Artificial art GAN be real art

Abstract

A class of artificial models motivated by biological vision, known as Deep Neural Networks, can be harnessed to perform artistic style transfer. This paper focuses on rendering an image non-photorealistically using Convolutional Neural Networks. However, I verify that in certain cases, this method produces perceptually unappealing results by producing artefacts in the image or distorting important aspects – such as colour, texture, and facial details in the case of portraits – of the image. I argue and conclude that although the production of art by machines has been limited to artistic style transfer so far, within accomplishing the task of artistic style transfer, the combination of Markov Random Fields and the dCNN is able to improve the perceptual quality of non-photorealistically rendered images – while introducing a method for photorealistic rendering of an image – in comparison to the initial method which was purely based on the Convolutional Neural Network (CNN). Finally, when discussing how Generative Adversarial Networks (GANs) trained on a dataset of artworks can be modelled to generate art rather than simply combine the style and content of input images, I argue that masterful and valuable traditional art can be neither replicated nor replaced by an algorithm.

1. Introduction

Image synthesis is a classical problem in computer graphics and vision [3]. With neural style transfer, we hope to replicate in a simplified manner the process an artist goes through while creating a painting. We proceed under the assumption that the artist is attempting to paint a given picture in a given style of painting, which can be photorealistic or non-photorealistic. Style describes the building blocks the image should be made of, while content constrains their layout [2]. Here, we take the content to be the picture we want to change the style of. Numerous attempts have been made in the past to tackle the problem of artistic style transfer.

Neural networks were first used by Gatys et. al [1] to perform artistic style transfer. The paper renders a given photograph in the style of a range of well-known artworks. This problem is usually approached in a branch of computer vision called non-photorealistic rendering [4]. Conceptually most closely related are methods using texture transfer to achieve artistic style transfer [3]. However, this approach mainly relies on non-parametric techniques to directly manipulate the pixel representation of an image. Also, previous work on separating content from style was evaluated on sensory inputs of much lesser complexity, such as characters in different handwriting, images of faces or small figures in different poses [5, 6].

In contrast, by using Deep Neural Networks trained on object recognition, Gatys et al. [1] carry out manipulations in feature spaces that explicitly represent the high level content of an image. Since this first academic paper on neural style transfer, applications like Prisma and DeepArt have been created to use artistic style transfer to render photographs in an art style. Prior to 2015, image classification and recognition [7] as well as handwritten digit recognition [8] applied the concept of backpropagation to the fully connected layers in the VGG-19 to classify the input. However, the neural style transfer paper is novel in its approach of applying the concept of backpropagation to a cost function rather than to the usual classifier portion of the VGG-19.

Gatys et al. [1] analyse the rather newly found deployment of CNNs in artistic style transfer. This will help evaluate the ways in which neural style transfer has improved over the years. For instance, a major drawback of neural style transfer explored in this paper is the local implausibility of the output image in terms of the input content image. Hallucinatory artefacts from the style image may, in a perceptually unsatisfying manner, make appearances in the output image, while the original hues of the content image are taken over by the local colours of the style elements. This paper will hence consider a newer attempt at artistic style transfer that uses a combination of the generative Markovian Random Fields and the discriminatively trained deep CNN. This method is a non-parametric one, unlike the neural style transfer method, and similar to the methods of artistic style transfer prior to that of Gatys et al. [1].

Further, the Generative Adversarial Network (GAN) is used in image super resolution [13] to upscale images to larger scales and with greater perceptual quality due to the GAN's generative capabilities influenced by the manifold dataset used in its training. I explore this ability of the GAN in the context of generating art.

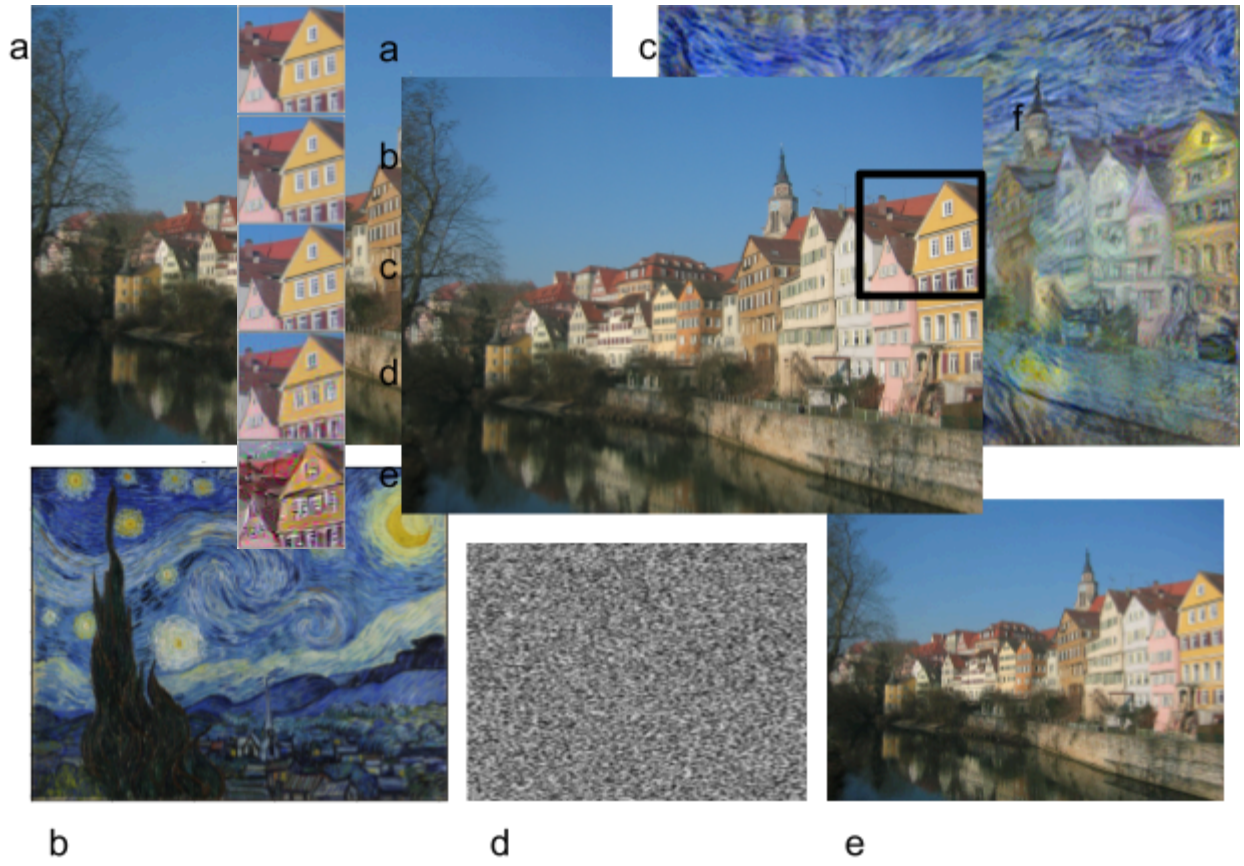
Section 2 largely outlines the step-by-step process to execute neural style transfer using the CNN and finally proposes a new method that overcomes the shortcomings (as verified and simulated in section 4) of the CNN based algorithm. Section 3 largely details the operation of the VGG-19 model and the calculation of the gram matrix. Section 5 discusses shortcomings of previous style transfer methods, describes a way of generating art, using GANs, in the absence of input content and style images, and outlines real world applications of the models outlined in this paper.

