DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY
Office for Scholarly Communication

Understanding Decisions and Tradeoffs of Neural Style Transfer

Citation

Morgan, Nick. 2021. Understanding Decisions and Tradeoffs of Neural Style Transfer. Master's thesis, Harvard University Division of Continuing Education.

Permanent link

https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37367685

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Understanding Decisions and Tradeoffs of Neural Style Transfer

Nick Morgan

A Thesis in the Field of Software Engineering

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2021

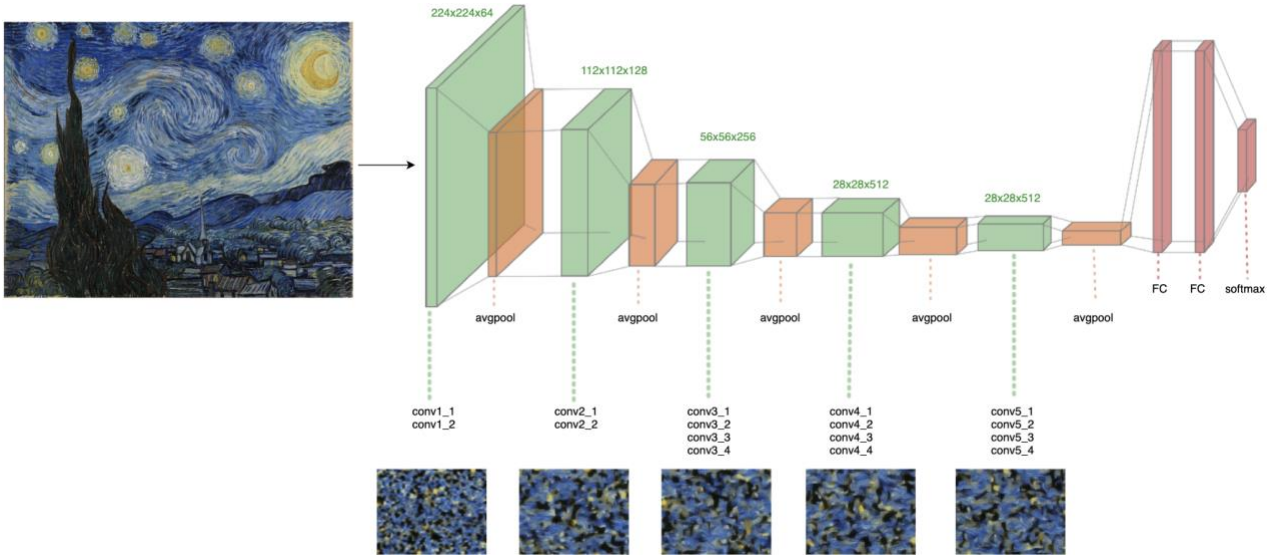
Copyright 2021 Nick Morgan

Abstract

A Neural Algorithm of Artistic Style introduced an artificial system to create artistic images. It does so by generating a set of feature maps from a content image and a style image. These feature maps are used to generate a loss, which is applied iteratively to a noisy image via gradient descent. This process results in the generated image having features which represent that of the target content and the target style images. The goal of this project is to explore the original research decisions, and explicitly document the tradeoffs being made.

The original research used the Visual Geometry Group's 19 weight-layer model. Many properties of this method were not fully documented - the optimization method is not specified (only broadly referred to as gradient descent), the architecture is altered to "improve gradient flow" without documenting how it was improved, no other models are evaluated, and many properties of the gradient descent process were not explored. This project aims to explore additional models, document various methods of gradient descent, explain characteristics of the loss, and propose solutions to improve the gradient flow.

Frontispiece



Acknowledgments

I want to thank my thesis advisor, Dr. Stephen F. Elston. His guidance throughout this research has been invaluable.

I would also like to thank my research advisor, Dr. Hongming Wang, for helping me draft my thesis proposal and for valuable advice throughout the process.

Finally, I would like to thank my 10th-grade math teacher, Mr. Way. He was the first teacher to get me excited about math. Without his influence, I don't believe I would have pursued this degree.

Table of Contents

List of Tables

List of Figures

Glossary

BFGS

Broyden-Fletcher-Goldfarb-Shanno Algorithm

Conv1_1, Conv2_1, Conv3_1, Conv4_1, Conv5_1

With regard to any of the VGG models, Convi_j is denoted such that i represents the convolutional block and j represents the specific convolutional layer within a given convolutional block i. Conv3_1, for example, refers to the first convolutional layer within the third convolutional block

L-BFGS

Limited-Memory Variant of Broyden-Fletcher-Goldfarb-Shanno Algorithm

Style A, Style B, Style C, Style D, Style E

Refers to the notation used by Gatys et al. (2015) to denote feature map configurations when calculating the loss.

SGD

Stochastic Gradient Descent

VGG-11

11 weight-layer variant of the Visual Geometry Group's convolutional neural networks (Simonyan et al., 2014)

VGG-13

13 weight-layer variant of the Visual Geometry Group's convolutional neural networks (Simonyan et al., 2014)

VGG-16

16 weight-layer variant of the Visual Geometry Group's convolutional neural networks (Simonyan et al., 2014)

VGG-19

19 weight-layer variant of the Visual Geometry Group's convolutional neural networks (Simonyan et al., 2014)

Chapter I.

Introduction

A Neural Algorithm of Artistic Style has become increasingly popular in recent years. Many websites and applications use its methodology to produce visually appealing images. Its implementation has been modified and expanded into many domains ranging from game development to artificial makeup removal from images.

The goal of this project is to explicitly document the convergence behavior of style transfer algorithms. The 19 weight-layer variant of the Visual Geometry Group's Large-Scale Image Recognition models was used for the original implementation, although there are many similar variants of this model that were also introduced by the Visual Geometry Group (Simonyan et al., 2014). Gatys et al. (2015) uses the 19 weight-layer variant to generate feature maps, which are used to generate two loss functions - content and style. The content loss is calculated by merely using the mean squared error, whereas the style loss is calculated by taking the Gram matrix of the activations before calculating the mean squared error. This loss is applied back to a noisy input image iteratively, resulting in a generated image that represents the feature maps of its targets.

The original implementation optimizes this process, referring vaguely to "gradient descent", but does not define which method of gradient descent was used, which hyperparameters were used, or any of the convergence properties associated with the process (Gatys et al. 2015). The style loss function applies normalization based on the shape of each activation but does not explore additional methods of normalization.

This project also aims to document the loss behavior and the characteristics of how the loss is generated. The style loss utilizes various hidden-layer activations in order to generate Gram matrices. These Gram matrices have interesting characteristics which have been found to influence the behavior of the loss.

1.1 Background

Neural Style Transfer is dependent on many areas of research which preceded it. This section serves as a brief overview of all the tools and techniques required to implement this algorithm.

1.1.1 ImageNet

Deng et al. (2009) released ImageNet with the goal of populating the majority of existing WordNet synsets. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition that allows researchers to compare algorithms' accuracy in object detection and image classification.

1.1.2 Visual Geometry Group

The Visual Geometry Group (VGG) is a research group from the University of Oxford that has participated in the ILSVRC. Simonyan et al. (2014) published six models as part of the Visual Geometry Group's 2014 ILSVRC submission. At the time of publication, the purpose of these image-recognition models was to evaluate convolutional neural networks of increasing depth, using an architecture with small convolution filters. These models achieved first place for the 2014 localisation track, and second place for the classification track of the competition.

Each of the six models were trained on the same subset of ImageNet images, and follow the same steps of preprocessing. The inputs are a batch of fixed-size 224x224 images with the pixel order of red-green-blue (RGB images).

The original implementation changed the pixel order to blue-green-red (BGR images) before subtracting the mean pixel value - calculated from the training dataset - from each individual image.

As illustrated in the table below, each of the model's variants have a similar architecture - with the key difference between variants being the number of convolutional layers within each convolutional block. Additionally, the 11 weight-layer model includes a variant which adds a Local Response Normalisation (LRN) layer. The 16 weight-layer model also includes a variant which replaces some of the 3x4 convolutional filters with a 1x1 convolutional filter.

Table 1. ConvNet Configurations

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Taken from Very Deep Convolutional Networks for Large-Scale Image Recognition

Metrics of success for these models are defined as the top-1 and top-5 error. Top-1 is a multi-class classification error; it measures the proportion of incorrectly classified images. Top-5 represents the proportion of images whose label falls outside of the model's top-5 predicted categories.

When evaluated against the 2014 ILSVRC test dataset, each of the six published models had top-1 val errors between 25.5% and 29.7%. Each of the six-published models had top-5 val errors between 8.0% and 10.5%. The most accurate of these models, in both categories, was the 19 weight-layer model

Table 2. ConvNet Performance Metrics

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

Taken from Very Deep Convolutional Networks for Large-Scale Image Recognition

1.1.3 Pooling

Yamaguchi et al. (1990) published the first implementation of max-pooling, which was applied to a time delay neural network (TDNN). Its original application was to support a speaker-independent isolated word recognition system. Since its introduction, pooling has become a popular method of dimensionality-reduction in convolutional neural networks.

Pooling operations reduce the dimensions of neural networks by combining multiple layer outputs in order to generate a single input for the following layer. Max pooling operates by taking the maximum value of the cluster of neurons at the prior layer, whereas average pooling operates by taking the average value of the cluster.

Max pooling and average pooling have key differences - maximum pooling is nonlinear, whereas average pooling is linear. For the purposes of implementing Neural Style Transfer, this effect may be significant; nonlinearity of the pooling layers may lead to a diminished gradient flow. Gatys et al. (2015) replaced the max-pooling layers with average-pooling when using the pre-trained 19 weight-layer Visual Geometry Group model.

1.1.4 VGG Implementations

The Visual Geometry Group published pre-trained models - trained on the 2014 ILSVRC subset of the ImageNet dataset - for their 16 weight-layer and 19 weight-layer models. Pre-trained models are often utilized, as training the models is time-consuming (these models in particular took between two and three weeks, according to the Visual Geometry Group).

The learnable weights of the model are saved to a file, and this file is used to initialize the weights of the network - instead of beginning with randomly-generated weights (which is typically done before training a model). This does not include the pooling layers, which do not have learnable weights.

Pre-trained models are available in many different frameworks, although the models vary slightly by implementation. The original models released by the Visual Geometry Group are implemented using the Caffe deep learning framework. This framework was utilized by the original implementation (Gatys et al., 2015)

The PyTorch framework includes pretrained models for all of the architectures introduced by the Visual Geometry Group, although its implementation varies slightly. The Caffe framework model was trained using the 2014 ILSVRC subset of the ImageNet

database. PyTorch's implementation was trained on a different subset of ImageNet data - the exact subset of which is currently unknown. At the time of writing, the published models were trained on a random subset of the 2012 ImageNet dataset, but the exact images (as well as the sample size) were lost.

1.1.5 A Neural Algorithm of Artistic Style

The original implementation by Gatys et al. (2015) uses the 19 weight-layer variant of the Visual Geometry Group's model. After the model's weights are loaded, the max-pooling layers are replaced with average-pooling in order to improve gradient flow. The weights of this model are frozen; the weights are not updated throughout the process.

The implementation consists of two loss functions - content and style - which are evaluated both separately and combined. All of the methods - content, style, and a combination of both - begin with a generated image consisting of random noise. The pixels of this generated image are updated iteratively in order to minimize the loss function.

All of the loss functions utilize the hidden-layer activations of the Visual Geometry Group's 19 weight-layer variant. Both the noise image and the target image[s] are sent through this neural network. Each of the convolutional layers has a corresponding rectified linear activation. The activations of the noise image and the target image are used to generate the loss, and the pixels of the generated image are updated iteratively in order to minimize this loss.

Gatys et al. (2015) has demonstrated that Content loss is more straight-forward to calculate than Style loss and has less variation in its generated image. This research aims to explore Style loss specifically; no exploration of Content loss is presented in the

findings of this research. However, Content loss is introduced by Gatys et al. (2015), and is constructed in a manner that is similar to that of Style loss. For that reason, and for the sake of a thorough background on the algorithm, a brief summary of Content loss is provided below.

