Recreating Traditional Indonesian Batik with Neural Style Transfer in AI Artistry

Michael Joseph
Computer Science Department,
Faculty of Computing and Media,
Bina Nusantara University,
Jakarta, Indonesia 11480
michael.joseph@binus.ac.id

Jeconiah Richard
Computer Science Department,
Faculty of Computing and Media,
Bina Nusantara University,
Jakarta, Indonesia 11480
jeconiah.richard@binus.ac.id

Calvin S. Halim
Computer Science Department,
Faculty of Computing and Media,
Bina Nusantara University,
Jakarta, Indonesia 11480
calvin.halim@binus.ac.id

Rowin Faadhilah
Computer Science Department,
Faculty of Computing and Media,
Bina Nusantara University,
Jakarta, Indonesia 11480
rowin.moenaf@binus.ac.id

Nunung N. Qomariyah
Computer Science Department,
Faculty of Computing and Media,
Bina Nusantara University,
Jakarta, Indonesia 11480
nunung.qomariyah@binus.ac.id

***Abstract*—Style transfer is a method of combining two images into one, taking reference of one of the image's styles and the other image's content. Convolutional neural networks have been applied to this method to produce what is known as neural style transfer. Using VGG-Network, the artificial system was able to recreate artistic images by combining different content and style. However research regarding the effects of different models and optimization to the quality of the image produced are limited. The aim of this experiment is to compare between the VGG19 and VGG16. We use data and results from Leon A. Gatys to compare between two different models which are VGG19, and VGG16 in terms of content loss and style loss. This architecture is also applied to the more focal style of Batik in this research to experiment the effects of a dominant color and pattern on another image. With Indonesia's rich culture and its diverse art portfolio, it is only natural that this paper explore neural style transfer's effects on the creative pattern forming of Batik.**

***Index Terms*—Style transfer, CNN, Deep Learning, Computer Vision, Artificial Intelligence, Batik**

I. Introduction

Computer-generated artwork has been estimated to be developed first back in the 1970s, where Harold Cohen began creating the AARON system [1]. It has since evolved, using higher and higher echelons of Artificial Intelligence (AI) and neural networks in order to generate a more convincing and anthropomorphic art. Fast forwarding to 2018, the first infamous instance where a machine learning algorithm called Generative Adversarial Network (GAN) generated a portrait to be sold at Christie's art auction. This AI generated portrait was sold at an astounding 432,500 USD [2]. This phenomenon has certainly exceeded most people's expectations including the auction host Christie, which at first glance valued the portrait at around 10,000 USD. This raises the question of artificial creativity, which was widely renowned to be monopolized by humans exclusively. The notion that Artificial Intelligence might be more creative than humans have also been floating around. Take chess for instance, it is a game strongly gravitating towards move calculation and opening preparation. At the highest level of chess, opponents can play the first few moves purely from memory and preparation. However, DeepMind's artificial chess engine AlphaZero has consistently beaten the world's top players. Vladimir Kramnik, the former world chess champion even goes as far to say that its ability to adapt and explore different variants of the game makes the game "more beautiful". Another one of DeepMind's arsenals, AlphaFold has been able to solve a 50-year old grand challenge of biology that no human was able to achieve. It is able to predict the structure of dozens of proteins and enzymes which is said to be a scientific breakthrough that can help doctors come up with new types of medicine. Through these metrics of discovering, creating and coming up with new variants of known instances, machine learning and artificial intelligence have more creative capability than humans. Hence, why not apply AI into the field where creativity is a highly observed and valued element, the field of art.

The applications of AI in art can be classified in one of two forms: Analyzing existing art; or generating new art. In our case, this paper will proceed to describe the works of artificial intelligence in the practice of a concept called style transfer to generate new art. Style transfer is an algorithm that takes 2 input images and blends together the two images to create a new image with the same content from the content image but using the style from the reference image [3]. This practice was further developed by Leon A. Gatys, Alexander S. Ecker, and Mathias Bethge by implementing convolutional neural networks (CNN). In our work, we show how Convolutional

978-1-6654-1697-9/21/31.00 ©2021 IEEE

Neural Networks are utilized to transform the content and style of images using another image as a reference. By extracting the features using state-of-the-art CNNs and applying some loss functions that will be minimized over a number of iterations, the goal is to generate an image retaining the main features extracted "drawn" in the style of another image.

