# A Neural Algorithm of Artistic Style: An Investigation

E4040.2018Fall.STYL.report
Aditya Sinha as5624, Praneet Vibhu Balabolu pb2735, Vinay Raghavan vsr2119
*Columbia University*

## Abstract

*In this work, we use a VGG-19 Deep Neural Network to perform style transfer from a given style image and content image to a target image, thus creating artistic renderings of the scene based on the work by Gatys et al. [1]. Furthermore, we investigate the dependence of the output image on various network parameters. This system creates a neural representation of style and content, thus providing us with a possible algorithmic understanding of how humans perceive artistic images. The main challenge of this is identifying the style and content layers, and then performing enough style transfer to make it psycho-visually appealing, which requires tuning of the style and content weights. We were able to obtain artistic images of high perceptual quality, albeit slightly different from those presented in [1], due to differences in initialization and training parameters.*

## 1. Introduction

With rising computational power, use of Deep Convolutional Neural networks have become more prevalent for image classification tasks [2]. Among these, VGG-19 is a very popular architecture [3], which is what we employ to intelligently transfer the style of a painting to the content image. Using the VGG-19 network pre-trained on the ImageNet dataset, we deconstruct the content and style components of the two images, and then feed this information back to a target image.

The main challenge with this task is to identify the layers in VGG-19 whose activations would optimally encode the content and style information. Furthermore, it is important to ensure that the style features do not contain any content information, so as to not mix the contents of the two images. The original paper tackles the first problem by backpropagating the loss from several layers and seeing what information they encode, and then assessing the best representations of content and style. To solve the latter problem, we take the spatial correlation of the style layer outputs to remove the content information, the details of which are discussed in the methods section. Another challenge is the careful choice of various parameters to make the resulting artistic image visually satisfying, which is a very subjective standard, since we don't have any measure of perceptual image quality that we can optimize.

## 2. Summary of the Original Paper

In this section, we will discuss the methodology employed by Gatys, et al. to perform neural style transfer, as well as their key findings that contribute to the fields of computational imagery and visual perception.

### 2.1 Methodology of the Original Paper

Here, we present the architecture employed and the algorithm used in the original paper.

The original work employs a pre-trained VGG-19 architecture. It first feeds the style image through the network and takes the output activations of the layers conv1_1, conv2_1, conv3_1, conv4_1 & conv5_1 and then takes their spatial correlation to get style representations. Next, it feeds the content image and takes the output activation from content layer conv4_2. Finally, it initialises a target image with noise and uses these activations along with the calculated loss to iteratively backpropagate into the target image and modify it, thus obtaining the artistic image.

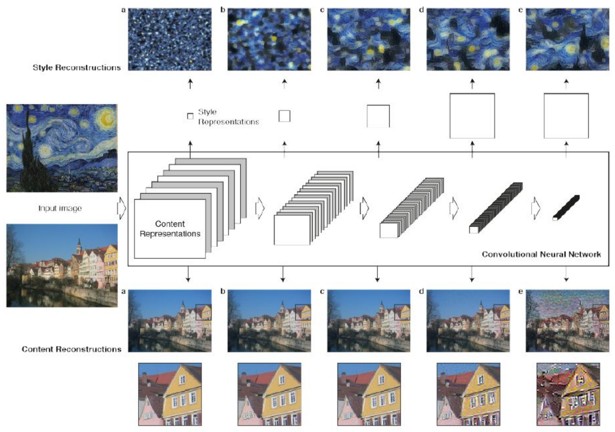

Figure 1. Content and style representations of input images at different scales in successive layers of VGG-19

In the above architecture and the corresponding style and content reconstructions, we see that successive layers correspond to more abstract reconstructions of the image content. Therefore, it makes sense to consider the higher layers for the content representation. For style, we need to identify colors and patterns (textures) at various levels (scales). For this, we take outputs at various levels and then discard any content information by taking the spatial correlation of the outputs (as Gram matrices).