Content loss is calculated by taking the mean squared error between the target activations and the noise activations:

$$\mathcal{L}_{\text{content}}\left(\vec{p}, \vec{x}, l\right) = \frac{1}{2} \sum_{i,j} \left(F_{ij}^{l} - P_{ij}^{l}\right)^{2}$$

Where $\vec{p}$ rand $\vec{x}$ represent the original image and the image that is generated and P^{l} and F^{l} represent their respective feature representation in layer l. The derivative of the loss with respect to the activations in layer l equals

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F_{ij}^{l}} = \begin{cases} \left(F^{l} - P^{l}\right)_{ij} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} < 0 \end{cases}$$

Where F_{ij}^{l} is the activation of the i^{th} filter at position j in layer l. An interesting observation about this derivative is the distinction between activations that are greater than 0 vs activations that are less than 0 - rectified linear activations by their very nature assert that the activations will never be below 0. A modification to this derivative is proposed in section 2.3.2, which modifies the derivative as follows:

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F_{ij}^{l}} = \begin{cases} \left(F^{l} - P^{l}\right)_{ij} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} \leq 0 \end{cases}$$

Style loss also utilizes the target activations and the noise activations but does so by generating a Gram matrix of the activations before calculating the mean squared error. The contribution of loss from an individual layer l is represented by

$$E_{l} = \frac{1}{4N_{l}^{2}M_{l}^{2}} \sum_{i,j} \left(G_{ij}^{l} - A_{ij}^{l}\right)^{2}$$

Where N and M represent the height and width of the activations, and G_{ij}^{l} and A_{ij}^{l} represent the Gram matrices of the activations ij at layer l. Whereas content loss is calculated by the activations of a single layer, style loss combines one or more layers. The total style loss is defined as

$$\mathcal{L}_{\text{style}}\left(\vec{a}, \vec{x}\right) = \sum_{l=0}^{L} w_{l} E_{l}$$

Where w_{l} are the weighting factors for each layer. Gatys et al. (2015) uses equal weightings across all layers of the network when calculating the loss. The derivative is calculated as follows:

$$\frac{\partial E_{l}}{\partial F_{ij}^{l}} = \begin{cases} \frac{1}{N_{l}^{2}M_{l}^{2}} \left(\left(F^{l}\right)^{\mathrm{T}} \left(G^{l} - A^{l}\right)\right)_{ji} & \text{if } F_{ij}^{l} > 0 \\ 0 & \text{if } F_{ij}^{l} < 0 \end{cases}$$

The same modification described above is also proposed in section 2.3.2, which modifies the distinction to include values that are less than *or equal to* 0:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - A^l\right) \right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l \leq 0 \end{cases}$$

Gatys et al. (2015) evaluates these losses on their own, and also combines them using α and β as weighting parameters:

$$\mathcal{L}_{\text{total}}\left(\vec{p}, \vec{a}, \vec{x}\right) = \alpha \mathcal{L}_{\text{content}}\left(\vec{p}, \vec{x}\right) + \beta \mathcal{L}_{\text{style}}\left(\vec{a}, \vec{x}\right)$$

Content and style loss functions are generated using various activations throughout the network. Content uses a single activation for all of its configurations, whereas style uses a weighted combination of one or more activations. There are five configurations of activations considered, which are labelled as A - E:

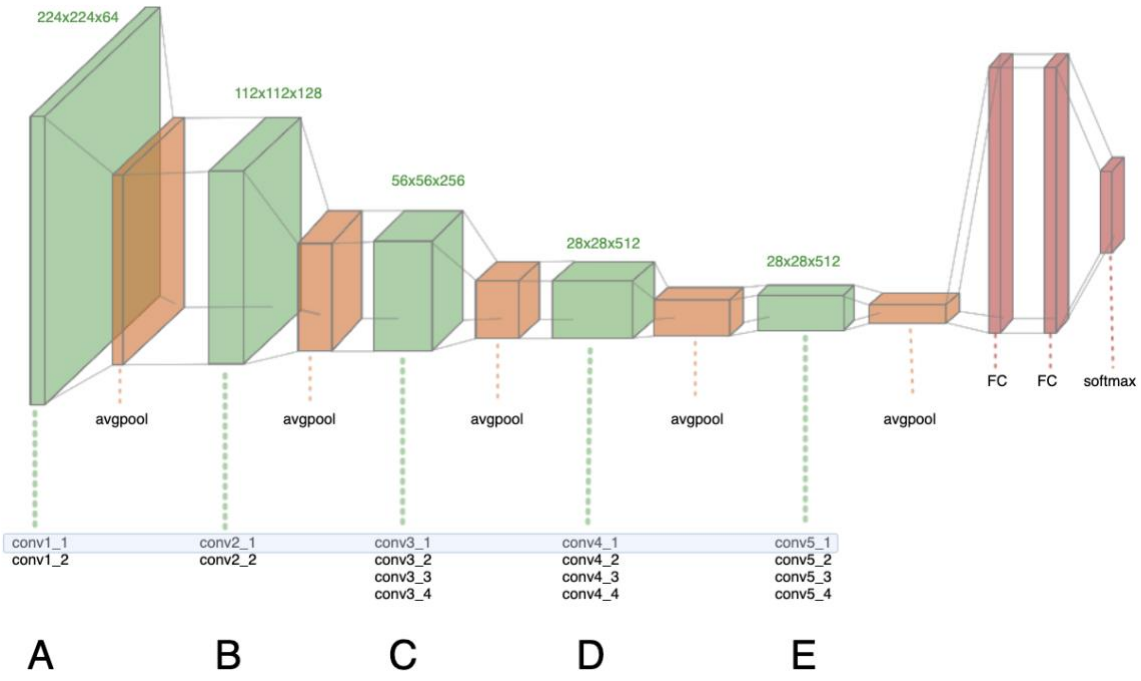


Figure 1. VGG 19 Convolutional Neural Network.

The first convolutional layer of each convolutional block (highlighted in the image above) are used to calculate each of the 5 loss configurations, which are labelled A-E. Content Loss functions use a single convolutional layer, whereas Style Loss functions use every layer that precedes it. Style E, for example, is the weighted average of losses for conv1_1, conv2_1, conv3_1, conv4_1, and conv5_1.

Although the original implementation only broadly refers to "gradient descent" without suggesting a specific optimization method, the author has suggested online to optimize via the quasi-newton method of Broyden-Fletcher-Goldfarb-Shanno algorithm.

1.1.6 Broyden-Fletcher-Goldfarb-Shanno Algorithm

Newton's method is an iterative method for finding the root of a differentiable function. It can be applied to the derivative of a twice-differentiable function in order to

find the minimum or maximum of the function. It does so by selecting a random starting point, and performing the following iteration until the optimal value is reached:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

When expanded to a higher-dimensional space, Newton's method can be generalized as follows:

$$x_{k+1} = x_k - [f''(x_k)]^{-1} f'(x_k)$$

where $[f''(x_k)]^{-1}$ represents the inverse of the Hessian matrix, and $f'(x_k)$ represents the gradient.

Although Newton's method is effective for finding optima, it can be very computationally expensive - calculating the inverse Hessian - which must be done each iteration - has a computational complexity of $O(n^3)$.

Quasi-Newton methods reduce the complexity imposed by Newton's method, as they approximate the Hessian instead of calculating it directly. Four authors independently published the algorithm now known as Broyden-Fletcher-Goldfarb-Shanno (Broyden, 1969; Fletcher, 1969; Goldfarb, 1969; Shanno, 1969). This algorithm gradually improves the approximation for the Hessian through a generalized secant method. These updates do not require matrix inversion, which reduces the computational complexity from $O(n^3)$ to $O(n^2)$. The iterative method can be summarized as follows:

$$f\left(x_k + \Delta x\right) \approx f\left(x_k\right) + \nabla f\left(x_k\right)^{\mathrm{T}} \Delta x + \frac{1}{2}\Delta x^{\mathrm{T}} B \Delta x$$

Where ∇f represents the gradient, and B represents the approximation of the Hessian.

While BFGS reduces the computational complexity, it does not reduce the memory required to store the Hessian - which has a memory requirement of $O\left(n^2\right)$ for the $n \times n$ matrix. Liu et al. (1989) effectively reduces the memory complexity of this algorithm (L-BFGS, for limited-memory BFGS) by approximating the Hessian via the history of the gradients. This results in a linear memory requirement, as only one vector is required to approximate the Hessian instead of a dense $n \times n$ matrix.

1.1.7 Strong Wolfe Condition on Curvature

The approximation of the Hessian may not represent the true objective function, which may lead to strange convergence behavior. If the quadratic approximation is shallower or steeper than that of the true function, a step may overshoot or undershoot and lead to an iteration that increases the cost.

Wolfe (1969) introduced a set of conditions that, when enforced, may lead to better convergence behavior. These conditions assert that the length of each iteration step is proportional to the decrease in the objective function. This also means that any individual step cannot increase the cost.

1.1.8 Stochastic Gradient Descent

Robbins et al. (1951) introduced the Robbins-Monro algorithm, which is an iterative method to solve root-finding problems. One such application of this algorithm is Stochastic Gradient Descent, which is an iterative method for optimizing differentiable functions. It does so by randomly selecting a sample and using that sample to approximate the true gradient. This process repeats until the approximate minimum is reached. The objective can be summarized as follows:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w)$$

Where w minimizes $Q(w)$. SGD finds a w that minimizes $Q(w)$ by performing the following calculation iteratively:

$$w := w - \eta \nabla Q(w)$$

Where $\nabla Q(w)$ represents the gradient at w and η represents the step size. SGD does not compute the entire gradient. Instead, the gradient is approximated from a random sample of rows on each iteration. Choosing an appropriate step size is important - a step size too small will take very long to converge, whereas a step size that is too large may overstep the true optima.

1.1.9 Adaptive Movement Estimation

Kingma et al. (2014) proposed a solution to this step-size problem by introducing a method which updates the step size dynamically. Each parameter is assigned an individual step size, and this step size is updated in each iteration, based on the gradient of the previous iteration.

1.1.10 Condition Numbers

For any given function, the condition number represents how much the output of that function will change for a small change in the input values. A problem with a low condition number is defined as being well-conditioned, and a problem with a high condition number is defined as being ill-conditioned. The solution to an ill-conditioned problem becomes hard to find, especially through gradient descent, as the cost can vary drastically even with a very small step size. Additionally, the gradient may only be well-defined in some directions and under-determined in others.

A condition number of 100 is often used as a baseline reference. A condition number below 100 is said to be well-conditioned and may have easier convergence properties. A condition number above 100 is said to be ill-conditioned and makes the optimization convergence slow and uncertain.

Chapter II.

Methods

Gatys et al. (2015) is used as a baseline by which all other trials are compared. Many trials were evaluated, and the results of trials were used to dictate the path of future trials. Once a superior configuration for a method had been established, that configuration was used as the baseline during comparisons of new methods.

2.1 Image Processing

A few steps are required before the process of gradient descent can begin. Some of these steps must be reversed after the optimization has been completed, and some additional steps are required after the optimization in order to enable rendering of the output image.

2.1.1 Preprocessing

All input images are resized to have a height of 256 pixels. The width of the images is resized such that the aspect ratio remains constant. Some trials apply normalization to the input images based on the ImageNet subset that PyTorch models were trained on. This normalization is applied over the channels of the image (red-green-blue).

Table 3. PyTorch ImageNet Subset Characteristics

Mean	0.485, 0.456, 0.406
Standard Deviation	0.229, 0.224, 0.225

Some trials apply histogram equalization to the input images, and some trials do not apply any method of normalization or equalization.

Images typically have pixels that are in the range of [0, 255]. For most of the trials, the input images are scaled to fall within the range of [0, 1] - the same range that was used while training the PyTorch models.

Once the target image has been loaded, a noise image of the same range and dimensions is generated by randomly selecting pixel values in the range of the target image.

2.1.2 Postprocessing

Any trials that utilize image normalization as part of its preprocessing step reverse the normalization as part of postprocessing. Since there is no constraint on the noise image during gradient descent, it is possible that pixel values will fall outside of the renderable range of [0, 1] or [0, 255]. The output image is thus clipped at its respective range.

Images that generate an output with range [0, 1] are re-scaled after clipping in order to have a range of [0, 255]. This has no effect on the visual properties of the output image but results in a smaller file as it is less computationally expensive to store integers (only possible with range [0, 255]) than it is to store floats (required with range [0, 1]).

Table 4. Preprocessing & Postprocessing Methods

		Method			
		1	**2**	**3**	**4**
Pre	**Resize**	Resize image to height of 256			
	Range of Input Pixels	[0, 1]	[0, 1]	[0, 1]	[0, 255]
	Normalization Method	PyTorch ImageNet	Histogram Equalization	None	PyTorch ImageNet
Post	**Reverse Normalization**	PyTorch ImageNet	None	None	PyTorch ImageNet
	Clip Tensor	[0, 1]	[0, 1]	[0, 1]	[0, 255]
	Range of Output Pixels	[0, 255]	[0, 255]	[0, 255]	[0, 255]

PyTorch ImageNet normalization refers to normalizing the input image in accordance with the subset of ImageNet data that was used to train the PyTorch Models.