Not to mention, we applied the architecture with hopes of projecting the style patterns of Batik, a product of coloring designs on textiles by dyeing them with wax resist. Batik is the embodiment of the Indonesian art culture and is deeply rooted in our day to day life. So much so that on the 2nd of October, 2009, batik was declared to be a Humanitarian Heritage for Oral and Intangible Culture by UNESCO [4]. It is mostly a textile worn to formal, or casual events and very commonly used in rituals, ceremonies, traditions, and celebrations. Originating from the island of Java, Batik's motifs vary from region to region. Since it originates from using wax to draw patterns on different textiles, the main color palette, although not limited, is brown. It is rich in symbols and philosophy, where every stroke is inspired by the diverse culture and environment surrounding the tropical artist. As a part of Indonesian's identity, Batik has been passed down from generation to generation to represent the Indonesian culture and acts as a medium of appreciation for Indonesia's history. Batik is also made by repeating the same patterns throughout the fabric with some variations and reflections which is why it will be interesting to see what its neural style transfer product will look like. The batik making process is not a single method of engraving wax but an entire process. The traditional process requires a lot of creativity, hard work, patience and consistency. Each batik is first designed on a cotton cloth and that design will be covered with wax. After that, the wax will be cleaned from parts of the design and another chosen fixed color will be coated onto the cloth. Finally the cloth is immersed into the main color palette and cleaned with hot water [5]. This long and grueling process produces a textile result that is absolutely worth the time and effort put into it, which is why a fine quality batik which takes four months to produce will cost at least 75 USD. Previous research was performed to produce batik patterns with style transfer, but there is a key significant difference in which they implemented a different cost function for the color of the images using a k-means algorithm in order to distinguish the dominant color of the style image [6].

II. Model and Techniques

There exists a number of strong algorithms that enable the synthesis of photorealistic natural textures by using the pixels of another given source texture. Photorealism can be defined as a genre of art that envelopes painting, sketch and other graphic media where an artist ventures in reproducing an image as realistically as possible after studying a photograph. One example is an algorithm that takes a sample image and a user drawn target pattern and creates an image that is visually similar to that base image, drawn in the user created pattern [7]. However, there is an extremely obvious constraint in these algorithms, which refers to its low-level features of the target image as seen in the middle image of Figure 1. This constraint implies that a prerequisite of finding target pattern image representations to draft its base model variation style must be satisfied [8].

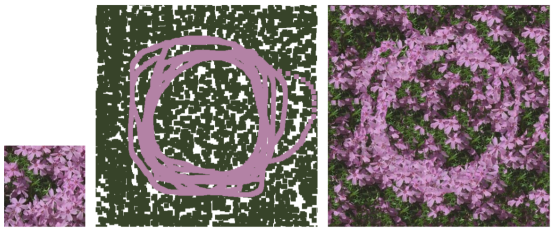


Fig. 1: An example of an existing texture synthesis algorithm

The main library that this program will derive from is called Keras. Keras is a high-level library API which is built on TensorFlow and Theano, providing the tools to swiftly build neural networks whilst hiding the mathematical aspects of tensors, optimization methods, etc. [9]. The grounds on using Keras is to access the Visual Geometry Group (VGG16 and VGG19) convolutional neural network model architectures along with its pre-trained weights on ImageNet. VGG16 and VGG19 [10] are pre-trained and can be used to perform image classification, transfer learning, and feature extraction. In this case the visual geometry group models are given the job of extracting ideal features (edges, interest points, contour, etc.) that can reflect the fundamental content of the images as complete as possible [11]. The number in the model name (16 & 19) refers to the number of weight layers that the model has. Figure 2 illustrates the architecture of VGG16 where the input layer takes an image of size (224 x 224 x 3) and outputs a softmax prediction on 1000 classes. The feature extraction portion of the architecture is highlighted in red (max pooling layer) with the size of (7 x 7 x 512) while the rest of the network is the classification part of the model. That being said, the pooling layers of the VGG16 and VGG19 will have a slight difference in that VGG19 will have one extra layer being used as the feature space. Feature spaces are used to obtain texture information by capturing the details about the style of an image. It is built on a collection of image filters in each layer which extracts the texture information of the input image [12].

