Neural Style Transfer with Learned Artist Representations

Evan Dogariu[1] **Ioana Marinescu**[1]
[1]Department of Computer Science, Princeton University
{edogariu, ioanam}@princeton.edu

Abstract

Previous research in the field of Neural Style Transfer is limited in the fact that transfer models are designed to learn style from a single template image. In this paper, we propose several generalizations of fundamental style transfer concepts that allow for learning representations of an artist's entire style rather than that of a single image. We evaluate models constructed with these approaches qualitatively and quantitatively to determine their feasibility and conduct exploratory analysis into this topic of artist representation. We find that it is very difficult to bridge the divide between computer understanding of artist style and our human understanding.

1 Introduction

The computer vision task of reproducing the artistic style of an image and using it to transform another one, called *neural style transfer*, was initially explored in [5]. This process is achieved by selecting two images: a style image whose style we want to reproduce and a content image which we want to alter in the style of the style image. The end goal is for the transformed image (called a *pastiche*) to have elements such as color selection, brushstroke style, and boldness of the style image while preserving the important content of the original image. As one might expect, quantifying and dealing with these elements in an algorithmic way is nontrivial and a rich area of research.

However, much of the existing research focuses on capturing the style of a given image. A much more open challenge in style transfer is to identify and learn the general style of an artist rather than solely one of their images. One large step was made by Dumoulin et al. [4], who proposed a method for capturing style by learning the style of not one but a diversity of paintings in the same style transfer model (N-styles). With this method, they were able to transform content images to generate pastiches that represent a combination of several styles from several different style images.

We propose and evaluate several methods to capture the style of a specific artist and use the information learned to create pastiches from content images. The goal of the newly created image is to appear as if it was painted by the artist whose style was learned while still maintaining its original form.

2 Related Work

Gatys et al. [5], in one of the first and most influential papers on the topic of neural style transfer, create the foundational approach of neural style transfer that is still used in cutting-edge style transfer techniques: the algorithmic separation of image content and image style. Since the single-style-image style transfer problem can be framed as creating a balance between content similarity to the original image and stylistic similarity to the style image, the technique created by Gatys et al. is to minimize a weighted sum of the differences between the generated image and the content and style image. More precisely, the algorithm starts from initializing p, the generated pastiche image, as the content image c and using gradient descent to minimize the loss function

$$\mathcal{L}(c, s, p) = \lambda_c \mathcal{L}_c(p) + \lambda_s \mathcal{L}_s(p)$$

where $\mathcal{L}_c(p)$ is the content loss, $\mathcal{L}_s p$ is the style loss and λ_c, λ_s are their respective weights. In their implementation, the content and style losses are computed using extracted image features via a pre-trained VGG [9] network. For $\mathcal{L}_c(p)$, both c and p are passed through the VGG and the responses after certain preselected layers are compared; $\mathcal{L}_s(p)$ is calculated similarly with a different selection of layers to extract features from. Backpropagation through the VGG layers

allows for calculating the loss gradient w.r.t p. With careful selection of which layers' extracted features to use, the constructed loss functions can ensure large-scale similarity with the content image while ensuring style similarity on smaller scales, discarding the global arrangement of the scene.

Since this method optimizes something initialized to the content image, transferring the same style to a different content image requires starting from scratch. Furthermore, this approach has the downside of being computationally expensive as it requires a forward and backward pass through the pretrained network (VGG) at each step of the training. To remedy this, Johnson et al. [7] proposed a feed-forward CNN as a style transfer network to learn the transformation. They train the network using the same loss functions defined above to measure perceptual differences in content and style. The use of a CNN to perform this transformation (instead of gradient descent optimization as in Gatys et al.) allows for advanced architectures and fast inference on any given content image.

However, both approaches are limited to learning a single style image, which implies that a new network has to be trained for each new style image. Therefore, in order to capture the style of an artist without training a new network for each of their paintings, we need to consider other methods.

3 Approach

We propose four methods of increasing novelty and complexity to achieve the goal of capturing the style of an artist and applying it to the content image.

3.1 Random Image

We select a random image from the list of paintings of an artist and apply the neural style transfer algorithm as outlined by Johnson et al. to the content image using the random painting as the style image. This approach can only capture the style of the artist represented in a specific painting, which likely does not completely encapsulate the style of the artist. However, this method has been shown to create robust transformation networks that can produce visually pleasing pastiches for a wide variety of input content images [7]. We consider this as a baseline, and we qualitatively evaluate the other methods in part based on their synonymy with this method.