2.2 Visual Geometry Group Models

Every model used for the purpose of this research is loaded using the PyTorch framework. After loading the weights, each of the weights are frozen to ensure that the weight values do not update during this process. Because each model only has a single output representing a classification, a modification is made to capture the activations of every convolutional layer.

2.2.1 VGG-19

The 19 weight-layer variant of the VGG models is the primary model that is considered across all trials. The first convolutional layer from each convolutional block is used to generate Styles A through E.

2.2.2 VGG-11, VGG-13, and VGG-16

The 11, 13, and 16 weight-layer variants are evaluated against the 19 weight-layer variant. While these models have varying amounts of convolutional layers, they share the same overall structure - five blocks of convolutional layers which are separated by a pooling layer. As with the 19 weight-layer variant, each of these variants generate its corresponding loss by taking the first convolutional layer from each respective convolutional block.

2.2.3 Pooling

In accordance with the Gatys et al. (2015) implementation, each of the max pooling layers are replaced with average pooling layers before gradient descent. Additionally, one set of trials serves to explicitly compare the output when using max pooling to the output when using average pooling.

2.3 Loss

Gatys et al. (2015) demonstrates that Style loss has a much larger variance in output when compared to content loss. As such, style loss is the primary method which is evaluated for the purpose of this research. The style loss of the original implementation is used as the baseline by which all modifications are compared.

The baseline loss for an individual layer is defined as

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

The baseline loss for all layers is then defined as

$$\mathcal{L}_{\text{style}}\left(\vec{a},\vec{x}\right)=\sum_{l=0}^{L}w_{l}E_{l}$$

And the baseline derivative from the loss is defined as

$$\frac{\partial E_{l}}{\partial F_{ij}^{l}}=\begin{cases}\frac{1}{N_{l}^{2}M_{l}^{2}}\left(\left(F^{l}\right)^{\mathrm{T}}\left(G^{l}-A^{l}\right)\right)_{ji} & \text{if } F_{ij}^{l}>0\\ 0 & \text{if } F_{ij}^{l}<0\end{cases}$$

The layers and labelling of each loss configuration mimic that of the original implementation:

Table 5. Style Loss Configurations

Style Loss ID	**Convolutional Layers**
A	conv1_1
B	conv1_1, conv2_1
C	conv1_1, conv2_1, conv3_1
D	conv1_1, conv2_1, conv3_1, conv4_1
E	conv1_1, conv2_1, conv3_1, conv4_1, conv5_1

The layers and identifiers mimic that of Gatys et al. (2015). When more than one layer is evaluated for the loss, all layers are weighted equally.

2.3.1 Normalization of Feature Maps

The Gram matrices of the corresponding activations were evaluated and found to be ill-conditioned with condition numbers at or near infinity. As a result, various normalization methods are considered to evaluate whether outcomes can be improved by modifying the activations before generating the Gram matrices.

Style loss is evaluated by calculating the Gram matrices of the activations - that is, the inner product of the activations transposed by itself: For each layer l, the Gram matrix can be represented as the inner product between the vectorized feature map i and j

$$G_{ij}^{l} = \sum_{k} F_{ik}^{l} F_{jk}^{l}$$

This approach often results in Gram matrices that are ill-conditioned with condition numbers at or near infinity. As such, multiple normalization methods are considered to evaluate whether outcomes can be improved by modifying the activations before generating Gram matrices. This modification can be written by

$$G_{ij}^{l} = \sum_{k} F_{norm_{ik}}^{l} F_{norm_{jk}}^{l}$$

Where F_{norm} represents the normalized representation of the activations. F_{norm} is calculated in numerous ways as defined below.

Min-Max Normalization

Min-Max normalization alters the activations such that the minimum is 0 and the maximum is 1. During each iteration, the following transformation is applied to the activations of both the generated image as well as the target image:

$$F_{norm} = \frac{F - F_{min}}{F_{max} - F_{min}}$$

Pulling Elements Towards Mean

Many of the activations were found to have extreme outliers with values greater than fifty standard deviations above the mean. One method of normalization aims to resolve this by pulling all elements closer to the mean

$$F_{norm} = \frac{(1-t) \times F}{t \times F_{mean}}$$

Where t is a constant

Dividing by Standard Deviation

Another method of accounting for outliers is dividing all elements of the activations by the standard deviation.

$$F_{norm} = \sum_{i,j} \frac{F_{ij}}{F_{std}}$$

Dividing by Standard Deviation, by Channel

This method of normalization is also considered with respect to the standard deviation of each individual channel of the activations

$$F_{norm} = \sum_{i,j} \frac{F_{ij}}{std_i}$$

Table 6. Summary of Normalization Methods

Name	**Formula**
Min-Max Normalization	$F_{norm} = \frac{F - F_{min}}{F_{max} - F_{min}}$
Pulling Elements Towards Mean	$F_{norm} = \frac{(1-t) \times F}{t \times F_{mean}}$
Dividing by Standard Deviation	$F_{norm} = \sum_{i,j} \frac{F_{ij}}{F_{std}}$
Dividing by Standard Deviation by Channel	$F_{norm} = \sum_{i,j} \frac{F_{ij}}{std_i}$

2.3.2 Feature Masks

The original implementation makes a distinction for derivatives based on the value of the activation being less than 0:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - A^l\right) \right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

Because rectified linear activations are lower-bound at 0, an activation will never have a value below 0. For this reason, a modification is considered which applies this same distinction at values that are less than *or equal to* 0:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - A^l\right) \right)_{ji} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l \leq 0 \end{cases}$$

The activations are found to have extreme outliers, with many elements having a value as much as fifty standard deviations above the mean. As such, an additional feature mask is considered, which only applies the derivative for activations that are within three standard deviations of the mean:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left(\left(F^l\right)^{\mathrm{T}} \left(G^l - A^l\right) \right)_{ji} & \text{if } |F_{ij}^l| \leq F_{mean} + F_{std} \times 3 \\ 0 & \text{if } |F_{ij}^l| > F_{mean} + F_{std} \times 3 \end{cases}$$

2.3.3 Early Stopping

Most trials are run with a maximum of five hundred epochs. Many trials reach a nonzero convergence limit before reaching five hundred epochs. Early stopping is evaluated every fifty epochs, and a trial is stopped if the value of the loss has not changed for fifty epochs at the time of evaluation.

2.4 Optimization Methods

Although Gatys et al. (2015) does not mention a specific optimization method, the author has recommended the Broyden-Fletcher-Goldfarb-Shanno algorithm online. Other optimization methods are also evaluated.

2.4.1 Broyden-Fletcher-Goldfarb-Shanno Algorithm

The limited-memory variant of this algorithm is implemented to satisfy the hardware constraints and improve processing time. This algorithm is evaluated with and without a line search function. The line search function evaluated is the Strong Wolfe method. All other hyperparameters are held constant throughout all trials.

Table 7. Limited-Memory Broyden-Fletcher-Goldfarb-Shanno Hyperparameters

Parameter	**Values Evaluated**
Learning Rate	1
Max Iterations Per Optimization Step	20
Max function Evaluations Per Optimization Step	25
Termination Tolerance on First Order Optimality	$1e^{-}5$
Termination Tolerance on Function Value	$1e^{-9}$
Update History Size	100
Line Search Function	None, Strong Wolfe

2.4.2 Stochastic Gradient Descent

Stochastic Gradient Descent is evaluated with varying magnitudes of a learning rate. All other hyperparameters are held constant throughout all trials.

Table 8. Stochastic Gradient Descent Hyperparameters

Parameter	**Values Evaluated**
Learning Rate	0.01, 0.1, 1, 10
Momentum Factor	0
Weight Decay	0
Dampening for Momentum	0

2.4.3 Adaptive Moment Estimation

Adam Optimization is evaluated with varying magnitudes of a learning rate. All other hyperparameters are held constant throughout all trials.

Table 9. Adaptive Moment Estimation Hyperparameters

Parameter	**Values Evaluated**
Learning Rate	0.01, 0,1, 1, 10
Beta Coefficients	(0.9, 0.999)
Denominator Epsilon	$1e^{-8}$
Weight Decay	0

2.5 Metrics

Gatys et al. (2015) provides output images. While these images are visually interesting, it makes comparison difficult as the quality of images is inherently subjective. This research aims to add the ability to quantitatively compare each of the methods considered.

2.5.1 Subjective Analysis of Generated Images

Each trial outputs the final value of the generated image, with its values clipped at its respective range in order to enable rendering. While these results are subjective, some discussion can be made about the variety of output generated through gradient descent.

2.5.2 Loss

The log loss of each activation is recorded for each epoch. Style A only includes the log loss for the activations corresponding to conv1_1, whereas Style E includes the log loss for the activations corresponding to conv1_1, conv2_1, conv3_1, conv4_1, conv5_1.

If the loss for every convolutional layer remains constant for fifty epochs, it is assumed that the nonzero convergence limit has been reached, and the trial is terminated.

2.5.3 Condition Number of Gram Matrices

Once a trial is concluded (after reaching 500 epochs or after reaching 50 consecutive epochs with a constant loss for all layers), the Gram matrices for the final output image is recorded, prior to clipping the output image in order for rendering.

The sorted eigenvalues of these Gram matrices are plotted alongside their corresponding condition numbers. Similar to the loss evaluation, the Gram matrix analysis is done with regard to each convolutional layer that comprises the loss.

In instances where normalization is applied to the Gram matrix, the sorted eigenvalues and condition numbers are plotted before the normalization and after the normalization, in order to visualize the effect of normalization against the Gram matrices.

Chapter III.

Results

Recreating the results of the original implementation is unattainable for two reasons:

1. The generated image begins as randomly-generated noise, which results in a different output between identical trials
2. The original implementation uses the Caffe framework of the Visual Geometry Group's model, which was trained on a different subset of ImageNet data than that of Pytorch.

With these two limitations in mind, many of the results have generated images that closely resemble that of the original implementation. The optimal configurations are summarized in the table below. Additional methods evaluated can be viewed in their corresponding sections.

Table 10. Optimal Configurations by Category

Category	**Optimal Configuration**
Normalization Method	Divide by Standard Deviation By Channel
Pooling Method	Average
Optimization Method	L-BFGS with Wolfe Constraint
VGG Variant	VGG-13
Feature Mask	None

Using these optimal configurations results in the following:

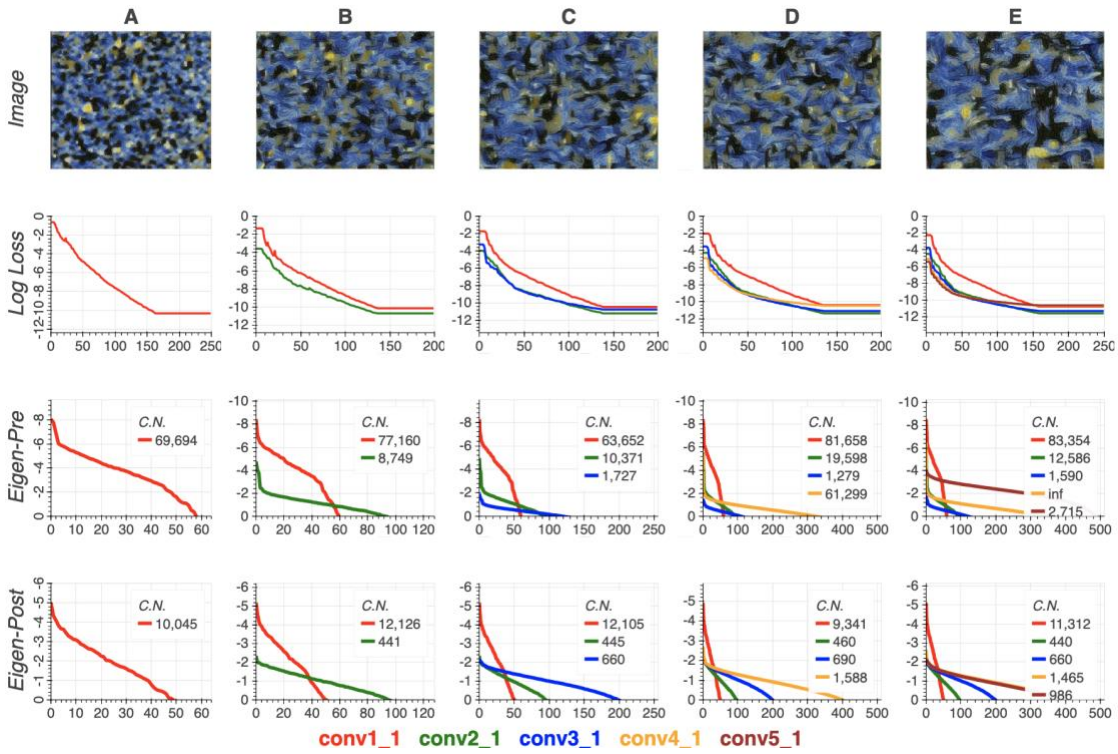


Figure 2. Results of Optimal Configuration

These trials normalize by dividing by the standard deviation by channel, utilize the VGG-13 model, replace max pooling layers with average pooling, and optimize via L-BFGS with a Wolfe Constraint on Curvature. Letters A-E refer to the Style Configuration as per Gatys et al. (2015). Eigen-Pre and Eigen-Post refer to the eigenvalues before and after normalization, respectively. C.N. refers to the condition number.