The Figure 3 shows a visualization of the image representations in a Convolutional Neural Network. In order to get the essences of the style and the content of the given input images, the input image is reconstructed in different layers. For the content image reconstruction the layers conv1_2, conv2_2, conv3_2, conv4_2, and conv5_2 are visited. The images reconstructed from these layers will then be compared with the original input image and a content loss will be calculated. The style reconstruction and loss undergoes a similar process except that each style reconstruction uses an extra layer than the previous stage. The total loss of content and style is then fed as input into the optimization function (limited-memory

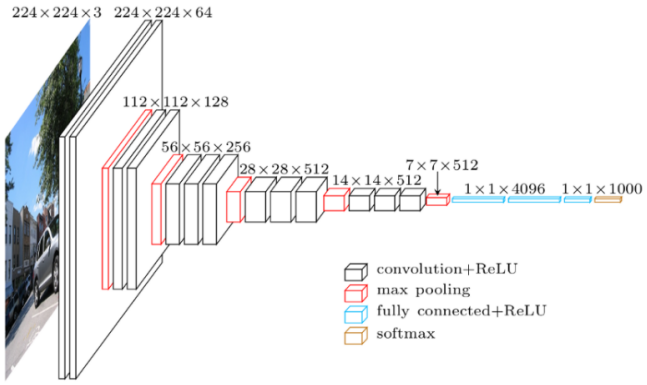


Fig. 2: VGG16 Architecture

BFGS algorithm) and is minimized whilst constantly being iterated through the set number of iterations to produce the result image.

The process of creating artistic style transfer is following the steps below:

1) Install and import the necessary packages and libraries (numpy, keras, pillow, etc.).
2) Load the two images (base and style) and make sure to have working directories for an output.
3) Process image using preprocess_image function to convert it into a tensor.
4) Input the tensor into the pretrained model of choice (VGG16 or VGG19) obtaining a model summary.
5) Calculate the content and style losses.
6) Obtain the weighted sum of content and style loss. Predetermine the convolutional layers for content and style feature extraction.
7) Calculate the gradient of the loss function with respect to the generated image
8) Process the array to get the RGB values and turn the array into an image to be saved.

A. Content Loss

Content Loss needs to be generated to ensure that the generated output image x retains some of the "global" characteristics of the content image, p [6]. For instance if p contains an image of a cat with a plant in the background, we need the output image to still be able to illustrate and distinguish those objects after the style transfer has been applied. The cat's facial features (ears, eyes, nose, etc.) must also be recognizable. Hence, a content loss function must be defined, and in this case, it is defined as the mean squared error between the feature representations of p and x at any given layer l as seen in Equation 1. [6]. A squared function is used instead of absolute values because it is a smooth function which is more tolerant of small errors [13]. From the formula in Equation 1, F and P are matrices with N rows and M columns, where N is the number of filters in the given layer l and M is the number of spatial elements in the feature map of the same layer. F and P also contain the feature representation of generated image (x) and the content image (p) respectively.

Content Loss function

$$L_c\left(p,x,l\right)=\frac{1}{2}\sum_{i,j}\left(F_{ij}^{l}-P_{ij}^{l}\right)^2 \tag{1}$$

B. Style Loss

In order to extract the style information from the VGG network, all the layers of the CNN are used. Additionally, the amount of correlation existing between the feature maps in a given layer can be defined as a "style" or style information. For the loss of a style, it is defined as the difference of correlation between the feature maps computed by the generated image and the original style image. The main idea is to calculate a style matrix (known as Gram matrix) for both the generated image and the style image, where the style loss is the root mean squared difference between the two style matrices. The definition of a gram matrix or style matrix can be seen in Equation 2. The gram matrix is a square matrix, meaning that it has the same number of rows as it has columns, and it contains the dot products of each vectorized filter in a given layer l [6]. As mentioned previously, F here is a matrix with a number of rows equal to N and a number of columns equal to M for the layer l.

Gram matrix

$$G^1=F^1\left(F^1\right)^T \tag{2}$$

Style loss for a given layer is shown in Equation 3, where A is the Gram matrix for the style image a and the Gram matrix for the generated image x being G.

Style loss function for a given layer l

$$E_1=\frac{1}{4N_l^2M_l^2}\sum_{i,j}\left(G_{ij}^{l}-A_{ij}^{l}\right)^2 \tag{3}$$

Additionally, receptive fields grow larger and larger as you ascend in the convolutional networks layers. To put it quite frankly, receptive fields can be referred to as the region in the input space that a particular CNN's feature is looking at. Intuitively, a receptive field's size grows bigger as you stack more layers, or by making the network deeper [14]. Due to the increase in the receptive field's size, the larger-scale characteristics of the input image are preserved. Which leads to the need for multiple layers to be selected for "style" to contain both local and global stylistic qualities. In order to create an effortless mesh between these different layers, a weight w is assigned to each layer and a total style loss function is defined as shown in Equation 4.

