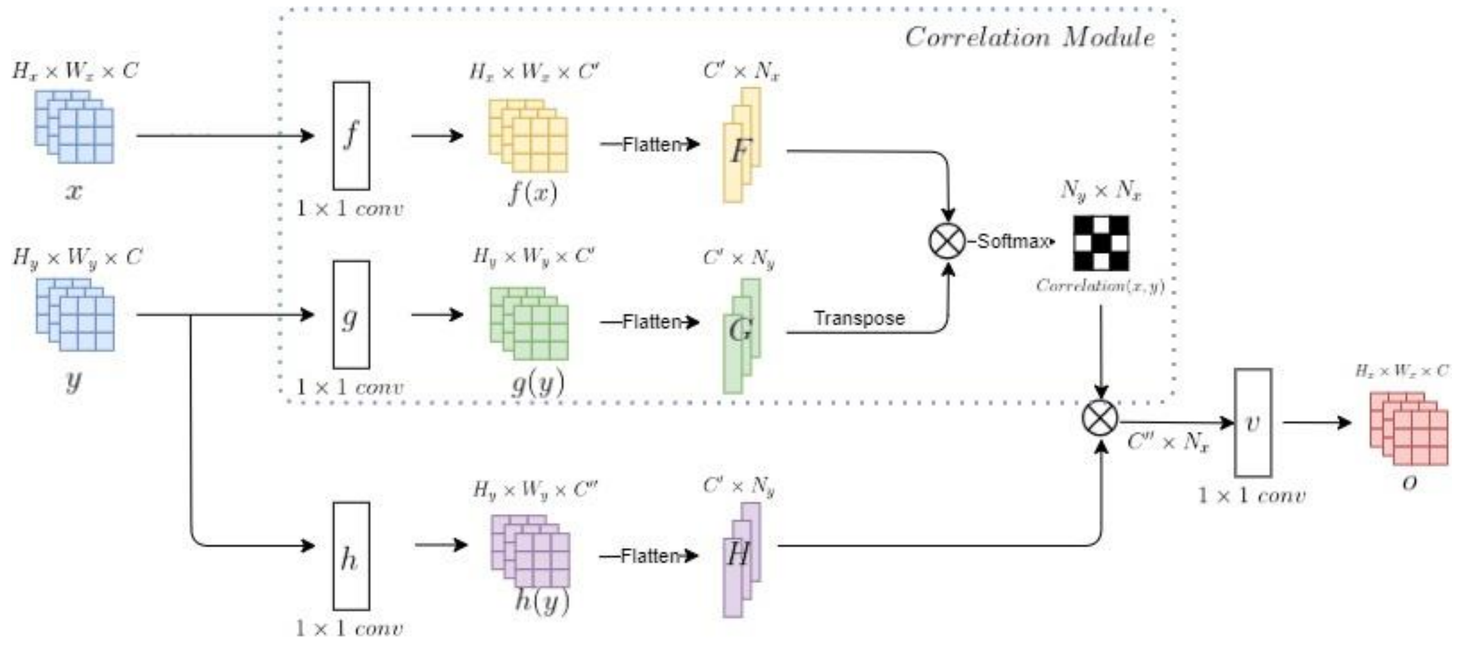

Figure 7: The restructuring module

The restructuring module is used in both the attention module and the style transfer module. The functionality of the restructuring module is to restructure a feature map by using the patches in another feature map with the same channel dimension. The module can be represented as the following function mapping:

$$Restructure: \mathbb{R}^{h_x \times w_x \times c} \times \mathbb{R}^{h_y \times w_y \times c} \mapsto \mathbb{R}^{h_x \times w_x \times c}$$

$$o = Restructure(x, y)$$

For example, there are two tensors x and y as the feature map extracted by the VGG network. The feature maps $x \in \mathbb{R}^{h_x \times w_x \times c}$ and $y \in \mathbb{R}^{h_y \times w_y \times c}$ are fed into the restructuring module to output a self-attention feature map (or residual) $o \in \mathbb{R}^{h_x \times h_x \times c}$ using the features in y. The module has four trainable 1×1 convolution layers f, g, h, and v. f and g are used to transform the information from the feature maps x and y, respectively. The transformed feature map $f(x) \in \mathbb{R}^{h_x \times w_x \times C'}$ and $g(y) \in \mathbb{R}^{h_y \times w_y \times c'}$ are stretched to 2D matrices $F \in \mathbb{R}^{c' \times n_x}$ and $G \in \mathbb{R}^{c' \times n_y}$. Then, the correlation matrix $Corr \in \mathbb{R}^{n_y \times n_x}$ between each pair of patches in x, y is defined as:

$$Corr = softmax(G^T \otimes F),$$

where $softmax$ is applied to each column vector of $G^T \otimes F$. The correlation matrix has the following properties:

- The $Corr(i, j)$ entry represents the similarity between the i-th element of feature map x and the j-th element of feature map y.
- $\forall i \in [0, N-1], \sum_{j=0}^{N-1} Corr_j^{(i)} = 1$ (each column vector sums to one).

The first property is achieved by the network structure. Since the 1×1 convolution layers f, g are trainable, it is optimized to automatically match the elements in two feature maps during the training stage. The second property is achieved by the $softmax$ function since it is applied along the column dimension. This property plays a significant role to construct an attention map that satisfies the specific mathematical requirement.

Similar to g, h is another convolution to transform the feature map y. $h(y)$ is stretched into a matrix $H \in \mathbb{R}^{c'' \times n_y}$. The matrix H is multiplied with the correlation matrix $Corr$ and recovered by the 1×1 convolution v to generate the restructured feature map.

$$o = reshape(conv_{1\times1}(H \otimes Corr))$$

Note that each column of the matrix H^T represents a patch in the feature map y, and each column of $H \otimes Corr$ represents a patch in the feature map x. Therefore, the matrix multiplication can be rewritten in this way to show the linear combination of patches:

$$M = H \otimes Corr$$

$$\forall i \in [0, n_x - 1], M_i = \sum_{j=0}^{n_y - 1} Corr_j^{(i)} H_j^T,$$

where M_i is the i-th column vector of matrix M, H_j^T is the j-th column vector of matrix H^T. According to the second property of $Corr$, each column vector of M is also a weighted average of column vectors of H^T and $Corr_j^{(i)}$ is the weight of H_j^T to construct M_i. **In other word, the entry $\boldsymbol{Corr_j^{(i)}}$ represents the frequency of the $\boldsymbol{i}$-th input patch that is used to construct the $\boldsymbol{j}$-th output patch.** As stated in the first property, $\boldsymbol{Corr_j^{(i)}}$ is also the similarity between i-th patch in feature map x and the j-th patch in feature map y. Therefore, the algorithm assigns a higher weight to the style feature patch that is similar to the content feature to restructure the content feature. The final output o is also called residual feature in the network.

3.4 Self-Attention Module

The self-attention module is modified according to [20]. The module takes the feature map as the input and outputs the attention map. Different from AAMS [20] , which directly takes average along the channel dimension of the restructured feature (or residual feature) o, we exploit the second property of the correlation matrix to generate an attention map with special mathematical property. The structure of self-attention module is shown in Figure 8.