3.2 Average Image

We create a 'representative' image as the RGB pixel average of an artist's paintings. Again, we apply the neural style transfer algorithm to the content image, this time with the average image as the style image. This approach loses most of the information about individual paintings as different scenes that are not at all related get combined and nearby pixels of the average image do not describe an object anymore. Furthermore, the averaged image loses color information of individual paintings, reflecting only general artist preferences for certain colors in certain image regions. Style information is also lost, as the appearance of elements of artistic style (brushstrokes, boldness, etc.) in the average image only occurs if these elements show up in similar locations, colors, orientations, and scales in many of the artist's paintings.

In general, inspection of the average image of several artists shows that it is usually a uniform, papery brownish image, reflecting the average of artist's preference of color tones. To combat this, we also implement a "smart" average technique. Rather than averaging the raw pixel values and using the VGG to find style features of this averaged image, we use the VGG to find style features of every painting of the artist and average them all together. Doing so grants us average style features that did not come from any particular image, but that are more representative of some of each artist's general style elements.

3.3 Cycling Images

For this method, we modify the neural style transfer algorithm such that the style image changes during training, cycling between all the paintings of the particular artist. By doing so, we allow the trained transformation network opportunities to learn the styles of many different images corresponding to the same artist; hopefully, this would allow the network to learn some general elements of the artist's style that appear in many of their paintings. Performing this cycling does not increase the computational difficulty of each training cycle, but it requires more training cycles to begin to converge.

3.4 Classifier

In the previous methods, we maintain the same neural style transfer setup of having a single style image per training cycle which we use to evaluate style

loss (with feature extraction through VGG). We could, however, use an auxiliary network more specialized to this problem than the VGG to evaluate the style loss. To do so, we create an artist classifier that predicts an artist given an input image. During training of the transformation network, since we know exactly which artist the pastiches are supposed to be classified into (after all, we design the network to adopt a certain artist's style) we can simply classify the generated pastiche and compare with the expected labels to get the style loss. We backpropagate this loss through the classifier to get loss gradients w.r.t. the transformation network's parameters.

4 Implementation

4.1 Transformation Network

4.1.1 Architecture

The transformation network is designed very similarly to the one used by Johnson et al. [7]. We have an architecture constructed from the following pieces:

- **ConvLayer(in_channels, out_channels, kernel_size, stride)**, which is a regular PyTorch Conv2D layer with ReflectionPad2D before the convolution and InstanceNorm2D and ReLU activation after.
- **ResidualLayer(channels, kernel_size)**, which is two ConvLayers with in/out channel dimensions and kernel sizes as given, plus a residual connection across these two layers.
- **DeConvLayer(in_channels, out_channels, kernel_size, stride)**, which is a regular PyTorch ConvTranspose2D layer with InstanceNorm2D and ReLU activation after.

The architecture consists of a Convolutional Block (4 ConvLayers), a Residual Block (5 ResidualLayers, each containing 2 ConvLayers), and a DeConvolutional Block (3 DeConvLayers and a ConvLayer without ReLU), totaling 15 convolutional and 3 deconvolutional layers with 5 residual connections in the middle. Specific channel dimensions, filter sizes, and strides can be found in the code. The code to implement this architecture in PyTorch was adapted from a GitHub repository for fast single-style-image style transfers [8]. The inclusion of reflection padding and 2-dimensional instance norms was inspired by the design choices made by Dumoulin et al. in their N-styles network [4]. The addition of an extra convolutional (and deconvolutional) layer with a 1 x 1 kernel was done to rectify some blocky pixel artefacts that show up in the transformed images. Aside from these choices, the architecture is pretty much exactly as implemented by Johnson et al. [7].

4.1.2 Training

We use a training process for our transformation network as described above. To be particular, during each training iteration, we take our (B x C x H x W) tensor batch of content images c, transform it with the transformation network to get a (B x C x H x W) tensor batch of pastiche images p, and pass them both through the pre-trained VGG-16 model. We grab the content features from the 'relu2_2' layer and use MSELoss between the p and c content features to find content loss. For the non-classifier approaches, we do the same thing with the style image s except that we grab style features from the 'relu1_2', 'relu2_2', 'relu3_3', and 'relu4_3' layers, calculate the MSELoss between the gram matrices of the features for each of these layers, and sum them to get the style loss. This process is summarized in Figure 1. For the classifier approach, we instead calculate the style loss by classifying p and calculating the CrossEntropyLoss between the output and the expected label.