2. Analysis

The analysis largely details the process of neural style transfer as carried out using the VGG-19 CNN model. Section 2.1 mentions methods for image compression during preprocessing. Sections 2.2 – 2.5 describe in detail not only the considerations to be made while choosing layers for neural style transfer but also the loss functions that will be optimized using the Limited memory-Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) Optimizer in Section 2.6. Section 2.7 introduces MRFs for patch matching so as to apply style transfer to produce more perceptually pleasing results.

2.1. Reducing Image Size while Preprocessing:

While preprocessing the image before running it through any models, reducing the size of an image (eg. 24 bit depth to 8 bit depth per pixel) while retaining visual similarity helps make processing quicker. Colour quantization and superpixel segmentation are two image compression methods. Both processes use *K*-means clustering to do so. In colour quantization, *K* number of cluster centroids span the three dimensional Red-Green-Blue (RGB) space and each cluster takes the colour of the centroid pixel, while during superpixel segmentation an additional process, Simple Linear Iterative Clustering (SLIC), is applied where each pixel in each of the *K* clusters takes on the average colour value of the pixels in that segment (or cluster). So the higher the *K* value, the greater the bit depth per pixel, and the more visual similarity to the original image (see Figure 10, in Methods)



2.2. Neural Style Transfer:

We'll use the pretrained VGG-19 CNN model. It takes in three images individually: a content image, a style reference image (such as an artwork by a famous painter), and the target image, which is initialised to either a white noise image or to the content image itself (see Figure 1). Style is added to the target image such that it is transformed to look like the content image, but "painted" in the style of the style image.

Figure 1: An example of neural style transfer using CNNs. We take a) the content image as the Neckarfront in Tubingen, Germany; b) the style image as *The Starry Night* by Vincent van Gogh, 1889; and possible target images as either d) a white noise image or e) the content image. The output of the neural style transfer process is shown by c), where the photograph retains its content but looks as though painted by Van Gogh.

2.3. Going deeper into the VGG-19:

As the VGG-19 applies more filters to an image input into it, so as we go deeper into the model, the details of the content are lost and the feature maps detect high-level textures and shapes (see Figure 2). Review Methods for the inner workings of the VGG-19.

Figure 2: Content reconstructions of the content image when passed through the VGG-19. a)-e) show part of the content image output from the first convolutional layer of each of the five blocks in the VGG-19. Shallower layers detect low-level features like edges and simple textures. Deeper layers detect high-level features like complex textures and shapes, and because of this the spatial arrangement of objects in the content image is obscured in later layers.

2.4. Content extraction and content loss:

For content extraction, choosing a feature map in between the VGG-19 architecture means some amount of content is preserved while still making the picture less sharp (or detailed) so it can be filled in by the style (in Figure 1, Van Gogh's brush strokes). We then need a measure of the extent to which the content of the target image deviates from that of the content image, so that we can minimise this deviation. We do this by defining a loss function using a suitable middle layer from both the target and the content images. The content loss is then computed by averaging the summation, throughout the layer, of the squared error between an activation map from the content image and the corresponding map from the target image.

$$\text{Content\_Loss} = \frac{\left[\sum_{i=1}^{a} \left(AG_i - AC_i \right)^2\right]}{a},$$

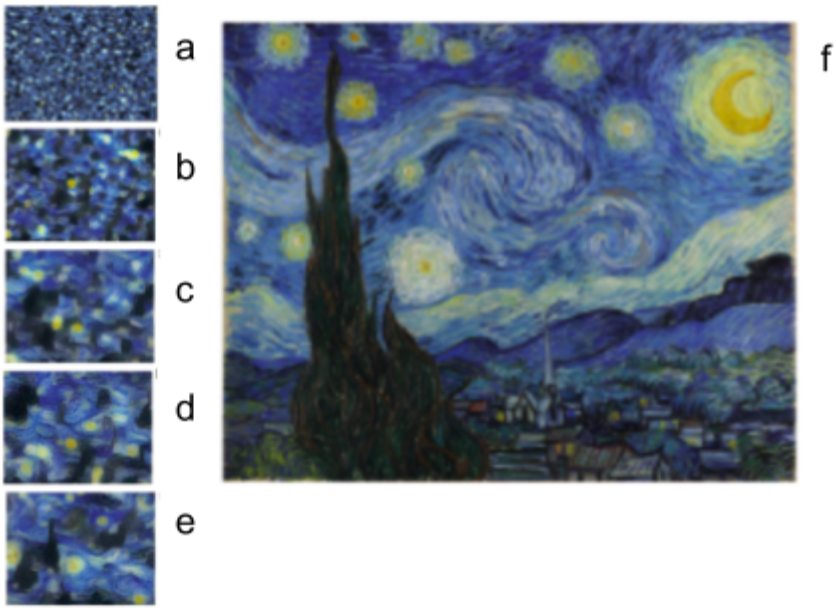
where a is the number of activation maps in the chosen layer, AG_i is the i^{th} activation map of the chosen layer produced by the generated image, and AC_i is the i^{th} activation map of the chosen layer produced by the content image. The content loss can be implemented as a python function [review Appendix] that takes in a layer (collection of feature maps) from the content image and the corresponding layer from the target image. Their squared error is then divided by the number of maps in that layer.

2.5. Style extraction and style loss:

VGG-19 extracts the features and content of an image well, as seen in content extraction. But the gram matrix needs to be applied to a layer to extract the style features of that layer. The gram matrix of a layer is essentially a measure of the correlation between the feature maps of that layer. To understand the implications of this, let's consider a toy example of a layer having just two feature maps. Say, for example, that map 1 extracts spirals and map 2 detects blue objects. If the gram matrix of that layer is high, then any spirals that may be present in the image have a high chance of being blue. Like this, all the maps in a given layer are correlated with each other using the gram matrix. See Methods for the calculation of the gram matrix.

Now that we have a way to measure the style in an image, we need a measure of the deviation between the style of the target image from that of the style image, so that we can minimise this deviation. We do this by defining a loss function that compares the gram matrices of a few layers from both the target and the style images.

Figure 3: Style reconstructions of the style image when passed through the VGG-19. In reference to the



blocks and layers produced by the VGG-19, the images show an overlay of the first layer(s) from the a) first block, b) first two blocks, c) first three blocks, d) first four blocks, and e) first five blocks (or all blocks). In style reconstructions closer to a), style features (for example, colour) are better detected while closer to e), style elements (for example, spatial arrangement and shapes of style structures) are prominent.