The feature map x is projected by ZCA [17] and then fed into the correlation module to generate a correlation matrix $Corr$. The matrix represents the correlation between each patch pair in the same feature map:

$$Corr = Correlation\_module(x, x).$$

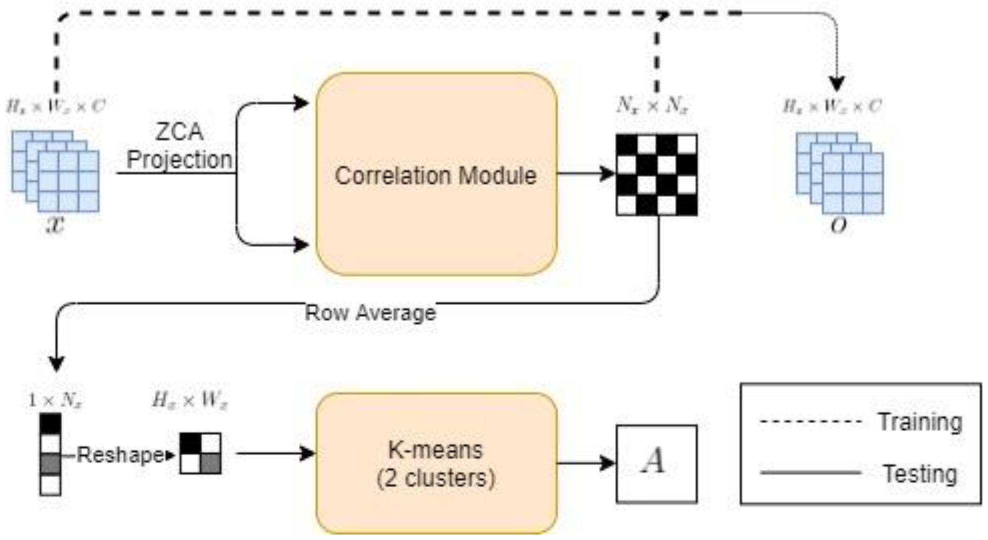

Figure 8: Self-attention module

Due to the change of function domain, the output matrix $Corr$ has more properties:

- $\forall k \in [0, n_x - 1]$, $argmax_{j \in [0, n_x - 1]}\{Corr(k, j)\} = k$; $argmax_{j \in [0, n_x - 1]}\{Corr(j, k)\} = k$.
- If the k-th patch contains the salient feature in the input feature map, the average value of the k-th row will be larger.

The first property is an induction of the first property of the general correlation matrix. The similarity of the patch with itself is higher than the similarity with other patches.

For the second property, note that although we only use the correlation module in this step, the whole restructuring module is optimized during the reconstruction training phase. The output is produced by the **whole restructuring module** during the training phase. As mentioned, $Corr_j^{(i)}$ represents the frequency of the i-th input patch that is used to construct the j-th output patch. The average value along the row direction

$$Average_i = \frac{1}{n_x} \sum_{j=0}^{n_x - 1} Corr_j^{(i)}$$

represents the frequency of i-th input patch that is used construct the whole output feature map. If the k-th input patch contains salient feature, it will be added as a residual in the output feature map according to [20]. Therefore, the output feature map o in restructuring module will contain more features in the k-th patch. Since the k-th input patch is the most similar to itself, the frequency of the k-th input patch to construct the output feature will be larger. According to this property, we define the attention value of the i-th patch as the average of the i-th row vector in $Corr$. The novel attention map sums to one according to the following proof:

$$\sum_{j=0}^{n_x-1} Corr_j^{(i)} = 1$$

$$\sum_{i=0}^{n_x-1}\sum_{j=0}^{n_x-1} Corr_j^{(i)} = n_x$$

$$\frac{1}{n_x}\sum_{i=0}^{n_x-1}\sum_{j=0}^{n_x-1} Corr_j^{(i)} = \sum_{j=0}^{n_x-1} A_j = 1.$$

Due to the limitation of computing resource, we use K-means with only two clusters to segment the attention map into two regions. One region contains salient features that belong to the non-void style, the other region contains features that belong to the blank-leaving style. This Boolean mask is passed to the SAVA-Net module to guide the style swapping.

3.5 SAVA-Net

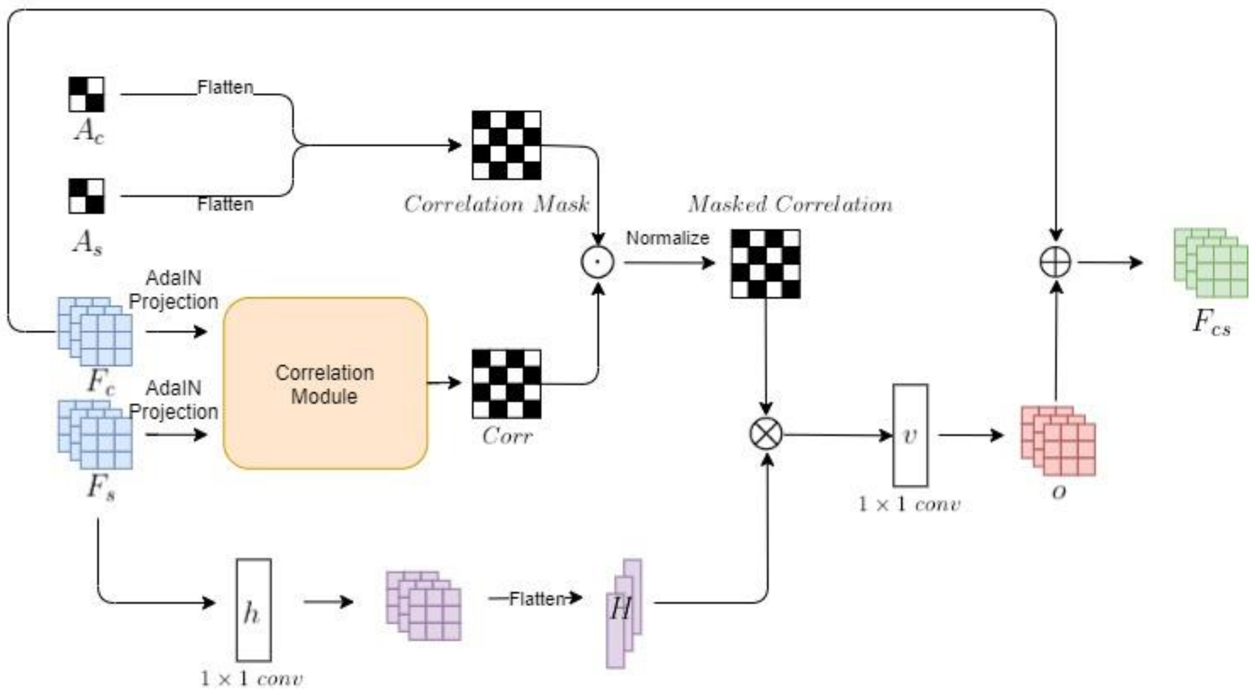

Figure 9: SAVA-Net architecture

As shown in Figure 9, SAVA-Net takes the feature and attention maps of content and style images as the input. The output is the combined feature map. Similar to the general restructuring module, SAVA-Net will generate a correlation matrix $Corr$ by using the correlation module. However, before multiplying the correlation matrix with the style feature H, the correlation matrix is modified by a correlation mask. The combined feature map o, is then computed and added as a residual to the original content feature map:

$$o = reshape(conv_{1\times1}(H \otimes Corr'))$$

$$F_{cs} = o \oplus F_c.$$

There are two modes to generate the masked correlation.

3.5.1 Hard Mode

In the hard mode, the attention maps of content and style images A_c, A_s are flattened to vectors in $\mathbb{R}^{n_c}$ and $\mathbb{R}^{n_s}$, respectively. Then the correlation mask M is defined as:

$$M_j^{(i)} = \mathbb{I}\left[A_s^{(i)} = A_c^{(j)}\right] (\forall i \in [0, n_s - 1], \forall i \in [0, n_c - 1]),$$

where $\mathbb{I}$ represents the indicator function. If $M_j^{(i)} = 1$, both patch i in the content feature map and patch j in the style feature map are in the non-void or void region. The mask is multiplied with the correlation matrix and column-wise normalization will be applied to the masked correlation matrix.