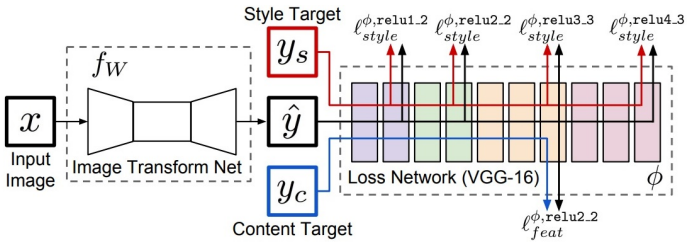

Figure 1: Training process (diagram from Johnson et al. [7])

4.1.3 Hyperparameters

To begin, we selected the batch size to be the largest that the NVIDIA GTX 1060 GPU we used to train can handle in memory; so, we used batches of $BATCH\_SIZE = 4$ images, each rescaled to size (224 x 224). Given the large number of networks that we needed to train (and the fact that AWS declined us for GPU cloud computing), we needed each network

to be trained in less than 4 hours. So, we selected $NUM\_EPOCHS = 32$ such that this is the case. Given this constraint, we then experimented with learning rates for the Adam optimizer that worked best with this NUM_EPOCHS. In the end, we found that using an initial $lr = 0.0096$ that halves every 16 epochs allowed for the best convergence given the limited training time. We also applied L_2 normalization with a weight decay of 0.0001 to our Adam optimizer.

The other training hyperparameters we needed to consider were the relative weights of the content and style losses. We selected a fixed $CONTENT\_WEIGHT = 17$, and varied the $STYLE\_WEIGHT$: for the non-classifier approaches in which style loss was evaluated with VGG features we had $STYLE\_WEIGHT = 45$, while for the classifier approach in which style loss was evaluated with classifier predictions we had $STYLE\_WEIGHT = 2000000$. These weights were gathered from brief experimentation; larger style weights result in models with more extreme styles whose outputs resemble the content images less.

4.2 Dataset

Our artist dataset consists of 8355 paintings belonging to 50 influential artists from different time periods collected from a Kaggle challenge [1]. The images are of varying size and are stored in .jpg format. We use OpenCV to load the image data, which we preprocess by rescaling the images to a smaller, uniform size (281 x 269, half the average dimensions of the dataset), and converting the images from (H x W x C) to (C x H x W) format, and transforming into PyTorch tensors. The two models that we pass these tensors into (VGG-16 and the classifier) require different preprocessing from this point.

We also utilize a dataset of random content images to cycle through during transformation network training. If we were to train with only one content image, we risk that the network would not be generalizable to style transfers on arbitrary content images. So, we use a subset of 26,927 images from the ImageNet training dataset [3] to serve as our random content images. These images are preprocessed similarly to as above - load, rescale, convert to (C x H x W), convert to PyTorch tensor dataset - with the difference that these images are rescaled to 224 x 224 for computational efficiency, since these are the images that are batched and used during training.

4.3 Pre-trained Models

4.3.1 VGG-16

We use the VGG-16 model from the torchvision.models package with the original caffe-framework pre-trained weights and biases. The code for an implementation that allows for grabbing outputs from desired layers is adapted from [8].

4.3.2 Classifier

The classifier that we use is based on the ResNet-50 residual CNN [6]. We copy the convolutional and residual blocks of the network from the torchvision.models ResNet-50 model, and replace the fully connected layer at the end with two Linear layers, ending in 19 outputs. Specific implementation dimensions and details can be found in the code. We choose to use 19 classes for the 19 artists with the largest number of paintings in our dataset; this allows us to use the pre-trained weights and biases for artist classification from a user submission on Kaggle [2]. This pre-trained classifier gets ¿90% accuracy across the 19 artists it classifies in our dataset.

5 Evaluation

We chose to deploy our approaches to create artist style transfer models for 4 different influential artists with distinct styles and large painting datasets: Pablo Picasso, Leonardo da Vinci, Rembrandt, and Vincent van Gogh. We evaluate each of our approaches and the generated pastiches both qualitatively with our eyes and expectations and quantitatively using the artist classifier.

