Multi-style Transfer: Generalizing Fast Style Transfer to Several Genres

Brandon Cui
Stanford University
bcui19@stanford.edu

Calvin Qi
Stanford University
calvinqi@stanford.edu

Aileen Wang
Stanford University
aileen15@stanford.edu

Abstract

This paper aims to extend the technique of fast neural style transfer to multiple styles, allowing the user to transfer the contents of any input image into an aggregation of multiple styles. We first implement single-style transfer: we train our fast style transfer network, which is a feed-forward convolutional neural network, over the Microsoft COCO Image Dataset 2014, and we connect this transformation network to a pre-trained VGG16 network (Frossard). After training on a desired style (or combination of them), we can input any desired image and have it rendered in this new visual genre. We also add improved upsampling and instance normalization to the original networks for improved visual quality. Second, we extend style transfer to multiple styles, by training the network to learn parameters that will blend the weights. From our work we demonstrate similar results to previously seen single-style transfer, and promising preliminary results for multi-style transfer.

1. Introduction

Earlier style transfer algorithms have the fundamental limitation of only using low-level image features of the target image to inform the style transfer [1, 2]. Only with recent advancements in Deep Convolutional Neural Networks (CNN) have we seen new powerful computer vision systems that learn to extract high-level semantic information from natural images for artistic style classification. In recent years we've seen the advent of Neural Style Transfer due to the intriguing visual results of being able to render images in a style of choice. Many current implementations of style transfer are well documented and produce good results using CNN, but have drawbacks in performance and are limited to learning a style from just one image and producing a single pre-trained style network. We hope to implement style transfer in a more generalized form that is fast to run and also capable of intelligently combining various styles.

1.1. Related Work

A number of research works have used optimization to generate images depending on high-level features extracted from a CNN. Images can be generated to maximize class prediction scores [17, 18] or individual features [18]. Mahendran and Vedaldi [19] invert features from CNN by minimizing a feature reconstruction loss; similar methods had previously been used to invert local binary descriptors [20, 21] and HOG features [22]. The work of Dosovitskiy and Brox [23] was to train a feed-forward neural network to invert convolutional and approximate a solution to the optimization problem posed by [19]. But their feed-forward network is trained with a per-pixel reconstruction loss. Johnson et al.[7] has directly optimized the feature reconstruction loss of [19].

The use of Neural Networks for style transfer in images saw its advent in Gatys et al., 2015 [3, 4, 5, 6]. This introduced a technique for taking the contents of an image and rendering it in the 'style' of another, including visual features such as texture, color scheme, lighting, contrast, etc. The result was at once visually stunning and technically intriguing, so in recent years many others have worked on refining the technique to make it more accurate, efficient, and customizable.

Johnson et al. 2016 [7, 8, 9] proposed a framework that includes a new specialized 'style transfer network' working in conjunction with a general CNN for image classification, which allows for the simultaneous understanding of an style and content in images so that they can be analyzed and transferred. This method is well documented and produces very good results, but the method still has drawbacks in performance and in being limited to learning a style from just one image and producing a single pre-trained style network.

Our goal in this project is first to understand the existing implementations of style transfer and the advantages/disadvantages of its many variations, then to devise a method extending one of these implementations so that the algorithm can have a more holistic understanding of 'style' that incorporates multiple images from a certain genre/artist rather than just one, and finally to implement our method fully and seek to optimize performance and accuracy along

the way.

2. Problem Definition

The goal of our project is to:

- Implement the most primitive form of Style Transfer based on iteratively updating an image and altering it to fit a desired balance of style and content. This can be framed as an optimization problem with a loss function as our objective to minimize through backpropagation onto the image itself.
- Improve upon the naive approach by implementing and training a feed forward Style Transfer network that learns a particular style and can convert any image to the style with a single forward pass (Johnson)
- Generalize this network to aggregate multiple styles and produce find the best combination of them without any manual specifications from the user
- Compare the results of these different approaches by analyzing both the visual qualities of the resulting images and numerical loss values

3. Data Processing

We will choose different type of datasets to test and validate our multi-style transfer algorithm:

- SqueezeNet for naive style transfer baseline
- VGG-16 and associated pre-trained ImageNet weights for loss network
- Microsoft COCO Image Dataset 2014 (80,000 images) for full training of our transfer network

4. Approaches

4.1. Baseline (Gayts et al. 2015)

The baseline implementation iteratively optimizes an output image (can start from blank pixels or random noise) and over time reaches a picture capturing the contents of one image in the style of another. It seeks to optimize a loss value that is a weighted sum of various Perceptual Loss Functions that allow us to mathematically compare the visual qualities of images. The details of these loss functions will be described in a later section.