Once gram matrices are applied, layers close to the beginning are usually more effective in recreating style features while later layers offer additional variety towards the style elements (Figure 3). So, we'll need to take layers from across the VGG-19 to achieve a good balance of both; five layers are chosen for this purpose. If shallower layers need to be prioritized for more style features, each of the five layers will be given different weightings – where the shallower ones have greater weightings. The style loss is then calculated by simply averaging the weighted sum of the five squared errors.

$$\text{Style\_Loss} = \sum_{l=1}^{5} w_l E_l ,$$

where l denotes each of the five layers considered and E_l is the averaged summation of the squared error – between the target image and the style image – of the gram matrix of two vectorized feature maps in that layer for every combination of paired feature maps possible throughout that layer.

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

In the style loss python function [see Appendix], E_l takes the form of converting a layer into a matrix and multiplying the matrix with its transpose, to give a dot product, before taking the average squared error between this dot product of one layer of the target image and that of the corresponding layer of the style image. This style loss python function calculates the mean of the squared error loss of the gram matrices of only two layers – one layer from the style image and its corresponding layer of the target image. A later function finds the weighted sum of the five averages that the style loss function can calculate. This weighted sum is the total style loss.

2.6. Optimization:

Neural style transfer is an optimization technique; the total loss function Total_Loss = αContent_Loss + βStyle_Loss is minimized using a suitable optimizer. When we plot the cost function with inputs as the content loss and the style loss, the graph will show a minimum point, hence this is the point where the values of the content loss and style loss are optimal and desirable to us. To get to the minimum point of the graph modelled by the cost function, we modify the target image in iterations after learning from the graph.

For example, before the first iteration, given the target image was initialized to the content image, the content loss is zero and the style loss is arbitrarily large, giving a high total cost value. During the first iteration, the optimizer computes the gradient of the cost function at this point. It then takes a step proportional to the negative of the gradient – to ensure that the optimizer is performing gradient descent, which is the motion toward the minima of a function. The random interval it steps in this iteration is determined by the learning rate, which changes iteratively. The taking of the step commonces the first iteration, where the content loss and style loss at the new point are used to alter and update the target image. Next, the gradient of the cost function in this new point is calculated, and the process iterates. This is the process of backpropagation, wherein the desired outcome – here, the minima of a cost function – dictates how the variable inputs – here, the content and the style loss – are revised.

2.7. An improvement using MRFs:

Figure 4: Comparison of Gatys et al. [1] with Li et al. [2] for artistic synthesis. Content images credited to flickr users *Christopher Michel* (top) and *Peter Dahlgren* (bottom). Output images credited to Li et al. [2]. The eyes in the results of Gatys et al. don't match those of the content image.

The MRFs allow for a comparison of neural activations (or patches) rather than pixel values. Patches having similar content from the style and content images are first matched, using activations from the intermediate ReLU layers, and then are blended together using optimization techniques as used by Gatys et al. [1] so that the patch in the content image is rendered in the style of the corresponding patch in the style image. In general, this method creates more stylish images (Figure 4 and Figure 5), but may contain artifacts when the MRFs do not fit the content. In contrast, the parametric method [1] is more adaptable to the content, but at the cost of deviating from the style [2] (Figure 5 and Figure 6). The method of Li et al. [2] also works exceptionally for non-photorealistic rendering (Figure 5). Since the method of Gatys et al. [1] uses parametric methods of style transfer unlike that of Li et al. [1] that matches patches before transferring texture, the style tends to be applied to all regions of the content image, producing undesirable marks and making the results (in Figure 4, Figure 5, and Figure 16. h)) perceptually displeasing in comparison to those of Li et al. [2]. See the results section for further shortcomings of the method of Gatys et al. [1] that can in theory be overcome by the method of Li et al. [2].

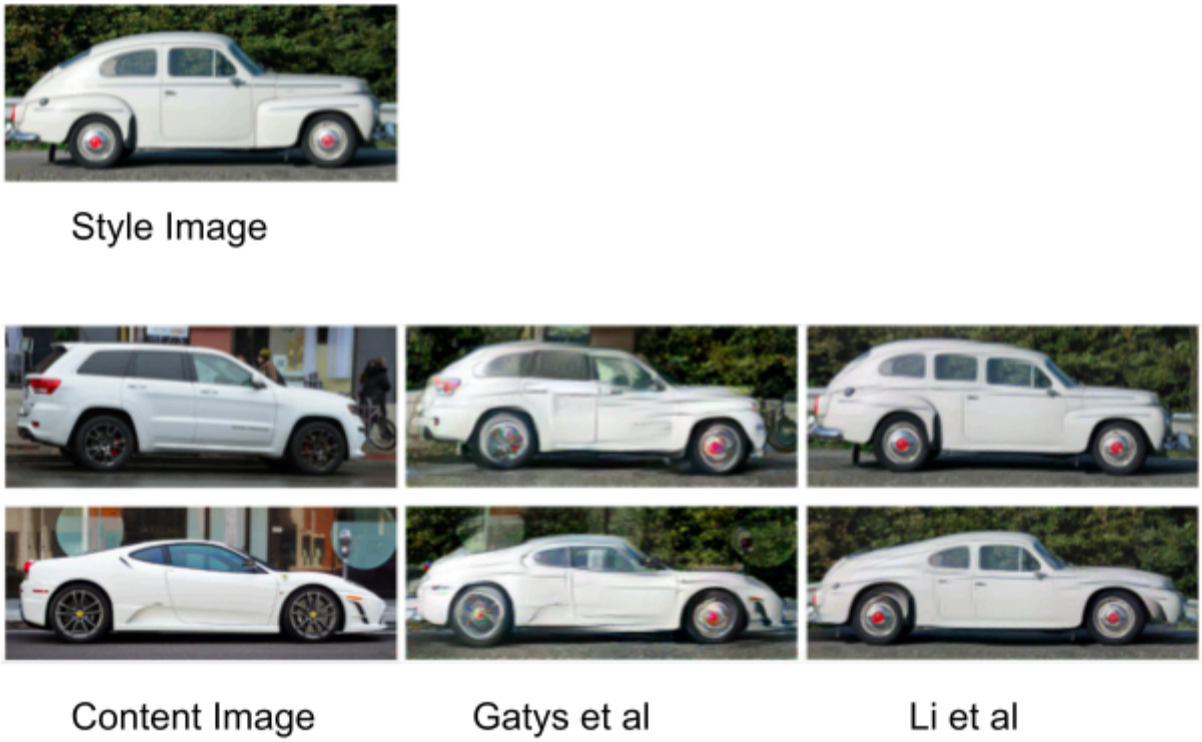


Figure 5: Comparison of Gatys et al. [1] with Li et al. [2] for photo-realistic synthesis. Input images credited to flickr users *Brett Levin*, *Axion23* and *Tim Dobbelaere*. Output images credited to Li et al. [2].

Figure 6: Illustration of the importance of patch matching in the method of Li et al. [2] and its differences with that of Gatys et al. [1]. The car altogether vanishes using the former method while the latter method remains true to the content, thereby undesirably producing a car of distorted style and colour.