5.1 Qualitative Analysis

5.1.1 Average Image

The RGB average image of all the paintings of an artist is a plain shade of brown image for all 4 artists. The average images vary very little between artists. Using this as the style image of an artist and applying it to the content image results in something similar to applying a simple brown filter to the content image, which fails to capture anything from the style of the artist (Figure 2). However, ”smart average” captures

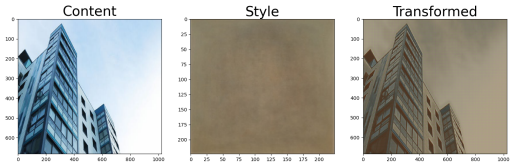

Figure 2: Vincent van Gogh regular average

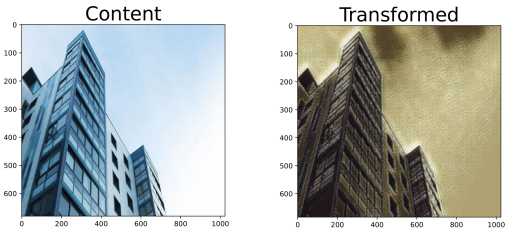

Figure 3: Vincent van Gogh ”smart” average

Figure 4: Leonardo da Vinci ”smart” average

more information about the style, as can be seen by the additional style complexity in Figures 3 & 4. There are differences in brushstroke size and style between van Gogh and da Vinci that are captured by the ”smart” average, which the RGB average cannot (van Gogh and da Vinci RGB averages are almost identical).

5.1.2 Random Image

Selecting a random painting as the style image and applying the simple neural style transfer algorithm achieves the desired goal. Features such as color selection, line boldness, and brushstrokes appear to be transferred successfully to the new image while preserving enough information from the content image (Figures 5-8). However, a random painting cannot capture the diverse style of the artist, which includes different colors and sometimes even different types of lines and brushstrokes (see **Discussions** 6).

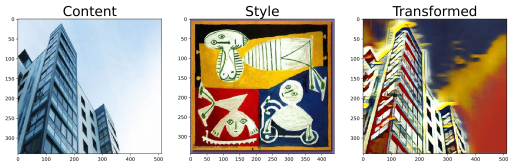

Figure 5: Picasso random

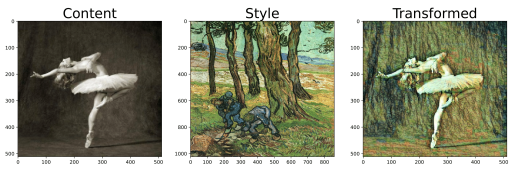

Figure 6: Vincent van Gogh random

5.1.3 Cycling Method

For this method we observed that some of the features that are prevalent in many of an artist's paintings are preserved, such as the dark, gloomy colors in Rembrandt's paintings, the thin red lines in da Vinci's drawings, and the characteristic brushstrokes of van Gogh. However, most artists use a wide range of colors in their paintings that differ from painting to painting. Cycling through these paintings every training iteration leads to overlapping different colors in the same region; so, the network learns to minimize content loss by sticking to the brownish colors which are somewhere in the middle (Figures 9-12).

Figure 7: Rembrandt random

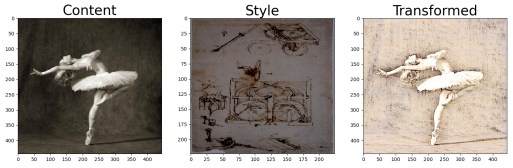

Figure 8: Leonardo da Vinci random

Figure 9: Picasso cycle

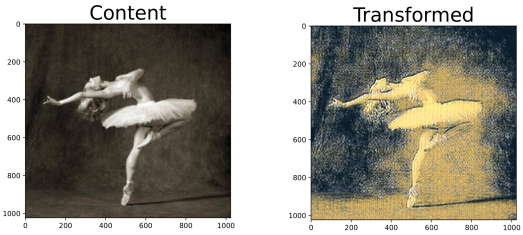

Figure 10: Vincent van Gogh cycle

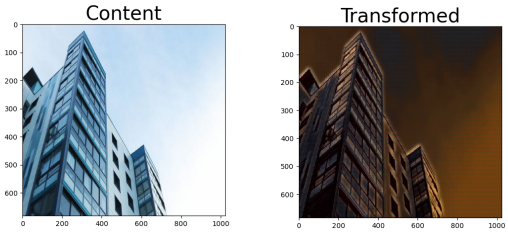

Figure 11: Rembrandt cycle

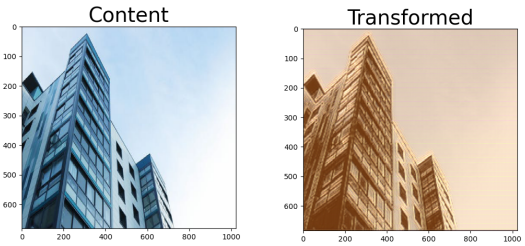

Figure 12: Leonardo da Vinci cycle

5.1.4 Classifier

Using the artist classifier predictions instead of the VGG features to calculate style loss generated very interesting results. The stylized image always looks like a colored, grid-like mask was applied to the content image (Figures 13 - 16). Also, it can be seen that there are some blurs added to certain locations in the images. However, we see very clearly that these generated images do not represent the styles of their respective artists in the way that we as humans expect. The results are not visually pleasing.