While this method sufficiently accomplishes the basic task of transferring styles, it has various shortcomings. Primarily, it requires iteratively perturbing every input image through backpropagation, which is very slow. This also does not lead to an understanding of what exactly takes place in this transformation and merely runs as an separate optimization problem each time. It would be beneficial for our algorithm to consolidate style transfer into a series of operation that can be applied to an image instantly.

4.2. Fast Style Transfer Architecture (Johnson et al. 2016)

We design a feed-forward CNN that takes in an image and outputs one of the same size after a series of intermediate layers, with the output as the result of converting the original image to a chosen style. The network begins with padding and various convolution layers to apply filters to spatial regions of our input image, grouped with batch normalization and ReLU nonlinearity. These are followed by the same arrangement of layers but as a residual block, since we estimate that parts of the original image only need to be perturbed slightly from their original pixels. Then, upsampling is needed to restore the matrices into proper image dimensions. (We standardized the dimensions to 256×256, but this can be customized.) Our initial implementation follows Johnson's method of using fractional (transpose) convolution layers with stride $1/2$ for upsampling, which gets the job done but leads to some minor undesirable visual artifacts that will be addressed and improved next.

We connect this transformation layer to feed directly into a pre-trained VGG16 network (Frossard), which we use as a feature extractor that has already proven its effectiveness. This presents us with many choices regarding which layer(s) of the VGG network to select to represent image features and styles. In addition, since total style loss is a weighted sum of the style losses at different layers, we need to decide how much to weigh each.

After much experimentation, we chose the 'relu2_2' layer for features since it yielded reconstructions that most contained both broad and specific visual contents of the original image. The style layers were taken to be

['relu1_2', 'relu2_2', 'relu3_3', and 'relu4_3']

with weights [4, 1, 0.1, 0.1] respectively to capture a variety of high and low level image qualities.

Our total loss function is defined by:

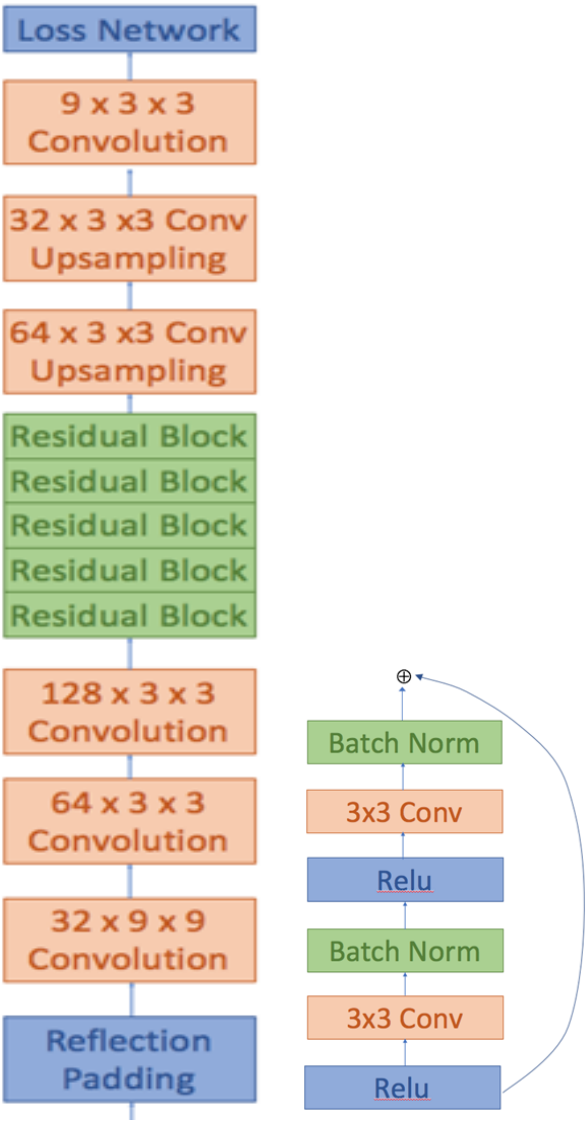
$$L = \lambda_c L_c + \lambda_s L_s + \lambda_{tv} L_{tv}$$

where these represent content, style, and total variation loss respectively, each with scaling weights as hyperparameters. These individual loss functions are described in much more detail below; essentially, content corresponds to the actual subject matter of the image, style represents the way it looks, and total variation measures similarity between neighboring pixels as a method for reducing noise. After extensive hyperparameter searching and tuning, we've found that in our implementation the best values are typically around

