# Deep Style **Link to Git Repo, Team Name : SVM**

**Srivaishnavi Sree Krishnan**
ssreekri@ucsd.edu
A53283562

**Shrinidhi Venkatakrishnan**
s3venkat@ucsd.edu
A53272432

**Manasa Kumar**
m5kumar@ucsd.edu
A53279839

## Abstract

When humans and machines collaborate, we can produce things neither would create on their own. The intersection of art and AI is an area that renders the human's artistic abilities via a machine. In today's world the portraits created by the artificial intelligence algorithm has sold for more than 432,500 dollars. In this project, we aim to build a model using Convolutional Neural Networks that can separate and recombine content and style of different images to form one resulting image of a different style but retaining it's original content. This further can be extended for high level image synthesis and manipulation.

## 1 Introduction

Art Style is the manner in which the artist portrays his or her subject matter and how they expresses their vision. Style is determined by the characteristics that describe the artwork, such as the way the artist employs form, color, and composition, to name just a few. It is astounding to note that computers using efficient algorithm will be able to render different styles to a photograph that has taken a human years to master. The first major step in this field was introduced in the paper 'A Neural Algorithm of Artistic Style' in September 2015 [1]. Gatys et. al showed that the task of fetching the style from one image to the content of another can be posed as an optimization problem which can be solved through training a Convolutional Neural Network. For each style transfer that we want to generate in this model, we need to solve a new optimization problem, which makes it unsuitable for real-time applications. The paper, titled 'Perceptual Losses for Real-Time Style Transfer and Super-Resolution' by Johnson et. al [2], shows that it is possible to train a neural network to apply a single style to any given content image with a single forward pass through the network, using the loss network suggested in [1]. In our project, we aim at improving and experimenting with the visual quality of this image synthesis by modifying the style and content factors from the input images.

## 2 Model Description

### 2.1 Neural Style Transfer

The fundamental idea behind the architecture of Convolutional Neural Networks (CNN) is that object features become more explicit as the network gets deeper. That is, reconstructions from the lower layers reproduce the exact pixel values of the original image whereas the deeper layers emphasize more on the content. To get the style information of the input image, we use a feature space designed to capture texture information. This feature space is built on top of the filter responses in each layer of the network by taking the correlation of the feature maps in that particular layer by forming the Gram Matrix. It basically consists of the correlations between the different filter responses over the spatial extent of the feature maps. The key finding of this paper is that the representations of content and style in the CNNs are separable hence can be modified to create the desired image. A white noise image is passed onto the network as the input and its content and style loss is minimised with respect to every layer. The pixels of the original image is then updated based on the calculated loss function.

#### 2.1.1 Architecture

The architecture consists of 16 convolutional and 5 pooling layers of the 19-layer VGG network [6]. The feature space provided by a normalised version of the VGG-19 layers were used. The normalization of the network is done by scaling

the weights so that the mean activation of each convolutional filter over images and positions is equal to one. The activation function used over the feature map is ReLU. No fully connected layer is used to get the results.

The specialised network for the particular application of style transfer used is as shown in [1]. The input white noise image is taken and then modified after every forward pass in the network.

#### 2.1.2 Algorithm

Let $\vec{x}$ be the input white noise image and $\vec{p}$ be the content image and $\vec{a}$ be the reference style image. Let $G^l$ and $A^l$ be the Gram matrices of $\vec{a}$ and $\vec{x}$ respectively at layer l formed by the correlation of their respective feature maps. The contribution of the style image at layer l to the total loss is given by the following equation.

$$\frac{1}{4 \times N_l^2 \times M_l^2} \times \sum_{i,j} (G_{i,j}^l - A_{i,j}^l)^2 = E_l \tag{1}$$

And the total style loss is given by the following equation, where $w_l$ are weighting factors of the contribution of each layer to the total loss.

$$L_{style} = \sum_{l=0}^{L} (w_l \times E_l) \tag{2}$$

The loss for the content between the white noise and the content image is given by the following equation, where $F_l$ and $P_l$ are the features of the computed image and the content image respectively.

$$L_s = \sum (F_l - P_l)^2 \tag{3}$$

The total loss is given by the following expression, where $\alpha$ and $\beta$ denote the weights for style and content loss respectively:

$$L_{total} = (\alpha \times L_{style}) + (\beta \times L_{content})$$

The update for the white noise image after every forward pass is given through gradient descent as:

$$\vec{x} := \vec{x} - \lambda \frac{\partial L_{Total}}{\partial \vec{x}} \tag{4}$$

### 2.2 Real-time style transfer

As mentioned above, the original algorithm proposed by Gatys et. al formulated a loss function consisting of a style loss and content loss for style transfer and reduced the problem down to one of optimizing this perceptual loss function alone. Johnson et. al show that, restricting ourselves to a single style image for many content images can help us train a neural network that solves the same perceptual optimization problem at a much higher speed in real-time.

