Artificial Artworks

Style Transfer using CNNs and CycleGANs

Erik Bossow Julia Fülling Lena Kagoshima
Peter Keffer Linda Ariel Ventura di Lorenzo Lopes
Jan Eric Wiemann

April 2022

Abstract

The main hurdles in transferring an artistic style onto another image lies in the conservation of the original images content features and in the detection of the common and recognisable stylistic features found in the artistic style source. This paper treats the Neural Style Transfer and the CycleGAN deep-learning approaches to these problems, illuminates their implementation and compares the resulting images. The CycleGAN seems to create results that resemble the underlying artistic style of a large set of image samples, whereas the Neural Style Transfer approach creates results that show heavy influences from a single artistic style reference image.

1 Introduction

The goal of this paper is to explore how two different methods approach the same task of style transfer. Style transfer is the meshing of the content of one image with the style of another image. The models translate the styles of famous artworks unto unpaired images without losing its content.(Yuan, 2018)

The first method we investigate is an Artificial Neural System that uses a simple **Convolutional Neural Networks (CNNs)** as its basis. CNNs are one of the essential architecture types of Deep Learning. Although they are usually used as a discriminatory tool, they can also be used to generate data (Saha, 2018). Gatys et. al (Gatys, Ecker, & Bethge, 2015) were among of the first to use CNNs in such a generative function while focusing on a style transfer between two images. In the following we have implemented their established method using the opensource software library TensorFlow.

We have decided upon implementing a CycleGAN as our second approach, they are a certain type of **Generative Adversarial Network (GAN)** that is particularly designed for image to image translation tasks. GANs use a combination of two adversarial CNNs pitted against each other. One model generates

the data (the *Generator*) and the other model determines if the output looks indistinguishable from the original data set - ie. if the data looks "real" (the *Discriminator*). The two networks contest against each other. This results in generating more realistic output(Nash, n.d.) GANs are not trying to generalise or draw predictions from the given data, but rather to figure out the underlying data density distribution and generate new examples of what it has learned(*CycleGAN*, n.d.).

2 Contextualisation - History of Artistic Style Transfer

Before Deep Neural Networks were used for image stylization, rendering an image in a different style had been studied extensively within the field of non-photorealistic rendering.

The earliest stylisation methods were *procedual*: people coded filters that tried to implement artistic styles directly. Simply using operations like low-level blurring and edge detectors could already produce an interesting artistic effect.

Later style transfer methods relied more on *texture transfer*: In the domain of texture synthesis, a popular method was to use non-parametric algorithms which could synthesise textures based on a source image. Some people took this idea and used it as a basis in the seemingly different field of image stylisation. They developed a number of different approaches to transfer a generated texture onto an unrelated target image (texture transfer). To preserve the objects in this target image and constrict the areas to which the texture should be applied to, there were a number of different methods available. Most of them used low-level content representations to infer image structures, such as image intensity or local image orientation angles, like in the case of image quilting (Efros, 2001).

3 CNN

The first method we implemented is the Neural Algorithm proposed in the paper "A Neural Algorithm of Artistic Style" (Gatys et al., 2015) . We chose Gatys et. al's ANN for our comparison, because this was the first instance of a Deep Neural Network being used for artistic style transfer. Also, despite having been implemented seven years ago, even from today's point of view their results are quite impressive.

CNNs trained on object recognition develop an internal representation of an image. Along the processing hierarchy of the network the input runs through a large number of various filters, which pick up on different features of the image. Throughout this process, the exact pixel values of the original picture are abstracted away from and what remains are the position and the types of objects, in other words the general scenery of an image. Along the processing hierarchy of the network, this semantic content representation becomes more

and more explicit. In this way, CNNs extract high-level semantic information from a given image.The higher layers of such a CNN contain what we will from here on out refer to as the content representation. The style of an image can also be captured by a CNN in form of a style representation. For this, a feature space consisting of the correlations between the different filter responses is built on top of the filter responses in each layer of the network. Including the feature correlations of multiple layers allows us to capture the general texture of an image but not the depicted objects.

3.1 Implementation

The key idea behind this CNN-based approach is that the content and style of any image can be separated and recombined. This paves the way for a very elegant algorithm which is able to extract the semantic content of an arbitrary photograph and combine it with the style of a piece of art. The resulting image still depicts the content and general scenery of the original photograph, but is rendered in the style of a specific artwork. Both, the content and the style representation are achieved by using the internal representations of Deep Neural Networks (Gatys et al., 2015).