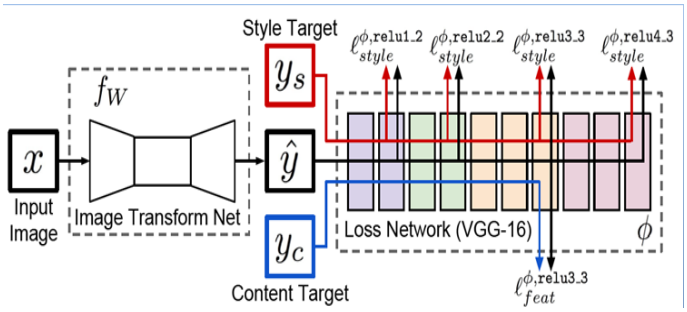
$$\lambda_c = 1.5, \quad \lambda_s = 5 \cdot 10^{-4}, \quad \lambda_{tv} = 3 \cdot 10^{-9}$$

We implemented the network in TensorFlow and trained it on 80,000 images from the Microsoft COCO 2014 dataset. Using minibatches of size 4 with two total epochs, training time is around 6 hours.

The resulting style transfer network can stylize images in less than a second, which is much faster than naive style transfer (See Figure 1 for the fast style transfer Architecture). However, it has the limitation of only being able to handle one chosen style fixed from the start. x‘



a) **b)**



c)

Figure 1: Neural Network Architecture for Style Transfer **a)** Image Transform Net **b)** Residual Connections **c)** Loss Network (Johnson et al. 2016)

4.3. Improved Upsampling (Google, 2016) and Instance Normalization (Ulyanov, 2016)

We found that the fractional convolution layers in Johnson's network produced images where pixels were loosely arranged in blocks, leading to slightly checkered visual patterns despite the overall result being visually convincing and effectively capturing the transfer in style. These were modified to use nearest neighbor upsampling instead, which gave much smoother results by avoiding the issue of box-shaped filters resulting from fractional convolution layers.

Also, we replace batch normalization with instance normalization. When training, we want the task of stylization to be viewed as a task performed on individual images rather than the entire batch, which is accomplished using instance normalization. The formula no longer takes an average across the entire batch, so our instance normalization rule is:

$$y_{tijk} = \frac{x_{tijk} - \mu_{ti}}{\sqrt{\sigma_{ti}^2 + \epsilon}}, \quad \mu_{ti} = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H} x_{tilm}$$

$$\sigma_{ti}^2 = \frac{1}{HW}\sum_{l=1}^{W}\sum_{m=1}^{H}(x_{tilm} - \mu_{ti})^2$$

After this change, we find that the new transformations applied are more effectively account for contents and sections of the image itself rather than appearing in repeated patterns throughout.

4.4. Multiple Styles

We simultaneously train multiple networks for multiple styles. For each network, we train it to blend combinations of styles. This blending can be made into learnable parameters.

4.5. Optimizing Total Loss

For all cases, we utilized the Adam optimizer to minimize total loss with learning rate 0.001. The loss is defined as a weighted sum of the following:

4.5.1 Content Loss

The content loss is the Mean Squared Error of the feature activations in the given content layers (CL) in the model, between the content mixed image and the source image. For any given pretrained model M, let FC and FS are the feature map of the current image and source image respectively. The content loss can be calculated as:

$$L_c = w_c \times \sum_{l \in CL} \sum_{i,j} (FC_{i,j}^l - FS_{i,j}^l)^2$$

where w_c is a scalar weight.

4.5.2 Single-Style Loss

The style loss is the Squared Error for the Gram-matrices in the given style layers (SL), where the Gram Matrix for an input image $x \in \mathbb{R}^{C_j \times H_j \times W_j}$ to be Gx$\in \mathbb{R}^{C_j \times C_j}$. The Gram Matrix is defined as the following:

$$Gx_{c,c'} = \frac{1}{C_j H_j W_j} \sum_{h=1}^{H_j} \sum_{w=1}^{W_j} \phi_j(x)_{h,w,c} \phi_j(x)_{h,w,c'}$$

where $\phi_j(x)$ is the C_j dimensional features for each point in the image.