## 2.2 Key Results of the Original Paper

The paper by Gatys et al. showed that in any image, style and content are separable parameters, and we can separate them using a simple image classification network like VGG-19. Here, they investigated reconstructions from different layers to see what information they represent. They found out that spatial correlations of layers conv1_1, conv2_1, conv3_1, conv4_1 and conv5_1 represent style while layer conv4_2 represents content. They went on to show that you can use these representations from different images to construct a visually-pleasing, artistically-styled image. Moving forward into possible future work, they have also hypothesized that this work represents a path forward to an algorithmic understanding of how humans perceive artistic imagery.

For detailed results of the paper and comparison with our own results, see Section 5.2: Comparison of Results

## 3. Methodology

In order to reproduce the results of Gatys, et al., we sought to use their approach of using a pre-trained network to extract the style and content features from their respective images as well as the target image, find the difference, and backpropagate the loss to the target image. Just like the original paper, we employed the VGG-19 network using weights pre-trained on ImageNet. Unlike the original paper, however, we implemented this whole architecture in Tensorflow, rather than the caffe-framework.

In the rest of the section, we will discuss the objectives of our investigation of neural style transfer, the challenges we faced in recreating and extending the work of the Gatys et al., the formulation of our objectives into an overall problem, and the design of our solution.

## 3.1. Objectives and Technical Challenges

The primary objectives of this project are to 1) understand the process used by Gatys et al. to produce artistic images of mixed style and content, 2) produce target images that perceptually resemble those produced by Gatys et al., and 3) experiment with different configurations and extensions of their design.

In pursuit of these objectives, we faced some challenges that resulted both from our lack of technical expertise and our uncertainty about some of the parameters used in the original paper. The architecture employed here is fundamentally different from our prior experience working with neural networks because it uses a pre-trained network with frozen weights and performs backpropagation into the input of the network. Therefore, we were unfamiliar with how to construct this architecture as described. This was ultimately overcome with lots of troubleshooting and the use of additional resources, such as [4], to understand how to go about constructing and using a pre-trained network.

Furthermore, some design choices shown in the original paper did not include sufficient support justifying their use. For example, the conv4_2 layer was selected as the content representation; however, content reconstructions were only shown for the conv1_1, conv2_1, conv3_1, conv4_1, and conv5_1 layers, most of which produced reasonable content reconstructions. Although a clear, intuitive argument was provided for why the higher layers of the network are better representations of content, an argument specifically for conv4_2 or against using multiple high-level layers was not provided. Similarly, equal weighting factors were used on the loss from each layer when calculating the style loss. Intuitively, this would cause local and global style elements to be represented equally; however no justification was provided regarding whether equal representation produces images that are perceptually optimal. Nevertheless, in the effort to produce target images that were most similar to those produced by Gatys et al., we retained their design selections.

## 3.2. Problem Formulation and Design

To accomplish our objectives, we adhered to the mathematical and conceptual formulation provided by Gatys et al. We first construct the VGG-19 network with weights pre-trained on ImageNet. We then pass the content image through the network and retrieve the content features, and pass the style image through the network and retrieve the style features. These features are stored in order to compute the loss, and the rest of the network activations are discarded. We then calculate the loss with respect to a variable input to the VGG-19 network, and optimize the input image to minimize this loss.

This loss function contains the most interesting parts of this project because it defines how style and content are each represented in the activations of the VGG-19 network and how to optimize the transfer of style and content to the target image.

The total loss, $L_{total}$, is given as the weighted sum of content loss, $L_{content}$, and style loss, $L_{style}$:

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

The weights on each of these types of loss depends on particular content and style images, but the ratio of $\alpha/\beta$ is generally either $10^{-3}$ or $10^{-4}$.