The log loss of these trials is comparable to many of the other trials - both in the amount of decrease, as well as the number of epochs reached before the nonzero convergence limit is reached. However, these trials are considered to be optimal for the following reasons:

1. The images have more “global” Style features as the Style configuration moves from A-E. Style A has smaller, local features (consisting primarily of dots). Conversely, Style E has larger, global features, including texture and broad brush strokes.
2. Normalization successfully reduces the condition number by orders of magnitude for every feature map within every trial
3. The post-normalized condition numbers are among the smallest achieved, and not a single post-normalized condition number is at or near infinity
4. The VGG-13 is chosen over the VGG-16 model (which achieves similar results when keeping all other values constant), because it is less complex - it has fewer convolutional layers which results in better computational performance.

3.1 Methods of Normalization

All methods of Normalization are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm with a Strong Wolfe condition on curvature. The 19 weight-layer variant of the Visual Geometry Group’s model is used for all trials. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.1.1 Generated Images

When the activations of each convolutional layer are not normalized prior to generating Gram matrices, the output images are very different from that of the original implementation. Many of the images appear mostly similar, and there is an unusual tint of grayish-yellow. The first convolutional layer of the first convolutional block, *conv1_1*, is

not successfully learned - as indicated by the output of Style A representing pure noise. Because *conv1_1* is used in every style configuration, the lack of successful learning of *conv1_1* may explain the varying results in Style B - Style E.

Pulling towards the mean by channel has results that are similar to the results of not normalizing at all. Styles B - E have few distinguishing characteristics, and also has the grayish-yellow tint throughout the image. Style A also represents pure noise, indicating that the style was not successfully learned.

Min-Max normalization has a detrimental effect for all Style configurations. Every Style configuration results in a generated image that represents pure noise.

Normalizing the activations by the standard deviation, and its variant which normalizes over each channel of the activation, have results that most clearly represent the results of the original implementation. Style A consists of small, local features - the same colors of the input image are represented, but the generated image consists of mostly small dots. Each successive Style configuration represents larger, global features - the texture of the input image is more clearly represented, and features which mimic brush strokes replace the dots found in Style A.

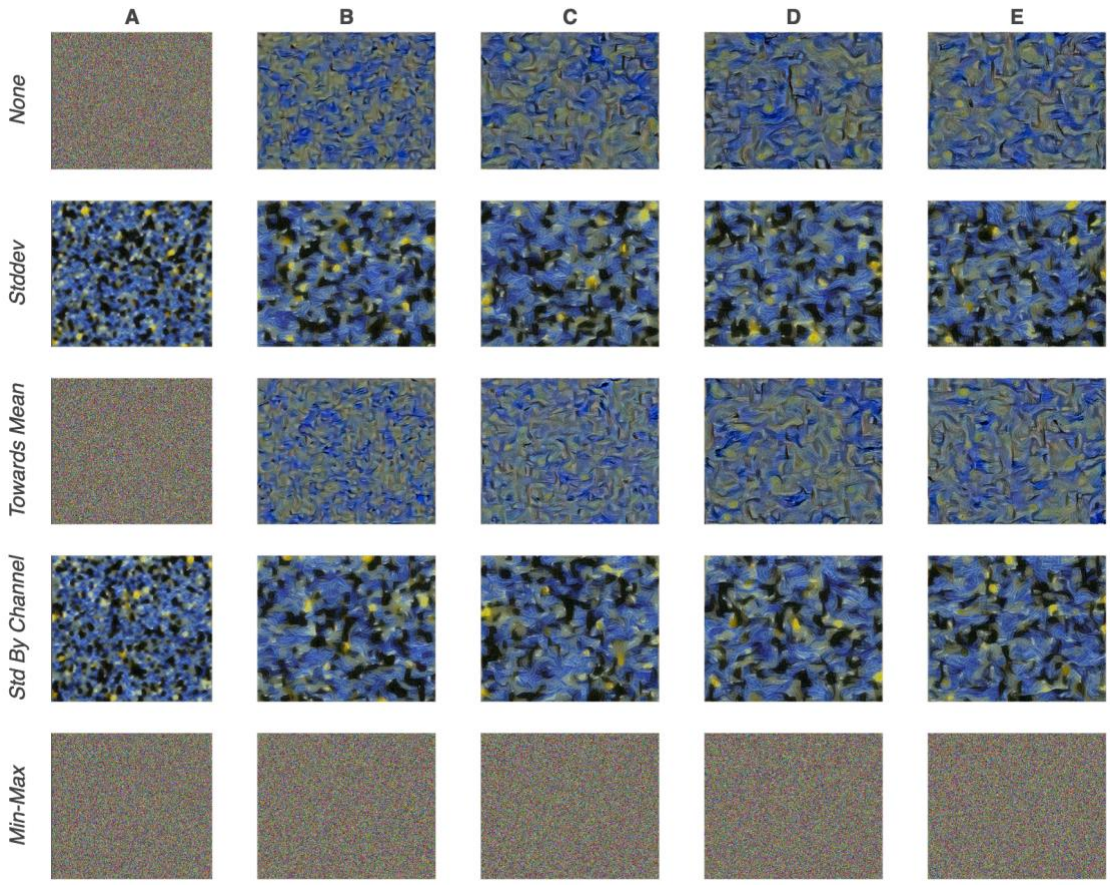


Figure 3. Generated Images of Various Normalization Methods

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature

3.1.2 Loss

Style A of the trials without any normalization method does appear to learn the style through the first 20 epochs - although a nonzero convergence limit is quickly reached at around 22 epochs. This results in the noisy image as illustrated above. Each successive trial has an increasing amount of epochs before a nonzero convergence limit is

reached. This same behavior is seen in the method of normalization which pulls each element towards the mean of its respective channel.

The min-max normalization method results in a mostly flat loss - there is some small amount of change that occurs within the first 50 epochs (as demonstrated by the total number of epochs reaching 100 before the nonzero convergence check resulted in an early stop), but the changes are small enough that the plots appear to be entirely flat.

Style A and B appear to have a better loss behavior when normalizing by the standard deviation when compared to no normalization - the nonzero convergence limit is not reached until epoch 200 for both configurations. The loss of Style C, Style D, and Style E does not indicate on its own that the style was learned better than the trials that did not use normalization - this indicates that loss, on its own, cannot be used to determine success of Neural Style Transfer.

An interesting characteristic of the methods which normalize by standard deviation is that conv1_1 is not learned at the same rate as subsequent activations - there is a noticeable gap in the loss of conv1_1 when compared to the other loss histories. This phenomenon does not occur in the trials which are not normalized.

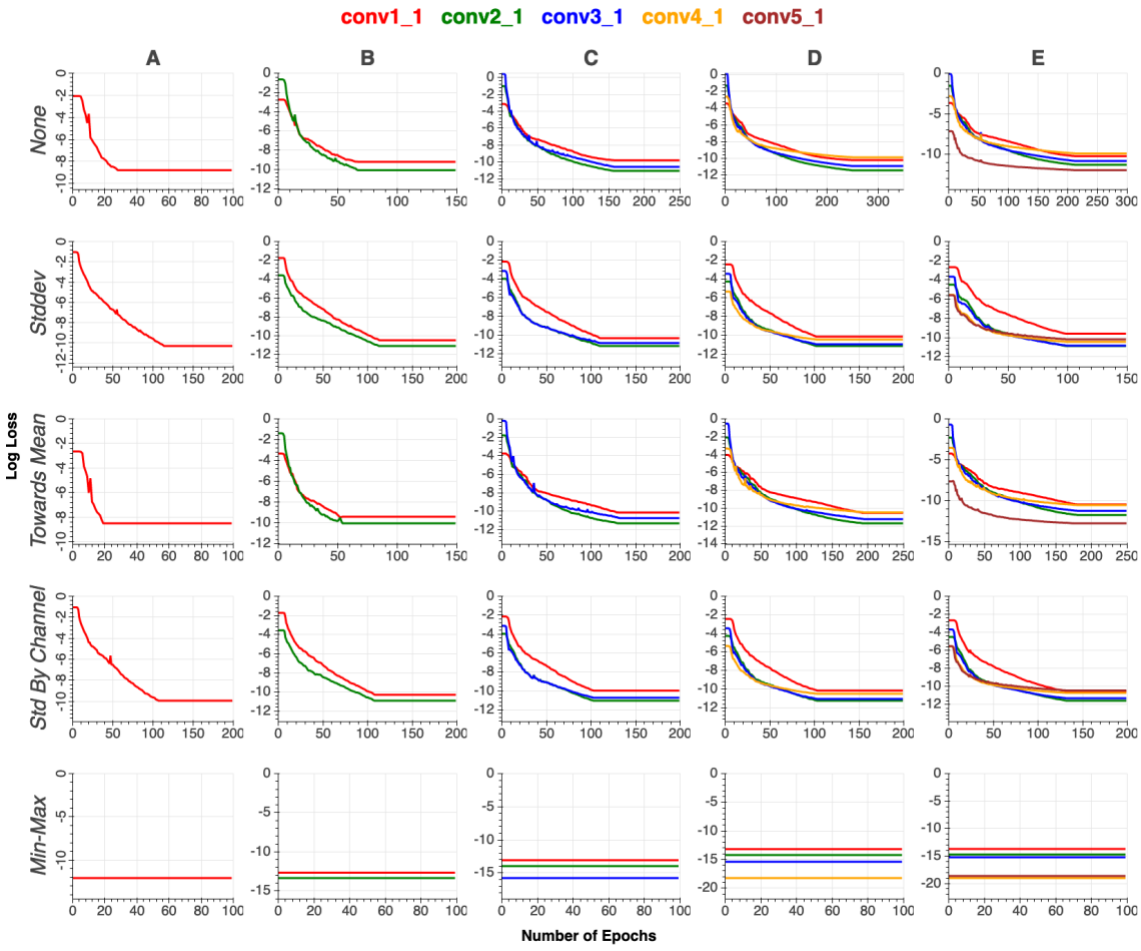


Figure 4. Loss Characteristics of Various Normalization Methods

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

3.1.3 Condition Numbers & Eigenvalues

There is a relationship between methods that generate a final image which is visually-appealing and methods that effectively reduce the condition number of the Gram matrices.

The methods of normalization that result in visually unappealing output images - *No Normalization*, *Pull Towards Mean By Channel*, and *Min-Max Normalization* - have feature maps that support the notion that the target image is not being learned successfully. The condition numbers after the normalization are of the same magnitude as the condition numbers before the normalization.

By contrast, the methods of normalization that result in visually appealing output images - *Normalize By Standard Deviation* and *Normalize By Standard Deviation By Channel* - have condition numbers that are many magnitudes smaller after the normalization. *Normalize By Standard Deviation By Channel* results in a Gram matrix that has condition numbers less than 1,000 for conv2_1 and conv3_1 for Style B and Style C configurations. This is especially noteworthy because all other methods of normalization have condition numbers at or near infinity for these same style configurations. While all methods include at least one style configuration which contains a Gram matrix with an infinite condition number, the methods which produce visually appealing images consistently lower (by many degrees of magnitude) condition numbers for every feature map.