#### 2.2.1 Architecture

The architecture proposed in the paper consists of two networks, namely: (i) Transformation Network and (ii) Loss Network. We first train the transformation network to transform images into output images. The loss network we use is a pre-trained net used for image classification (such as VGG-19) to define perceptual loss functions. The loss network remains fixed during the training process.

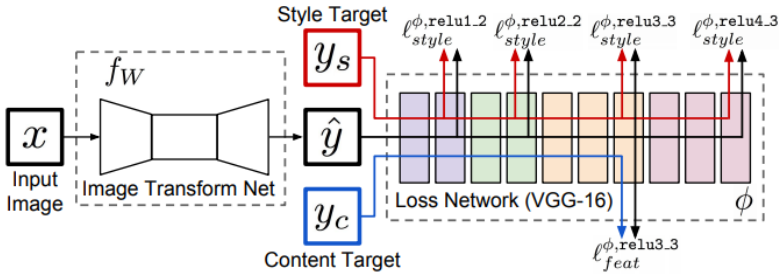


Figure 1: System Overview (Justin Johnson et. al 2016)

As shown in Fig. 1, training the transformation network involves using the pre-trained loss network to compute the loss for a given training example and then propagating back the error through every layer in the image transformation network. At each layer we compute the gradient of the weights with respect to the loss function and use it to make small changes to the weights in the negative direction of the gradient. The technique we use during this back propagation is called gradient descent, and this will improve the performance iteratively by improving the network weights until the loss is below acceptable threshold.

#### 2.2.2 Algorithm

The image transformation network consists of 3 layers of convolution and ReLU non-linearity, 5 residual blocks, 3 Upsampling layers and finally a non-linear tanh layer which produces an output image. Convolution layers are used to apply filters to an input image, by moving a fixed-size kernel filter across the input image and applying the convolution operation to the pixels within the kernel window to compute each corresponding pixel in the output image. After applying both stride 2 convolution layers our image resolution is reduced to $\frac{1}{4}$ of it's original size. Mean, zero, duplication,reflection, and symmetric padding are all valid and widely used methods of padding for convolutional layers. In our opinion, there is no padding that is better performed than the other for all cases. Intuitively, we think that reflection and symmetric padding alters the local image structure and global statistics the least and adds certain plausible values as padding from the edges rather than zero padding which adds extra unrelated information to the input[8]. The desired output of our transformation network is a styled image with the same resolution as the original content image.

In order to achieve this, two convolution layers with stride $\frac{1}{2}$ had been introduced in the original implementation of the Real-time Style Transfer. These are referred to as transpose convolution layers or deconvolution layers. These layers (in the form of stride) were introduced to increase the receptive field size of the kernel filter in the layers that follow. However, the deconvolution can easily have "uneven overlap," putting more of the metaphorical paint or image intensity in some places than others, creating a checkerboard like pattern [3].

To avoid these artifacts, we have replaced the traditional deconvolution layer with an Upsampling Layer. In this layer, we interpolate the input feature map (with methods such as nearest neighbour or bilinear interpolation) and then perform the operation of a convolution layer.

Between the downsampling and upsampling layers we have 5 residual layers. These are stride 1 convolution layers but input of the network being added back to the generated output. These layers make it easier to train the network for identity output. Every sampling layer is followed by a Batch Normalization operation applied on the resultant feature maps, that reduce the internal covariance shift, avoids exploding or vanishing gradients and thereby speeds up the training process. In our project, we have experimented with another normalization technique like Instance Normalization to improve the stylization effect on the image.

The Transformation network parameterized by weights $W$ transforms input images $x$ into output images $\hat{y}$ via the mapping $\hat{y} = f_W(x)$. Each loss function computes a scalar value $l_i(\hat{y}, y_i)$ measuring the difference between the output image $\hat{y}$ and a target image $y_i$ and is trained using stochastic gradient descent to minimize the loss between our generated output image and our target content and style images. This loss is calculated in the same way as the previous model in section 2.1, taking that model to be the loss network. As we reconstruct from higher layers, image content and overall spatial structure are preserved, but color, texture, and exact shape are not [2]. To maintain the perceptual visual quality and also achieve the styled version of the image we take the loss from the second layer of the network.

