Arbitrary Style Transfer with Enhanced Edge Awareness

William Greim
Student of Computer Science
University of Missouri
Columbia, United States
lgreim33@gmail.com

***Abstract*—Trends in research for new methods of arbitrary style transfer have focused on making improvements in computationally efficient style transfer, as well as preserving important features and edges from the content image. This paper proposes a model that combines the efficiency of using an Adaptive Instance Normalization (AdaIN) layer with a Convolution Block Attention Module (CBAM) to increase the weight of important features. A traditional loss formula adding content and style loss is also compared to a loss model that adds the calculated SSIM between the gradient magnitude of the generated image and the gradient magnitude of the original image. One benefit of this approach to reducing edge loss is the lack of overhead at inference time, which we see when we compare performance between the model that uses SSIM loss and the model that does not. The model that uses SSIM loss does result in a clear reduction of artifacts in the generated images, likely due to the model being penalized for adding edges that do not appear in the content images.**

***Index Terms*—Style Transfer, AdaIN, CBAM, SSIM, Arbitrary Style Transfer**

I. Introduction

Style transfer is the process in which the abstract "style" of one image is applied to the content of another image. It was first discovered by Gatys et al that deep convolutional neural networks were suitable for extracting stylized features from images [1]. Continued work in this area with style transfer models have focused on utilizing an encoder-decoder model, with variations of a pretrained VGG network serving as the encoder. Methods for improvement of the model generally come in the form of additional components that transform the feature maps generated by the encoder before they get passed to the decoder [2], [3], [5]. However, these models generally have trouble preserving important content features in the stylized image, without significantly increasing model complexity.

In this paper, a novel model architecture is proposed that combines components of architectures seen in previous work on style transfer. Specifically, a CBAM is used to increase the attention given to important features, and Adaptive Instance Normalization is used to efficiently combine the low-level content and style feature maps. Furthermore, different loss functions are tested to see if the model output can be improved without additional overhead during the inference time. The architecture was primarily inspired by the work of Woo et al. , who proposed the use of CBAM in style transfer models [4], and the works of Huang and Belongie, who proposed the use of Adaptive Instance Normalization for style transfer [3].

II. Background

A. AdaIN

Huang and Belongie propose a method of arbitrary style transfer using adaptive instance normalization to align the mean and variance of the content image with the mean and variance of the style image. This approach is computationally efficient due to only requiring a few operations, and does not contain trainable weights, reducing training time significantly [3]. The AdaIN layer takes the encoded feature maps up to $relu4\_1$ from a style image y and a content image x, it then normalizes the content feature map, which is then multiplied by the style standard deviation and summed with the style mean.

$$AdaIN(x, y) = \sigma(y)(\frac{x - \mu(x)}{\sigma(x)}) + \mu(y) \tag{1}$$

Huang and Belongie provide a very nice intuition for why AdaIN works for style transfer. Imagine a feature channel that identifies brushstrokes characteristic of a particular artistic style. For a style image with such brushstrokes, this feature channel would exhibit a high average activation. AdaIN modifies the content image's feature representation so that it matches these feature statistics, specifically the mean and variance of the style image. This ensures the same high average activation for the brushstroke feature while preserving the structure of the content image [3]. The decoder should then be able to learn to reconstruct the combined feature maps into a stylized output that matches the dimensionality of the input image. The code from the original paper was written in Lua, however others have adapted it in Python. This paper uses an implementation that directly credits Huang and Belongie, and can be found here: https://github.com/naoto0804/pytorch-AdaIN [8].

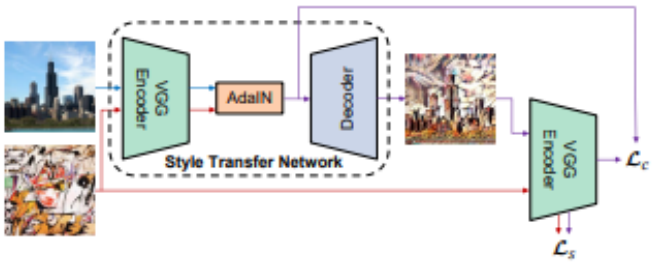

Fig. 1: Huang and Belongie AdaIN Architecture

B. CBAM

The convolutional block attention model was originally proposed by Woo et al, contributing the addition of a spatial attention module to the pre-existing channel attention module. It was later implemented in a style transfer network by Xin and Li [2]. Xin and Li show that the use of CBAM does improve the model's ability to retain important content post stylization. However, the addition of multiple trainable AdaAttN networks significantly increases model complexity and overhead during inference time, making it difficult to achieve real time stylization.