3. Methods

3.1. The VGG-19:

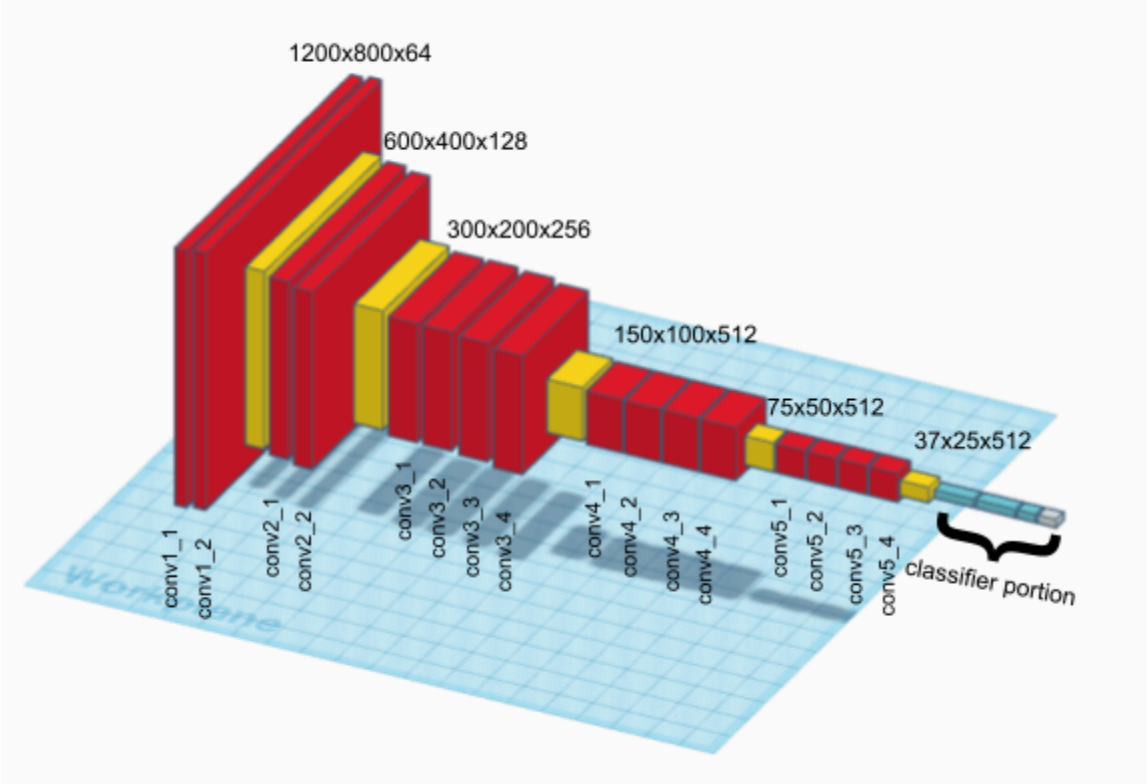


Figure 7: Visual representation of the VGG-19 model complete with the various blocks and layers it produces, where conv3_4 labels the fourth layer in the third block.

To understand neural style transfer, we need to understand the VGG-19 model, a pre-trained CNN model trained on the large ImageNet data set in a supervised manner. When an image is passed into the VGG-19 (this includes each of the three images of content, style, and target), a number of filters (or kernels) are applied to it. Each kernel extracts a specific feature of that image (see Figure 8).

The feature maps produced as a result of the filters can be grouped into 47 layers, which can be further clustered into 8 blocks, where the last 3 are the fully connected and the softmax layers, which form the classifier portion (as in Figure 7). Given an input image of dimensions 1200x800x3, the figure above shows the corresponding dimensions of each layer of the image after it has been run through the VGG-19 model. The last three blocks classify the image by finding its probability resemblance to other images, and hence are not important to us.

Each block has at least one convolution layer, a ReLU layer (which helps include non-linearity of pixel values) and a max pooling layer. A dot product between the filter and input is taken at every convolutional layer. This computation is called a convolution, where two functions (here, the kernel and the image) are combined to get a third function that describes how one is modified by the other. The kernel overlaps with part of the input image, a dot product is generated between the kernel and the overlapped portion of the image before the kernel moves by the stride value. The stride is the number of pixels the kernel moves

after each dot product. Each dot product taken makes a single pixel value of the feature map produced at the end of the convolution.

Convolution uses padding (adding zeroes around the boundary) so that the size of the feature map output from each convolutional layer equals the size of input of that layer. Max pooling doesn't pad the input, resulting in a smaller feature size, which is the process of downsampling. The size of a resulting feature map is given by $[W - K + 2P]/S + 1$, where W is the size of the input image, K the size of the kernel, P the size of padding, and S the stride of the kernel. By default at a convolution, the stride and the padding values are set to one and each kernel is of size three. So, we can see that the resulting feature map is of size W.

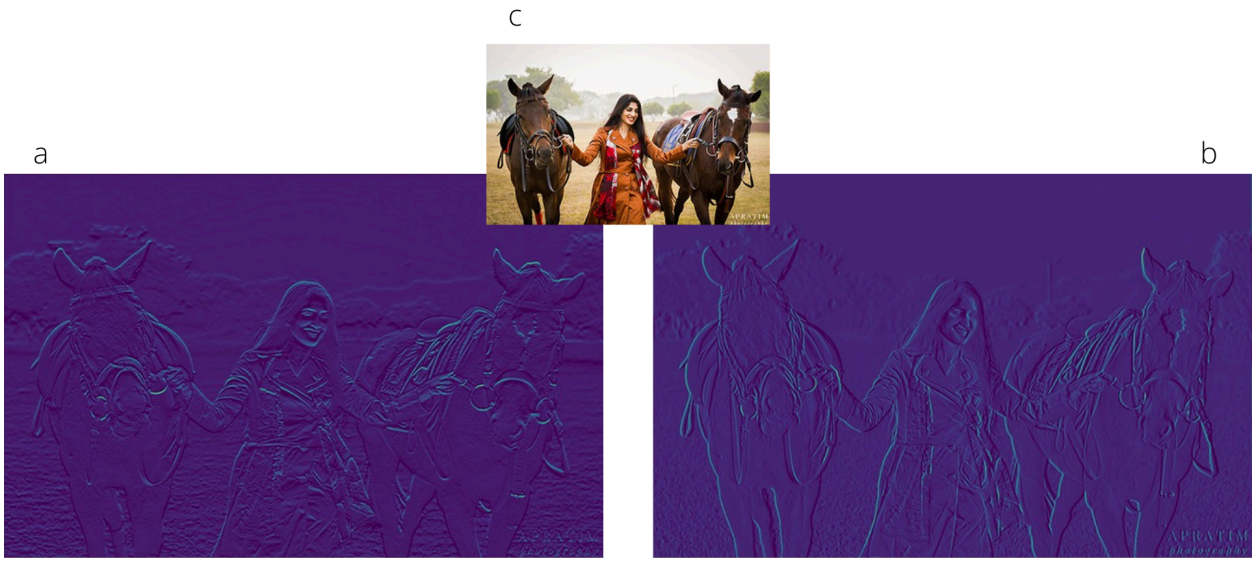


Figure 8: Feature responses from the VGG-19. a) the product of a kernel that detects horizontal edges, with a preference for topmost edges and b) the product of a kernel that detects vertical edges, with a preference for leftmost edges, are different activation maps from conv1_1, produced on passing image c) through the VGG-19. Images credited to Medium user *Apratim Sahu*.