## 3 Experimental Settings and Results

### 3.1 Neural Style Transfer

#### 3.1.1 Baseline model

The algorithm and model as described in section 2.2 was implemented. More specifically, following the setup in [1], the ratio $\alpha$ to $\beta$ was set to $10^{-3}$ . The content representation on layer conv4_2 and the style representations on layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 of the pretrained VGG-19 network were weighted with 0.20. The optimizer used with gradient descent was LBFGS [7]. Some examples of style transfer for one content image with different style images are shown in Fig. 2. The variation of the total loss, content loss and style loss with epochs are as shown in Fig. 3. We can see that the total loss which is the sum of content and style loss decreases steadily with the number of epochs.

### 3.1.2 Initialize gradient descent with content image

Instead of initializing the gradient descent with a noisy image, the experiment was done by initializing it with the content image. This was to get a clear picture of how the gradient descent updates the pixel values based on the style transfer loss. We can clearly see more and more style getting added to the output image with increasing number of epochs. The results obtained were as shown in Fig. 4.

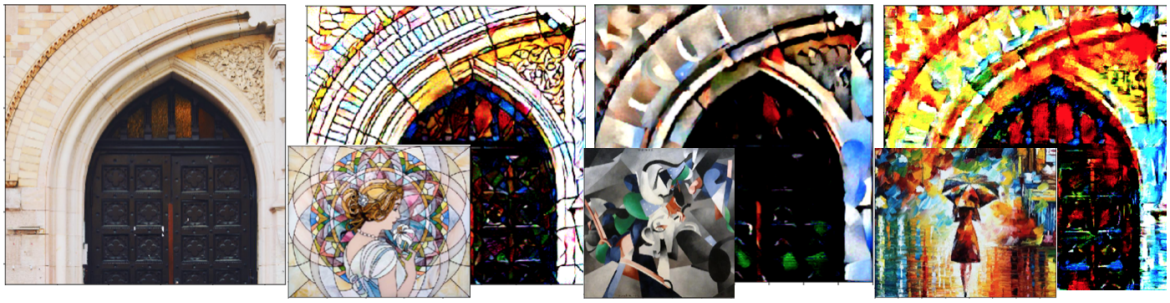

Figure 2: Baseline results for Neural Style Transfer

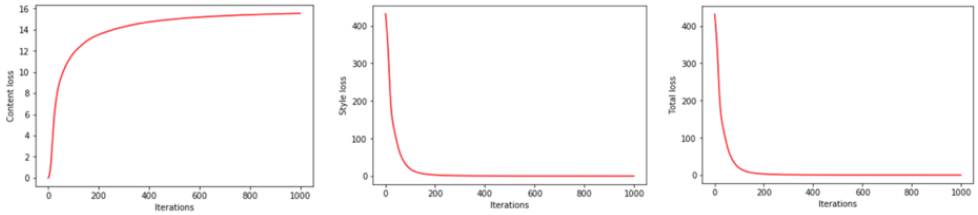


Figure 3: Content, Style and Total loss variation with number of epochs


Figure 4: Initializing with content image. Base images, output at 100, 200 and 500 epochs from left to right.

Figure 5: Left side corresponds to a higher value of alpha, and the right side to a higher value of beta.

### 3.1.3 Content Style Trade Off

The values of alpha and beta were varied from (1, $10^{-5}$) to ($10^{-5}$, 1) and the resulting images obtained in order were as shown in Fig. 5. Higher value of $\alpha$ which corresponds to giving more weight to content loss retains more content as opposed to style. Higher value of $\beta$ which corresponds to giving more weight to style loss retains more style as opposed to content.

### 3.1.4 Weighting layers in VGG-19

In the baseline implementation, the content representation on layer conv4_2 and the style representations on layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 were weighted with 0.20. In this experiment, the layers selection and weights were engineered to capture just pixel wise features, all the way to more content specific features. First, just the conv1_1 layer's features were considered, then conv1_1 and conv2_1 layers' features were taken, and so on until all the layers' features were considered.

The results in order from left to right are as shown in Fig. 6. Analysing this, it can be observed that when just conv1_1 layer features were considered, the image obtained is more grainy as the lower layers of the VGG net encode more pixel rich details. When all the layers were considered, there is a higher blend of content and style, as the final layers of the VGG net encode more content rich details.