Let GC and GS are the Gram-matrix of the feature map of the current image and source image respectively. The style loss can be caculated as

$$L_s = w_s \times \sum_{l \in SL} \sum_{i,j} (GC_{i,j}^l - GS_{i,j}^l)^2$$

where w_s is a scalar weight.

4.5.3 Total-variation regularization

To smooth the image, we will add another term to our loss that penalizes wiggles or "total variation" in the pixel values. The TV loss can be calculated as:

$$L_{tv} = w_{tv} \times \sum_c \sum_i \sum_j ((x_{i,j+1,c} - x_{i,j,c})^2 + x_{i+1,j,c} - x_{i,j,c})^2)$$

4.5.4 Multi-Style Loss

Weighted average of all the input style loss for blending different styles.

4.6. Multi-Style Blending Weights

The loss of each input style (S_i) can be blended as part of total multi-style (S) loss as follows:

$$L_s = \sum_{i=1}^{n} w_i L_{s_i}$$

The blending weights can be calculated by

$$w_i = \frac{L_{s_i}}{\sum_{i=1}^{n} L_{s_i}}$$

5. Results and Analysis

5.1. Fast Single Style Transfer Loss Curve

Below we plot the loss curve for a network that was trained on an image of Hoover Tower as the target image and *Starry Night* as the style image for 40,000 iterations.

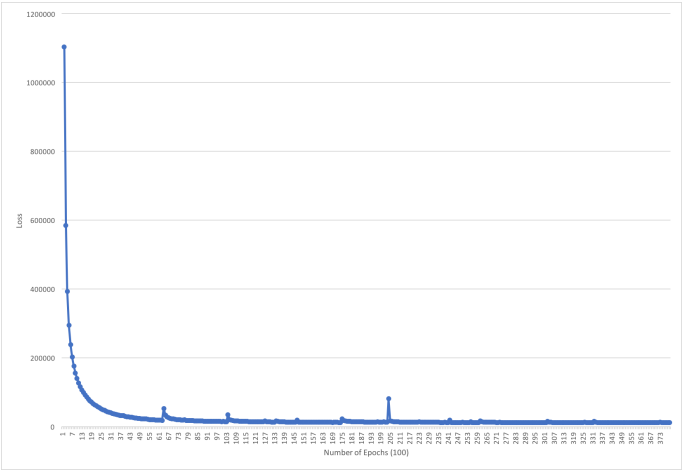
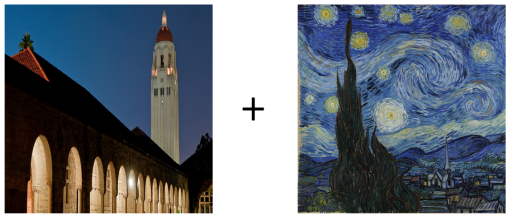


Figure 2: Loss Curve

We note that the loss

5.1.1 Fast Single Style Transfer

Below we present our results for single-style transfer using different methods of upsampling, convolution transposes and nearest neighbor upsamlping. Additionally, we use the same picture of Tübingen as our content image in these results unless otherwise noted.



a)

b) **c)**

Figure 3: **a)** An image of Stanford's iconic Hoover Tower with *Starry Night* **b)** Fast style transfer with fractional convolutions (causes grainy checkered texture) **c)** Fast style transfer with nearest neighbor upsampling

b)

Figure 5: **a)** Tübingen with *Alley by the Lake* Fast style transfer with nearest neighbor upsampling on Tübingen with *Alley by the Lake*

a)

b) **c)**

Figure 4: **a)** Tübingen with *Great Wave off Kanagawa* **b)** Fast style transfer network with fractional convolution layers **c)** Fast style transfer with nearest neighbor upsampling

a)

b)

Figure 6: **a)** Tübingen with *The Scream* **b)** Fast style transfer with nearest neighbor upsampling on Tübingen with *The Scream*.

a)

a)

b)

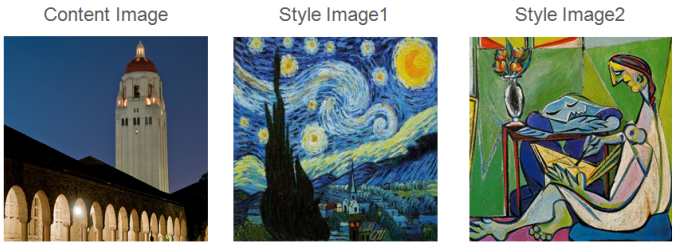
Figure 7: **a)** Tübingen with Donut **b)** Fast style transfer with nearest neighbor upsampling on Tübingen with *Donut*.