Total Style Loss Function

$$L_s=\left(a,x,l\right)=\sum_{l}^{L}w_lE_l \tag{4}$$

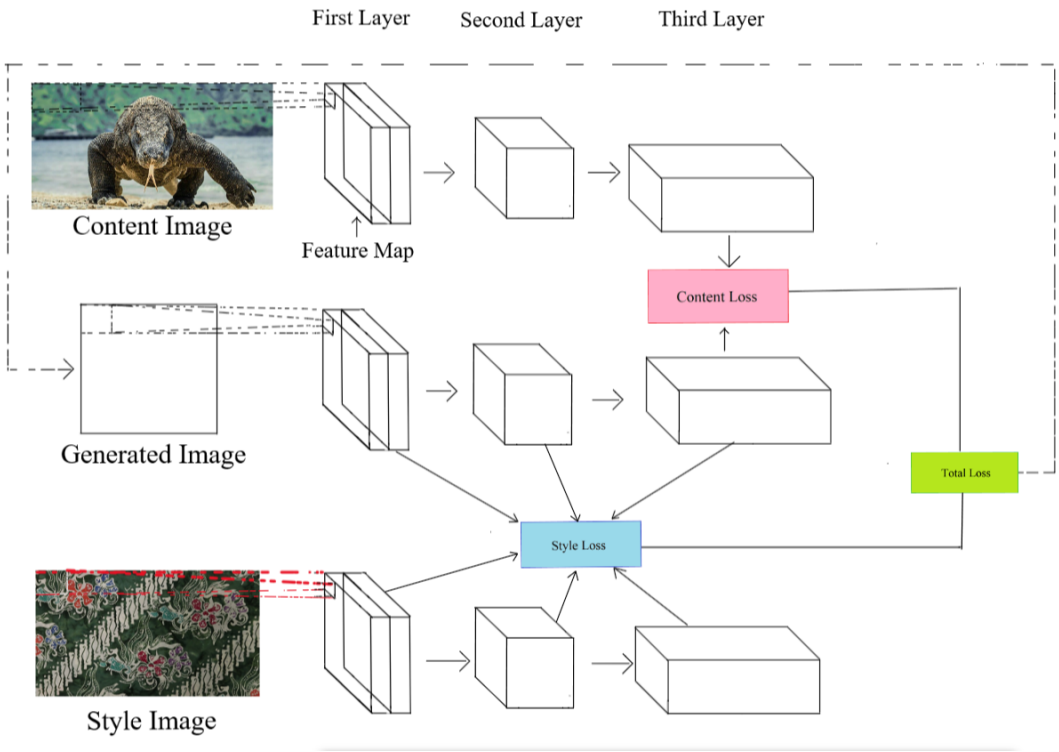


Fig. 3: Content and Style Reconstruction and Loss generation

For the final part of the loss functions, we need to combine them both together to allow us to fine tune the relative influence of the content image and the style image on the result image. Moreover, coefficients to the content and style loss functions need to be applied in order to set how much of each image do you want to "apply" to the generated image. Since we want to transfer the style of the style image onto the content image, it would only make sense that the coefficient before the style image is significantly greater than the coefficient on the content image. Gatys et al. [8] have experimented with the coefficient's ratios and found that an alpha value of 1 and a beta value of 10000 giving a ratio of 0.0001 produces a fairly good looking image. Figure 4 illustrates their experiments on tweaking the total loss function formula.

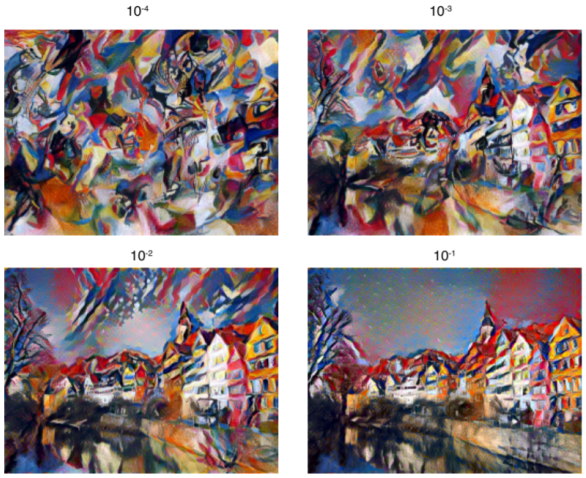


Fig. 4: Gatys et al. [8] experimenting on loss function's coefficients

Total loss function formula

$$L(p, a, x, l) = \alpha L_c(p, x, l) + \beta L_s(a, x, l) \tag{5}$$

The end of the program's cycle feeds both of these loss functions into a scipy optimization function and gradient function respectively and a local search optimization algorithm called BFGS is applied to total loss function over 600 iterations. Explaining in greater detail, optimization is a process which finds values for input parameters that maximize or minimize an objective function [15]. In these optimization algorithms, they can be classified as First-Order or Second-Order depending on which order of derivative that the algorithm uses to find the optima (minimum or maximum value) of a function. In the case of the Broyden, Fletcher, Goldfarb and Shanno (BFGS) algorithm, it makes use of the second-order derivative of an objective function. The algorithm tries to identify the variables that remain fixed and variables that are free using a gradient method, and then uses the L-BFGS method to modify the free variables to optimize the function.