To calculate the content loss, we first need to understand how the content information is represented. Let each convolutional layer, l, be defined as containing $N_l$ distinct filters producing $N_l$ feature maps, each of size $M_l$, where $M_l$ is the height times the width of the feature map. Therefore, all of the activations of layer l can be stored as a matrix, $F^l \in \mathbb{R}^{NxM}$, where $F^l_{ij}$ is the activation of the ith filter at position j in layer l. These activation matrices, $F^l$, are how the content information is represented. If we let p and x be the content image and the target image, respectively, and $P^l$ and $F^l$ be their respective activations in layer l, we can define the content loss as the squared-error loss between the two content representations:

$$L_{content}(p,x,l) = \frac{1}{2} \sum_{i,j} (F^l_{ij} - P^l_{ij})^2$$

To calculate the style loss, we first need to understand how the style information is represented. The style information is represented by the texture of the image at multiple scales. In order to extract the texture at a given scale, we must compute the correlation between the different activations over the spatial extent of the feature map at the given layer because patterns that are correlated throughout the image will be representative of the overall texture of the image. For small scale textures, we use the correlation at a low level of the network, and for large scale textures, we use the correlation at a high level of the network. This combination of textures at multiple scales is how style is represented. Therefore, to compute the style loss, we must first compute the feature map correlations, which are given by the Gram matrix, $G^l \in \mathbb{R}^{NxN}$, where $G^l_{ij}$ is the inner product between the vectorized feature map i and j in layer l:

$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$

These Gram matrices, $G^l$, are how the style information is represented. If we let a and x be the style image and the target image respectively, and $A^l$ and $G^l$ be their respective Gram matrices at layer l, we can define the style loss at layer l as the normalized squared-error loss between the two style representations:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G^l_{ij} - A^l_{ij})^2$$

And the total style loss as the weighted sum of these over all layers:

$$L_{style}(a,x) = \sum_{l=0}^{L} w_l E_l$$

The values of the layer weights, $w_l$, are 0 for all layers except for the layers immediately following the pooling, where they are ⅕ (all equal and normalized to sum to 1).

In an attempt to extend the result of the original paper, our further experiments include a third loss term, called total variation loss. This term was included to encourage spatial smoothness in the target image. It functions by penalizing large differences between neighboring pixels in the target image. We used the anisotropic variation of total variation loss because it is differentiable and easier to minimize. With x as the target image, total variation loss, $L_{variation}$, is given as:

$$L_{variation} = \sum_{i,j} \sqrt{\left|x_{i+1,j} - x_{i,j}\right|^2} + \sqrt{\left|x_{i,j+1} - x_{i,j}\right|^2}$$

and the new total loss, $L_{total}$, is given as:

$$L_{total} = \alpha L_{content} + \beta L_{style} + \gamma L_{variation}$$

where $\gamma$ can be tuned to optimize for smoothness in the output, and was often matched with $\beta$ in our experiments.

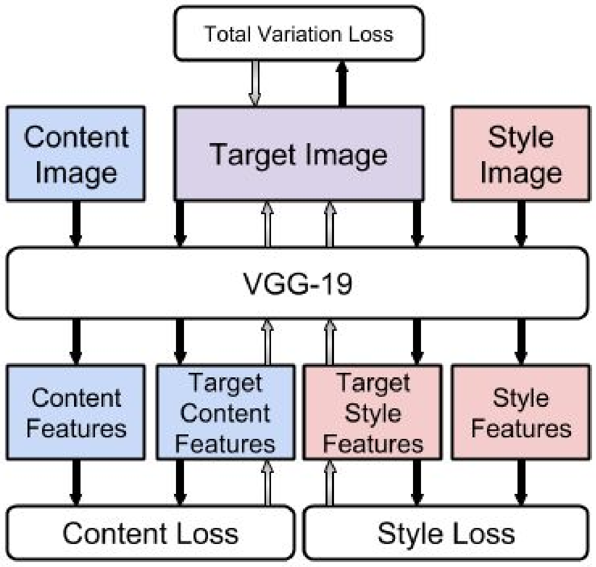

Figure 2. Block diagram of the training process with loss sources, including the optional total variation loss. Black lines represent forward passes and gray lines represent backpropagation of loss gradients. Only the target image is changed by backpropagation.

## 4. Implementation

In this section, we will discuss the implementation of this architecture in code. We will begin by discussing the implementation of the VGG-19 network, and follow that with the exact training process used during experimentation.