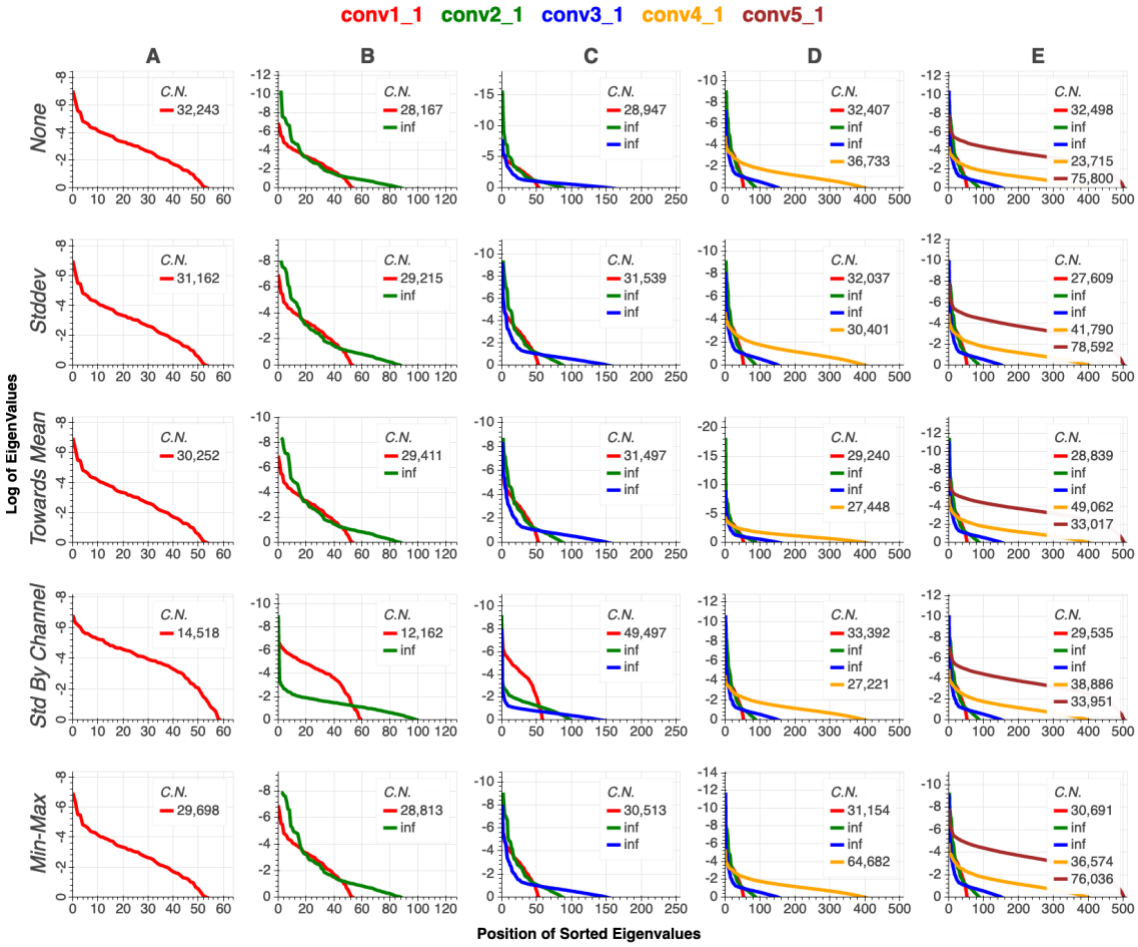


Figure 5. Eigenvalues & Condition Numbers of Various Normalization Methods Before Applying Normalization

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. C.N. refers to condition number.

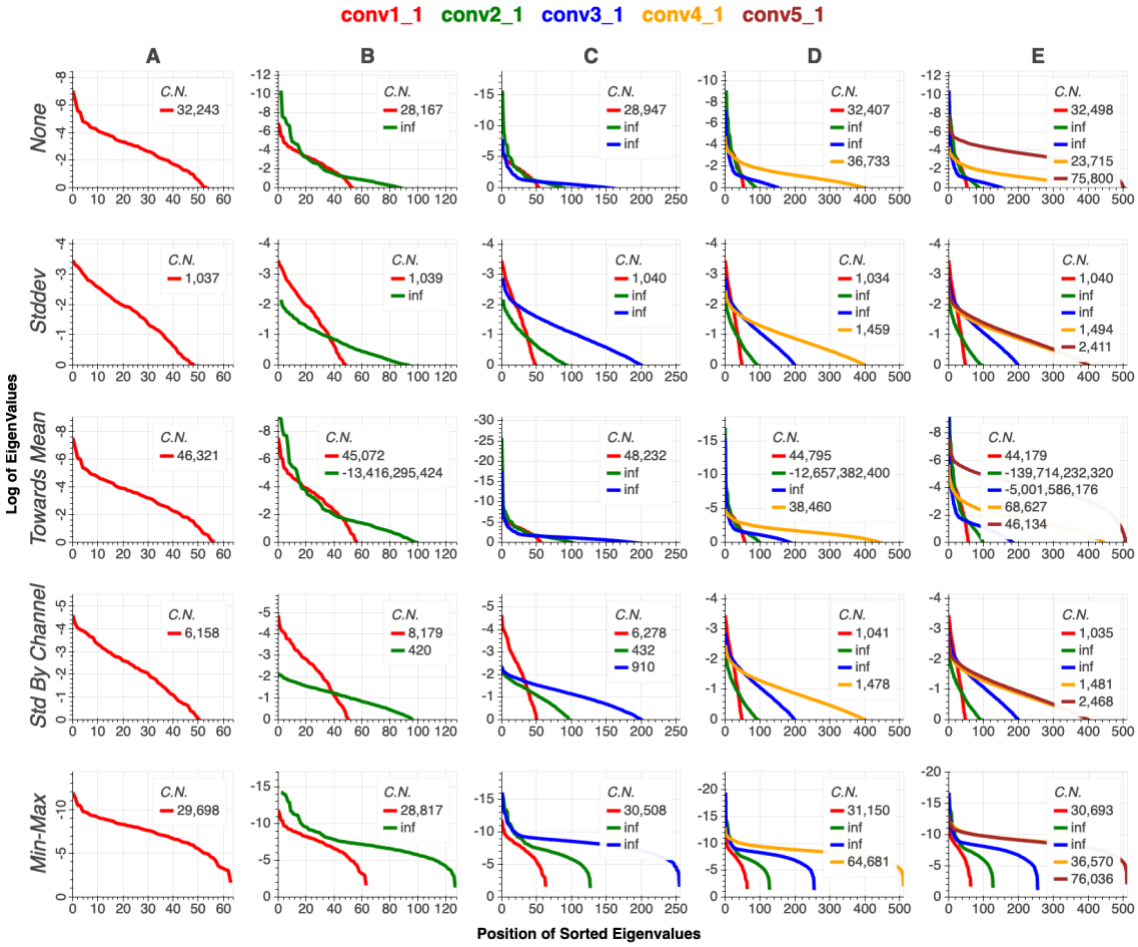


Figure 6. Eigenvalues & Condition Numbers of Various Normalization Methods After Applying Normalization

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. C.N. refers to condition number.

3.1.4 Influence of Future Trials

All of the results outlined above suggest that normalizing the feature maps by dividing by the standard deviation across the channels results in the best outcomes - the generated images are more visually appealing, and the reduction of its feature map's

condition numbers indicate that the normalization is working effectively. This method of normalization will be used as the baseline by which all other trials are implemented.

3.2 Methods of Pooling

Because max-pooling is non-continuous, it may be more difficult to effectively find the gradient. The original implementation of Neural Style Transfer replaces all of the max-pooling layers with average-pooling layers for this reason - stating that the gradient flow is improved when using average-pooling.

This section aims to test this hypothesis explicitly. All trials are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm with a Strong Wolfe condition on curvature. The 19 weight-layer variant of the Visual Geometry Group's model is used for all trials. The feature maps are normalized by dividing by the standard deviation over channels. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.2.1 Generated Images

While the generated images are visually similar when using the Style A configuration, the images generated diverge between methods of pooling for the style configurations deeper in the network. Average-pooling has results with an increasing range of learned features, whereas max-pooling generates images with primarily localized features. This is most noticeable on Style C - which has larger features and textures associated with average-pooling vs max-pooling

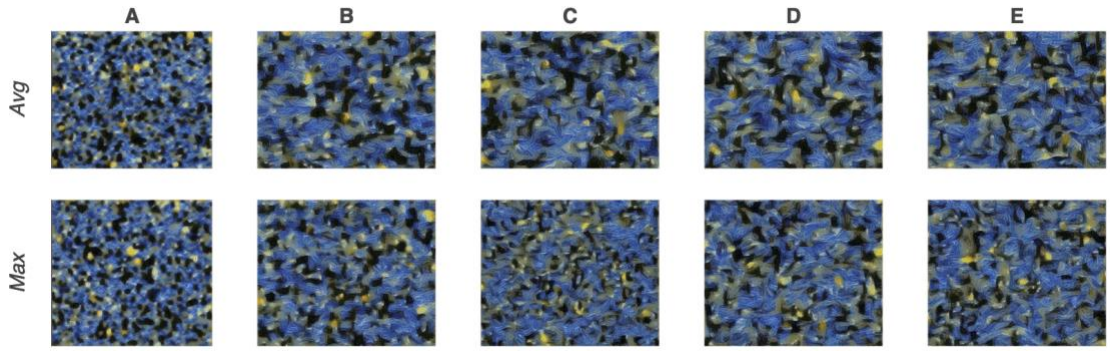


Figure 7. Generated Images of Average-Pooling & Max-Pooling

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels.

3.2.2 Loss

Both methods reach a nonzero convergence limit near the same amount of epochs and have similar final losses - both in terms of degree of loss, and with regard to the relationship between loss and respective convolutional layers. The most notable difference between the lost characteristics can be observed for Style C and Style D - both of which have a volatile loss associated with conv3_1 for max-pooling. This may support the notion that gradient flow is improved when replacing max-pooling layers with average-pooling

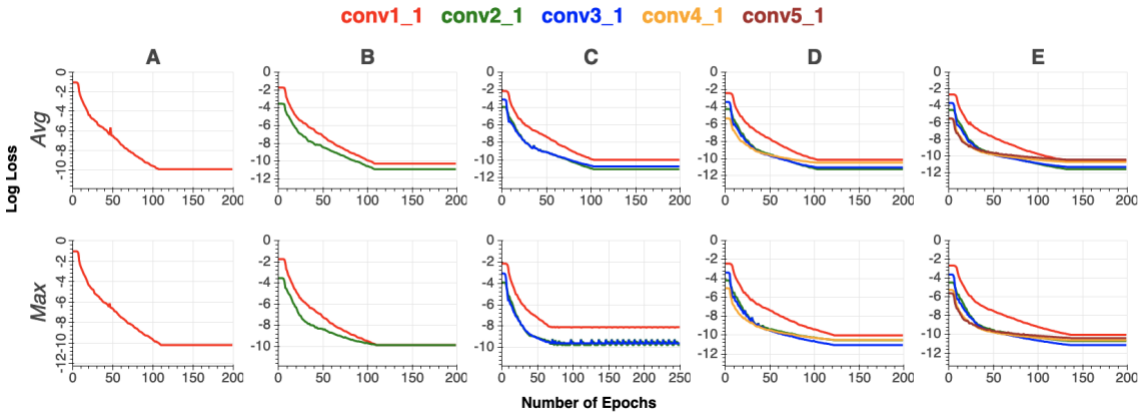


Figure 8. Loss Characteristics of Average-Pooling & Max-Pooling

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

3.2.3 Condition Numbers & Eigenvalues

After normalizing the feature maps by dividing by the standard deviation over the channels, average-pooling has fewer observations with a condition number at infinity. Notably, Max-pooling has an infinite condition number for conv2_1 on Style B, as well as an infinite condition number for both conv2_1 and conv3_1 on Style C. Average-pooling, by contrast, has a condition number below 1,000 for every convolutional layer for Style B and Style C.

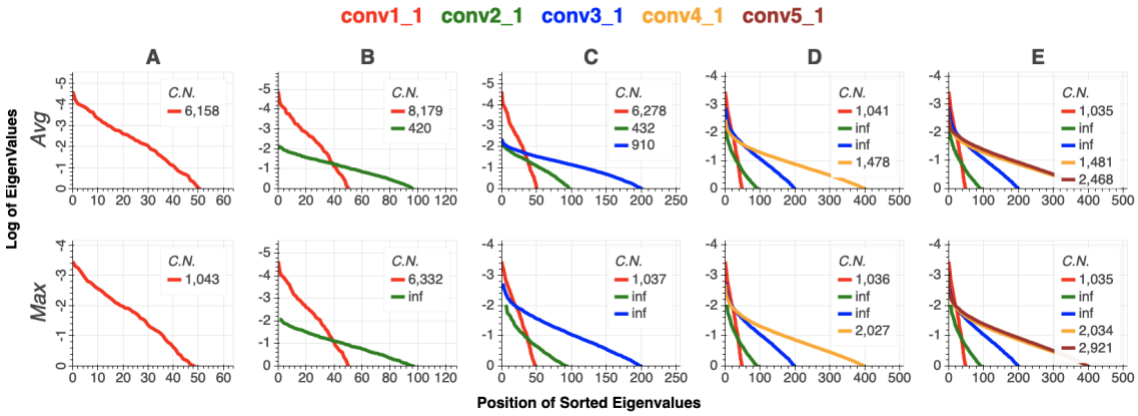


Figure 9. Post-Normalized Eigenvalues & Condition Numbers of Average-Pooling & Max-Pooling

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. C.N. refers to condition number.