This program is then tested against two models Visual Geometry Group (VGG) 16 and 19. Since both these models are based on the same architecture, classifying its detailed

differences must come from a deeper understanding of convolutional neural networks. Convolutional Layers are the primary building blocks in neural networks where each layer has a certain amount of filters. A convolution is the application of a filter to an input which does not have to be an image. This reapplication of the same filter on the input starts to indicate the locations and strengths of a significant feature in an input in the form of a feature map [16]. In the case of images, these will accent the lines, curves, edges, contours and other features of the image. These filters in VGG are of 3×3 dimensions which means that they could be thought of as a square matrix with 9 elements. For example, a hand crafted 3×3 element filter for detecting vertical lines might look something like the matrix in Figure 5. These filters are then applied repeatedly, overlapping each other on the input in order to create the feature map as seen in Figure 6.

```
0.0, 1.0, 0.0
0.0, 1.0, 0.0
0.0, 1.0, 0.0
```

Fig. 5: Filter matrix for vertical lines accenting

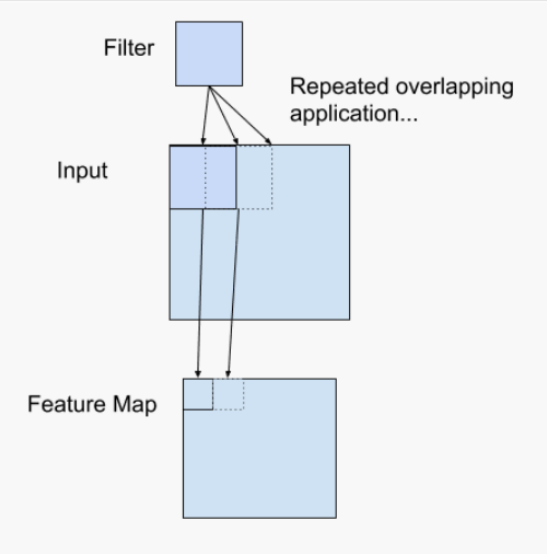


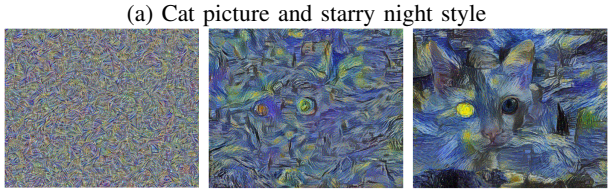
Fig. 6: Creating a Feature map

With a stronger grasp of how convolutional neural networks and their layers function, we can proceed towards the difference in the layers between VGG16 and VGG19 as seen in Table I. VGG16 has 41 layers in total in its network architecture and VGG19 has a total of 47 layers [17]. From the row indicating the different filter sizes of both the models we can clearly see that VGG19 has more filters. All of these "upgraded" values of the convolutional layers point to one purpose, to increase the accuracy of feature representation of an image. This means that the features of an image will be more visually identifiable and depicted by the model of VGG19 as compared to the VGG16.

TABLE I: Table Comparison of VGG16 and VGG19 Layers

Layer	**VGG16**	**VGG19**
Size of Layer	41	47
Image Input Size	224x224 pixel	224x224 pixel
Convolutional Layer	13	16
Filter Size	64 & 128	64,128,256, & 512
ReLU	5	18
Max Pooling	5	5
FCL	3	3
Drop Out	0.5	0.5
Softmax	1	1

(a) Cat picture and starry night style



(b) Results of iterations from 10 to 600

Fig. 7: Style transfer with VGG16 at different iterations

III. Results and Discussion

The images from Figure 7b are the results for the VGG16 when run on 10 iterations, 100 iterations, 600 iterations, and 800 iterations of Figure 7a. The variation of loss value is produced using the Keras program on multiple picture models, resulting in various outputs. The higher the loss value, the higher the clarity and quality of the resultant pictures. With every iteration of the L-BFGS algorithm, the loss function is minimized and it gives a better output image which will better illustrate the content and style images meshed together. However, there is a time constraint with running all these iterations as the number of layers affect the computation speed. Even though a greater amount of layers are desired to extract the style content to its highest capability, the computation time observed is extraneous.