3.1.1 Dataset

To generate an image which mixes the style and content of two separate pictures we do not need a big dataset. The pretrained VGG19 network merely requires one image from which it can extract the content and a separate image from which it can extract the style. For the purpose of comparison we decided to use some of the images of the *ukiyoe2photo* dataset, which we will later on also use in our CycleGAN implementation (for details see CycleGAN dataset).

3.1.2 Preprocessing

For data preprocessing we used the preprocessing function of the VGG19 Keras implementation

```
tf.keras.applications.vgg19.preprocess_input
```

Doing this converts the content and the style image from RGB to BGR. As a result, each color channel is zero-centered with respect to the ImageNet dataset.

3.1.3 Architechture

In recent years all winning architectures of the annual "ImageNet Large Scale Visual Recognition Challenge", which evaluates algorithms for object detection and image classification, have been some sort of CNN. In 2014, the winner of the ImageNet challenge was a CNN created by the Visual Geometry Group (VGG), boasting with an error rate of 7,0%. Similarly to Gatys et al., we use this network as a basis for our network. VGG 19 consists of 16 convolutional

layers, each of them having a non-linear ReLU activation function. The whole network is separated by 5 pooling layers and ends with 3 fully connected layers. We replaced the max-pooling layers with average pooling, because this gave us better results in image synthesis (Singh, 2019).

Content reconstruction
We can synthesize an image whose feature maps at a specific convolutional layer match the corresponding feature maps of the content image . The newly generated image will have the same content as the original photograph, but the exact pixel values are not important for the activation to be similar, so the texture may be different. Concretely, we are performing gradient descent on a white noise image to find another image that matches the feature responses of the original image.

How well the content is recreated can be represented by a loss function:

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

Figure 1: Cycle Loss Function
(Singh, 2019)

The responses in a given layer can be stored in F^l, where F^l is the activation of the different filters in layer l. P^l is the respective feature response of layer l of the generated image. i and j correspond to the ith filter at position j in layer l. The content loss is in a specific layer is therefore defined as the Mean Squared error between the feature maps F of our content image and the feature maps P of our generated image.

Style reconstruction
The feature correlations we built on top of the responses in each layer l of the network are given by the Gram matrix G^l. G^l essentially is the dot-product of the vectorised feature maps in layer l. Similarly to the content reconstruction, we perform gradient descent on a random background to find an image which matches the Gram matrix of the original style image. The newly synthesized image will have the same style as the piece of art, but not necessarily the same content. This can also be represented in a loss function:

$$\mathcal{L}_{style} = \frac{1}{2} \sum_{l=0}^{L} (G_{ij}^{l} - A_{ij}^{l})^2$$

Figure 2: Style Loss Function
(Singh, 2019)

G^l is the style representation in layer l of the original artwork and A^l is the respective style representation in the synthesised image. The style loss is then computed as the Mean Square Error between G^l and A^l. Style represenations from earlier layers capture more of the finer texture details of the artwork, while later layers capture more of the higher-level structures. The best results are achieved by taking combinations multiple layers as a style representation for the final image.

Synthesising the final image
The whole task of finding an image, which combines the style and content of two separate images is essentially the search for a minimum within a loss landscape.

The total loss is the weighted sum of both the content and the style loss.

$$\mathcal{L}_{total} = \alpha \mathcal{L}_{content} + \beta \mathcal{L}_{style}$$

Figure 3: Style Loss Function
(Singh, 2019)

The final image is constructed by performing gradient descent on this loss landcape on a random white noise image:

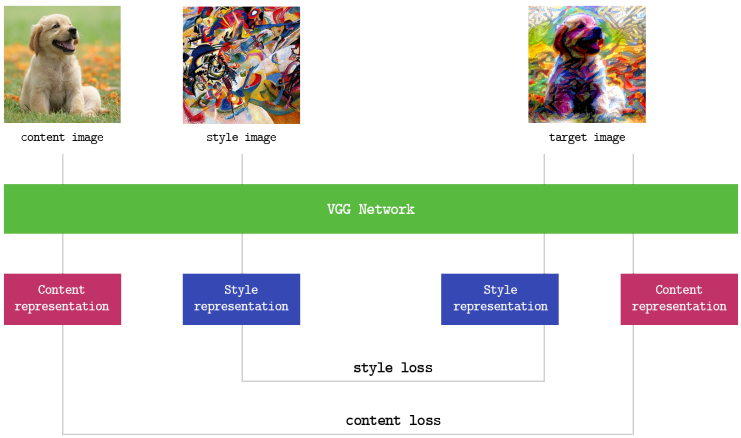

Figure 4: Neural Style Architecture

3.2 Training

For the training process we first decided to use sgd as an optimizer with an exponentially decaying learning rate of 100. This seemed to be going well at