3.3 Methods of Optimization

Although Gatys et al. (2015) refers only broadly to "gradient descent", the author has indicated online that the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno algorithm achieves quality results. This section compares this method to Stochastic Gradient Descent and Adaptive Movement Estimation.

The 19 weight-layer variant of the Visual Geometry Group's model is used for all trials. The feature maps are normalized by dividing by the standard deviation over channels. A learning rate of 1.0 is used for all trials (Note: additional learning rates were evaluated for SGD as per *Table 7* and Adam as per *Table 8* but did not lead to different results). Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.3.1 Generated Images

Adam and L-BFGS have results that are visually similar to one another - both optimization methods transition between learning local features (at the earlier layers of the network) to global features (at the later layers of the network). Stochastic Gradient Descent generates an image that represents pure noise - indicating that very little information was learned about the representation of the target image. This phenomenon is present throughout all trials; modifying the learning rate does not have an effect on the ability to learn the style features.

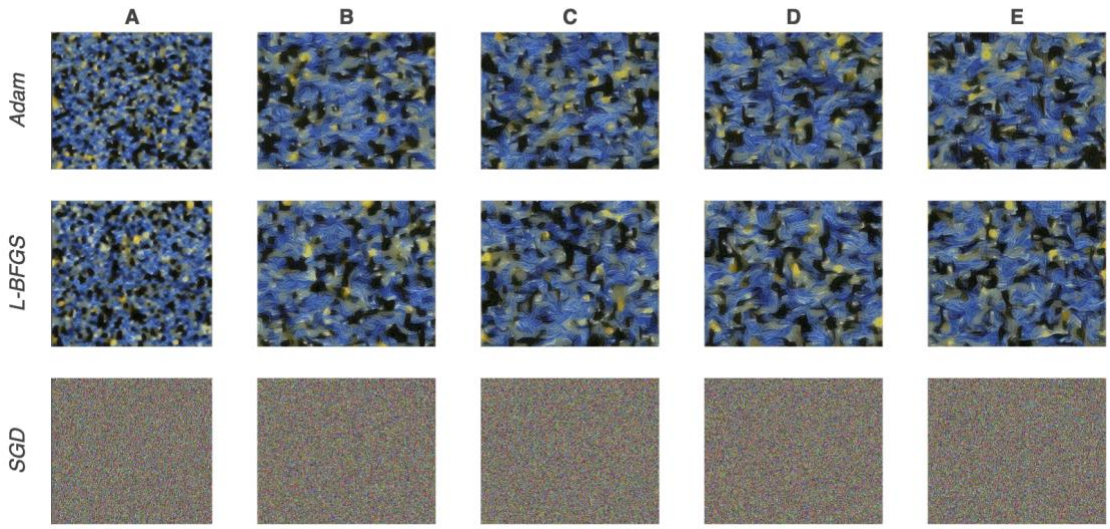


Figure 10. Generated Images of Various Optimization Methods

All trials generate their feature maps using the VGG-19 model. All trials have feature maps that are normalized by dividing by the standard deviation over the channels.

3.3.2 Loss

Adam optimization never reaches a nonzero convergence limit through the duration of all epochs. There is some degree of volatility associated with the loss in the later epochs, which can be explained by the dynamic nature of the learning rate.

Despite never learning the features, Stochastic Gradient Descent never reaches a nonzero convergence limit. Although the loss history appears to be mostly flat, there is enough variance between epochs such that an early stop is not triggered.

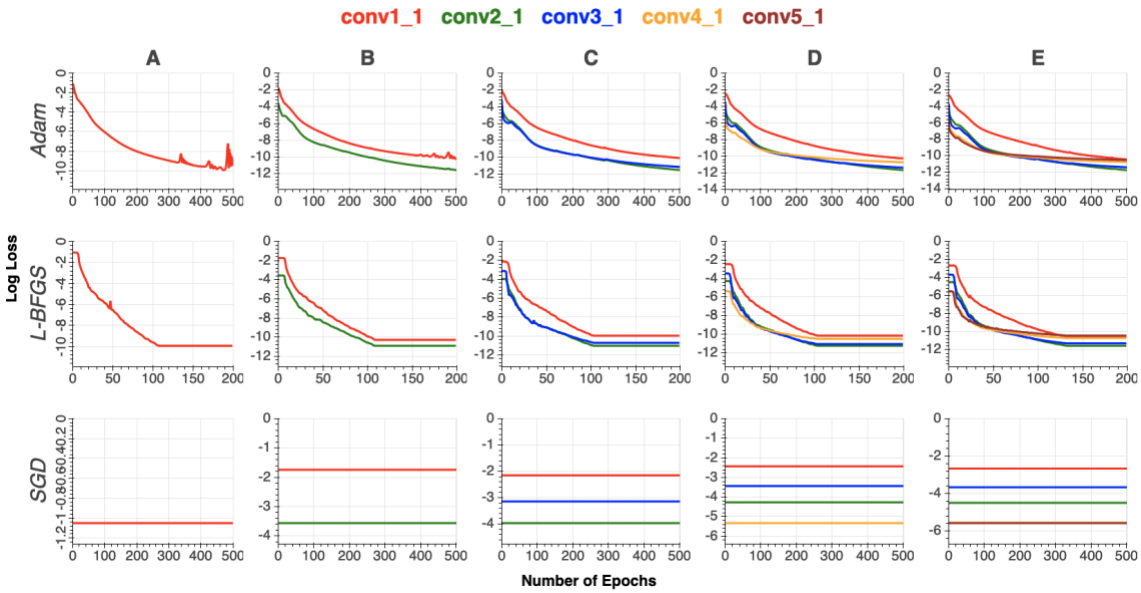


Figure 11. Loss Characteristics of Various Optimization Methods

All trials generate their feature maps using the VGG-19 model. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

3.3.3 Condition Numbers & Eigenvalues

L-BFGS optimization has fewer infinite condition numbers than Adam optimization. Excluding infinite values, Adam optimization has lower condition numbers, although they are within the same magnitude. Stochastic Gradient Descent has condition numbers that are similar to that of Adam optimization, despite having an output image consisting primarily of noise. This suggests that feature-map stability on its own is not enough for an optimizer to learn the style features.

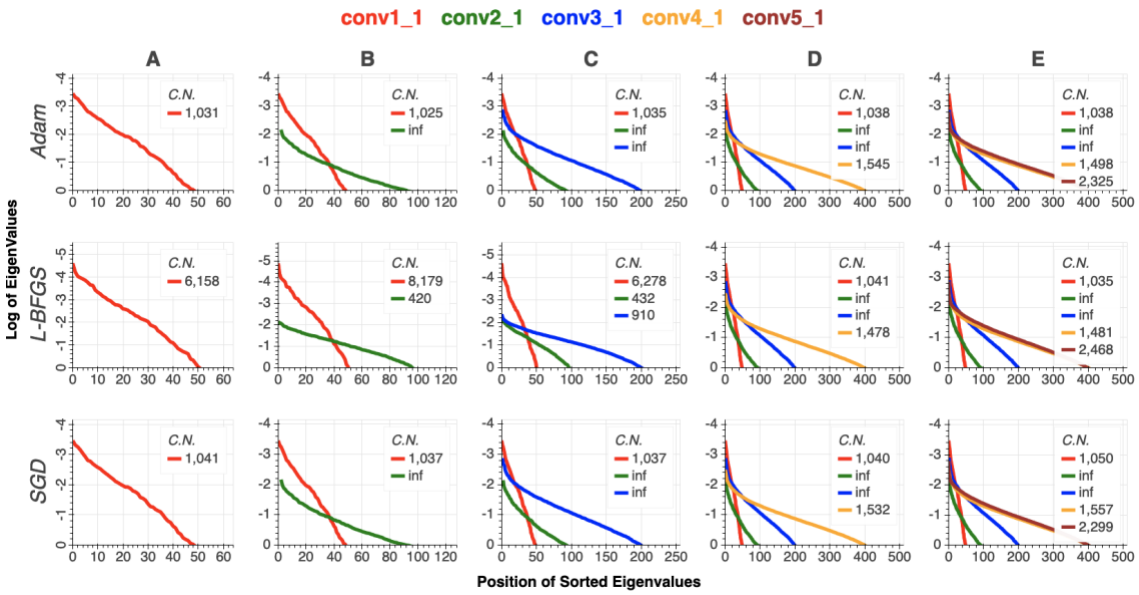


Figure 12. Post-Normalized Eigenvalues & Condition Numbers of Various Optimization Methods

All trials generate their feature maps using the VGG-19 model. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. C.N. refers to condition number

3.4 Additional Variants of the Visual Geometry Group's Models

While there is a relationship between the depth of each network and its corresponding top-1 and top-5 score, each of the weight-layer variants of the Visual Geometry Group's models have feature maps that appear similar to one another.

Each of the models have a varying amount of convolutional layers, but each model follows the same basic structure - five convolutional blocks separated by pooling layers, followed by two fully-connected layers. Each of the trials presented generate the feature map via the activations of the first convolutional layer of each convolutional block.

All trials are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm with a Strong Wolfe condition on curvature. The feature maps are normalized by dividing by the standard deviation over channels. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.4.1 Generated Images

The same pattern is observed across all variants of the model, with localized features being present in the earlier style configurations, and more global style features becoming present as the feature maps go deeper and deeper into the networks. Few discernable differences can be spotted across any of the model variants.

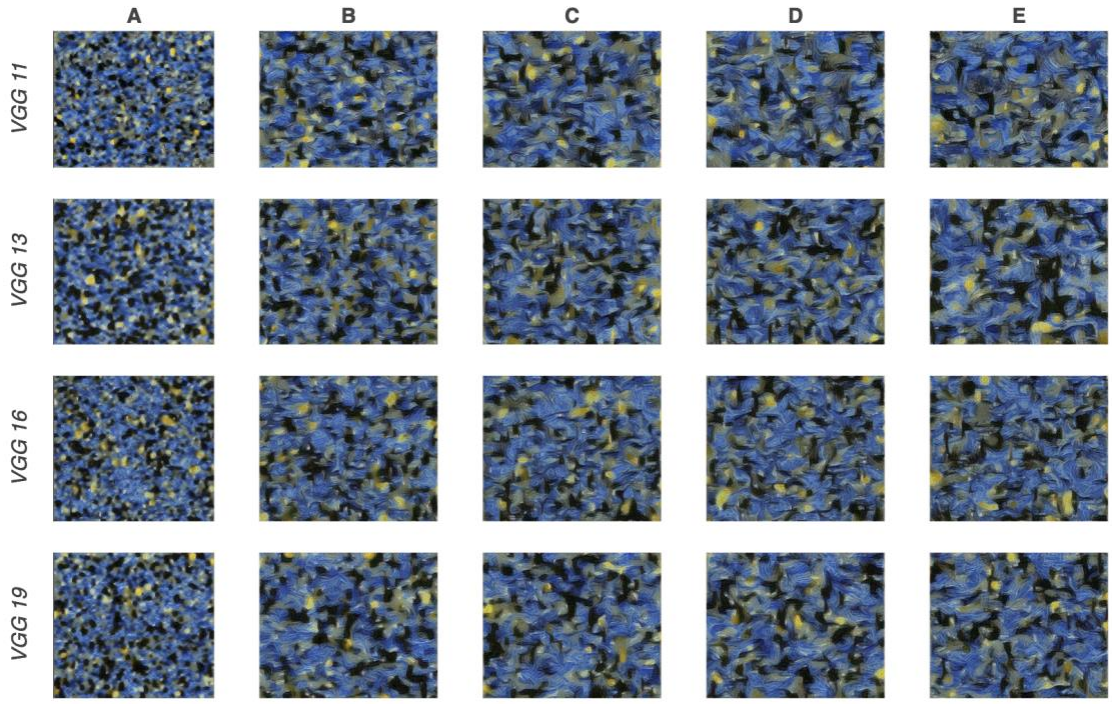


Figure 13. Generated Images of VGG Variants

All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels.

3.4.2 Loss

None of the trials reach 500 epochs; each trial reaches a nonzero convergence limit around the 200-epoch mark. The same general loss behavior is observed across all variants of the model - nonzero convergence limits are reached around the same point, the final loss is similar, and the relationship between convolutional layers is similar across all model variants.