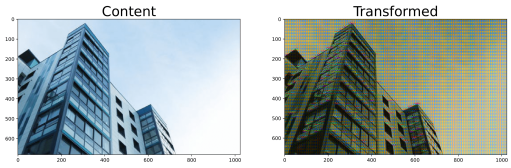

Figure 13: Picasso classifier

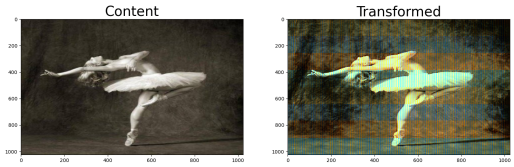

Figure 14: Rembrandt classifier

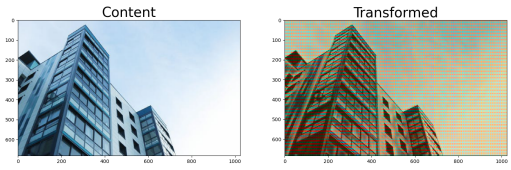

Figure 15: Leonardo da Vinci classifier

In general, we see that the content images that worked best for da Vinci were simple images with sharp lines, such as the building that ended up looking like a drawing. For van Gogh and Rembrandt, more complex content images like landscapes and the ballerina worked best. For Picasso, due to the wide range of styles used in his paintings, the results were more confusing and it was overall more difficult to capture his artistic style in a single model.

5.2 Quantitative Analysis

In order to perform quantitative analysis of these models, we decide to make use of the pre-trained artist classifier. In particular, we generate pastiche images for a sample of random content images for each artist using each model. We then pass these images

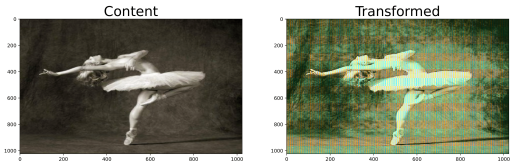

Figure 16: Vincent van Gogh classifier

into the artist classifier and count the proportion that were classified as being made by the desired artist. We summarize the findings showing percentages of pastiches in the style of a specific artist were classified as paintings of that artist by the classifier in Table 1.

	Average	Random	Cycle	Classifier
Picasso	0.93%	16.54%	8.27%	99.1%
da Vinci	6.77%	49.62%	79.05%	98.08%
Rembrandt	9.37%	53.38%	75.94%	98.5%
van Gogh	5.21%	86.47%	41.35%	99.1%

Table 1: Percent of stylized images recognized by the classifier

The images generated using both the **average** and **smart** average approaches for each artist were classified poorly, as expected. Even with the **smart** average method, the averages are very similar for many artists, and they don't really resemble the images in our dataset that the classifier was trained on.

A **random** image of the artist seemed to generate images that were recognized by the classifier as belonging to the right artist half of the times or more in the case of da Vinci, Rembrandt, and van Gogh, but not in the case of Picasso. The Picasso pastiches generated with the random painting method were most often miscategorized as Andy Warhol paintings, which appears to be understandable (see Figure 5).

The Picasso pastiches generated with the **cycle** method were most often miscategorized as Rembrandt paintings, which is again understandable given the color selection (see Figure 9, Rembrandt uses similar, darker colors).

Pastiches generated with the **classifier** method are almost always classified as belonging to the desired artist no matter the given content image. This is expected, as these models were designed to partially minimize the classification loss, yet it also demonstrates the differences between human-perceived and computer-perceived style.

6 Discussion

Some of the results reflected our initial expectations while some were very surprising. The average image used as a style image resulted in a brown version of the content image, capturing nothing of the style of the artist, other the nuances they used. The average image for artists that mostly did drawings in black/brown and white/beige (such as da Vinci) was a lighter shade than the average color of artists who mostly did paintings in vivid colors. The random image also worked well considering that it is applying the original neural style transfer algorithm that takes as input a single image. However, a random image fails to capture the style of an artist, and sometimes, depending on the content image, the result can even look similar to the style of another artist. For example, an image of a building transformed with the style from a Picasso painting ended up looking like an Andy Warhol painting (Figure 17).