Figure 6: Feature extraction from lower to deeper layers of VGG-19.

### 3.1.5 Feature extractors

In place of the VGG-19 features, Resnet[8], AlexNet[9] and VGG-16 features were used with the same algorithm. The resultant images in order were obtained as shown in Fig. 7.

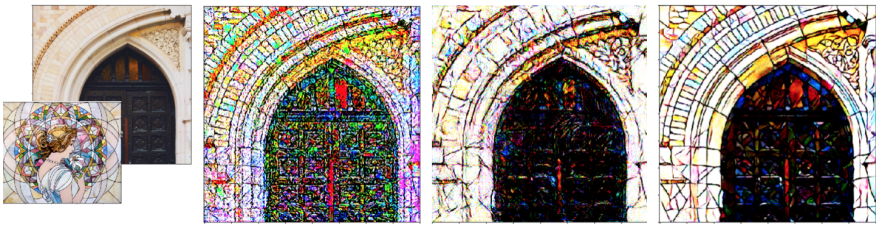

Figure 7: ResNet, AlexNet and VGG-16 feature extractors from left to right.

In comparison to VGG nets, AlexNet retains more unrelated background information in last convolutional layer, which often disturbs the final prediction. Upon inspection of the Resnet features, they were found to be more specific and less general as compared to VGG features. We can see from Fig. 7 that VGG-19 performs better than the other three for the same set of hyper parameters.

### 3.1.6 Pooling Layers

The Max Pooling layers in the VGG-19 net were replaced with Average Pooling layers while extracting the features alone, and not during the training phase. The results obtained were as shown in Fig. 8. Max pooling extracts the more extreme features like edges. Average pooling takes everything into account and outputs an average value. In general, average pooling extracts more smooth features. Hence, Average Pooling performs better than Max Pooling for Neural Style Transfer.

### 3.1.7 Different Optimizers for gradient descent

The experiments were run with fixing the learning rate, alpha-beta values, content and style images for the same number of epochs with different optimizers such as SGD, Adagrad, Adam and LBFGS. SGD and Adagrad had a hard time minimising the loss as they oscillated more with increasing number of epochs. Adam and L-BFGS converged faster and in a more stable manner and had almost the same error. L-BFGS performed the best, as indicated in [1]. The results obtained were as shown in Fig. 9. We can see that for the same number of epochs, there is a much higehr blend of content and style for L-BFGS.

### 3.1.8 Scaling Style image

By resizing the style image before extracting features, the types of artistic features transfered can be controlled. The style images were scaled to fixed percent of it's original dimensions before running the algorithm using nearest neighbor interpolation. This resulted in different styles as shown in Fig. 10. This intuitively follows from the fact that the richness of the style changes when the images are scaled.

### 3.1.9 Rotating the Style image

The style images were rotated before running the algorithm. This resulted in different styles captured in the mixed image as shown in Fig. 11. This follows from the fact that when you rotate style images, important style feature such as direction of strokes change. This could be considered as a whole different style of its own.

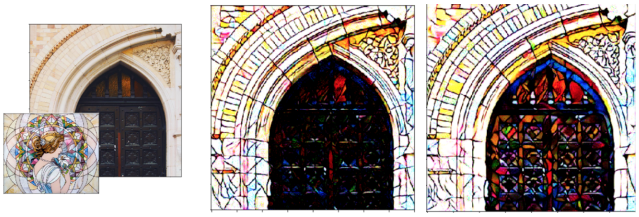

Figure 8: Max Pool and Average Pool from left to right.