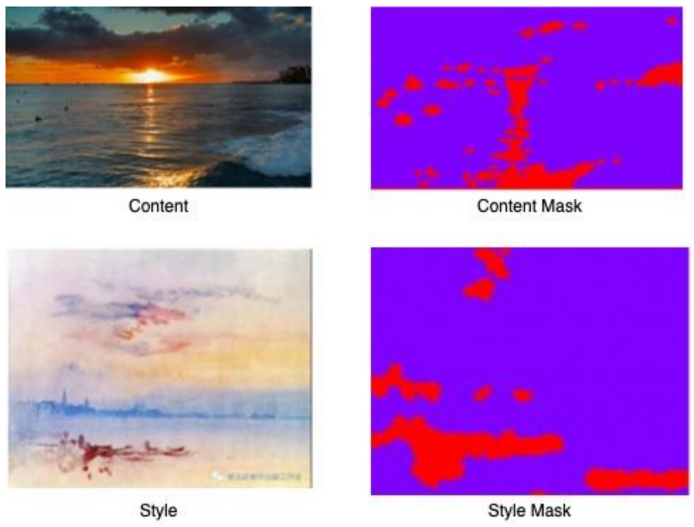

Figure 10: An example of content and style mask

In the original setting of SA-Net, each patch in the output feature map is the linear combination of all patches in the style feature map. But after using the correlation mask, as shown in Figure 10, the patches in the red part of the content mask will be restructured by a linear combination of all patches in the red part of the style feature map. Similarly, the blue part of the content feature map will only learn the style information from the blue part of the style feature map. In this way, the transferred patches only learn the style patches' features in the same category.

3.5.2 Soft Mode

In the self-attention module, if we follow the setting of [20], we can get a reconstructed feature map by using the input feature map itself. In the soft mode, the algorithm computes a correlation matrix for the reconstructed self-attention feature of content and style feature maps:

$$o_c = Restructure(F_c), o_s = Restructure(F_s)$$

$$M = Correlation\_module(o_c, o_s).$$

Then the mask is multiplied with the original correlation matrix elementwise.

3.6 Self-Attention Reconstruction Network

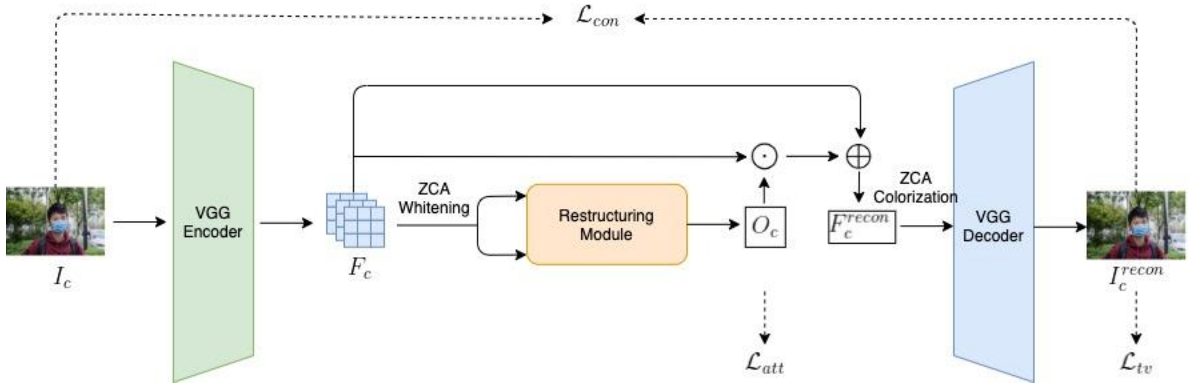

Figure 11: Self-Attention reconstruction network

As shown in Figure 11, we train the attention module by using the VGG-19 encoding-decoding network. The feature map in the fourth layer of VGG network f is fed into the self-attention module to get the attention feature (residual) A. Similar to [20], the reconstructed feature is defined as the multiplication of the feature map and the residual plus the feature map:

$$O_c = Restructure\ (F_c, F_c)$$

$$F_c^{recon} = F_c + F_c \odot O_c$$

The reconstructed feature is then passed to the decoder to obtain the reconstructed image. This network is optimized by reducing the difference between the reconstructed image I_c^{recon} and the original image I_c. Similar to [20], the loss function is defined as a combination of reconstruction loss, attention loss, and variational regularization loss.

$$\mathcal{L} = \lambda_{con}\mathcal{L}_{con} + \lambda_{att}\mathcal{L}_{att} + \lambda_{tv}\mathcal{L}_{tv},$$

$$\mathcal{L}_{con} = \sum_{l \in l_c} ||\phi_l(I_c^{recon}) - \phi_l(I_c)||_2^2 + \lambda_p \left|\left|I_c^{recon} - I_c\right|\right|_2^2,$$

$$\mathcal{L}_{att} = ||O_c||_1$$

$$\mathcal{L}_{tv} = ||I_c^{recon} - shift\_left(I_c^{recon})||_1 + ||I_c^{recon} - shift\_up(I_c^{recon})||_1,$$

where ϕ_l represents the feature map of the content image in the l-th layer of VGG network. The reconstruction loss $\mathcal{L}_{con}$ is used to encourage the network to learn the salient feature in the residual O_c. If the i-th patch is important, it will appear more frequently in the residual feature map O_c. The loss is a summation of L_2 distance in each layer of VGG feature maps and the L_2 distance between the original image and the reconstructed image.

The attention loss $\mathcal{L}_{att}$ is used to restrict the region with a high attention value, which is defined as the residual's L_1 norm. The total variational regularization loss $\mathcal{L}_{tv}$ is used to smoothen the texture in the reconstructed images. It is computed as the L_1 distance between the new image and shifted new images.

During training, we use Adam optimizer [24] with learning rate 0.0004 and learning rate decay 0.00005. The batch size is 8 and the maximum iteration is 80000. Each image will be fed into the network 4 (8*80000/40000) times. The restructuring module and VGG decoder contain trainable parameters, but the VGG encoder is not trainable. The program divides the whole network into different parts, and explicitly define whether a part should be trainable. The hyper-parameter setting is:

$$\lambda_{con} = 10, \lambda_{att} = 6, \lambda_{tv} = 10, \lambda_p = 1$$

3.7 Style Transfer Network

The structure of the style transfer network is shown in Figure 6. The self-correlation module is pre-trained by the self-attention reconstruction network, it will be used to generate the attention maps for content and style image. The loss function is defined as a combination of content loss, style loss, and identity loss:

$$\mathcal{L} = \lambda_c \mathcal{L}_c + \lambda_s \mathcal{L}_s + \mathcal{L}_{identity}$$

$$\mathcal{L}_c = ||VGG_{encoder}(I_{cs})^{r\_4\_1} - F_c^{r\_4\_1}||_2 + ||VGG_{encoder}(I_{cs})^{r\_5\_1} - F_c^{r\_5\_1}||_2$$

$$\mathcal{L}_s = \sum_{i=1}^{L} ||\mu(\phi_i(I_{cs})) - \mu(\phi_i(I_s))||_2 + ||\sigma(\phi_i(I_{cs})) - \sigma(\phi_i(I_s))||_2$$

$$\mathcal{L}_{identity} = \lambda_{identity1}(||I_{cc} - I_c||_2 + ||I_{ss} - I_s||_2) + \lambda_{identity2} \sum_{i=1}^{L} (||\phi_i(I_{cc}) - \phi_i(I_c)||_2 + ||\phi_i(I_{ss}) - \phi_i(I_s)||_2)$$

The content loss $\mathcal{L}_c$ is used to preserve the structural information in the content image. By using $\mathcal{L}_c$, the learnable convolution in the correlation module will automatically build up the correlation matrix. It is computed as the summation of L_2 distance between the content feature map to the combined feature map in different layers of VGG. The style loss $\mathcal{L}_s$ is used to preserve the textural (or style) information in the style image. $\mathcal{L}_s$ is computed for each layer of the VGG-19 network as the L_2 distance of the mean and variance. The identity loss $\mathcal{L}_{identity}$ is used to ensure the consistency of the model. I_{cc} represents the content image transferred by itself, and the I_{ss} represents the style image transferred by itself. The loss is defined as a weighted summation of L_2 distance of images and L_2 distance of feature maps in different layers.