first, but changing the style weights, content weights or total variation weights resulted in the error message NaNs (not a number). Picking higher values for the learning rate also resulted in nans. To solve this problem we decided to use the Adam optimizer instead. We chose a learning rate of 10.0, because this gave us the best results.

The total variation loss (which is also part of the loss function) ensures smoothness on the pixel level of the final image. Leaving it out led to incoherent images with pixel artifacts.

3.3 Results

The quality of the final mixed image highly depends on how well the content and the style image fit together. We observed more interesting textures in the final images with the use of style images that contain many structural features instead of homogenous areas. The results also look best when the texture of the style image and the lines of the general scenery of the content image resemble each other.

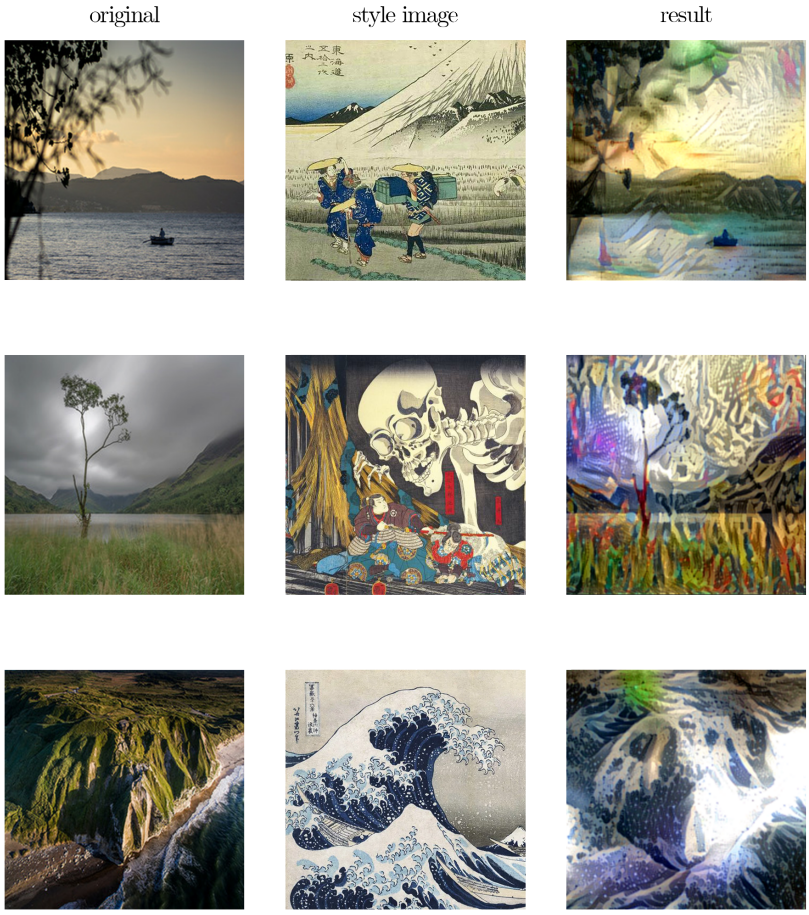

Figure 5: Neural Style Result Images

4 CycleGAN

Cycle GANs were introduced in 2017 specifically to perform image-to-image translation with unpaired images. CycleGANS consist of 2 Generators, 2 Discriminators and the important feature of a Cycle Loss. Generator 1 aims to transform the original content-image to look like the style-images. Discriminator 2 checks the result and promotes it to look like the style-images. Generator 2 tries to transform the result of Generator 1 back to look like the original content-image. Discriminator 2 checks whether the result from generator 2 looks like the original content-image and promotes such resemblance. The difference between

the original content-image and the result from generator 2 is then measured by the cycle loss. The cycle loss promotes the result from generator 2 to look like the original and the results from generator 1 to look like the style-images. This ensures that the result from generator 1 will still contain important features of the original image instead of try to adapt to the style-images by disregarding the original content(Zhu, 2020).

