Synthetic Batik Pattern Generator using Wasserstein Generative Adversarial Network with Gradient Penalty

Kus Andriadi
Computer Science Department, Binus Graduate Program, Doctor of Computer Science Bina Nusantara University
Jakarta, Indonesia
kus.andriadi@binus.ac.id

Yaya Heryadi
Computer Science Department, Binus Graduate Program, Doctor of Computer Science Bina Nusantara University
Jakarta, Indonesia
yayaheryadi@binus.edu

Lukas
Cognitive Engineering Research Group (CERG)
Universitas Katolik Indonesia Atma Jaya
Jakarta 12930, Indonesia
lukas@atmajaya.ac.id

Wayan Suparta
Department of Electrical Engineering, Faculty of Industrial Technology, Institut Teknologi Nasional Yogyakarta, Yogyakarta
55281, Indonesia
wayan@itny.ac.id

Ilvico Sonata
Computer Science Department, Binus Graduate Program, Doctor of Computer Science Bina Nusantara University
Jakarta, Indonesia
ilvico@binus.ac.id

2023 6th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI) | 979-8-3503-5834-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISRITI60336.2023.10467561

***Abstract*—Batik is an Indonesian world cultural heritage. Batik consists of many kinds of patterns depending on where the batik comes from, Batik-making techniques continue to develop along with technology development. Among the batik making techniques that are widely used are hand-written, stamping, and printing. Batik motifs have been widely used as research material, especially in the field of artificial intelligence. The diverse appearance of batik motifs has attracted many researchers to carry out research on making synthetic batik patterns, one of which uses a Generative Adversarial Network. This paper presents a synthetic batik pattern model based on the Wasserstein Generative Adversarial Network with Gradient Penalty. This model has been proven to create new synthetic batik patterns quite well and almost identical with images provided in the dataset, with the notes if the dataset provided is large.**

Keywords—Batik, Pattern, Generator, GAN, WGANGP

I. INTRODUCTION

Batik is an Indonesian world cultural heritage. Every year, Indonesia celebrates National Batik Day on October 2 as a form of appreciation and pride for this cultural heritage. Batik contains many kinds of patterns depending on where the batik comes from. In the past, batik also was only created for the palace (keraton) environment and used for the king, family and the followers [1]. Now, Batik has developed rapidly and is produced by various regions in Indonesia, each with it is own characteristics. Apart from that, batik has also achieved international recognition and become part of the global fashion world. [2].

In recent years, there has been a topic that has attracted quite a lot of researchers attention, namely research to produce quite varied abstract patterns. There is art font image generation using a Conditional Generative Adversarial Network (CGAN) [3], the idea behind that research is to generate an abstract pattern and put the pattern in a characters image, in this case, the object is a character. There is other research also that generate abstract pattern and puts the pattern in hand images use Deep Convolutional Generative Adversarial Network (DCGAN) [4]. Both have different objects to implement the pattern (character and hand images), but the aim is quite the same: put the abstract art pattern in the object.

Aside from putting the abstract pattern in the object, research related to Generative Adversarial Networks that change the standard image to art looks image is also growing rapidly. For example, there is research to change image to tile image [5] or illustration image [6]. For the tile image research, the object in the image is not very visible because of the abstract pattern of tiles produced, but this is what makes the image look like a beautiful work of art. Different things are presented for illustration image research; the result does not change the main object, so we can still clearly see the main object even with the illustration style. The aim of both ideas is quite the same: to build a more convenient art-style generation system for users. The model generated from those research can be implemented in various kinds of image editing applications.

The research conducted on batik pattern topic is quite numerous and varied, such as a research that matches batik clothing objects with people in the picture using a Conditional Generative Adversarial Network (CGAN) [7], the architecture is to perform semantic segmentation and batik texture synthesis, including encoder-decoder generator implementing U-Net and a convolutional discriminator. Some batik pattern generation techniques also appeared, like the generation of batik fractal patterns using Julia Set [8] and the generation of synthetic batik pattern using Deep Convolutional Generative Adversarial Network (DCGAN) [9]. The batik pattern generated using DCGAN is quite impressive, almost imitating the parang batik pattern, even though not entirely similar.

As research related to abstract art generation arises, research related to batik pattern generation also arises with different kinds of techniques, especially using Conditional Generative Adversarial Network and Deep Convolutional Generative Adversarial Network. With that fact, this study aims to enrich kind of techniques and expand knowledge about another method to generate batik patterns. In this case, we will use Wasserstein Generative Adversarial Network with Gradient Penalty.

II. Research Method

A. Dataset

This study used two datasets. In the first experiment, we were using dataset Batik 300 [10, p. 300]. Batik 300 is a dataset collected by capturing 50 types of Batik cloth. Each cloth consists of six random images and then resized to 128x128 pixel size in JPEG format. The total picture in this dataset is 300 images in 50 classes. Examples of Batik 300 dataset can be seen in Figure 1.