During training, we use Adam optimizer with learning rate 0.0004 and learning rate decay 0.00005. The batch size is 4, and the maximum iteration is 200000. Each content image will be fed into the network 20 (4*200000/40000) times, while each style image will be fed into the network 10 (4*200000/80000) times. The SAVA-Net and VGG decoder contain trainable parameters, but the VGG encoder and the self-attention module are not trainable. The hyper-parameter setting is the same as [1]:

$$\lambda_c = 1, \lambda_s = 3, \lambda_{identity1} = 50, \lambda_{identity2} = 1$$

4 Design & Implementation

4.1 Development Environment

- **Operating System** Ubuntu 18.04, macOS 10.15
- **Device** Macbook Pro 2017, Dell Optiplex 9020
- **Programming Language** Python 3.7.6
- **Deep Learning Framework** Pytorch 1.5.0 [23]
- **Computing Resources** GPU: NVIDIA GeForce GTX 2080 TI
- **Auxiliary Tools** Visual Studio Code, Jupyter Notebook, Git, iTerm, Vim, CUDA toolkit 10.0.130
- **Important Libraries** Matplotlib, Numpy, PIL, Torch, Torchvision, tqdm

4.2 Implementation

This part will give some overall and significant information about the implementation of the project.

We implement the style transfer system mainly in Python language by Visual Studio Code and Jupyter Notebook. Here is the link of the Github repository.

Table 1: Lines of code in each repository

Component Name	Language	Number of Files	File List	Lines of Code
Data	Python	2	*./filter.py* *./filter_percentage.py*	95
Model	Python	2	*./net/models.py* *./net/network.py*	570
Utilities	Python	1	*./net/utils.py*	190
Training	Python	2	*./train_attn.py* *./train_sava.py*	463
Testing	Python	2	*./test_attn.ipynb* *./transfer_sava.ipynb*	-
SUM	Python	7	-	1318

The functionality of each component is shown below:

- **data**: This component is used to filter the large image in MS-COCO [2] and WikiArt [3] training set.
    - *./filter.py*: Filter the images with wrong format.

    - *./filter_percentage.py*: Filter the images that is too large to be fed into the network.

- **model**: This component defines all the models used in the project, including self-attention module, self-attention network, SAVA-net, VGG network etc.
    - *./net/models.py*: Contains two classes: *Transform, SAVA_test.*
    - *./net/network.py*: Contains the implementation of VGG network and six classes: *Encoder, Correlation, SelfAttention, SAVANet, SANet, and AttentionNet.*
- **utilities**: This component contains miscellaneous functions, including statistic metrics calculation, feature map (tensors) manipulation, and other helper functions.
- **training**: This component is used to train the self-attention module in the first step and the style transfer module in the second step.
    - *./train_attn.py*: Contains the implementation for phase I training.
    - *./train_sava.py*: Contains the implementation for phase II training.
- **testing**: This component is used to test and visualize the style transfer module.
    - *./test_attn.ipynb*: Contains the code to run the attention module and visualize the attention map generated.
    - *./transfer_sava.ipynb*: Contains the code to run the whole style transfer model and visualize the combined images.

4.3 Data

4.3.1 Data Filtering

The data will be filtered by two filters. The first filter removes all images that cannot be opened by the PIL image library, images with more than three dimensions, and images with more than three channel dimensions. Due to the limitation of the computational resource, GPU memory overflow will happen if the large images are fed into the VGG network. The second filter removes large images with a size larger than a specific threshold.

4.3.2 Data Preprocessing

Although the images are already filtered, the training will be prolonged if we direct feed the images into the VGG-19 network. Therefore, the following preprocessing steps are applied to reduce the size and improve the performance:

- The images are resized to (512, 512, 3) for content and style sets.
- The images will be randomly cropped to the size (256, 256, 3) for training. The images will be center cropped to the size (256, 256, 3) for testing.

- After transferring to tensor, the dataset is normalized by subtracting the mean and dividing the standard deviation in each color channel.

4.4 Model

Table 2: Architecture of the VGG-19 encoder

Layer	Type	Kernel Size	Stride
1	conv1_1	1×1	$[1,1]$
2	conv1_2	3×3	$[1,1]$
3	max pooling	2×2	$[2,2]$
4	conv_2_1	3×3	$[1,1]$
5	conv_2_1	3×3	$[1,1]$
6	max pooling	2×2	$[2,2]$
7	conv_3_1	3×3	$[1,1]$
8	conv_3_2	3×3	$[1,1]$
9	conv_3_3	3×3	$[1,1]$
10	conv_3_4	3×3	$[1,1]$
11	max pooling	2×2	$[2,2]$
12	conv_4_1	3×3	$[1,1]$
13	conv_4_2	3×3	$[1,1]$
14	conv_4_3	3×3	$[1,1]$
15	conv_4_4	3×3	$[1,1]$
16	max pooling	2×2	$[2,2]$
17	conv_5_1	3×3	$[1,1]$
18	conv_5_2	3×3	$[1,1]$
19	conv_5_3	3×3	$[1,1]$
20	conv_5_4	3×3	$[1,1]$
21	max pooling	2×2	$[2,2]$

4.4.1 VGG Architecture

VGG encoder and decoder are very important to extract and recover feature maps in this project. The setting of the encoder is shown in Table 1. We use the features from the 12-th and 17-th layers. The decoder architecture is shown in Table 3.

Table 3: Achitecture of the VGG-19 decoder

Layer	Type	Kernel Size	Stride
1	conv1_1	3×3	$[1,1]$
2	nearest upsampling	-	-
3	conv_2_1	3×3	$[1,1]$
4	conv_2_2	3×3	$[1,1]$
5	conv_2_3	3×3	$[1,1]$
6	conv_2_4	3×3	$[1,1]$
7	nearest upsampling	-	-
8	conv_3_1	3×3	$[1,1]$
9	conv_3_2	3×3	$[1,1]$
10	nearest upsampling	-	-
11	conv_4_1	3×3	$[1,1]$
12	conv_4_2	3×3	$[1,1]$

4.4.2 GPU VS. CPU

During the implementation, I find that different algorithm should be computed in different process units. For example, the VGG-19 network should be computed the GPU, but the CPU should compute the Singular-Value-Decomposition in the ZCA whitening phase to improve the computation speed. Due to the GPU memory limitation, the program should also empty the catch frequently to delete intermediate results.

4.5 Training

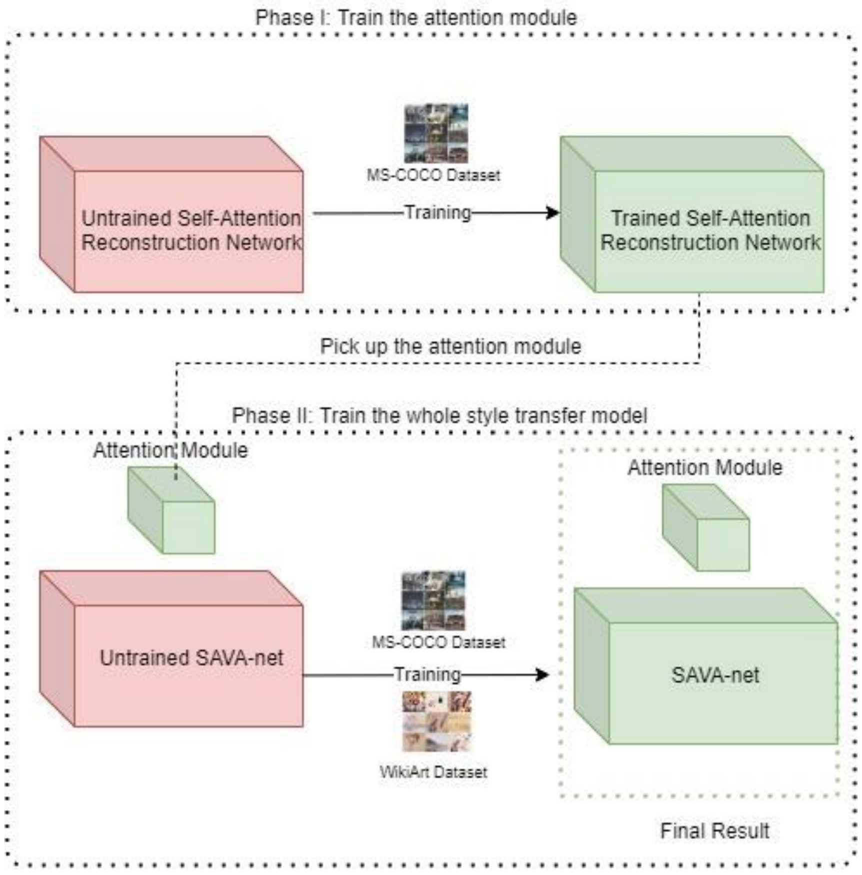
Phase I: Train the attention module
MS-COCO Dataset
Untrained Self-Attention Reconstruction Network
Training
Trained Self-Attention Reconstruction Network
Pick up the attention module
Phase II: Train the whole style transfer model
Attention Module
Attention Module
Untrained SAVA-net
MS-COCO Dataset
Training
WikiArt Dataset
SAVA-net
Final Result

Figure 12: Training pipeline

Due to the limitation of computing resource (GPU memory especially), the model cannot be trained jointly. Therefore, we divide the whole training procedure into two phases.