3.2. Calculating the Gram Matrix:

We first need to reshape the layer so that it can be represented as a rank two tensor (or a matrix) rather than the rank three tensor it is now. All the pixels from a feature map are lined up into a single row, so the two dimensional map is now transformed into a one dimensional array of pixels. The number of feature maps in that layer (128 in Figure 9) then becomes the number of rows of the new matrix formed. The gram matrix of a number of vectors having the same dimensions is most efficiently calculated by first lining the vectors into a matrix and then taking an inner product between the matrix and its transpose (matrix formed by rotating another matrix by 90 degrees), as done in the style loss function in the Appendix. In contrast, the formula for E_l outlined previously tediously takes the gram matrix between every pair of vectors (again, derived from feature maps) possible within a layer.

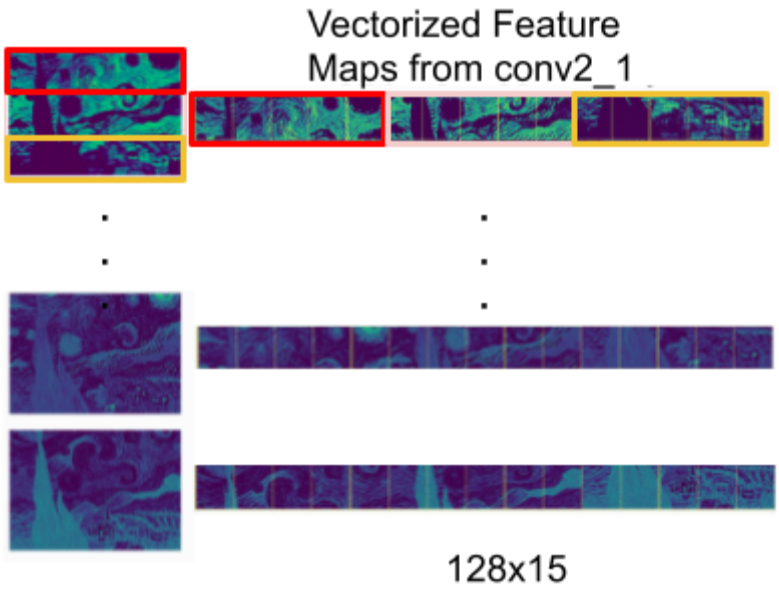


Figure 9: A visualisation of feature map vectorization; we operate under the assumption that each map is of dimension 3x5, or has 15 pixels. Images credited to Medium user *Apratim Sahu*.

3.3. Reducing Image Size while Preprocessing:

As in section 2.1, the size of an image can be reduced while retaining perceptual similarity. To do so, an optimal number of clusters needs to be chosen to prevent too small a pixel depth (Figure 10. a)), which reduces visual appeal or too large of pixel depth (Figure 10. c)), which may not reduce the size much.

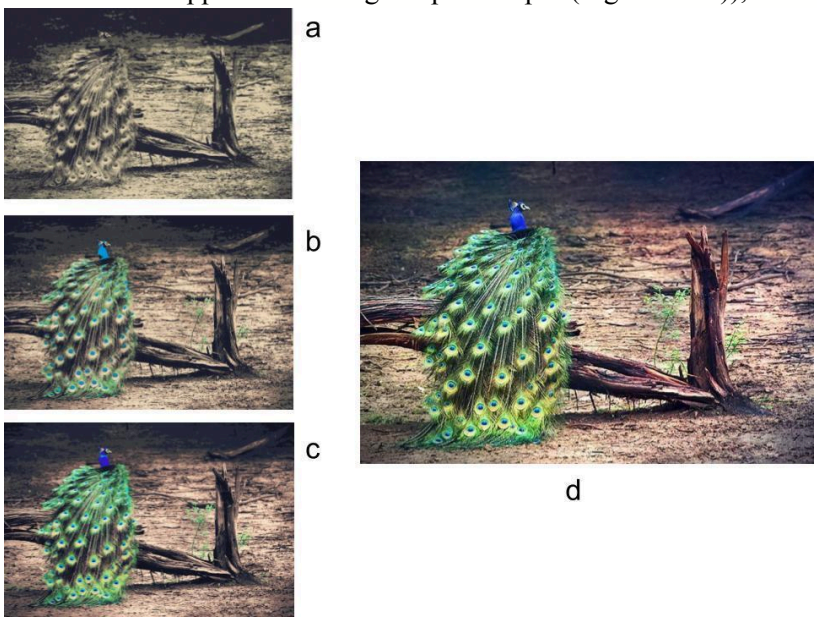


Figure 10: Reducing image size using *K*-means clustering. d) shows the original image. a) - c) show the image made up of *K* colours where $K = 4, 8, 15$ respectively. Increasing pixel depth (or *K* value) increases visual similarity to the original.

4. Results

The content images Figure 11. a), Figure 11. d), Figure 12. g), Figure 13. c), Figure 13. e), Figure 14. e), Figure 15. b) and Figure 15. e) were photographed by me.