Further experimenting on the relative weighting of matching content and style of the respective source images shows the different effects on the feature visibility and the style compromise. Different values of the ratio were used as seen from Figure 8, and it can be assumed that a greater value on the style seems to produce a slight variation of the style image whilst a lower value produces less of a painting and more of a picture of the content image with a filter applied on to it.

One extraneous constraint when trying to generate output images and make observations is processing time. As shown in Table III, to perform what is considered to be the average threshold producing a convincing enough result image over

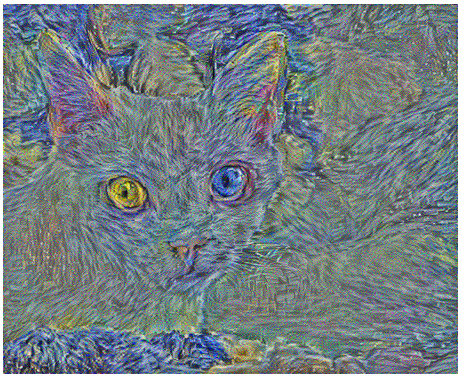


Fig. 8: Image of starry night cat with a beta of 1000

TABLE II: Hardware Specifications

Machine	CPU	GPU	RAM
Michael	Intel i7-8550u	Intel(R) UHD Graphics 620	8GB
Calvin	Intel i3-9100F	NVIDIA GeForce GT 1030	8GB
Jeconiah	Intel i5-10400	NVIDIA GeForce GTX 1650	16GB

TABLE III: Time Taken For Each Process

Iterations	Hyper parameter values	Machine	Time Taken
10	VGG19, Beta = 10000	Michael	128.4s
100	VGG19, Beta = 10000	Michael	3000.5s
600	VGG19, Beta = 10000	Michael	43907.5s
600	VGG19, Beta = 10000	Jeconiah	43828.4s
600	VGG19, Beta = 1000	Calvin	37668.3s

Note: separate processes were done on different computers with different hardware specifications

600 iterations takes a large period of time. Not to mention, during the period of running the process, the CPU utilization goes up to 90 percent, making the computer experience a huge jolt of lag. Intuitively, multitasking on the device where the processes are running will divide the computer's resources between different applications and slow down the iteration as well. With the current specifications from Table II, time management and schedule planning has to come in and allocate the right periods to run these iterations as well.

Now that a better understanding of our desired hyperparameters, model, number of iterations is achieved we can start applying these conditions towards transferring a batik motif onto a content image. From the figure 9b below we can see that the motif is successfully applied on Indonesia's national animal, the Komodo dragon.

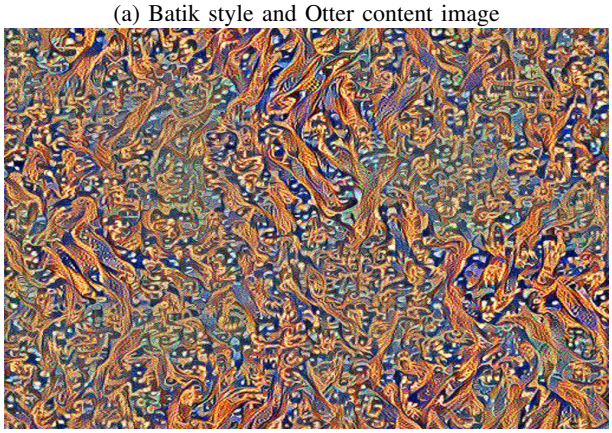
Obtaining the style transfer results on the content and style images from figure 9a, the left picture of figure 9b uses the hyperparameter beta of 1000 and an alpha value of 1. Whereas, the picture on the right of figure 9b has a higher beta value of 10000 with the same alpha value, implying a stronger influence by the style image on the result. As we can see the features on the left has a grittier feature complexion as it retains more information of the content image and less of the style image. Since batik is largely centered on pattern engraving with the occasional swirl of variation, it would not make sense to use the image with a higher similarity to the Komodo dragon as a motif for batik textiles.