As shown in Figure 12, we first train a self-attention reconstruction network based on the MS-COCO dataset. This model will be saved as a PyTorch checkpoint which is reusable. After that, the trained self-attention module will be loaded into the SAVA-net to create an attention map for the content and style images. The blank-leaving information is predicted by the attention map, and the SAVA-net will learn the style-content mapping guided by the attention map. Finally, the SAVA-Net is trained by MS-COCO and WikiArt dataset to generate the final model, which can transfer the content image based on style and voidness information.

5 Evaluation & Result Analysis

5.1 Experiment Setup

5.1.1 Training Experiment

- **Operating System** Ubuntu 18.04
- **Device** Dell Optiplex 9020
- **Language&Framework** Python 3.7.6**,** Pytorch 1.5.0
- **Computing Resources** NVIDIA GeForce GTX 2080 TI
- **Model** Self-attention reconstruction network, Style transfer model

5.1.2 Testing Experiment

- **Operating System** Ubuntu 18.04
- **Device** Dell Optiplex 9020, CS Lab HTGC
- **Language&Framework** Python 3.7.6**,** Pytorch 1.5.0
- **Computing Resources** NVIDIA GeForce GTX 2080 TI, NVIDIA Tesla V100
- **Model** Style transfer model

5.2 Dataset Details

Table 4: Summary of datasets used in this project

Name	**Size**	**Type**	**Training Phase**	**Data Property**
MS-COCO [2]	40,000	Image	Phase I	Real life content images
WikiArt [3]	80,000	Image	Phase I, II	Artworks

5.3 Evaluation of Different Approaches

5.3.1 Self-attention Training

Two settings can be changed during the first training phase. The training set for phase I could be the MS-COCO, WikiArt, or a mixed dataset of the two datasets. The projection method before the restructuring module could be ZCA projection or AdaIN projection.

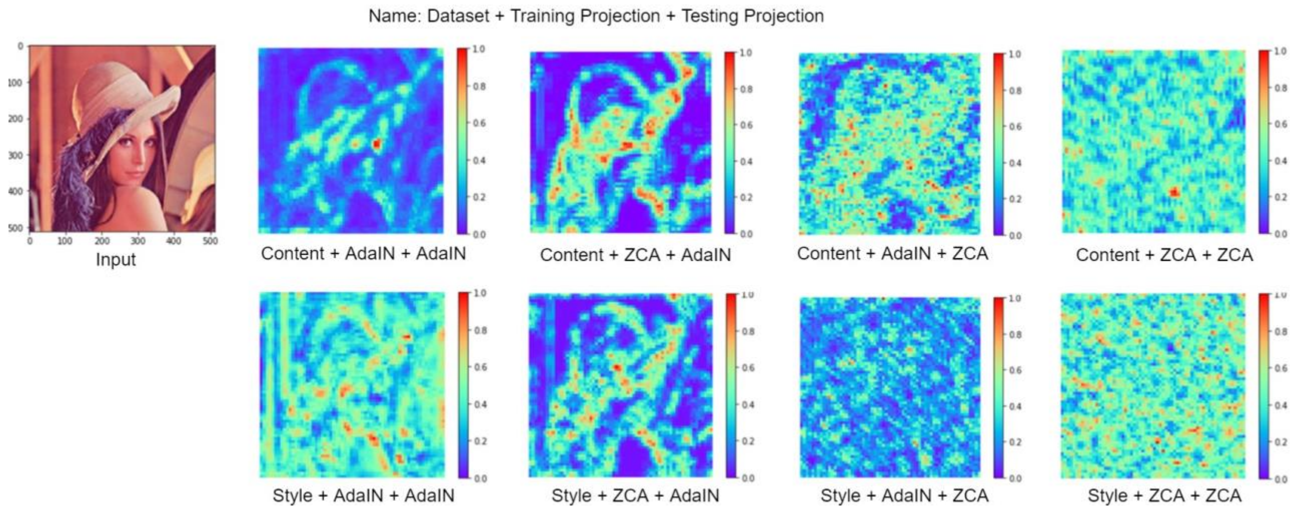

Figure 13: Experiment of self-attention training

In the experiment, I tried to use different combinations of dataset and training projection. And then visualize the average residual in the attention module with different testing projection. The result is shown in Figure 13. It can be observed that the ZCA projection has the best performance. Therefore, different from [20], we use ZCA projection instead of AdaIN projection to calculate the attention map.

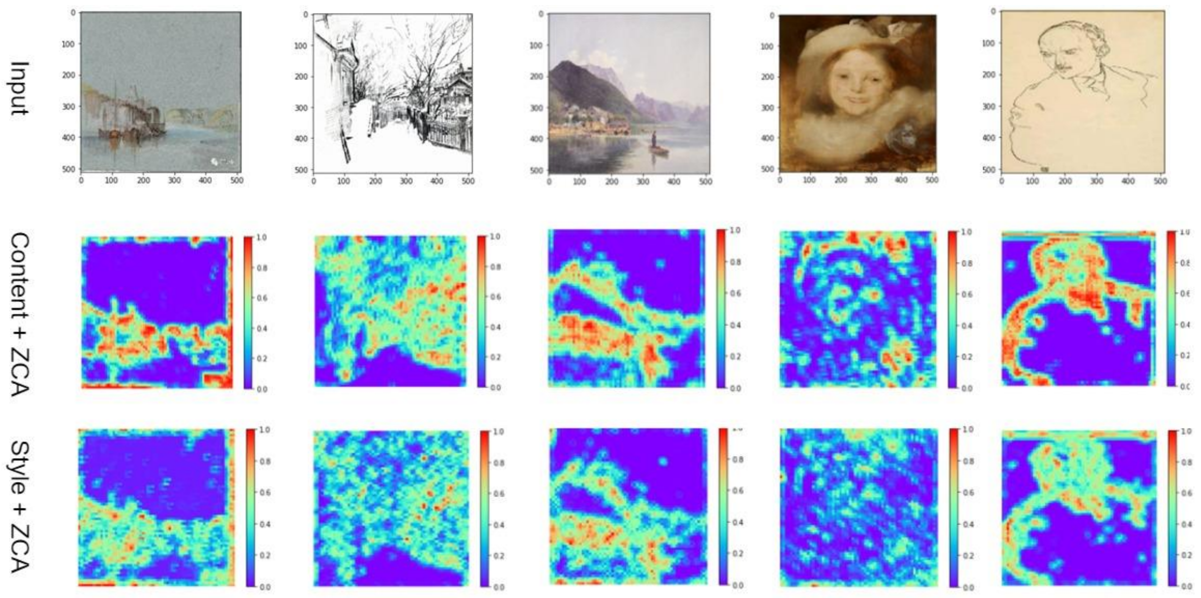

Figure 14: Experiment of training set

Then I visualize the channel-wise average of the residual feature map for the blank-leaving style images. The result is shown in Figure 14. We can observe that the model trained by the content dataset has a better performance even for the images in the style dataset. Therefore, we only use the MS-COCO dataset to train the self-attention module in phase I.