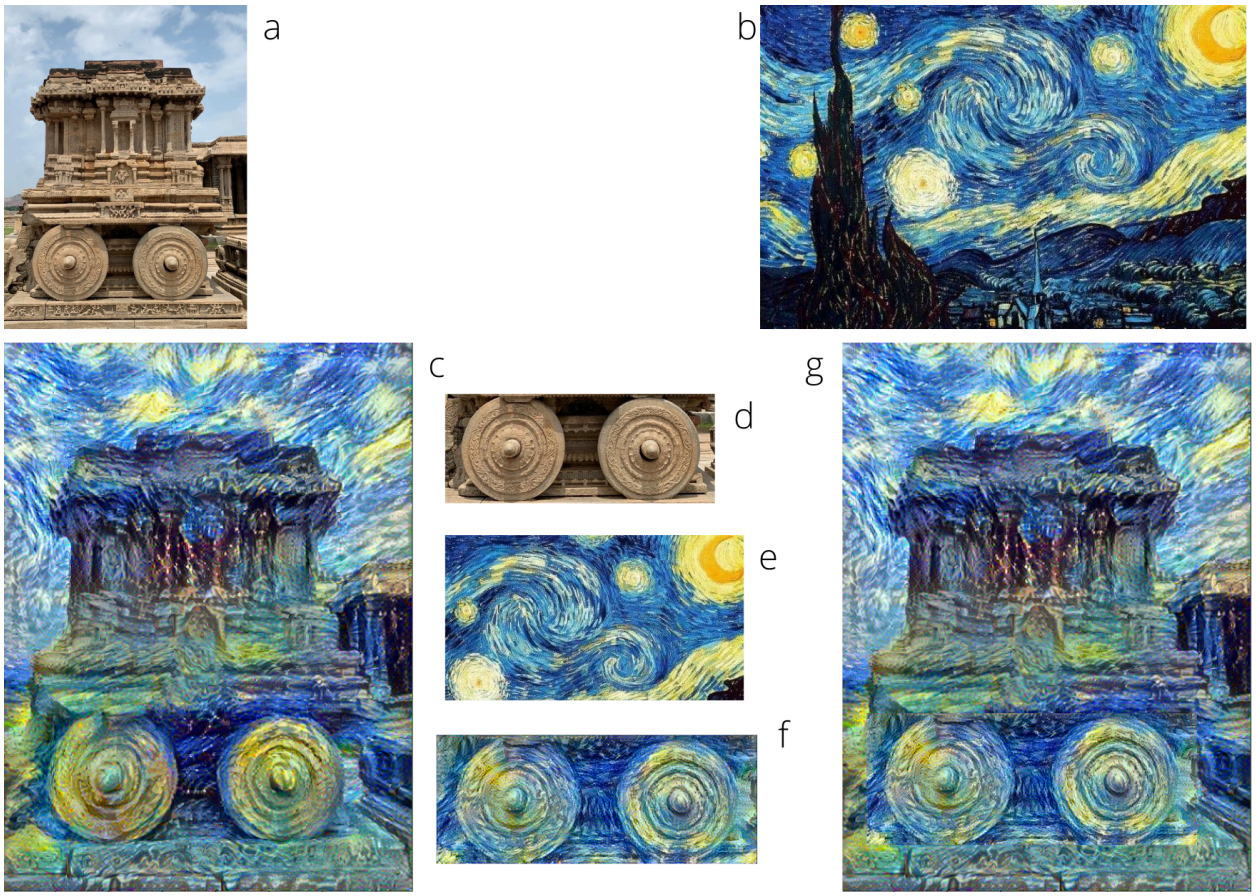


Figure 11: a) and d) are rendered with style images b) and e), giving results c) and f) respectively. A lack of swirls in the wheels of c), as a result of the method of Gatys et al. [1] that transfers the style of the ground in b) to the wheels in a), can be compensated by synthesising d) and e) and then stitching f) and c) to produce a more visually pleasing g). Also, the bright daylight of the content image is replaced with a dull and gloomy blue in the output images, while the ornamental details of the architecture are lost to the larger mesostructures of the style elements.

Figure 12: Rendering g) in the style of a) complements the colours in g), and this is desirable since g) is beautified by its sunset. Rendering f) in the style of b) gives c) while rendering g) in the style of b) gives e).

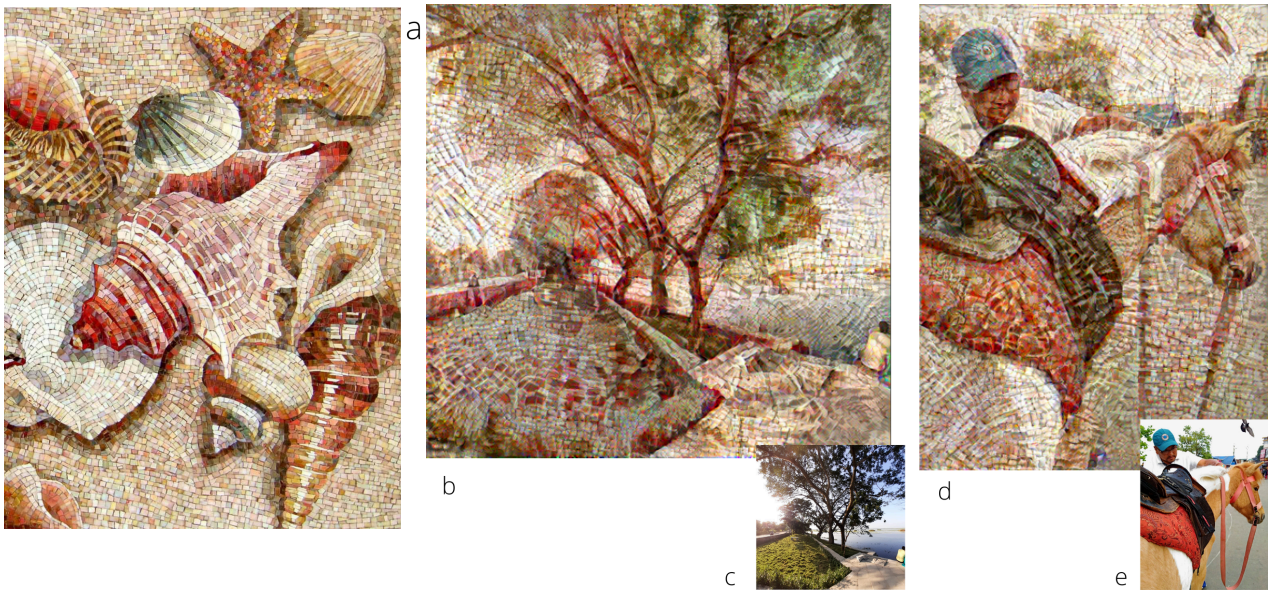


Figure 13: c) and e) are rendered in the style of a) to produce the

stylish outputs b) and d) respectively, with the hues of the content images more or less retained in the output images.

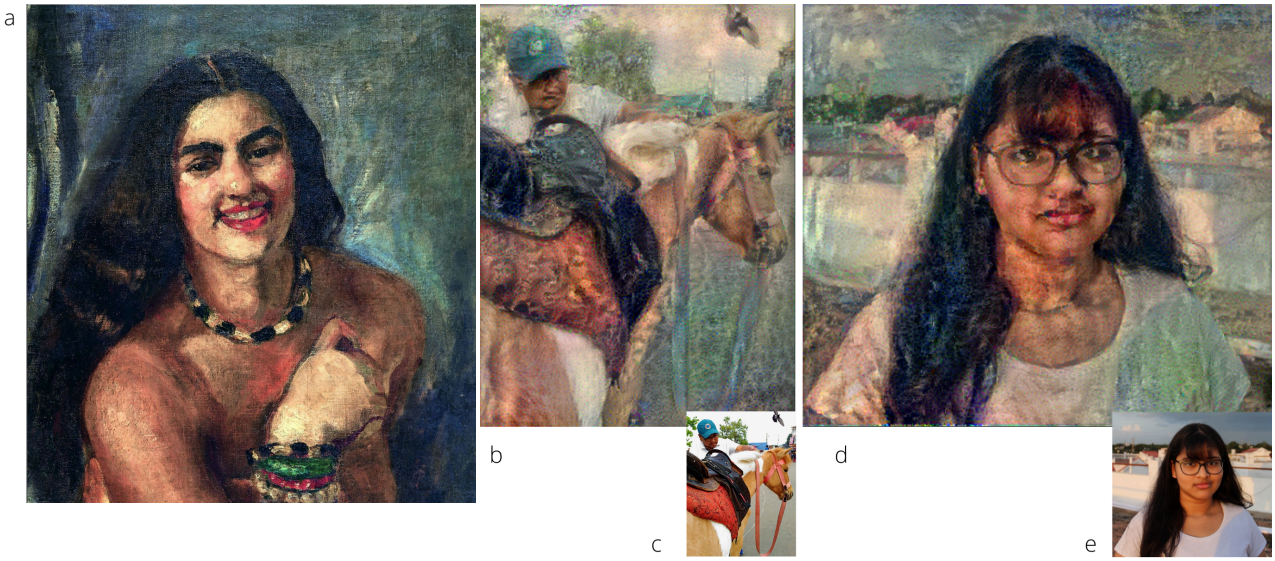


Figure 14: c) and e) are rendered in the style of a), a Self-portrait by Amrita Sher-Gil (1930), to produce the majestic outputs b) and d) respectively.