## 4.1. Deep Learning Network

The backbone of this project is the VGG-19 network pre-trained for image classification on the ImageNet dataset. In order to build VGG-19, we took the model details from [4]. There, we were able to view the layer configuration of the network and download the pretrained weights. To build the network and define all layer activations, we first built a list of network layers and iterated through the list, computing the activations at each layer and saving them in a dictionary with layer names as the keys and activations as the values. The computation of each activation was done by first parsing the layer name to determine what kind of layer we are on (convolution, ReLU, or pooling) and then computing the output of the layer by performing the corresponding operation. If the layer was a convolution layer, the corresponding kernel and bias was taken from the pretrained weights and used to perform the convolution.

The full architecture details for VGG-19 are as follows: all convolutions are performed using 3x3 kernels and all activation functions are rectified linear units (ReLUs). The first block of convolutions, conv1_1 and conv1_2 each contain 64 filters. The second block of convolutions, conv2_1 and conv2_2, each contain 128 filters. The third block of convolutions, conv3_1, conv3_2, conv3_3, and conv3_4, each contain 256 filters. Finally, both the fourth and fifth blocks of convolutions, conv4_1, conv4_2, conv4_3, conv4_4, conv5_1, conv5_2, conv5_3, and conv5_4 each contain 512 filters. The fully connected layers, fc_1, fc_2 and fc_3 contain 4096, 4096, and 1000 units respectively; however these layers were not used at all in this implementation because we only utilized the activations from the convolution layers. In standard VGG-19, max pooling is used for pooling, but Gatys et al. found better results using mean pooling due to improved gradient flow. We elected to use mean pooling but experimented with max pooling as well.

An important first step when using this network is the pre-processing of the images used as input. To preprocess the input, the mean pixel value from the training set must be subtracted from the input. This mean pixel value was contained with the pre-trained weights of the network. If this step is overlooked, the activations would not contain the information we would expect.

As opposed to the traditional VGG-19 network, the only trainable parameter in this network is the input, which was initialized as the content image, rather than noise, to speed up convergence. All weights were frozen, and the calculated loss, as described in section 3.2, was backpropagated to the input. To optimize the input, we employed the Adam optimizer with a learning rate of 10 over 200 iterations of training. Every 20 training iterations, we would compute the new total loss, and if it was less than the previous best total loss, the loss value would be saved as the new best and the target image at that training step would be saved to disk.

The Python notebook kernel takes negligible time to load and save the images. The code for loading the VGG-19 network, calculating the activations of the style and content images takes a short while, but it is a one-time overhead. The training of the network (training being a misnomer) was done on GCP using a K80 GPU and each iteration took about 3 seconds. However, when the network has to calculate the forward activation and loss of target image for the first time, it takes longer (~ 30s). In total, the whole process takes about 10 minutes to get the artistic image as the output.