5.3.2 Attention Map

As mentioned, we modified the self-attention module to generate mathematical meaningful attention maps. The original method [20] use the averaged residual feature as the attention map. Although it includes part of the voidness information, the attention map is an extracted feature map. Therefore, two images with the same level of voidness but different color will have different attention map. However, our attention value represents the frequency of the patch that is being used to construct the residual feature. As shown in Figure 15, the first two input images have the same voidness and different color. The original attention map is influenced by the color information, but our attention map assigns high attention values to the corner precisely. For the third input, we can observe that our attention map assigns high attention value to a more precise region compared with the original algorithm.

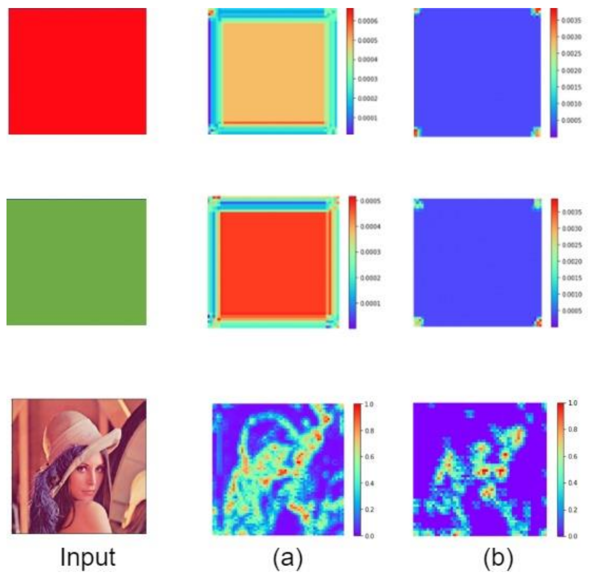

Figure 15: Comparison of different attention algorithms (a) self-attention from [20] (b) our attention map

5.3.3 Attention Mask

As shown in Figure 8, the attention mask is generated by K-means to assign a Boolean value to each patch in the input image. The Boolean value represents whether this patch belongs to the void or non-void region. An intuitive method to generate the attention mask is to set a threshold on the attention map. However, the variance of the attention value distribution is low, and many patches have the same attention value. In this case, the algorithm chooses the patches according to the array order and causes the “overflow” of the attention mask.

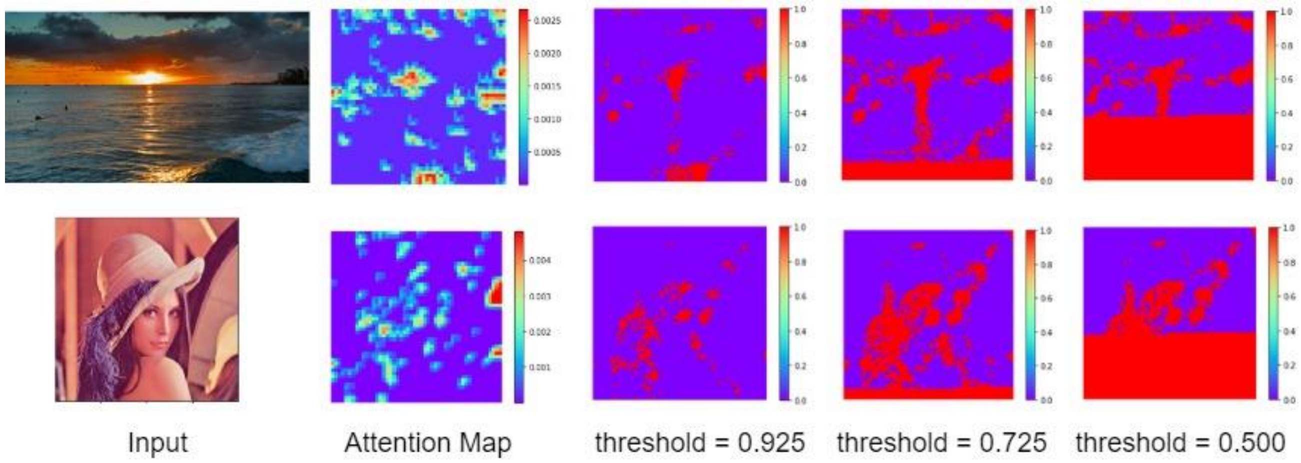

Figure 16: Mask overflow with different thresholds

As shown in Figure 16, if we use a threshold to filter the attention map, the attention mask may "overflow" due to the uneven distribution of attention value in the attention map. Therefore, we decide to use K-means to automatically generate two clusters in the attention map to create the attention mask.

5.3.4 Soft Attention Mask VS Hard Attention Mask

As mentioned in SAVA-Net part, there are two modes to use correlation mask to guide the style transfer model. The performance of the soft attention mask and hard attention mask is shown in Figure 17. We can observe that the performance of the hard attention mask is better than the soft attention mask because the background in the soft attention result should not be depicted in detail.

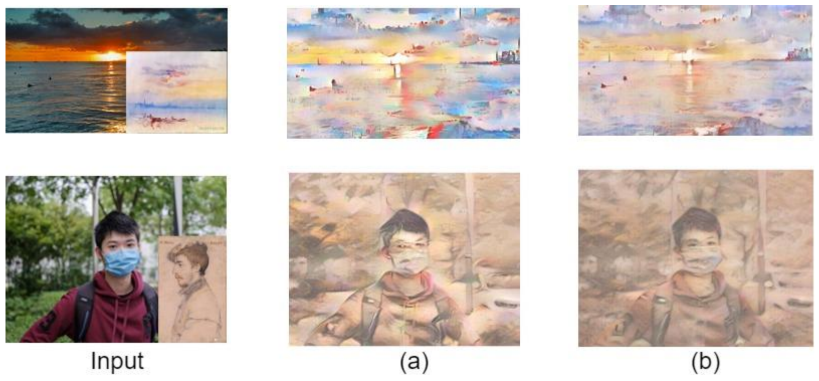

Figure 17: Comparison of (a) soft attention mask and (b) hard attention mask

5.4 Result Analysis

5.4.1 Hard Attention Mask

The result of the attention mask is shown in Figure 18. We can find that the region of interest is precisely detected by the algorithm and labelled as red.