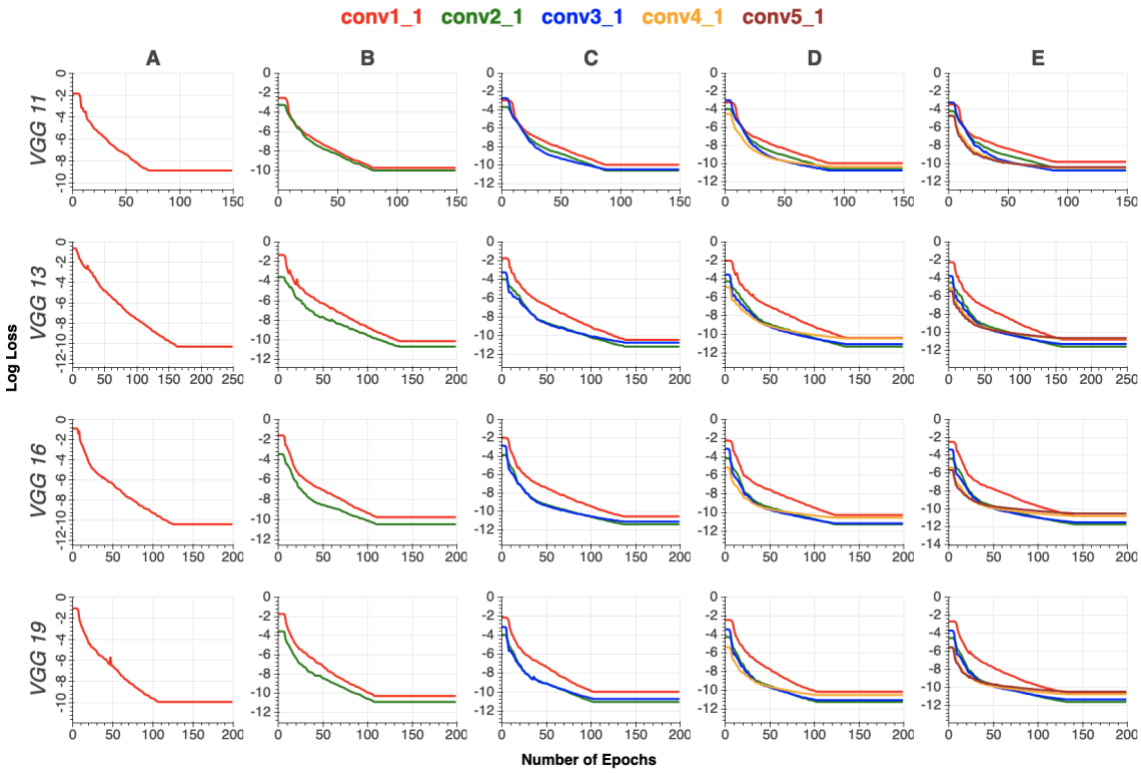


Figure 14. Loss Characteristics of VGG Variants

All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

3.4.3 Condition Numbers & Eigenvalues

The largest difference between trials is observed through the condition numbers of each set of feature maps. Most of the trials thus far have feature maps which contain an infinite condition number for conv2_1 and conv3_1 for Style D and Style E - even after normalization has been applied. VGG 13 and VGG 16, however, have condition numbers less than 1,000 for all Style Configurations. This behavior is not consistent, however -

VGG 13 and VGG 16 have a conv1_1 condition number that is consistently 2-10 times larger than their VGG-11 and VGG-19 equivalents.

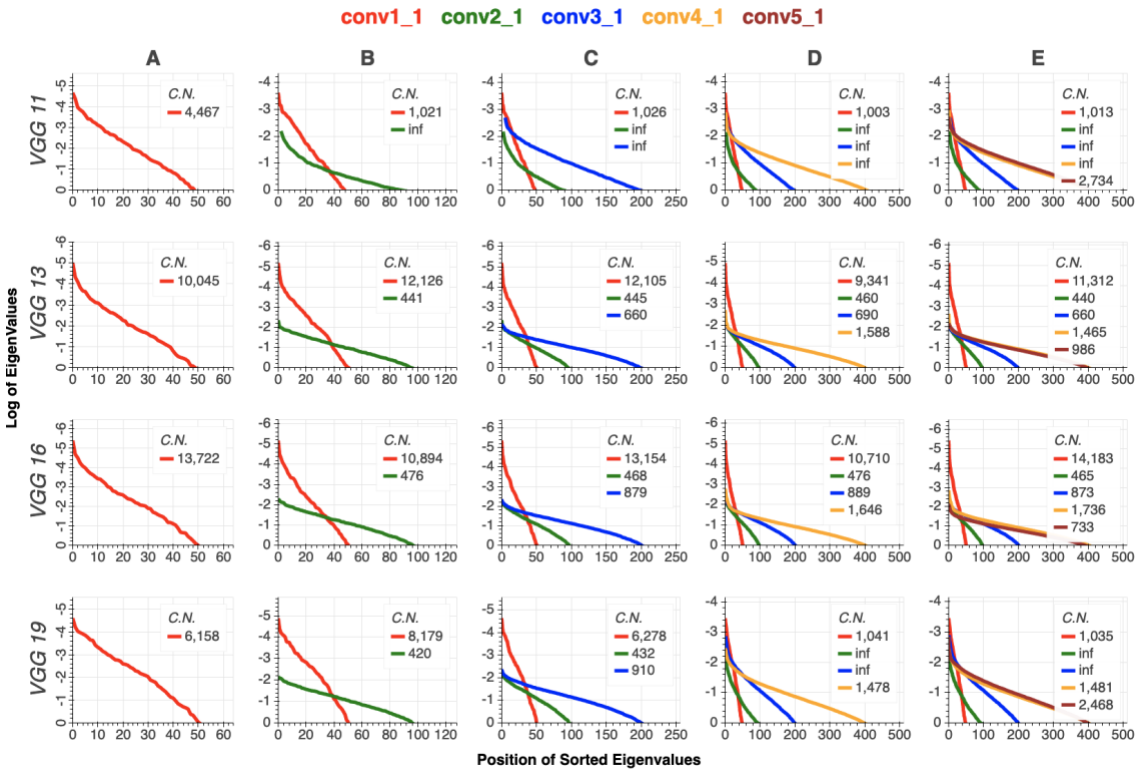


Figure 15. Post-Normalized Eigenvalues & Condition Numbers of VGG Variants

All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. C.N. refers to condition number

3.5 Effect of Wolfe Condition on Curvature

One of the most crucial components of convergence appears to be the Wolfe condition on curvature. Because of the Hessian approximation associated with the L-BFGS algorithm, the Wolfe condition proves to be critical in convergence.

All trials are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm. The 19 weight-layer variant of the Visual Geometry Group's model is used for all trials. The feature maps are normalized by dividing by the standard deviation over channels. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.5.1 Generated Images

Although the later style configurations - Style C, Style D, and Style E - have a final output image that is visually similar regardless of the Strong Wolfe constraint, Styles A and Styles B do not appear to have learned any of the stylistic representations.

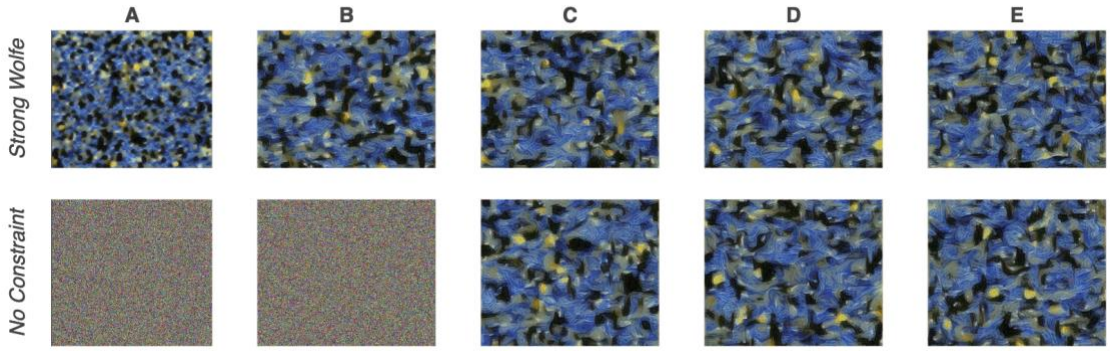


Figure 16. Generated Images of Strong Wolfe and No Line Search

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS. All trials have feature maps that are normalized by dividing by the standard deviation over the channels.

3.5.2 Loss

Despite some of the final output images being visually similar, the loss takes on very different behavior. This is most evident with Style D - when the Strong Wolfe constraint is not imposed, nearly 100 epochs pass before the algorithm begins to learn the feature space.

The loss behaviors for Style A and Style B support the notion that the stylistic features are not effectively learned without the Strong Wolfe constraint. However, similar to the results of SGD, there is just enough variance between epochs such that the early stopping conditions are not met.

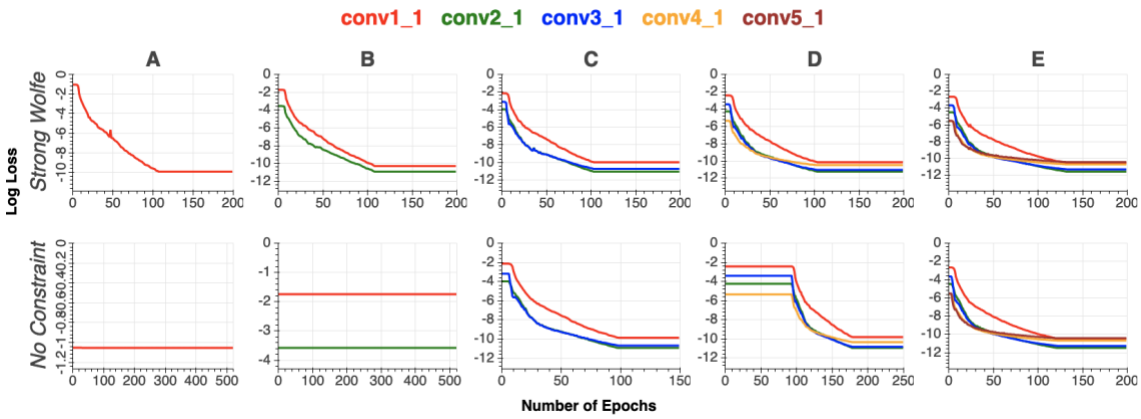


Figure 17. Loss Characteristics of Strong Wolfe and No Line Search

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

3.5.3 Condition Numbers & Eigenvalues

There are few differences in post-normalized condition numbers - the condition numbers for conv2_1 and conv3_1 reach infinity at earlier style configurations when the Strong Wolfe constraint is not enforced, but the same general pattern can be observed across the remainder of the trials.

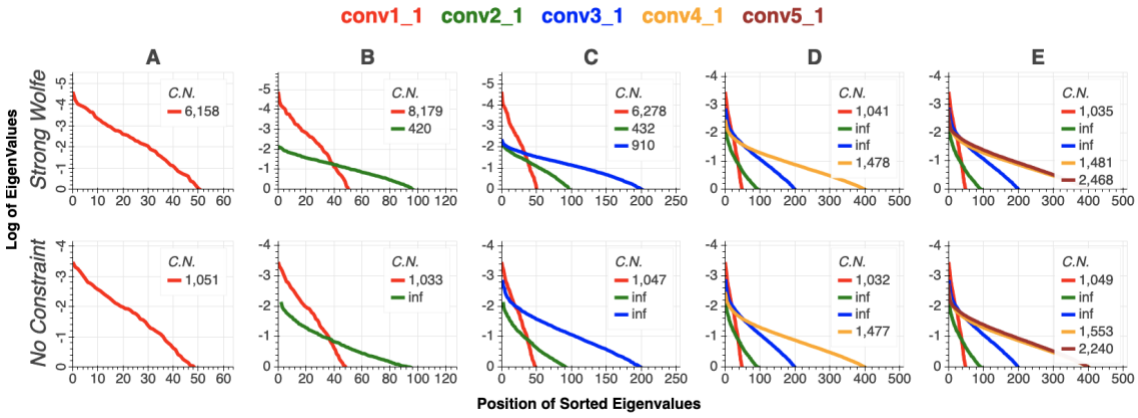


Figure 18. Post-Normalized Eigenvalues & Condition Numbers of Strong Wolfe and No Line Search

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. C.N. refers to condition number

3.6 Feature Masks

None of the methods of applying a feature mask appear to have any notable effect on the output; normalization is much more effective with regard to improving gradient flow. All trials are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm with a Strong Wolfe condition on curvature. The 19 weight-

layer variant of the Visual Geometry Group's model is used for all trials. The feature maps are not normalized. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.6.1 Generated Images

There are no notable differences between any of the images when applying various feature masks. If normalization is not applied, Style A cannot be effectively learned, and results in an output image representing pure noise. Styles B - E have the presence of a strange yellow-gray tint, and the output images do not appear to have any degree of increasing global features as the Styles increase from Style A through Style E.