After training the single-style transfer network, we can perform style transfer with one fast forward pass taking less than a second. We note the following results:

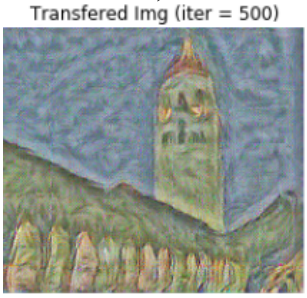
- Much faster than naive style transfer which iterates on the image
- Qualitatively the images are equally convincing, and quantitatively (based on values for style/feature loss) comparable in performance
- Disadvantages: training time is much longer, and each model is restricted to only a limited set of styles

5.1.2 Multi-style Transfer

For multiple style transfer we built on-top of the vanilla single image style transfer proposed in Gatys et al. 2015 paper. Our results are presented below (Fig 8, 9). Additionally, we modified our fast-feedforward neural network to be able to be trained on multiple styles. The results are shown below (Fig 10, 11). Overall, our multiple style transfer results are reasonable for both the single image and the feed-forward neural network implementation as we obtain high quality visual results when we try to transfer multiple styles. Additionally, we'd like to note that our multiple-style algorithm uses trainable weights, so it doesn't require hand-picking any hyperparameter scales for weighting different styles and automatically chooses the best combinations. Lastly, due to the implementation of our algorithms, both implementations of multiple-style transfer can be expended to an arbitrary number of styles.

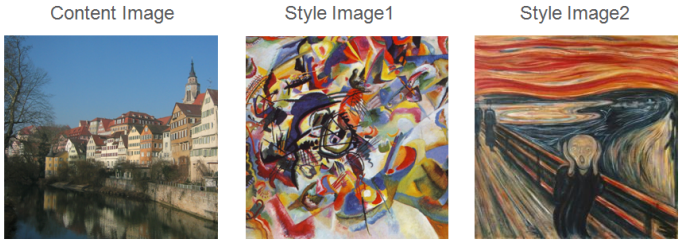


a)

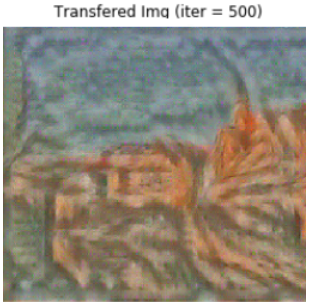


b)

Figure 8: **a)** Hoover Tower with multi-style: *A Muse* and *Starry Night* **b)** Multi-style *A Muse* and *Starry Night* transferring onto Tübingen using iterative optimization.

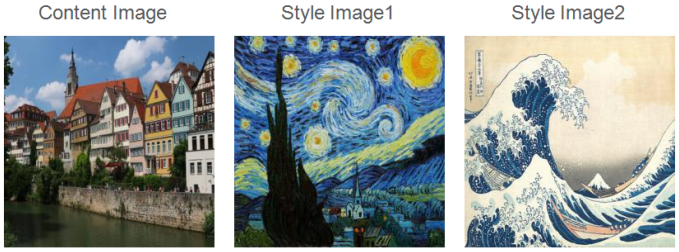


a)



b)

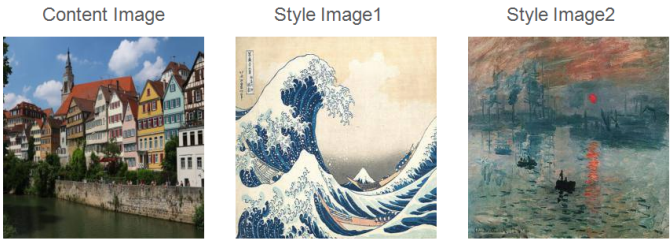
Figure 9: **a)** Tübingen with multi-style: *Great Wave* and *Impression, Sunrise* **b)** Multi-style: *Composition vii* and *The Scream* transferring onto Tübingen using iterative optimization.



a)

b)

Figure 10: a) Tübingen with multi-style: *Starry Night* and *Great Wave* **b)** Multi-style: *Starry Night* and *Great Wave* onto Hoover Tower using our transfer network.



a)

b)

Figure 11: a) Tübingen with multi-style *Great Wave* and *Impression, Sunrise* **b)** Multi-style: *Great Wave* and *Impression, Sunrise* transferring onto Hoover Tower using our transfer network.