(a) Modern batik style and Komodo dragon

(b) Results of iteration with Komodo
The left picture uses the hyperparameter beta of 1000 and an alpha value of 1. Whereas the picture on the right has a beta value $>$ 10000 with the same alpha value

Fig. 9: Experiment on batik and Komodo image content

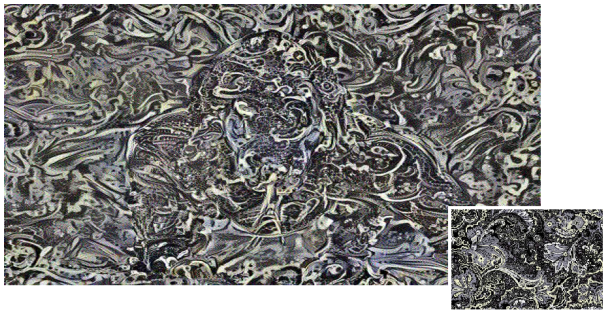
(a) Batik style and Otter content image



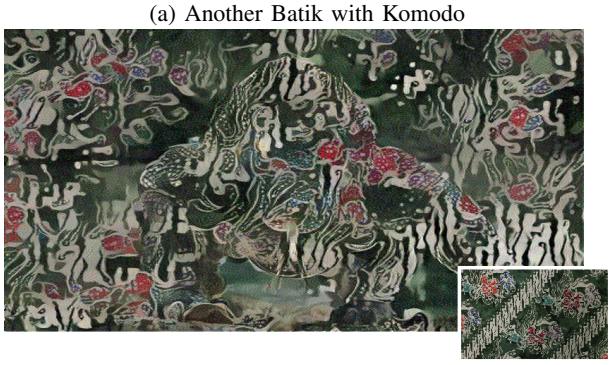
(b) Batik transferred on Otter content image
with beta $>$ 10000 and 600 iterations

Fig. 10: Experiment on a stronger batik pattern with Otter

However, there was an unexpected outcome from running the model on the images in Figure 10a. As we can see, we used an otter for the base content image and a Batik pattern as the style image. The output image however after being run on 600 iterations (Figure 10b), do not portray the features of the content image (otter image). This might be the cause of many factors but since the Komodo style transfer works, it might be safe to assume that the second style image is non ideal for style transfer because of its more uniform pattern. Although the outcome, does not show the content image features, it still results in a new batik motif which is still useful.



(a) Another Batik with Komodo



(b) Another Batik with Komodo

Fig. 11: Result of more experiments with beta $>$ 10000 and 600 iterations. The inset picture shows the original batik pattern.

These are further experiment results done on different batik styles on the same Komodo dragon image. As seen on Figure 11, the pattern and the base features of the content image are blended nicely to produce a new motif. The Komodo's more detailed features like it's scales, eyes, claws cannot be clearly made out which is desirable in batik as the main focus is to come up with a pattern and not an image.

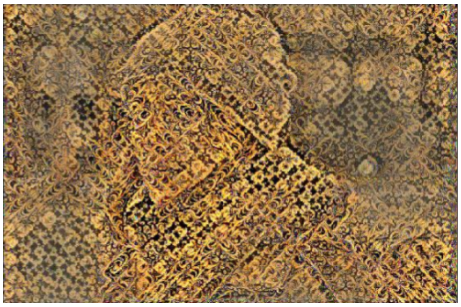


Fig. 12: Image of a generated batik from previous research.

The Figure 12 shows the generated image from a research conducted by Irawan et al. The produced image was produced through VGG16 whilst implementing an extra color cost function to minimize and account for. In hopes of obtaining a more "solid" color, the batik pattern was passed through a K-means algorithm to assemble the clusters of color that can be found in the batik style image. Additionally, the aforementioned color cost function will calculate the distance between the produced image and the style image which is directly proportional to the total cost value. Putting the result image of our model configurations and the configurations of the research don by Irawan up against each other, we can recognize that there are patches of dulled coloring around the images where as the komodo result images have the style image constantly applied throughout the output. The batik from 12 is iterated for 10000 times which might make the image get more details from the content image resulting in the blurred patches.

IV. Conclusion

In this paper, we have demonstrated how to make use of deep Convolutional Neural Networks to perform feature extraction and transfer image style between two images. We have decided to go with the VGG19 architecture instead of VGG16 because the added layers in the architecture was able to pronounce the content image features more, which creates a better balance of the content and the style images. Experiments with a cat image and the starry night style were performed in order to spot the differences and conclude the usage of VGG19. Although some of the produced images seem palatable and passable as art, there are still many technical limitations to the algorithm. The most limiting factor in this project would have to be the resolution of the result images. As for now, the quality of the image produced, are severely constrained since the number of units in the CNN and the dimensions of the input image are proportional to the computational time. The current computation time for producing an image with 600 iterations equates to an approximate of 44000 seconds translating into 12 hours without stopping. Moreover, the images contain some swirls of random colors which can be considered noise and irrelevant to the content and style of the images.