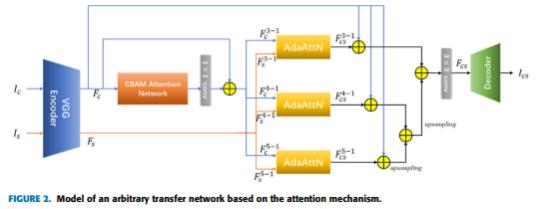

FIGURE 2. Model of an arbitrary transfer network based on the attention mechanism.

Fig. 2: Xin and Li CBAM AdaAttN Architecture

CBAM works sequentially across both channel and spatial dimensions to produce a refined feature map with increased weight on important areas. For our model, an existing implementation of CBAM is used, and can be found here: https://github.com/Jongchan/attention-module/blob/master/MODELS/cbam.py [7]. The CBAM takes some intermittent feature map $F \in \mathbb{R}^{C\times H\times W}$, in our case the output of $relu4\_1$ from the VGG-19 encoder. CBAM then infers a one dimensional channel attention map $M_c \in \mathbb{R}^{C\times 1\times 1}$ and a two dimensional spatial attention map $M_s \in \mathbb{R}^{1\times H\times W}$. Then, it determines the most important channels in the passed feature map before multiplying the increased channel attention feature map back into the main input features, giving us F', which is then given to the spatial attention module to apply max and average pooling to identify spatially important features. This attention map is then multiplied with F' to get our final refined feature map F'' [4]. The incorporation of the CBAM should help reduce the loss of details in important structures in the original content image.

$$F' = M_c(F) \otimes F \tag{2}$$

$$F'' = M_s(F') \otimes F' \tag{3}$$

Where $\otimes$ denotes element wise multiplication. F'' is the final refined feature map output [4].

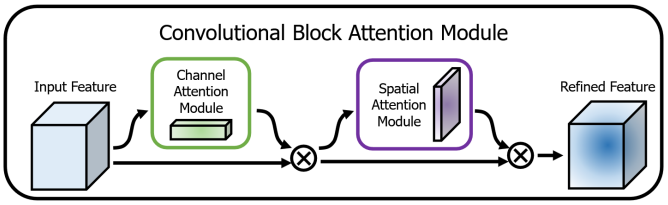

Fig. 3: CBAM Overview

III. Model Architecture

The model is made up of four main components, the first layer is a pre-trained VGG-19 model taken from the Pytorch models library [6], which receives a content image c and a style image s. It then extracts the intermittent feature maps produced by layers: $relu1\_1$, $relu2\_1$, $relu3\_1$, and $relu4\_1$, the final layer of which we will call F_c and F_s respectively. F_c is then passed to the CBAM component, the output of which we pass to the AdaIN function, along with F_s. AdaIN then aligns the mean and variance of the style and content feature maps, producing the target feature map F_{cs}. The target feature map is finally passed to the decoder, which produces an image $g(F_{cs})$, with the same dimensions as the input images. At this point, loss is calculated, and the weights of the CBAM and decoder are adjusted based on how well the style was applied and how well the content was retained. As stated before the structure is inspired by Xin and Li [2], as well as Huang and Belongie [3].

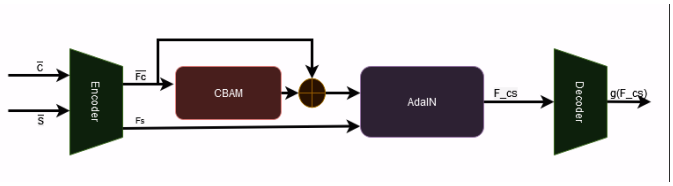

Fig. 4: Proposed Model Architecture

IV. Loss

This paper utilizes two different loss functions used to train two versions of the same model. One will be taken directly from the research conducted in Huang and Belongie [3], which is done to verify baseline performance for our architecture. The second loss function will incorporate the SSIM between the original and generated image. We will then compare the performance and output of the two models to analyze the effects of the modified loss function.