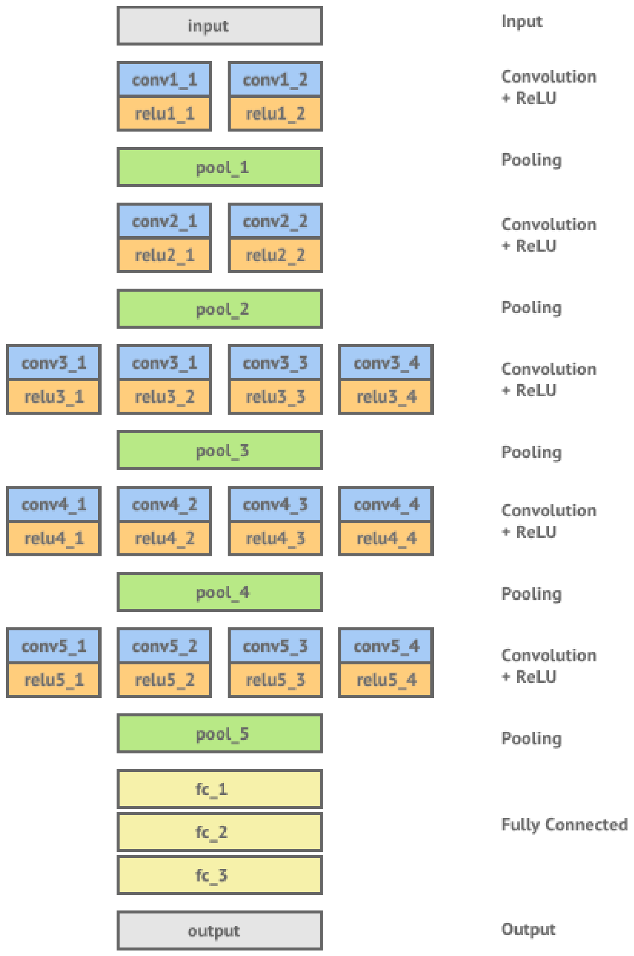

Figure 3. Block diagram of the full VGG-19 network, as found in [5]. For our purposes, the fully connected layers were not used. The conv1_1, conv2_1, conv3_1, conv4_1, and conv5_1 layer activations were used to extract style information, and the conv4_2 layer activation was used to extract content information.

## 4.2. Software Design

The code for this project was written using the Tensorflow package for Python. The code is divided into 4 main parts: a Python file with a VGG class that contains the VGG-19 model, the functions for computing the activations at each layer, and helper functions for preprocessing and unprocessing images (see section 4.1); a Python file with the total loss function and each of the individual loss functions (see section 3.2); a Python file with helper functions for reading and writing images; and a Jupyter Notebook to bring all of these files together, load the images, compute the content and style features, calculate the loss, and optimize the target image. We opted to create a class for the VGG-19 network because we needed to reuse the weights but with different network inputs, and a class was the simplest way to implement this. The loss and utilities were contained within functions, as only simple input-output relationships were needed. The training procedure was included directly in the Jupyter Notebook for easiest access to hyperparameters and for training transparency.

## 5. Results

In the end, we were able to achieve style transfer images that were of high artistic quality when the style images in [1] were applied to an original content image. When performing a direct comparison of our results with [1], we see that our result are different but still visually pleasing and more representative of the content in the image. In addition, we made hyperparameter modifications that change various stylistic qualities of the image for different visual effects.

## 5.1. Project Results

To determine the perceptual quality of the images produced by our neural style transfer model, we first attempted style transfer on a content image we selected using the style images from [1]. These results can be seen in figure 4. We believe that these results are very artistically and visually appealing. They all clearly show the original content image and have many of the features of the style images, with the exception of Image E, which captured some of the geometry of the style image but captured neither the colors nor the outlines.