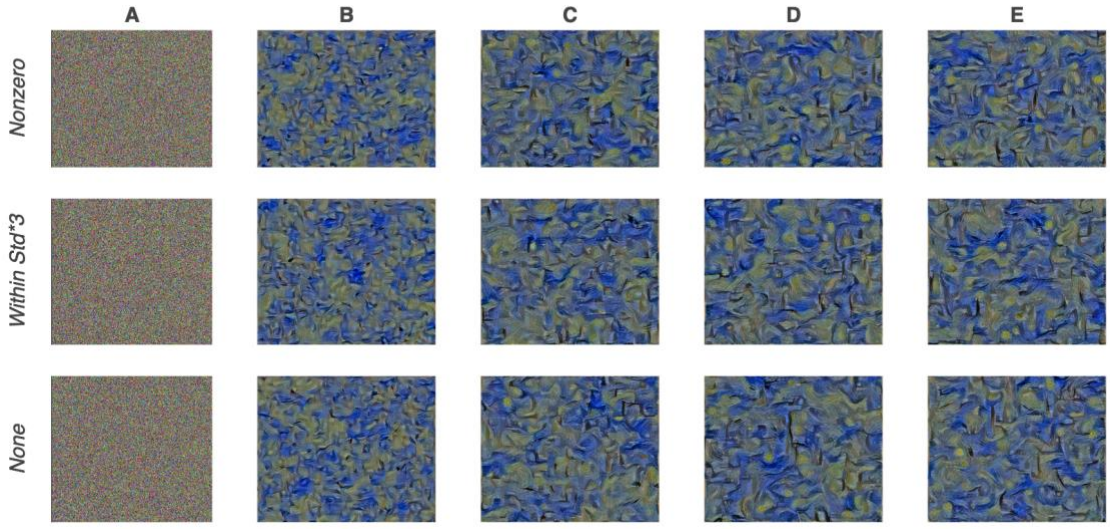


Figure 19. Generated Images of Various Feature Masks

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. Nonzero refers to only calculating

*the mean squared error with respect to activations that are greater than zero. "Within Std*3" refers to only calculating the mean squared error with respect to activations that are within 3 standard deviations of the mean.*

3.7 Range of Input Images

An interesting characteristic that may be unique to the PyTorch implementation of the Visual Geometry Group's models is that the range of input images seems to have an effect on the convergence properties of the Style Loss functions.

The PyTorch implementation is trained on images that are in the range of [0, 1] - but seem to also accept images that are in the range of [0, 255]. Images in the range seem to converge without any adjustments to normalization.

All trials are evaluated using the Limited-Memory variant of the Broyden-Fletcher-Goldfarb-Shanno Algorithm with a Strong Wolfe condition on curvature. The 19 weight-layer variant of the Visual Geometry Group's model is used for all trials. The feature maps are normalized by dividing by the standard deviation over channels. Each trial consists of 500 epochs, unless a nonzero convergence limit is reached.

3.7.1 Generated Images

Images in the range of [0, 255] have more visually appealing results when normalization is not applied. The generated images are similar to that of other trials where normalization is applied. Two key differences in these generated images is that they are a darker shade of blue, and there are more pure-black features when compared to the normalized equivalents in the range of [0, 1].

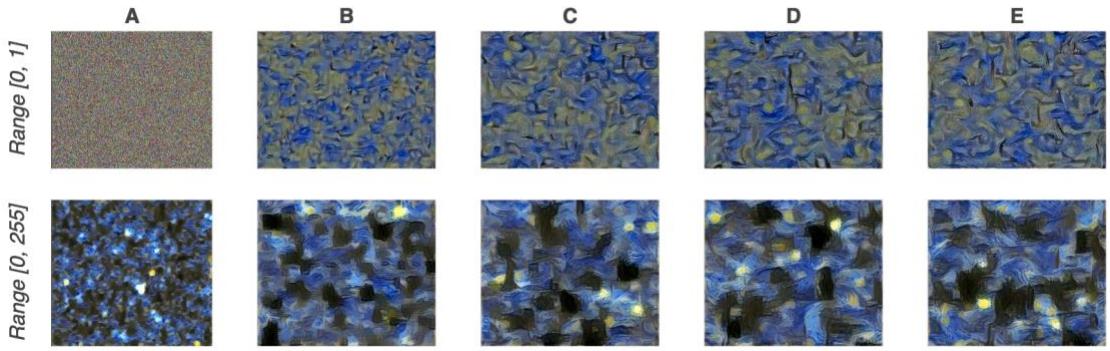


Figure 20. Generated Images of Various Image Ranges

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels.

3.7.2 Loss

A nonzero convergence limit is not reached when images in the range of [0, 255] are used, even without using any normalization techniques. The magnitudes of the loss are very different - the activations of each layer are observed to be much larger when the input images are in the range of [0, 255].

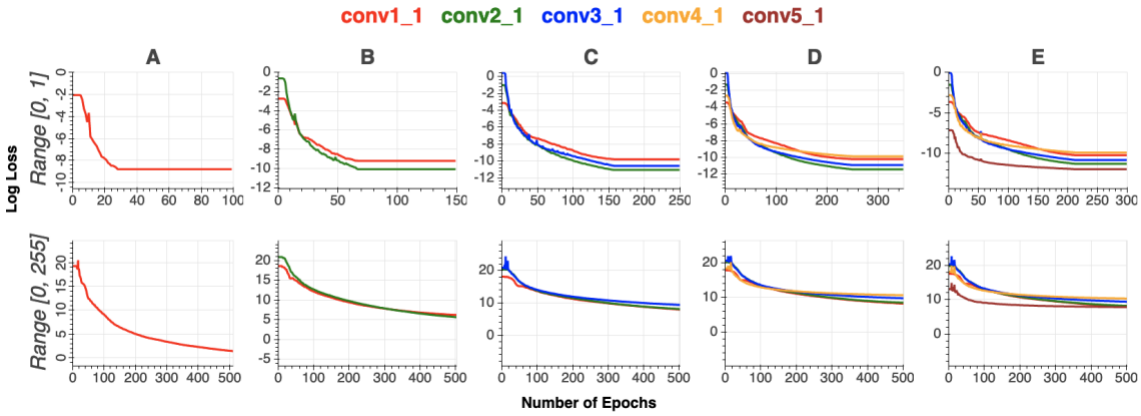


Figure 21. Loss Characteristics of Various Image Ranges

All trials generate their feature maps using the VGG-19 model. All trials optimize via L-BFGS with a Strong Wolfe Condition on Curvature. All trials have feature maps that are normalized by dividing by the standard deviation over the channels. Each x-axis is scaled in accordance with the number of epochs that occur before the nonzero convergence limit is reached.

Chapter IV.

Conclusion

This thesis evaluated multiple normalization methods, two methods of pooling, multiple methods of optimization, multiple variants of the VGG models, and multiple feature masks. The following table reflects the optimal configuration as determined by the results from all trials evaluated:

Table 11. Recap of Optimal Configurations by Category

Category	**Optimal Configuration**
Normalization Method	Divide by Standard Deviation By Channel
Pooling Method	Average
Optimization Method	L-BFGS with Wolfe Constraint
VGG Variant	VGG-13
Feature Mask	None

Using this this optimal configuration results in the following set of images:

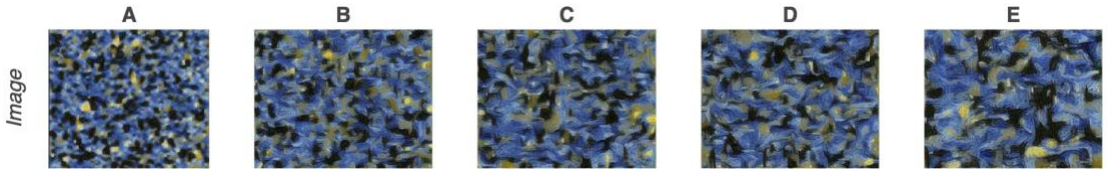


Figure 22. Output Images of Optimal Configuration

Many interesting characteristics can be observed in the loss behavior of Neural Style Transfer. The ill-conditioned nature of the Gram matrices can lead to convergence problems. This behavior can be solved by applying normalization before calculating the mean squared error - leading to results that are both more visually appealing, as well as not reaching a nonzero convergence limit as early. Conversely, applying a feature mask before calculating the mean squared error does not appear to have any effect on optimization.

Replacing max-pooling layers with average-pooling layers does seem to improve gradient flow. Average-pooling results in images with increasingly global features and has a loss function that is less volatile.

Although a specific optimization method is not mentioned by Gatys et al. (2015), it seems to play a large role. Stochastic Gradient Descent is unable to reach convergence regardless of its learning rate, whereas Adam optimization and L-BFGS are able to achieve much better results. Because of the Hessian approximation of the L-BFGS algorithm, a Strong Wolfe condition on curvature leads to better results - and the lack of such a condition leads to strange loss behavior.

The weight-layer variants of the Visual Geometry Group's models have a large degree of variance when measuring their corresponding top-1 and top-5 error rate, but their results are very similar when generating feature maps in order to capture the stylistic features of a given image. Of the four models evaluated, the 13 weight-layer variant seems to have the best results - it has fewer layers than the 16 and 19 variants (reducing computational complexity), its generated images capture more global features in Style D

and Style E, and its condition numbers do not reach infinity across any of its Style configurations.

Throughout the vast majority of the trials, a nonzero convergence limit is reached before reaching 500 epochs. Despite this phenomenon, visually appealing results can be generated - suggesting that loss on its own is not a suitable measurement of success of learning the style of an image.

References

Broyden, C. G. “The Convergence of a Class of Double-Rank Minimization Algorithms 1. General Considerations.” *IMA Journal of Applied Mathematics*, vol. 6, no. 1, Mar. 1970, pp. 76–90. *Silverchair*, doi:10.1093/imamat/6.1.76.

Deng, J., et al. “ImageNet: A Large-Scale Hierarchical Image Database.” *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–55. *IEEE Xplore*, doi:10.1109/CVPR.2009.5206848.

Fletcher, R. “A New Approach to Variable Metric Algorithms.” *The Computer Journal*, vol. 13, no. 3, Jan. 1970, pp. 317–22. *Silverchair*, doi:10.1093/comjnl/13.3.317.

Gatys, Leon A., et al. “A Neural Algorithm of Artistic Style.” *ArXiv:1508.06576 [Cs, q-Bio]*, Sept. 2015. *arXiv.org*, http://arxiv.org/abs/1508.06576.

Goldfarb, Donald. “A Family of Variable-Metric Methods Derived by Variational Means.” *Mathematics of Computation*, vol. 24, no. 109, Jan. 1970, pp. 23–23. *DOI.org (Crossref)*, doi:10.1090/S0025-5718-1970-0258249-6.

Kingma, Diederik P., and Jimmy Ba. “Adam: A Method for Stochastic Optimization.” *ArXiv:1412.6980 [Cs]*, Jan. 2017. *arXiv.org*, http://arxiv.org/abs/1412.6980.

Liu, Dong C., and Jorge Nocedal. “On the Limited Memory BFGS Method for Large Scale Optimization.” *Mathematical Programming*, vol. 45, no. 1–3, Aug. 1989, pp. 503–28. *DOI.org (Crossref)*, doi:10.1007/BF01589116.

Robbins, Herbert, and Sutton Monro. “A Stochastic Approximation Method.” *The Annals of Mathematical Statistics*, vol. 22, no. 3, Sept. 1951, pp. 400–07. *DOI.org (Crossref)*, doi:10.1214/aoms/1177729586.

Shanno, D. F. “Conditioning of Quasi-Newton Methods for Function Minimization.” *Mathematics of Computation*, vol. 24, no. 111, Sept. 1970, pp. 647–647. *DOI.org (Crossref)*, doi:10.1090/S0025-5718-1970-0274029-X.

Simonyan, Karen, and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” *ArXiv:1409.1556 [Cs]*, Apr. 2015. *arXiv.org*, http://arxiv.org/abs/1409.1556.

Wolfe, Philip. “Convergence Conditions for Ascent Methods.” *SIAM Review*, vol. 11, no. 2, Apr. 1969, pp. 226–35. *DOI.org (Crossref)*, doi:10.1137/1011036.

Yamaguchi, Kouichi, et al. *A Neural Network for Speaker-Independent Isolated Word Recognition*. 1990, pp. 1077–80.