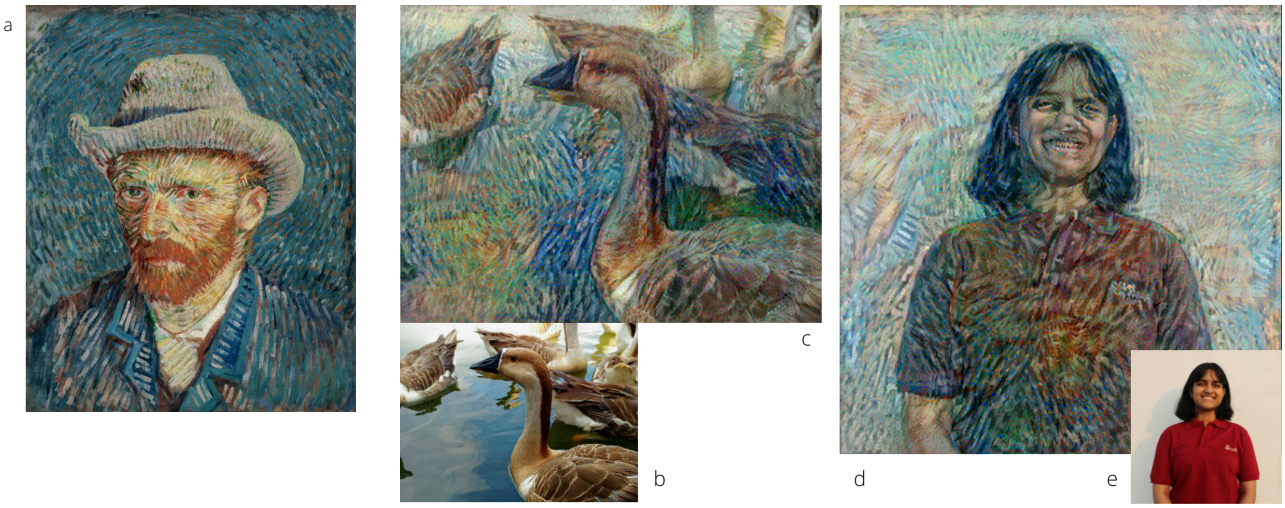


Figure 15: b) and e) are rendered in the style of a), a *Self-Portrait with Grey Felt Hat*, Van Gogh (1888), to produce the beautiful outputs c) and d) respectively.

Figure 16: A basic digital portrait doodle a) is rendered in the styles of d) and e) to produce b) and c) respectively. The hallucinatory artefacts and marks on the persona and the distorted backgrounds in b) and c) show that the method of Gatys et. al [1] may need to be improved using that of Champandard [12] for doodles as content inputs. h) shows artefacts in the background as well as a darker background when f) is rendered in the style of g), necessitating the non-parametric method proposed by Li et al. [2].

The method of Gatys et al. [1] produces images where the hues of the content image have been overridden by those of the style image, sometimes destroying the original beauty of the content image as seen in Figure 12. e) where the golden sunset has turned purple. The results in Figure 11 have also lost the brightness of daylight in the content image. Whereas, in Figures 13-15, the results are exquisite pieces of art. However, they can be improved. In the results of Figure 13, the mosaic pattern of the shells can be better applied to the content image with little distortion of individual squares, something that can be accomplished in a more stylish manner using the method of Li et al. [2]. The eyes in the results of Figure 14 don't match those of the content image, which can be fixed by the method of Li et al. as in Figure 4.

5. Discussions

Since the generative MRFs are performing patch matching (or generation) between the content and style images before actually blending them, the method of Li et al. [2] would require the style and content images to be considerably similar in not only content but also the spatial arrangement of the objects in the images. So, it only works if the content image can be re-assembled by the MRFs in the style image – for example, images of strong perspective or structure difference are not suitable for this method [2], as in Figure 6. Also, this method fails to produce output images as sharp as the original images. This is due to the loss of non-discriminative image details during the training of the network [2]. Intriguing future work that tries to combine dCNNs with stitching based texture synthesis like [9] can perhaps produce photorealism at the pixel-level in output images. Retraining of the dCNN can also be sought after in an attempt to mitigate the shortcomings of the methods of Li et al. [2].

On a more realistic note, image processing methods can be deeply beneficial to the artist community. The methods discussed above can be used to deliver quick responses to artistic experimentations, helping the artist save time and effort since he can promptly judge if the visualization in his mind for a new piece of art will yield productive results when pursued using traditional media. The sometimes unexpected results produced by the machine have the ability to inspire the artist and even influence his style. Hence, instead of competing with traditional artists, these algorithms can be harnessed by artists to support and reform art in the traditional sense. AI can be used to the advantage of that creator who may have a creative bearing but may lack the skill and time to implement their ideas. For instance, Champandard [12] works on creating masterful art pieces using simple and quickly made two-bit doodles. This model does however require an input of finer art pieces, such as those of Vincent Van Gogh, to copy style elements from.

However, the creator is still required to learn how to use these high-level models – for example, where and when he must tweak the parameters. Or, he must at least be able to communicate his ideas to a professional. It is often not easy to formulate words to describe a creative visualization. These are some considerations to be made before a creator makes use of artistic style transfer algorithms. For instance, Van Arman, the programmer of Cloudpainter, served as more of an "art director" for the painting

produced by the software. Although Cloudpainter made all of the aesthetic decisions independently, the machine was given parameters to meet and was programmed to refine its results in order to deliver the desired outcome [10]. This goes to show how none of the models discussed thus far can possibly produce or imagine any new images, scenes, and aesthetic visualizations worthy of being painted or rendered. It works purely on the directions and inputs of the programmer.