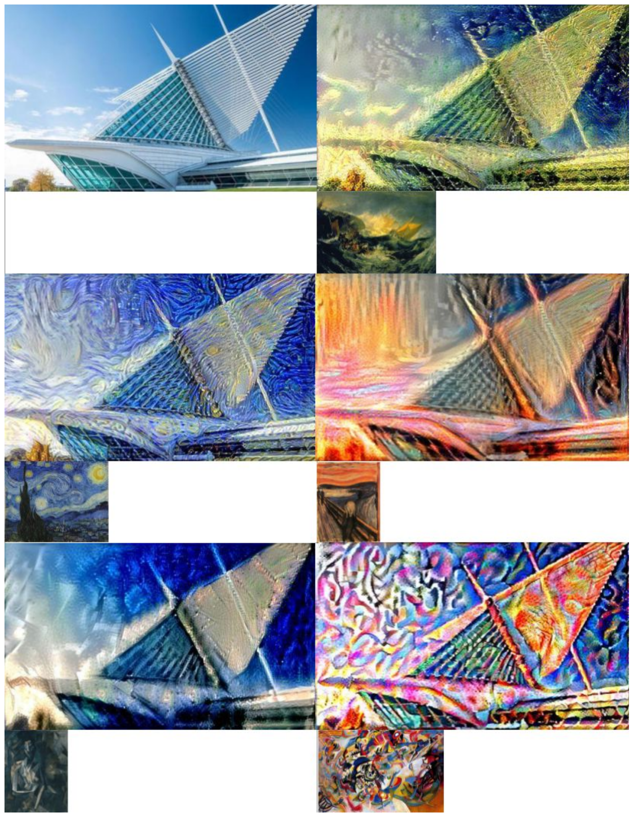

Figure 4. Style transfer results produced by our model on a photo of the Milwaukee Art Museum using the style images from Gatys et al. **Image A** is the original content image, Milwaukee Art Museum (Photo: TripExpert). The paintings that provided the style for the respective generated image is shown in the bottom left corner of each panel. **Image B** is styled using *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805. **Image C** is styled using *The Starry Night* by Vincent van Gogh, 1889. **Image D** is styled using *Der Schrei der Natur* by Edvard Munch, 1893. **Image E** is styled using *Figure dans un Fauteuil (Femme nue assise)* by Pablo Picasso, 1910. **Image F** is styled using *Composition VII* by Wassily Kandinsky, 1913.

## 5.2. Comparison of Results

In order to make the best comparison possible with [1], we used the same style images as they did. The original results from [1] can be seen in figure 5. We also present our results alongside in figure 6 for comparison.

.

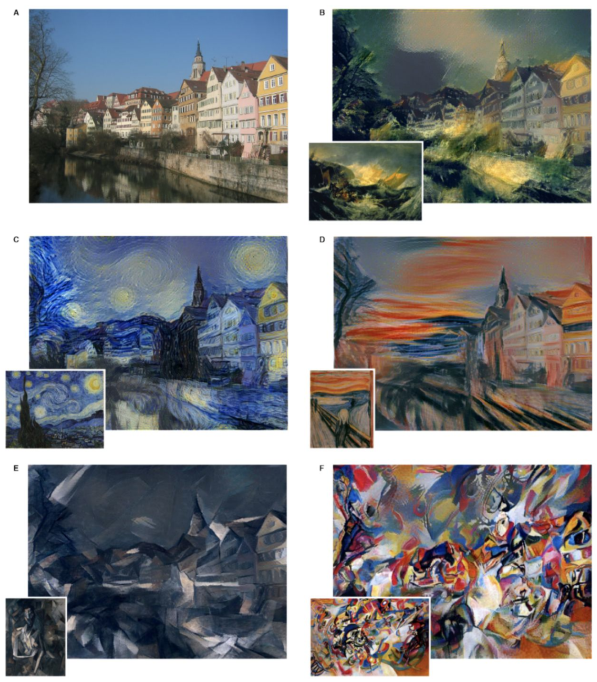

Figure 5. Original style transfer results from Gatys et al. **Image A** is the original content image, Neckarfront (Photo: Andreas Praefcke). The paintings that provided the style for the respective generated image is shown in the bottom left corner of each panel. They are the same style images as in figure 4.

Compared to the images in [1], our images are closer to the content images, while adding significant style features from the paintings. The results in the original paper had some unintended content from the style images. In our results however, the content from the style image is negligible, showing better separation of content and style. For example in figure 5.(c), we can clearly see the remnants of the yellow moon in the sky, while in figure 6.(c), the content is truer to the content image.

## 5.3. Discussion of Insights Gained

Our results were perceptually different from the results produced by [1]. Intuitively, this could be because we initialized the target image as the content image rather than simple Gaussian noise. This would have biased the training towards the content image, while incorporating the style features gradually. Initializing the target image as the content image was useful for speeding up convergence, but this change was most likely the reason why our results and the result produced by [1] were so different.