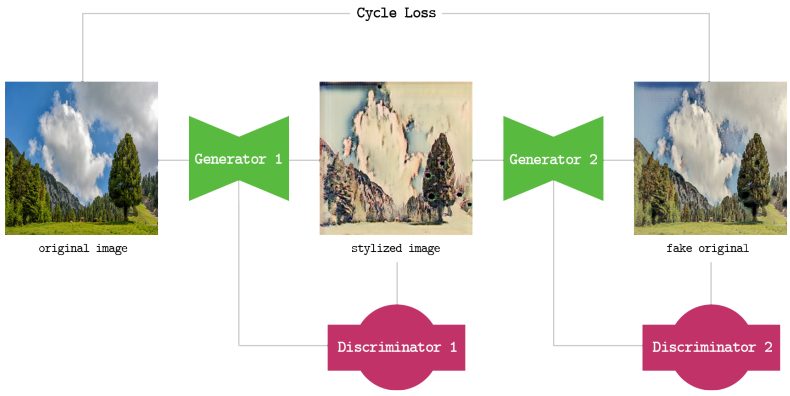

Figure 6: Cycle GAN Architecture

inspiration: (Matchue, 2020)

4.1 Implementation

4.1.1 Dataset

For our task we use the *ukiyoe2photo* dataset by *helloeyes* from kaggle The dataset contains 2 sets of unpaired 256 x 256 pixel RGB-images. Set A consists of images of the japanese Ukiyo-e art style, mostly portraits and scenes depicting characters and some landscape motives. Set B consists of landscape photographs depicting architectural and landscape motives. Set A will be used as the artistic-style source from which we apply the artistic style to the content images in set B (Helloeyes, 2022).

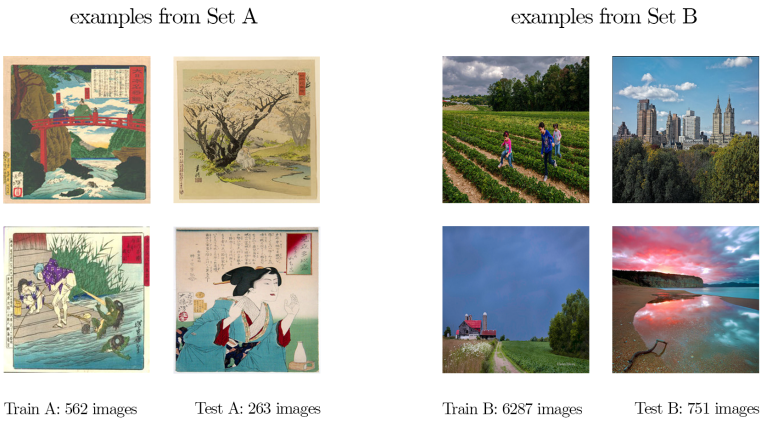

Figure 7: Cycle GAN Images

4.1.2 Preprocessing

To import the dataset from a local folder structure we used the

`tf.keras.utils.image_dataset_from_directory`

function. This allowed us to perform first steps of our preprocessing.

Initially we set the batch size to 64. That was followed by a long and tedious debugging process with unspecific errors, the sources of which were hard to discover. We tried various batch sizes ranging from 4 to 64. Due to the complex structure, these batch sizes still proved to be set too high. The loss calculations needed too much vRAM (we used a computer with 11GB of vRAM). Setting the batch-size to 1 (it is recommended to use batch-sizes of 1-2) resolved the problem. The resulting dataset was then split into training (70%), test (20%) und validation (10%). To speed up convergence and improve stability we normalised the pixel values from 0 to 255 to a range from -1 to 1.

4.1.3 Model Architecture

The networks main features are the 2 generators, 2 discriminators and the cycle loss. The following implementation is in accordance with the paper *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* (Zhu, 2020)

Generators
Each generator consists of an encoding layer, 9 residual blocks and a decoding layer. Each encoding layers consists of 3 Conv2D layers with ReLU activation functions. Each decoding layer consists of 2 transposed Conv2D layers with ReLU activation functions and a Conv2D layer with a tanh activation function.

Discriminators
Each discriminator consists of 5 Conv2D layers with leaky ReLU activation functions and a Conv2D patch layer to output the adversarial loss.

Cycle Loss
The cycle loss is calculated by the mean square error between the original content-image and the output of generator 2. The cycle loss is then added to the generators total loss.