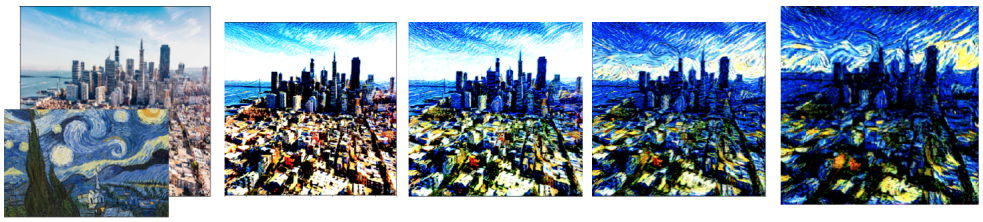

Figure 9: SGD, Adagrad, Adam, L-BFGS from left to right

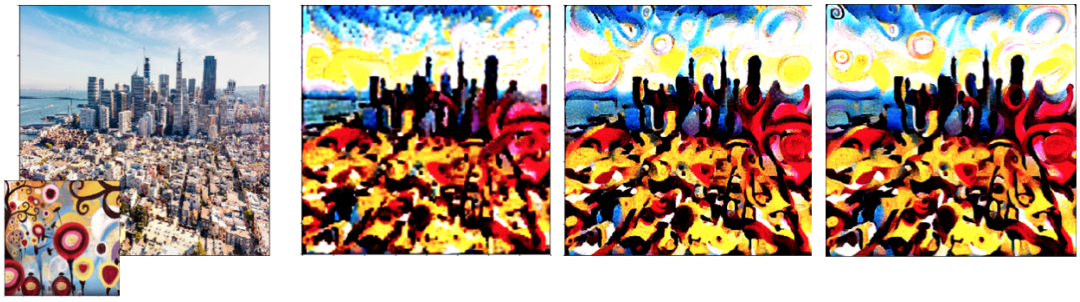

Figure 10: Base images, 10, 30, 60 % of the original style images' dimensions from left to right.

### 3.1.10 Multiple Style Images

Instead of finding the gram matrix and transferring features for a single style image, it is done for multiple style images and this is embedded into the style loss function. Now, this combined style loss function with a single content image's content loss function is considered. This incorporates style from multiple images as shown in Fig. 12.

### 3.1.11 Style Interpolation

In this experiment, we consider multiple style images and we also control which style image we want to give more importance to. This is done by interpolation of the different style losses contributed by the style images such that their interpolation coefficients sum to 1. In Fig. 13, we can observe the varying importance given to the style images.

### 3.1.12 Video

Along with the challenge of producing visually appealing and pleasing styled images, stylizing video frames in real time defines additional constraints on the speed of the neural network. As mentioned previously, the style transfer algorithm suggested by Gatys et. al involves solving a new optimization problem for every pair of style and content image and thus, the process takes approximately 5 minutes to generate in the DSML GPU server. Thus, the method can't be used for real-time applications.

Figure 11: Original, 30, 60, 90 degrees rotation of style image from left to right.

Figure 12: Two style images with a single content image.

Figure 13: Interpolation of style images.

Also, when going from one frame to the next, there isn't much smoothness in the transition as you can see the style features popping up in the present frame irrespective of its presence in the previous frame. This can be seen in Fig. 14. The circular style patterns from the style image is present in the background of the middle image (to the right most corner) even though it isn't present in the previous image. This is also known as 'popping effect'.

### 3.2 Real-Time Style Transfer

#### 3.2.1 ResNet vs VGG

The core idea of ResNet is introducing an identity shortcut connection that skips one or more layers. But for our application we don't need the number of layers to be very deep though it may or may not reduce the total loss function. In VGG it is easier to visualise and change the features of the layers as it is lesser when compared to the Resnets. After experimentation with different feature extractors such as Resnets, AlexNets and VGG Nets, as in section 3.1.5, VGG-19 feature extractors gave the best results for Real-Time Style Transfer. Hence, We have used VGG19 pretrained features instead of Resnets for implementing our final versions.

#### 3.2.2 Baseline model

We train style transfer network initially on the MS-COCO dataset [5] with a batch size of 4 for 40K iterations, giving roughly two epochs over the training data, adhering to the specifications outlined in [2]. The resultant mixed image obtained is as shown in Fig. 15.

#### 3.2.3 Improving the Normalization

It was observed that using just Batch Normalization after convolution layers in the transformation net gives a fading effect as it preserves the contrast of the content image in the stylized output image. However, we would like to design the style loss to transfer elements from a style image to the content image such that the contrast of the stylized image is

Figure 14: Popping effect in video style transfer.

Figure 15: Baseline model results for Real time style transfer.

similar to the contrast of the style image. Thus, we would have to replace the batch normalization with certain operation that would combine the effects of instance-specific normalization and batch normalization. Instance Normalization, as suggested by Ulyanov et. al [4] was applied after every convolution layer to modify the Transformation net such that it discards contrast information in the content image.