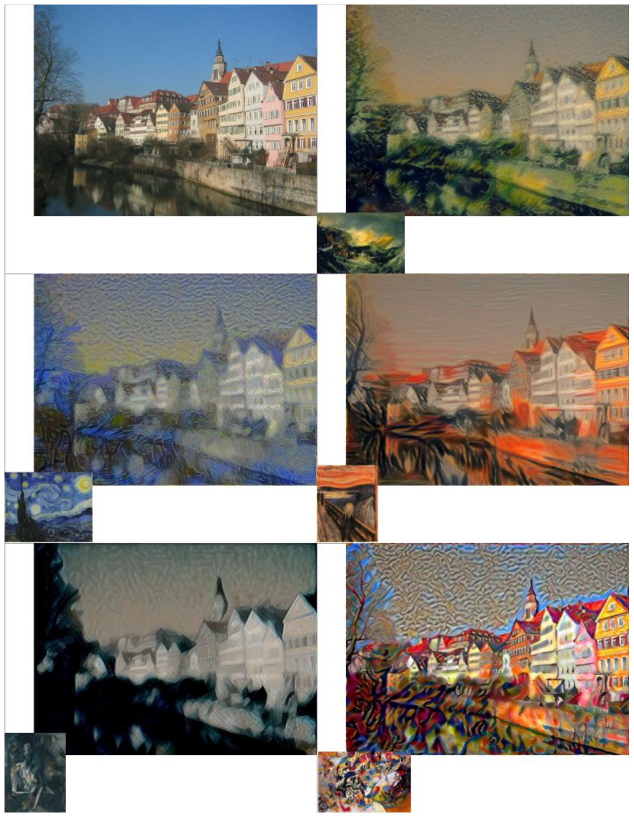
Figure 6. Style transfer results produced by our model on the same content and style images.

However, we think that our balance of content and style is more optimal for the task, since unlike the original paper, it retains the general structure of the content image, with style imposed on it. In the original paper, for images E & F, the results look more like a mixture of images and less like style over content.

Another reason for difference of results might be the difference in choice of learning rate and number of training iterations, since that information was not mentioned in the paper. Moreover, the dataset they have used is also not a standard, publicly-available dataset, and differences in image color intensities and resolutions of the content and style image could lead to different results.

We have also implemented mean pooling rather than the usual max pooling in VGG-19. The paper suggests mean pooling, since it improves the gradient flow in each iteration. Since, we use the content image as the initial target image, better gradient flow would mean better incorporation of style. This difference can be clearly seen in figure 7, as image A looks closer to a painting with smoother features than image B.

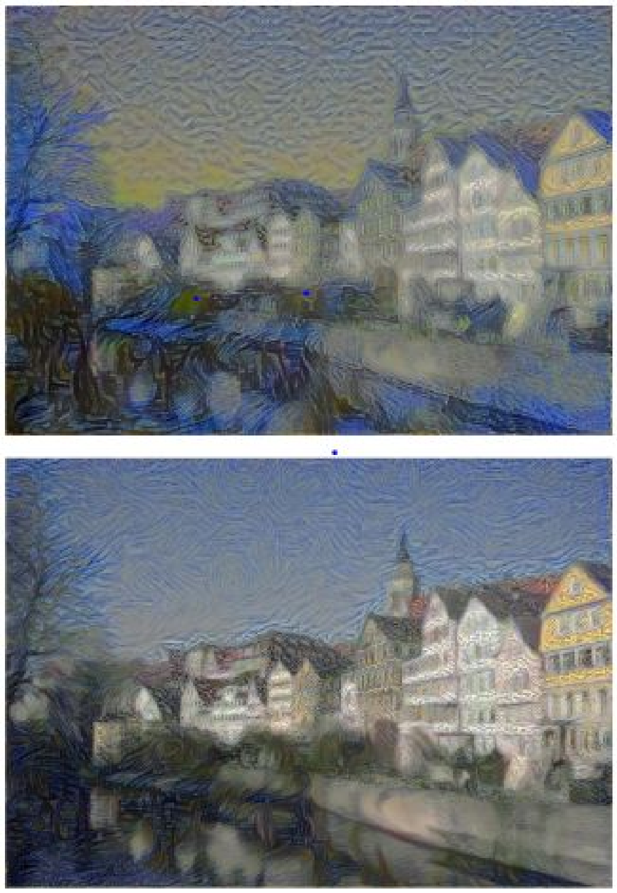

Figure 7. **Image A** (Top): Figure 6(c) with mean pooling; **Image B** (Bottom): Figure 6(c) with average pooling.

We go on to compare images with and without total variation loss in figure 8. The features in image A are smoothened out, whereas image B has more prominent edges. Since in painting, the transitions between strokes are smoother, image A is better. Therefore, using total variation loss improves the quality of the artistic rendition.