Visualisation
In order to turn the results into visible images we created a visualiser class that turns the tensors back into images and a callback function to generate and save images after each epoch.

4.1.4 Training

For the training process we chose the ADAM optimiser. It has proven to be useful for many applications including ours. We used a learning rate of 0.0002 with a beta 1 value of 0.5 for a decaying learning rate.

For safe and fast global reconfiguration of training hyperparameters we implemented a configuration class. This allows us to save all settings and hyperparameters of each training run and transmit them to experiment tracking tools *Weights and Biases and Tensorboard* like for comparison of training results.

4.2 Results

The resulting images do resemble the desired Ukiyo-e style. The intensity of the effect varies depending on the original image content. We are impressed by the content detection ability of the network important features seem to be confidently retained.

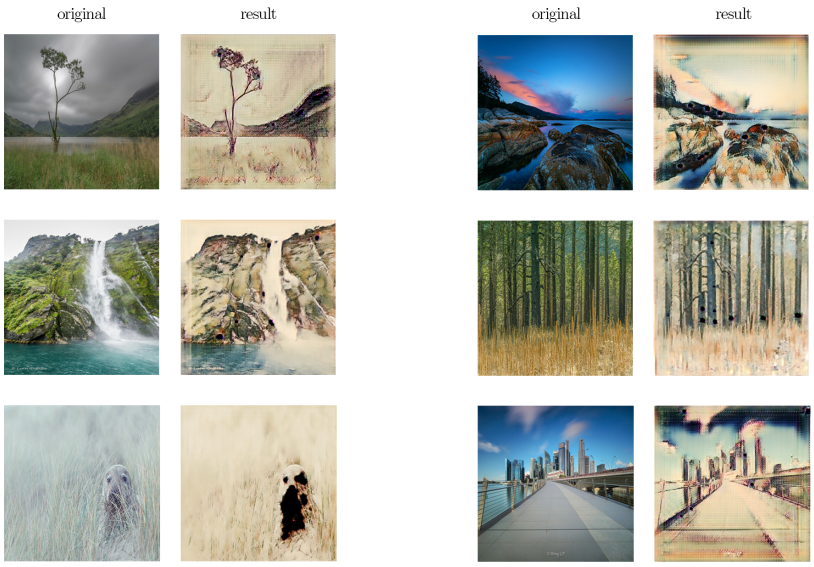

Figure 8: Style Transformations

A curious finding is that the results do not progressively get better with every generation. We also noticed black artefacts form in later generations of the same picture. Some images have them, some don't, so we are unsure about the source of these artefacts.

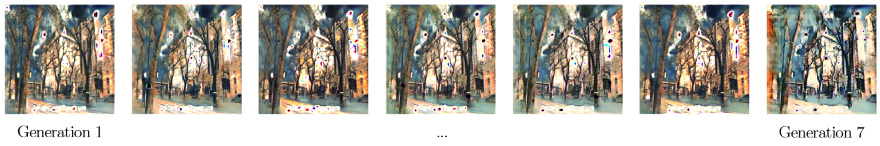

Figure 9: Style Evolutions

5 Discussion

Comparison of Model Outputs
When comparing the two results one must keep in mind the different sources of the style-information. Neural Style Transfer uses only one picture, whereas the CycleGAN uses a large set of samples to figure out the underlying style. This results in a more accurate translation by the CycleGAN. The CycleGAN creates

visually compelling images in which you can detect some common features to original Ukiyo-e printings: Broader areas often have one relatively coherent colour (colour blocking) or a smooth colour gradient, like in the case of the sky in the third CycleGAN example below. Both of these are methods used in the traditional Ukiyo-e printing technique. We assume that the CycleGAN was able to capture the underlying style principles from the Ukiyo-e genre better thanks to a bigger sample size. The Neural Style Transfer algorithm uses one style image as a reference per style transfer, which results in highly interesting textures that stem from prominent features in the style image. The results here vary widely picture by picture depending on how well the two input images fit together, but they mostly seem to have a more detailed and specific texture than the CycleGan images in common. This becomes very apparent in the first Neural Style Transfer example below, where Hokusai's waves are still clearly visible in the mixed picture.