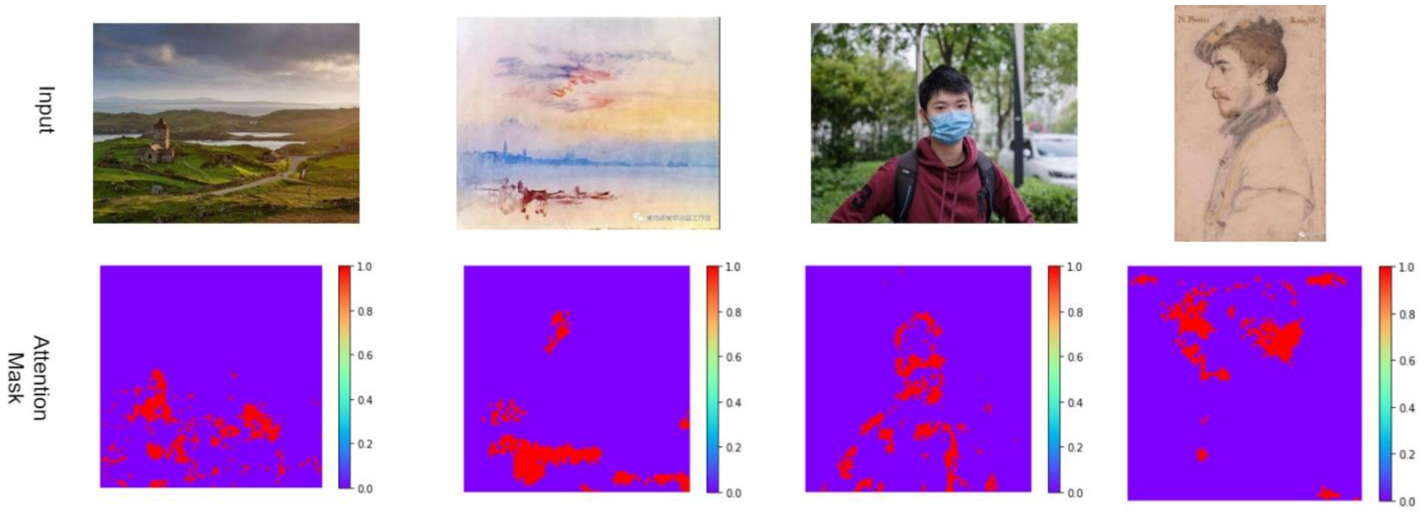

Figure 18: Result of attention mask

5.4.2 Qualitative Comparison

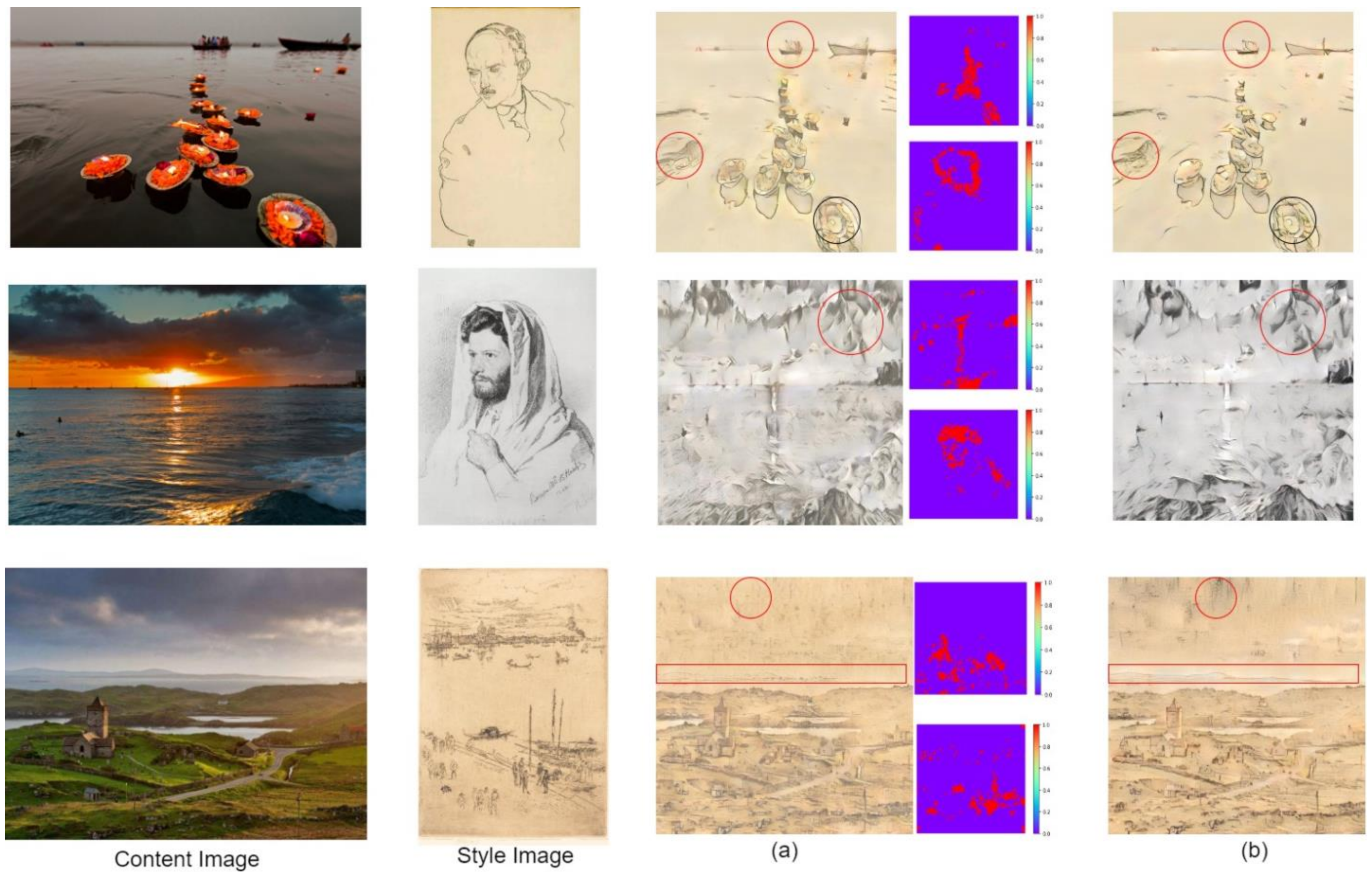

Figure 19: Comparison of (a) proposed algorithm with content and style attention masks and (b)SA-Net

As shown in Figure 19, we compare the proposed SAVA-Net with the baseline SA-Net to assess the ability to learn the voidness information. Red represents different patterns, and black represents similar patterns between the result of two algorithms. For the first transfer, we can find that the result of SA-Net tends to have the same line thickness in different scales. The candle, boat, and wave are depicted by similar texture. However, the texture is different for different scale in our result. For the second transfer, we can find that the result of SA-Net has false matching. The sky is depicted two a human face structure. For the last transfer, the sky also contains false matching in the SA-Net result. Besides, the mountain in the red rectangle is transferred to the blank-leaving style in our result but preserved in the SA-Net result.

Therefore, our method achieves the requirement of learning the style information and voidness information at the same time. The unwanted property such as false matching and texture repeatability is reduced.

5.4.3 Impact of Hard Attention Mask

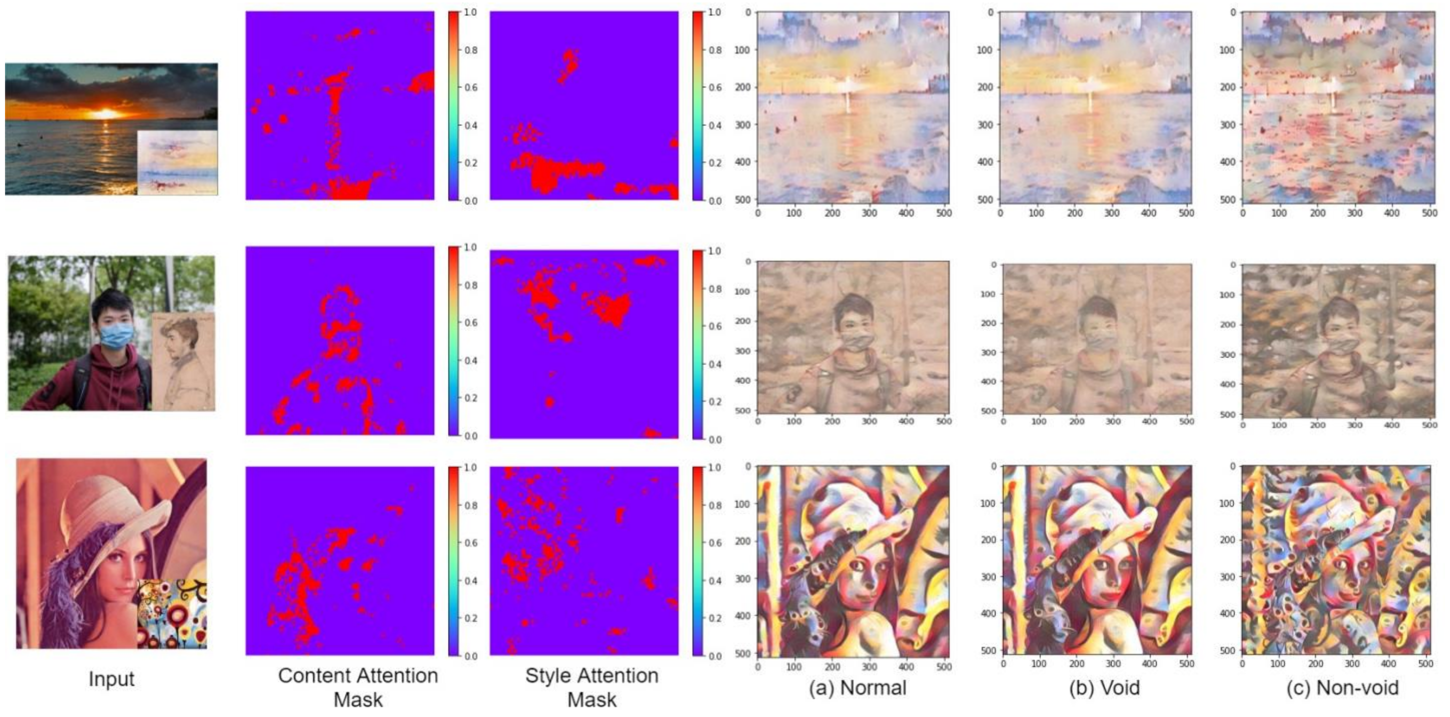

Figure 20: Comparison of (a) normal setting with (b) output transfered by void feature (c) output transfered by non-void feature

As mentioned in 3.5.1, the hard attention mask can restrict the domain of style learning. In the normal setting, the patch in the void region of the content image will only learn from the patches in the void region of the style image. In order to show the impact of hard attention, we transfer the whole content image by using patches in only one region to compare with the normal setting. As shown in Figure 20, the image transferred by the void feature tends to have a larger blank-leaving region, while the image transferred by the non-void feature tends to have the larger region depicted in detail.