One of the practical applications of style transfer is the ability to create new designs for batik. Batik is a part of Indonesian culture, it is a technique used to create clothing with interesting motifs. By applying style transfer to designing batik, it will be easier to create new motifs. Batik has developed from many different styles throughout the ages, with style transfer, a new batik style might emerge. Creating new motifs and styles for batik will greatly expand the culture of Indonesia.

An issue that might be overlooked by some is the effect that running these overnight simulations have on our devices, processor. The processor usage is constantly pushed to the limit resulting in an significant build up thermal energy. The processor is measured to go up to 86 degrees Celsius multiple times with an average high of 82 to 83 degrees Celsius. Intel has a standard processor temperature range of 100-105 degrees Celsius to act as the threshold before the CPU's lifespan is affected and a safeguard is activated where the power is reduced. Noticing that our processors come close to the threshold in an air-conditioned room, further researches should be careful to not running the program in a warmer climate as it might deteriorate the processors lifespan. We also recommend that a better graphics card be used to help lighten the load of image processing.

Currently, our program is limited to projecting the style of a single image onto the content image. However, since modern art is constantly shifting, it might be interesting to have the flexibility of different textures and colors as the style image that can be applied to different sections of the style image to create a more robust and diverse result image. Although the needed modifications to support multiple style images are not worked out yet, a rough idea of the implementation might include blending the weights of the different style images into a single array with equal weighting and normalizing those weight to sum to 1.

When all is said and done, the entire process of employing a computer in order to "re-create" art has been one of the most fascinating and emotionally rewarding experiences to be gained. Computational limitations are still a hurdle to overcome as well as critics under appreciating the generated products to be non-photorealistic. However, there is much more to be gained than loss from performing style transfer, from enriching one's culture, to pure entertainment and admiration, neural networks will continue to develop and increase in quality.

REFERENCES

[1] H. Cohen, "The further exploits of aaron, painter," *Stanford Humanities Review*, vol. 4, no. 2, pp. 141–158, 1995.

[2] Z. Epstein, S. Levine, D. G. Rand, and I. Rahwan, "Who gets credit for ai-generated art?" *Iscience*, vol. 23, no. 9, p. 101515, 2020.

[3] F. Liu, "Cs 229 project final report : Neural style transfer," 2019.

[4] L. P. Lusianti and F. Rani, "Model diplomasi indonesia terhadap unesco dalam mematenkan batik sebagai warisan budaya indonesia tahun 2009," *Transnasional*, vol. 3, no. 02, 2012.

[5] L. W. Aji, "Ta: Pembuatan motif batik menggunakan deep style transfer," Ph.D. dissertation, Institut Teknologi Nasional, 2021.

[6] Y. A. Irawan and A. Widjaja, "Pembangkitan pola batik dengan menggunakan neural transfer style dengan penggunaan cost warna," *JuTISI (Jurnal Teknik Informatika dan Sistem Informasi)*, vol. 6, no. 2, 2020.

[7] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 symposium on Interactive 3D graphics*, 2001, pp. 217–226.

[8] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image style transfer using convolutional neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2414–2423.

[9] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[11] D. ping Tian *et al.*, "A review on image feature extraction and representation techniques," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 4, pp. 385–396, 2013.

[12] V. Gupta, R. Sadana, and S. Moudgil, "Image style transfer using convolutional neural networks based on transfer learning," *International Journal of Computational Systems Engineering*, vol. 5, no. 1, pp. 53–60, 2019.

[13] H.-H. Zhao, P. L. Rosin, Y.-K. Lai, M.-G. Lin, and Q.-Y. Liu, "Image neural style transfer with global and local optimization fusion," *IEEE Access*, vol. 7, pp. 85 573–85 580, 2019.

[14] W. Luo, Y. Li, R. Urtasun, and R. Zemel, "Understanding the effective receptive field in deep convolutional neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 4905–4913.

[15] I. Fister Jr, X.-S. Yang, I. Fister, J. Brest, and D. Fister, "A brief review of nature-inspired algorithms for optimization," *arXiv preprint arXiv:1307.4186*, 2013.

[16] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017, pp. 1–6.

[17] W. Setiawan and F. Damayanti, "Layers modification of convolutional neural network for pneumonia detection," in *Journal of Physics: Conference Series*, vol. 1477, no. 5. IOP Publishing, 2020, p. 052055.