6. Discussion

6.1. Error Analysis

We'd like to begin to note that although our loss curves converged with the vanilla implementation presented in Johnson et al. 2016 that the residual filters left in the images was problematic. However, this problem is not unknown to neural networks Odena et al. 2016. As a result, we turned to a solution proposed by Dumoulin et al. 2017, where they propose implementing nearest neighbor upsampling as a replacement for the convolution transpose proposed in Johnson et al. 2016.

6.2. Limitations

While our algorithm for fast style transfer is sufficiently general and can handle any variety of input images, it is still limited to a fixed selection of style images to be trained on, followed by a very long training process of a few hours. We can certainly stop the algorithm earlier, but doing so leads to noticeably weaker results since the network wasn't able to run as many epochs and receive its fullest training. It is also restricted to a predetermined set of parameter weights on content and style losses, so we aren't able to alter these afterwards to see how the algorithm can transform the image to be highly stylized / more abstract on one end versus less styled / more realistic on another.

6.3. Future Work

In the future, we would like to expand our work to incorporate other aspects of image understanding. We could apply image segmentation so that our algorithm identifies different components of a picture as it applies styles to them, leading to more visually organized and consistent results. Another possibility would be to create a network that can have its parameters (content weight, style weight, selected layers, layer weights) customized without the need to be trained again from scratch. This would allow us to observe the exact visual features that each parameter contributes, giving us a richer understanding of the inner workings of the network.

References

[1] Efros A.A., Freeman W.T. Image quilting for texture synthesis and transfer. In Proceedings of the 28th annual conference on Computer graphics and interactive techniques, pp.341-346. ACM, 2001.

[2] Elad M., Milanfar P. Style-transfer via texture-synthesis. arXiv preprint arXiv:1609.03057, 2016.

[3] Gatys, L.A., Ecker, A.S., Bethge, M. A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576, 2015.

[4] Gatys, L.A., Ecker, A.S., Bethge, M. Texture Synthesis Using Convolutional Neural Networks. In Advances in Neural Information Processing Systems 28, 2015.

[5] Gatys, L.A., Ecker, A.S., Bethge, M. A neural algorithm of artistic style. arXiv preprint arXiv:1508.06576, 2015b.

[6] Gatys, L.A., Ecker, A.S., Bethge, M., Hertzmann A., Shechtman, E. Controlling perceptual factors in neural style transfer. CoRR, abs/1611.07865, 2016b.

[7] Johnson, J, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In European Conference on Computer Vision, pages 694711, 2016.

[8] Johnson, J. neural-style. https://github.com/ jcjohnson/neural-style, 2015.

[9] Johnson, J. fast-neural-style. https://github.com/ jcjohnson/fast-neural-style, 2016.

[10] Ioffe, S., Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: Proceedings of The 32nd International Conference on Machine Learning, 2015.

[11] Jing, Y. Neural Style Transfer: A Review. arXiv:1705.04058v1 (2017)

[12] Ulyanov D., Vedaldi, A., Lempitsky, V. Instance normalization: The missing in- gredient for fast stylization. arXiv:1607.08022, 2016.

[13] Dumoulin, V., Shlens, J., Kudlur, M. A learned representation for artistic style. ICLR, 2017.

[14] Frosard, D. VGG in Tensorflow. https://www.cs.toronto.edu/ frossard/post/vgg16/

[15] Krizhevsky A., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[16] Odena, A., Olah. C., Dumoulin, V. Avoiding checkerboard artifacts in neural networks. *Distill*, 2016.

[17] Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classi cation models and saliency maps. In: ICLR Workshop. (2014)

[18] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., Lipson, H.: Understanding neural networks through deep visualization. In: ICML Deep Learning Workshop. (2015)

[19] Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: CVPR. (2015)

[20] d'Angelo, E., Alahi, A., Vandergheynst, P.: Beyond bits: Reconstructing images from local binary descriptors. In: ICPR. (2012)

[21] d'Angelo, E., Jacques, L., Alahi, A., Vandergheynst, P.: From bits to images: Inversion of local binary descriptors. IEEE transactions on pattern analysis and machine intelligence 36(5) (2014)

[22] Vondrick, C., Khosla, A., Malisiewicz, T., Torralba, A.: Hoggles: Visualizing object detection features. In: ICCV. (2013)

[23] Dosovitskiy, A., Brox, T.: Inverting visual representations with convolutional net- works. In: CVPR. (2016)