A. Unmodified Loss Function

The unmodified loss function taken from Huang and Belongie will consist of separate components for content loss and style loss. Content loss, denoted by L_c will be the euclidean distance between the $relu4\_1$ layer of the encoded generated stylized image and the corresponding $relu4\_1$ layer of the

encoded content as the target. While the original paper uses the output of AdaIN as the content loss target, the feature map from the original content image would make a better target for the detail preserving purposes of this paper. The second component will be the style loss, L_s scaled by some λ, which will effect how strongly the model learns to apply styles. To calculate L_s, the style and generated images will be passed to the VGG-19 encoder, where the intermittent feature maps of the encoded generated image: $relu1\_1$, $relu2\_1$, $relu3\_1$, $relu4\_1$, will be compared with the corresponding feature maps of the encoded stylized image, using euclidean distance. L_s will be the summation of all four mean squared errors [3]. The implementation for the original loss function was borrowed from the same Github implementation as the AdaIN function [8].

$$L_c = ||f(g(L_{cs})) - L_{cs}||_2 \tag{4}$$

$$\begin{aligned} L_s = \sum_{i=1}^{L} &||\mu(\phi_i(g(L_{cs}))) \\ &- \mu(\phi_i(s))||_2 + \sum_{i=1}^{L} ||\sigma(\phi_i(g(L_{cs}))) - \sigma(\phi_i(s))||_2 \end{aligned} \tag{5}$$

$$L = L_c + \lambda * L_s \tag{6}$$

B. Modified Loss Function

The proposed modified loss function is based on the loss function of Huang and Belongie [3], retaining the same methods of calculating the loss of style and content. However, to ensure that the content of the generated image is not overly altered in the style transfer process without increasing the complexity of the model, the SSIM between the generated image and the input is also calculated and added to the total loss. SSIM is a measurement of how similar two images are, it outputs between 0 and 1. More similar images score closer to 1, and less similar images score closer to 0. However, SSIM is not invariant to luminance and contrast, which are properties that we do not necessarily wish to retain from the content image. Therefore, to only calculate the similarity between the edges of the content and generated images, the gradient magnitude of the generated image and content image are calculated, and passed to the SSIM loss function. To get the gradient magnitude of the images, we define two Sobel filter kernels in both the x and y direction, applying them to the images gives us I_x and I_y, which are the directional gradients of the image. The gradient magnitude is then equal to $\sqrt{I_x^2 - I_y^2}$. To allow for increased control over how much the SSIM loss effects the weight updates, we scale the loss factor by a predefined λ_{ssim}. The SSIM loss function used in this paper is a builtin function from the Pytorch library, and the result is subtracted from one, as we want to minimize the value.

$$L_{ssim} = 1 - ssim(GM(g(F_{cs})), GM(c)) \tag{7}$$

$$L = L_c + \lambda_s * L_s + \lambda_{ssim} * L_{ssim} \tag{8}$$

V. Setup

A. Training

Models were trained on a 4060 ti 16gb GPU for 100 epochs on a random subset of 7000 images from the MS-COCO [9] dataset for content images, and the Wikiart [10] dataset for style images. While multiple batch sizes were attempted, batch sizes greater than 16 resulted in an out of memory error, and increasing the batch sizes at all resulted in significant slowdown during training. Therefore, the batch sizes were set to one, with a learning rate of 0.0001. Before being passed to the model, all images are resized to 512x512, and randomly cropped to 256x256. This is partially to maintain consistency with past experiments with style transfer, however it does reduce training overhead and helps homogenize the dataset. The random cropping also helps reduce the chances of the model overfitting due to seeing the exact same content and style images each epoch.

B. Testing

For testing, both models were frozen and given the same subset of 5 random image pairs to perform style transfer on, outputting and graphing all 25 possible combinations of style transfer, for subjective analysis of performance. Then, both models passed the same random 1,000 image pairs from the dataset. This phase of testing is used to gather hard metrics for evaluation, where we calculate the average SSIM and MSE between the content image and the generated image, and the average time it takes to stylize one image.