## 6. Conclusion

With the methods proposed in the paper, we were able to produce artistic images of high perceptual quality, which were different from the ones presented in the paper due to the modifications we made. We believe that our results are better in the sense that they preserve the structure of the content image and add style on top of it, making them look like artistic renditions in style of the original content image. Additionally, we noted that mean pooling resulted in much better color and style in the output image. Moreover, we also found that addition of a total variation loss term helps smoothen out very high frequency details, since artistic images rarely have very detailed strokes.

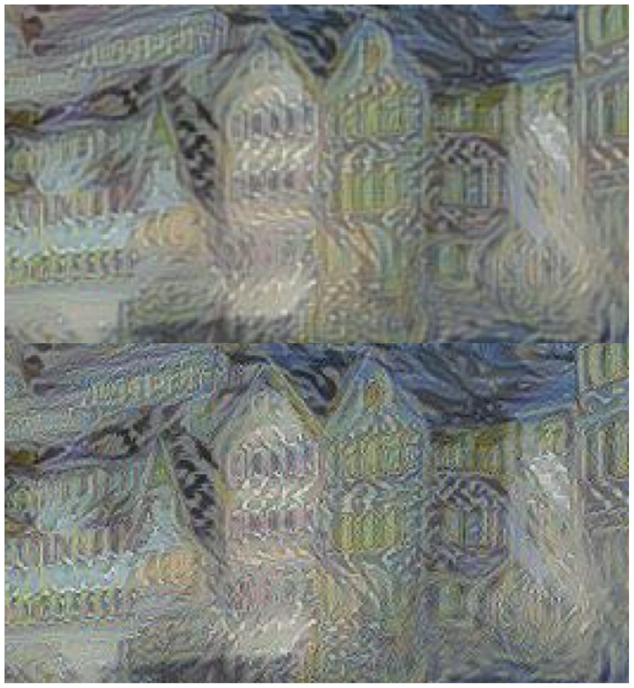

Figure 8. **Image A** (Top): Snippet of Figure 6(c) with total variation loss; **Image B** (Bottom): Snippet of Figure 6(c) without total variation loss.

## 7. Acknowledgement

We thank Dr. Zoran Kostic for teaching Neural Networks and Deep Learning this semester. We also thank the course TAs, Shilin Hu, Zixiao Zhang, Yilin Lyu, Yong Yang, Zixi Huang and Zekun Gong, for their help and responsiveness when questions arose throughout the semester.

## 8. References

[1] Gatys, L.A., Ecker, A.S., & Bethge, M. (2015). A Neural Algorithm of Artistic Style. *CoRR, abs/1508.06576*.
[2] Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012) “Imagenet classification with deep convolutional neural networks”, Advances in neural information processing systems, 1097–1105.
[3] Canziani, A., Paszke, A., & Culurciello, E. (2016). An Analysis of Deep Neural Network Models for Practical Applications. *CoRR, abs/1605.07678*.
[4] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. http://www.robots.ox.ac.uk/~vgg/research/very_deep/
[5] Shafeen Tejani (2018), From Bits to Brains: Artistic Style Transfer with Deep Neural Networks https://shafeentejani.github.io/2016-12-27/style-transfer/

[6] Link to Bitbucket
Aditya Sinha -
bitbucket.org/ecbm4040/2018_assignment2_as5624/
Praneet Vibhu Balabolu -
bitbucket.org/ecbm4040/2018_assignment2_pb2735
Vinay S Raghavan -
bitbucket.org/ecbm4040/2018_assignment2_vsr2119/

## 9. Appendix

### 9.1 Individual student contributions in fractions - table

| | as5624 | pb2735 | vsr2119 |
|---|---|---|---|
| Last Name | Sinha | Balabolu | Raghavan |
| Fraction of (useful) total contribution | 1/3 | 1/3 | 1/3 |
| What I did 1 | Procured dataset, data parsing for I/O | Key components, interpretation of algorithm flow | Created the style transfer routine in ipynb |
| What I did 2 | Code for tv loss, simulations | Code for interfacing with VGG class | Wrote the training, loss functions |
| What I did 3 | Abstract, introduction, summary in report | Results of original paper & ours,conclusions | Objectives, methodology, implementation, software design. |