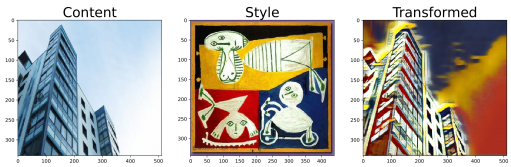

Figure 17: Picasso style image misclassified as Andy Warhol

We applied two different style images that both belong to Picasso to the same content image in Figures 18 and 19. As hypothesized, a random image is not enough to capture the style of an artist as the colors and brushstrokes differ quite a lot between the 2 style images. This difference is heavily reflected in the generated pastiches.

The most important methods we used are the "smart" averaging, the cycling, and the classifier. Initially, we thought that the classifier would give the best results. However, it turns out that there were shortcuts that the transformation networks could take during training that drastically minimized the classification loss without gaining any true representation of artist style.

The "smart" average and the cycling often gave comparable results: they capture some of the artist's style but lose color information, resulting in a generally brown color for the final image. For some artists with constantly dark backgrounds, such as Rembrandt, this works well, but for others such as Picasso, who uses a lot of colors, these pastiches lack a certain dimension of artist style.

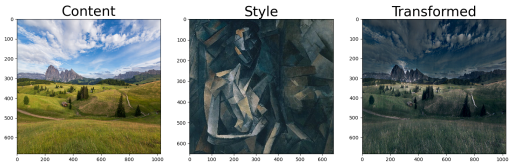

Figure 18: Picasso style image 1

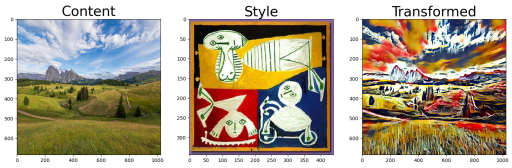

Figure 19: Picasso style image 2

In general, we have seen that there is not a clear way to broaden the problem from learning style of a painting to learning style of an artist. It is difficult to concoct an algorithmic loss function that measures artist style representation, and different methods can produce very different results. Furthermore, we observe that there is a great divide between how we interpret artistic style as humans and how a computer (for example, an artist classifier) interprets artistic style. The classifier approach to training a transformation network produced results that looked unsuccessful and unappealing to human eyes, but flawless in the classifier's perspective. This divide adds another dimension of difficulty to the problem, as crafting a good style loss function heavily relies on being able to bridge this gap in stylistic understanding.

7 Future work

Many of the models generated with our methods (particularly the average and cycle approaches) resulted in pastiches that captured artist style to some degree, but without the impressive color palette that these artists often employ. For visually pleasing results, perhaps it would be plausible to improve our transfer pipeline by taking the outputs of these networks and running them through a coloring algorithm to produce final outputs with both impressive colors and the artists' styles.

As for the training process itself, one improvement we could make is to utilize adaptive content and style loss weights as opposed to fixed, predetermined ones. We notice that the style loss overpowers the content loss in the beginning of training, but towards the end the content loss is what dominates most of the gradients. Perhaps an algorithmic way to control this balance could result in better models.

There is also a plausible artist representation approach, implemented in [4], that we did not attempt to replicate: the N-styles network. This network allowed for capturing of styles from an arbitrary number of images in a single network, which we could use to represent an artist's overall style much better than any single image could. Given more time, we would likely implement this method before any other improvements.

8 Acknowledgements

We would like to thank Prof. Russakovsky and our project mentor Sunnie Kim for all their teaching, support, and help!

References

[1] URL: https://www.kaggle.com/ikarus777/best-artworks-of-all-time.

[2] Jacopo Attolini. *Paintings classifier fastai - ResNet50 [90.2% acc]*. https://www.kaggle.com/attol8/paintings-classifier-fastai-resnet50-90-2-acc/notebook. 2019.

[3] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[4] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. "A Learned Representation For Artistic Style". In: *CoRR* abs/1610.07629 (2016). arXiv: 1610.07629. URL: http://arxiv.org/abs/1610.07629.

[5] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "A Neural Algorithm of Artistic Style". In: *CoRR* abs/1508.06576 (2015). arXiv: 1508.06576. URL: http://arxiv.org/abs/1508.06576.

[6] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[7] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. "Perceptual Losses for Real-Time Style Transfer and Super-Resolution". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 694–711.

[8] Rusty Mina. *fast-neural-style: Fast Style Transfer in Pytorch!* `https://github.com/iamRusty/fast-neural-style-pytorch`. 2018.

[9] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.