Let $x_{jijk}$ denote its $tijk$-th element, where $k$ and $j$ span spatial dimensions, $i$ is the feature channel, and $t$ is the index of the image in the batch. Then, the output after Instance Normalization is given by $y_{jijk}$ as:

$$y_{jijk} = \frac{x_{jijk} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}, \mu_i = \frac{1}{MNT}\sum_{t,l,m} x_{tilm}, \sigma_{ti}^2 = \frac{1}{MN}\sum_{l,m}(x_{tilm} - \mu_i)^2 \tag{5}$$

The comparative results obtained are given in Fig. 16 for starry-night style image and the amber content image. From this, it is clear that instance norm is visually more appealing than batch normalisation for the same content image. Hence, we will be using instance norm for all further experimentation.

Figure 16: Comparison between instance normalisation and batch normalisation.

### 3.2.4 Video Style Transfer

The second implementation is being trained for a particular style image and the styling takes place almost instantly, in 10ms. We see that the popping effect which was observed in the Neural Style Transfer method is removed while using Real Time Style Transfer. From this, especially for real time applications, the Real-Time Style Transfer method is much more suitable than Neural Style Transfer approach.

Figure 17: Video Style Transfer

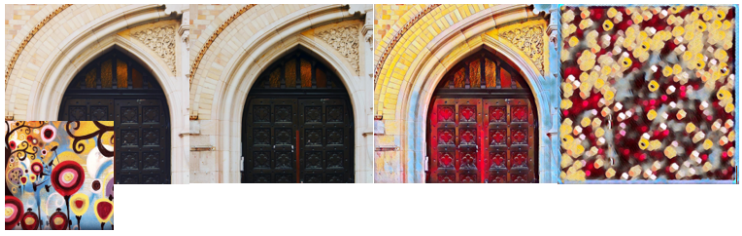

Figure 18: Results of increasing the [alpha/beta] ration.

### 3.2.5 Content Style Trade-off

Alpha and Beta values determine the percentage of importance given to the style and content during the training process. When the alpha value is greater than the beta value, the style is given more importance than the content and vice versa. The alpha beta parameters also affect the overall absolute value of the loss function. This also affects the visual perception of the image. The alpha and beta values were varied and the results are shown in Fig. 18. It depict the following; high content low style to high content less style. Here the alpha by beta value increases.

$$L_{total} = (\alpha \times L_{style}) + (\beta \times L_{content}) \tag{6}$$

## 4 Discussion and Conclusion

In this project, we implemented two major contributions towards Style Transfer in recent years. We also experimented with various possible modifications that could be applied to these algorithms, as mentioned in Section 3.

There were two key challenges that we faced: The first one was the trade off between the content and the style in the resultant image. We lost content when we tried to increase the style of our test image. The optimal value was found after experimenting with different weights for content and style loss. Hence, this could be a potential direction for the future.

The second major challenge is that most of the evaluations of neural style transfer algorithms are qualitative, and the most common method of evaluation would be a user-study. Though comparing the final value of loss function is a possible metric, the values of loss function do not always correspond precisely to the quality of the output image.

For the future extension of this work, we could try to retain the color of the original content image and change just the texture and style based on the style image which was used for training. Certain other loss functions like pixel based loss function and Total Variation Regularization can also be considered to give smoothness to the image. A simultaneous implementation of deep dream and style transfer can be done by minimising the loss functions of both at once. Apart from these, many more experiments could be performed to explore style transfer better.//

### Acknowledgments

We would like to express our sincere thanks to our Professor. Charles Deledalle who facilitated us in implementing this project.

### References

[1] 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, pages 2414–2423, 2016*

[2] Justin Johnson, Alexandre Alahi, Li Fei-Fei."Perceptual Losses for Real-Time Style Transfer and Super-Resolution."*European Conference on Computer Vision, 2016*

[3] Odena, et al., "Deconvolution and Checkerboard Artifacts", Distill, 2016. http://doi.org/10.23915/distill.00003

[4] Ulyanov, Dmitry et al. “Instance Normalization: The Missing Ingredient for Fast Stylization.” CoRR abs/1607.08022 (2016)

[5] Lin TY. et al. (2014) Microsoft COCO: Common Objects in Context. In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8693. Springer, Cham

[6] Karen Simonyan,Andrew Zisserman *Very deep convolutional networks for large scale image recognition*

[7] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, Ping Tak Peter Tang *A Progressive Batching L-BFGS Method for Machine Learning*

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778"Deep Residual Learning for Image Recognition"28 Nov 2018

[9] Alex Krizhevsky , Ilya Sutskever , Geoffrey E. Hinton, ImageNet classification with deep convolutional neural networks, Proceedings of the 25th International Conference on Neural Information Processing Systems, p.1097-1105, December 03-06, 2012, Lake Tahoe, Nevada