Although this makes it seem like neural network based algorithms can hardly replicate traditional art, it opens up a greater field of research that can explore the utilization of hyper-parameter based generative models (especially Generative Adversarial Networks, or GANs) with the discriminator machine of the GAN trained in an unsupervised manner on the vast database of artistic images, paintings, prints, and visulas available to us from across centuries. Depending on the period, style, and colour palette of the piece one would like to generate, the model can be trained on the set of paintings that identify with that category. The discriminator then detects the underlying pattern and structure in the dataset so as to be able to calculate the difference between the images in this 'real' dataset and the 'fake' image generated by the generator. As we minimize this difference or loss function using backpropagation in iterations that generate better images each time, the discriminator gets worse at telling the difference between the generated image and the dataset while the generator produces art whose style, content, and colour palette have a high correlation with those of the art in the training dataset. The popular website 'ThisPersonDoesNotExist.com' [14] does exactly this in using a dataset of human faces to generate fake but hyperrealistic portraits of human beings. We must note, however, that the process is not yet fully automated as the human being must still pick the appropriate learning rate and dataset in consideration of various features – like colour, content, style, form, shape, texture, and so on – for training.
Regardless of the type of machined art generation, we bring attention to the sources of any artistic dataset: skillful artists of the past. And, to be able to continue their legacy of talent, creativity, and productivity is an honour for contemporary artists. Moreover, besides being a therapeutic form of expression, a traditional art piece is a product of the individual experiences, values, and knowledge a singular artist brings to the community; the traditional art scene will not and must not be dampened regardless of technological progress.

6. Conclusion

The key insight of this paper is to elaborate on the techniques used by Gatys et al. [1] while carrying out non-photorealistic rendering using neural style transfer. We discuss the choosing of certain layers of the VGG-19 for content and style extraction, and form content and style loss functions. The method of Gatys et al. is unique as it utilizes the concept of backpropagation, which was previously used in image classification tasks, to minimize loss functions. However, disadvantages to this model exist and are explored in their distortion of the original hues of the content image, production of artifacts, and perceptually unpleasant applications of style to the content image. Further, improvements to this with non-parametric techniques of patch matching and blending using the generative MRFs and discriminative dCNNs are discussed. These not only produce more perceptually satisfying results by detecting mesostructures, which are structures of intermediate size, in the style images, but also aid in photorealistic rendering that can, for example, combine the content of one car image with the style of another car image to produce a brand new car model (Figure 5). Drawbacks to these two models are then outlined while proposing two potential fields of study to improve the methods discussed thus far.

Appendix

All code snippets below are derived from the GitHub repository [11] contributed to by GitHub users *MarkDaoust, raymond-yuan, Tylersuard, sihyeon-kim, jurgenizer,* and *fuzzythecat.*

Calculating the content loss for a given layer:

```
def get_content_loss(base_content, target):
        return tf.reduce_mean(tf.square(base_content - target))
```

Calculating the style loss for a given layer:

```
def get_style_loss(base_style, gram_target):
        """Expects two images of dimension h, w, c"""
        # height, width, num filters of each layer
        height, width, channels = base_style.get_shape().as_list()

        gram_style = gram_matrix(base_style)

        # We scale the loss at a given layer by the size of the feature map and the number of filters
        return tf.reduce_mean(tf.square(gram_style - gram_target))
```

Calculating the gram matrix for a given layer:

```
def gram_matrix(input_tensor):
        #input_tensor is the 3 dimensional layer

        #yields the no. of channels
        channels = int(input_tensor.shape[-1])

        #input rank 3 tensor is converted to a matrix
        a = tf.reshape(input_tensor, [-1, channels])

        #this yields the number of pixels in 1 feature map (no. of rows in a)
        n = tf.shape(a)[0]

        #calculates the gram matrix
        gram = tf.matmul(a, a, transpose_a=True)

        #converts n to float32, divides gram matrix by n
        return gram / tf.cast(n, tf.float32)
```

Acknowledgements

This work has been supported by Lumiere Education. I thank my mentors Mr Kharel and Mr Moulton for inspiring discussions and useful tips.

References

Gatys, Leon, et al. “A Neural Algorithm of Artistic Style.” *Journal of Vision*, vol. 16, no. 12, 2016, p. 326, doi:10.1167/16.12.326.

[2] Li, C., & Wand, M. (2016, January 18). *Combining Markov random fields and convolutional neural networks for Image Synthesis*. https://arxiv.org/abs/1601.04589.

[3] Efros, A.A., Freeman, W.T. (2001). *Image quilting for texture synthesis and transfer.* https://people.eecs.berkeley.edu/~efros/research/quilting.html.

[4] Kyprianidis, J. E., Collomosse, J., Wang, T. & Isenberg, T. (2013). *State of the ”Art”: A Taxonomy of Artistic Stylization Techniques for Images and Video.* https://ieeexplore.ieee.org/document/6243138.

[5] Tenenbaum, J. B. & Freeman, W. T. (2000). *Separating style and content with bilinear models.* https://direct.mit.edu/neco/article/12/6/1247/6381/Separating-Style-and-Content-with-Bilinear-Models.

[6] Elgammal, A. & Lee, C.-S. (2004). *Separating style and content on a nonlinear manifold*. https://ieeexplore.ieee.org/document/1315070.

[7] Ciresan, D., Meier, U., Masci, J., Gambardella, L. M., Schmidhuber, J. (2011, July 16). *Flexible, High Performance Convolutional Neural Networks for Image Classification.* https://www.aaai.org/ocs/index.php/IJCAI/IJCAI11/paper/view/3098/3425.

[8] Calderón, A., Ovalle S. R., Victorino, J. E. (2003, January). *Handwritten Digit Recognition using Convolutional Neural Networks and Gabor filters.* https://www.researchgate.net/publication/277285568_Handwritten_Digit_Recognition_using_Convolutional_Neural_Networks_and_Gabor_filters.

[9] Kwatra V., Schodl A., Essa I., Turk G., & Bobick A. (2003). *Graphcut textures: Image and video synthesis using graph cuts*. https://www.cc.gatech.edu/cpl/projects/graphcuttextures/.

[10] Weiner, K. (2018, November 12). *Can AI Create True Art?* Scientific American Blog Network. https://blogs.scientificamerican.com/observations/can-ai-create-true-art/.

[11] Tensorflow. (2020, April 25). *Models/4_Neural_Style_Transfer_With_Eager_Execution.Ipynb at b2d3a05c82858650f4dc63ec1a86197d1fdd77b6 · tensorflow/models*. GitHub. https://github.com/tensorflow/models/blob/b2d3a05c82858650f4dc63ec1a86197d1fdd77b6/research/nst_blogpost/4_Neural_Style_Transfer_with_Eager_Execution.ipynb.

[12] Champandard, A. J. (2016). *Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artwork.* https://www.semanticscholar.org/paper/Semantic-Style-Transfer-and-Turning-Two-Bit-Doodles-Champan

dard/3e837ba2bfcb028e2331cd3b0e5fefddc5b1c2a4.

[13] Ledig, C., Theis, L., Huszár, F., Caballero, J., Aitken, A.P., Tejani, A., Totz, J., Wang, Z., & Shi, W. (2017). *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network.* https://www.semanticscholar.org/paper/Photo-Realistic-Single-Image-Super-Resolution-Using-Ledig-Theis/df0c54fe61f0ffb9f0e36a17c2038d9a1964cba3.

[14] *This person does not exist*. This Person Does Not Exist. (n.d.). https://thispersondoesnotexist.com/.