VI. Results

A. Model Performance Metrics

Model Performance		
	No SSIM Model	SSIM Model
SSIM	0.324	0.381
MSE	0.07	0.074
Time(sec)	0.033	0.033

Fig. 5: Results, No SSIM Loss

Fig. 6: Results, SSIM Loss

(a) Content Image With Style Image

(b) SSIM Loss Output

(c) No SSIM Loss Output

Fig. 7: Close Examination

(a) Content Image With Style Image

(b) SSIM Loss Output

(c) No SSIM Loss Output

Fig. 8: Poor Results

VII. Conclusion

A. Subjective Analysis

Both models appear to apply the style images fairly well, while there is room for improvement that could potentially be achieved through more training and modified parameters, at a base level the models do apply a style similar to the input style to the content image. Both models also appear to retain image finer details. Looking at figure 7, we see that both models are able to retain important spatial and structural information, such as the reflection of the banana in the sunglasses, and the car in the background. However, the model that does not utilize SSIM in it's loss function has a non-negligible amount of artifacts, which are not present in the SSIM Loss Output. This is likely due to the artifacts appearing in the gradient magnitude of the generated images, but not in the original content image, resulting in a high SSIM score, and then a higher total loss. This would have encouraged the model to learn a combination of weights for the decoder that did not result in as many artifacts.

B. Poor Results

Of course, not every style that was tested was successfully applied to the content. Directing your attention to figure 8, you will see that in this specific case the model did not apply the style successfully. This is of course, subjective, but the resulting image in both cases is primarily green, while the "style" of the input image is some variation of pointillism. The consistent issue across both models with different loss function indicates that the issue is likely with how the model applies the style rather than how the model calculates loss. AdaIN is likely having trouble aligning the statistics of the style and content loss features in some cases.

C. Objective Analysis

In terms of objective statistical results, the SSIM loss model only performed slightly better than the model without SSIM loss. The SSIM model resulted in an average 17.59% increase in SSIM, and had a very small increase in average MSE. Even more importantly, both models were able to infer the stylized output in an average of 0.033 seconds. Showing that the improved performance did not come at the cost of overhead.

D. Final Thoughts and Future Work

The subjective and objective analysis of our results show that not only does our SSIM loss model appear to reduce the number of artifacts seen in the model without SSIM loss, but it also does it without additional computational complexity at inference time. Future work would likely include experimentation with more combinations of lambda values during training, as well as training with a larger dataset and larger batch sizes. Modifying the architecture to pass the style feature maps to it's own dedicated CBAM could also be worth testing, as it may increase the attention given to the style features and help reduce poor results like in figure 8.

REFERENCES

[1] L. A. Gatys, A. S. Ecker and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 2414-2423, doi: 10.1109/CVPR.2016.265. ICCV.2017.167.

[2] H. Xin and L. Li, "Arbitrary Style Transfer With Fused Convolutional Block Attention Modules," in IEEE Access, vol. 11, pp. 44977-44988, 2023, doi: 10.1109/ACCESS.2023.3273949.

[3] X. Huang and S. Belongie, "Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization," 2017 IEEE International Conference on Computer Vision (ICCV), Venice, Italy, 2017, pp. 1510-1519, doi: 10.1109/ICCV.2017.167.

[4] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "CBAM: Convolutional Block Attention Module," arXiv.org, Jul. 17, 2018. https://arxiv.org/abs/1807.06521

[5] S. Liu et al., "AdaAttN: Revisit Attention Mechanism in Arbitrary Neural Style Transfer," 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 2021, pp. 6629-6638, doi: 10.1109/ICCV48922.2021.00658.

[6] "vgg19 — Torchvision main documentation." https://pytorch.org/vision/main/models/generated/torchvision.models.vgg19.

[7] Jongchan, "GitHub - Jongchan/attention-module: Official PyTorch code for 'BAM: Bottleneck Attention Module (BMVC2018)' and 'CBAM: Convolutional Block Attention Module (ECCV2018),'" GitHub. https://github.com/Jongchan/attention-module

[8] Naoto, "GitHub - naoto0804/pytorch-AdaIN: Unofficial pytorch implementation of 'Arbitrary Style Transfer in Real-time with Adaptive Instance Normalization' [Huang+, ICCV2017]," GitHub. https://github.com/naoto0804/pytorch-AdaIN

[9] "COCO Dataset 2017," Kaggle, Mar. 18, 2023. https://www.kaggle.com/datasets/sabahesaraki/2017-2017

[10] "WikiArt," Kaggle, Sep. 14, 2022. https://www.kaggle.com/datasets/steubk/wikiart