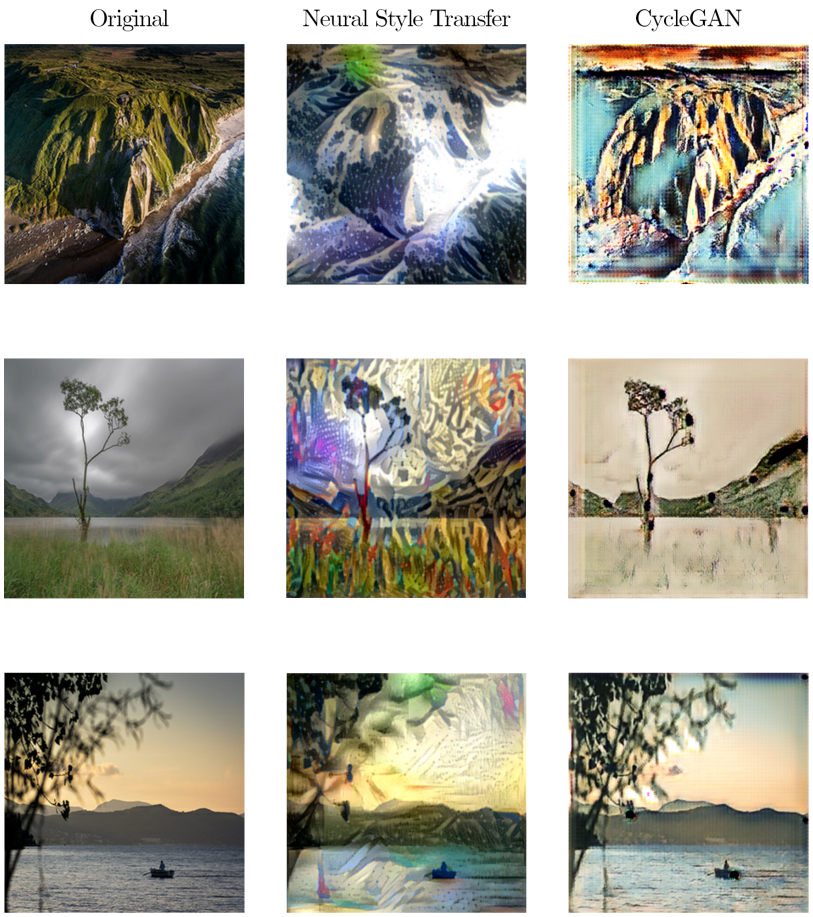

Figure 10: Comparison of Network Outputs

For the Ukiyo-e genre it is hard to say whether or not one of the networks performed better at style transfer, as both of them picked up different important aspects of the printing technique. However, the CycleGAN seemed to retain a more coherent quality in synthesising images in this particular art style. For further research it would be interesting to see how these two networks perform at transferring other art styles than Ukiyo-e.

References

Cyclegan. (n.d.). Retrieved from https://www.tensorflow.org/tutorials/generative/cyclegan

Efros, A. A. (2001). *Image quilting for texture synthesis and transfer.* Retrieved from https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/papers/efros-siggraph01.pdf

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015, Sep). *A neural algorithm of artistic style.* Retrieved from https://arxiv.org/abs/1508.06576

Helloeyes. (2022, Mar). *Ukiyoe2photo.* Retrieved from https://www.kaggle.com/datasets/helloeyes/ukiyoe2photo

Matchue. (2020, Apr). *Cyclegan explained in 5 minutes!* Retrieved from https://www.youtube.com/watch?v=-8hfnlxEPn4&ab_channel=Matchue

Nash, C. (n.d.). *Create data from random noise with generative adversarial networks.* Retrieved from https://www.toptal.com/machine-learning/generative-adversarial-networks

Saha, S. (2018, Dec). *A comprehensive guide to convolutional neural networks - the eli5 way.* Towards Data Science. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Singh, M. (2019, Oct). *Artistic style transfer with convolutional neural network.* Data Science Group, IITR. Retrieved from https://medium.com/data-science-group-iitr/artistic-style-transfer-with-convolutional-neural-network-7ce2476039fd

Yuan, R. (2018, Sep). *Neural style transfer: Creating art with deep learning using tf.keras and eager execution.* TensorFlow. Retrieved from https://medium.com/tensorflow/neural-style-transfer-creating-art-with-deep-learning-using-tf-keras-and-eager-execution-7d541ac31398

Zhu, J. Y. (2020, Aug). *Unpaired image-to-image translation using cycle-consistent adversarial networks.*