6 Conclusion

6.1 Critical Review

Overall, I consider this project is of high fulfilment. Firstly, our modified attention map has both mathematical meaning and better performance compared to the original approach. Meanwhile, by applying K-means to our attention map, the hard attention mask is used to guide the style transfer module to transfer patches with the same category. Then, the whole model is trained in two phases to optimize the self-attention module and style transfer module separately. Finally, the test result shows the style transfer model successfully learns the voidness information and texture information at the same time.

I think the self-attention module and hard attention guided style transfer module are the most significant contribution of this project. The self-attention module has satisfactory mathematical property and performance, while the hard attention mask is incorporated with the restructuring network to improve the performance of the style transfer module.

On the other hand, I am also obstructed by many problems. Firstly, the principle of many settings used in this project is not well explained. For example, when we train the self-attention reconstruction network, we use data only from the content set, ZCA projection, and add the multiplication of residual feature with original feature map to the original feature map. The setting is tested by the experiment but not explained by theories. Secondly, the VGG decoder is not properly trained. The output image decoded will have strange color sometimes. Last but not least, the limitation of computational resource makes it difficult to run more complex algorithms. For example, the hard attention mask is only divided into two regions (void and non-void). But if I divide the attention mask into more regions, the GPU memory will be overflowed. Besides, the limitation of GPU memory also makes it difficult to add new loss function such as attention loss to match the voidness statistics of the style image.

Despite the problems I faced, I have learned a lot of knowledge from this topic. Under the guidance of my supervisors, my ability to learn new algorithms, identify the problem of the existing approach, design new algorithms to solve the problem, set up the environment, and implement the algorithm is improved during this project.

6.2 Summary of Achievements

The summary of the contribution in this project is as follows:

- A novel attention map calculation algorithm to precisely identify the void and non-void region.
- A style transfer module based on restructuring module guided by the hard attention mask (derived from the attention map)

- A novel training pipeline to ensure the optimization degree of self-attention module and style transfer module with the limited computational resource.
- Experiments to test the performance of different settings for the attention map generation, attention map training, attention mask generation, correlation mask mode.

6.3 Future Improvement

In the future, If the computational resource (GPU memory especially) is fulfilled, we can further improve the performance of the algorithm. Firstly, we can divide the image into more regions in the hard attention mask to map the features more precisely. We can even integrate segmentation or object detection algorithms into the algorithm to replace the attention mapping. Secondly, we can add the attentional loss function to match the statistics of the combined image and the style image to ensure the voidness information is learned by the model.

7 References

[1] D. Y. Park and K. H. Lee, "Arbitrary Style Transfer with Style-Attentional Networks," in *CVPR*, 2019.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *Proc. ECCV,*, 2014.

[3] F. Phillips and B. Mackintosh, "Wiki Art Gallery, Inc.: A case for critical thinking," in *Accounting Education*.

[4] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural Style Transfer: A Review," in *IEEE Transactions on Visualization and Computer Graphics*, 1 Nov. 2020.

[5] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[6] Y. Li, N. Wang, J. Liu, and X. Hou, "Demystifying neural style transfer," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 2017.

[7] E. Risser, P. Wilmot, and C. Barnes, "Stable and controllable neural texture synthesis and style transfer using histogram losses," in *ArXiv e-prints*, Jan. 2017.

[8] S. Li, X. Xu, L. Nie, and T.-S. Chua, "Laplacian-steered neural style transfer," in *Proceedings of the 2017 ACM on Multimedia Conference*, 2017.

[9] J. Johnson, A. Alahi, and L. Fei-Fe, "Perceptual losses for real-time style transfer and super-resolution," in *European Conference on Computer Vision*, 2016.

[10] D. Ulyanov, V. Lebedev, A. Vedaldi, and V. Lempitsky, "Texture networks: Feed-forward synthesis of textures and stylized images," in *International Conference on Machine Learning*, 2016.

[11] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Improved texture networks: Maximizing quality and diversity in feedforward stylization and texture synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[12] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *European Conference on Computer Vision*, 2016.

[13] V. Dumoulin, J. Shlens, and M. Kudlur, "A learned representation for artistic style," in *International Conference on Learning Representations*, 2017.

[14] D. Chen, L. Yuan, J. Liao, N. Yu, and G. Hua, "Stylebank: An explicit representation for neural image style transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] Y. Li, F. Chen, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Diversified texture synthesis with feed-forward networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[16] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization," in *IEEE International Conference on Computer Vision (ICCV)*, Venice, 2017.

[17] Y. Li, C. Fang, J. Yang, Z. Wang, X. Lu, and M.-H. Yang, "Universal Style Transfer via Feature Transforms," in *Advances in Neural Information Processing Systems*, 2017.

[18] T. Q. Chen and M. Schmidt, "Fast Patch-based Style Transfer of Arbitrary Style," in *Proceedings of the NIPS Workshop on Constructive Machine Learning*, 2016.

[19] L. Sheng, Z. Lin, J. Shao, and X. Wang, "Avatar-Net: Multi-scale Zero-shot Style Transfer by Feature Decoration," in *CVPR*, 2018.

[20] Y. Yao, J. Ren, X. Xie, W. Liu, Y. Liu, and J. Wang, "Attention-aware Multi-stroke Style Transfer," in *CVPR*, 2019.

[21] K. Simonyan and A. Zisserman, " Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[22] Z. Lin, M. Feng, C. N. D. Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio, " A structured self-attentive sentence embedding," in *ICLR*, 2017.

[23] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan, "FROM RESEARCH TO PRODUCTION," Facebook's AI Research lab (FAIR), September 2016. [Online]. Available: https://pytorch.org/. [Accessed 2021].

8 Appendix

8.1 A. Monthly Logs

1. October:
   (a) Get familiar with the field of video quality assessment.
   (b) Review the important works of VQA.
   (c) Focus on VQA with deep learning, understand how does the DeepVQA, GMSDFLOW, and C3DVQA solve the VQA problem
   (d) Based on the previous methods, start to consider how to design the new VQA algorithm
2. November:
   (a) Observe the distortion of the database
   (b) Set up the machine in the FYP lab
   (c) Learn about the Pyramid
3. December
   (a) Revise the preliminary design based on the requirement
   (b) Learn about video classification
   (c) Try to propose an end-to-end learnable and injective mapping for VQA, instead of relying heavily on DISTS.
4. January
   (a) Switch the topic to "Style-Attention Void-Aware Style Transfer"/
   (b) Summarize the previous work.
   (c) Write the Interim Report II.
5. February
   (a) Design and implement the attention loss to improve the performance.
   (b) Adjust the interim report.
   (c) Adjust the hyperparameters in the style transfer module.
6. March
   (a) Change the style loss and content loss to improve the style transfer result based on attention.
   (b) Design the demonstration video.
7. April
   (a) Conduct experiments on the comparison of the detailed algorithms.
   (b) Finish the final report, slide and video.