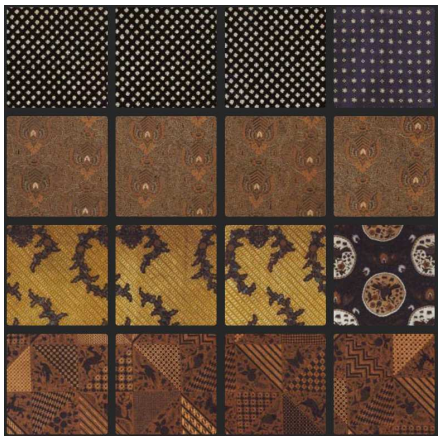

Fig. 1. Batik 300 Dataset

For the last experiments, we used dataset Batik Nitik 960 [11]. Batik Nitik 960 was captured from a piece of fabric consisting of 60 Nitik categories with a total of 960 images, and each class has 16 photos. Examples of Batik Nitik 960 can be seen in Figure 2.

Fig. 2. Batik Nitik 960 Dataset

B. Computation Environment

The model was trained and tested on a Paperspace platform (https://www.paperspace.com/), used 48GB A6000 GPU, 45GB Memory and 8 vCPUs.

C. Preprocessing Data

Before the training model starts, this step must be initiated to check whether the dataset aligns with the source code. We created a source code to train for images with size 128x128 pixels and RGB color. Thus, we need to ensure that the images inside the dataset are the same size as our allowable input.

The Batik300 dataset already has a 128x128 pixels image, so we do not need to do anything with this dataset. On the other hand, we need to resize all images from the Batik Nitik 960 dataset because the actual size is 512x512 pixels.

D. Implementation

Generative Adversarial Network Framework consists of two models, Generator (G) and Discriminator (D) like shown in Figure 3. Both model is a common neural network and trained each iteration simultaneously [12].

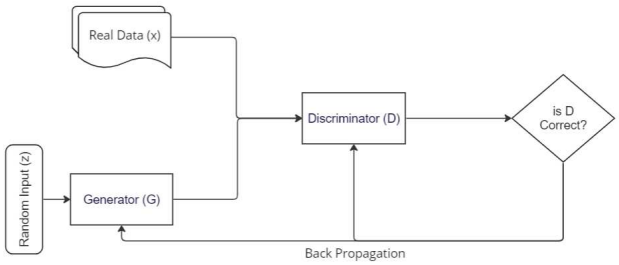

Fig. 3. Basic Generative Adversarial Network Framework [12]

The G model aims to generate data that is almost identical with real data. The D receives either real or fake data and must distinguish between the two. The trained will be done until the D could not differentiate between real data and fake data. So, this is like a game between two neural networks. Formally, the game between G and D is the minimax objective like in Equation 1 below.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

Equation 1. Minimax objective function

Training model with GAN is hard, there is a chance your model may never converge, and mode collapses are quite common. In this paper, we did not use basic GAN, we used an improved version from Wasserstein Generative Adversarial Network (WGAN) [13] called Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [14]. The objective function of WGAN is based on the Wasserstein distance (Earth Mover's Distance) that counts the distribution of real data and the generated data, with WGAN, generator still can learn even if discriminator is doing a good job. the WGAN objective function defined in Equation 2.

$$\min_G \max_D [\mathbb{E}_{x\sim P_{\text{data}}}[D(x)] - \mathbb{E}_{z\sim P_z}[D(G(z))]]$$

Equation 2. WGAN objective function

For training stability, WGAN introduces weight clipping on the discriminator to enforce a Lipschitz continuity constraint, but there is probability the network not performed well because of severe clipping of weights.

Training WGAN may needs a longer time compared to other basic GAN models, the convergence will come but may be slower due to weight clipping calculations. The algorithm of WGAN can be seen in Figure 4 below.

Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.
Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.
while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

Fig. 4. Wasserstein GAN Algorithm [13]

WGAN-GP [14] tried to solve weight clipping issues that introduces by WGAN. The paper introduces a gradient penalty to the Wasserstein distance objective function (Equation 3).

$$\min_G \max_D \left[\mathbb{E}_{x\sim P_{\text{data}}}[D(x)] - \mathbb{E}_{z\sim P_z}[D(G(z))] + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right]\right.$$

Equation 3. WGAN-GP objective function

Gradient Penalty is the main difference between WGAN and WGAN-GP. It is used to enforce the Lipschitz constraint. The concept is penalized the norm of the discriminator gradient depends on the input (real or fake data). This penalty encourages the discriminator model to have gradients close to 1.This penalty helps to achive Lipschitz continuity without the needs of aggressive weight clipping.

With WGAN's use, the discriminator model also differs from Basic GAN. The WGAN paper [13] mentioned that the discriminator model should not use a Batch Normalization layer. So, we remove that layer from our model configuration.

E. Optimizer

This paper conducts several experiments using RMSProp and Adam; both are good, but RMSProp converges faster than Adam Optimizer. The image result after training ends when using RMSProp also looks better Adam's.

So, for the final experiment, we used RMSProp optimizer with a learning rate 0.0005 for both the Generator and Discriminator.

III. Experiments and Results

This research conducted two experiments for two datasets. The first experiment used Batik300 dataset, second experiment used Batik Nitik 960 Dataset.

A. Experiment with Batik300 Dataset

We did several experiments using The Batik300 datasets and found that the optimum training is when using epochs 2000 and batch size 8. In Figure 5, we can see at epoch 100 that the Discriminator (D) and Generator (G) losses still do not converge. As shown in Figure 6, The loss starts converging near 2000 epoch.

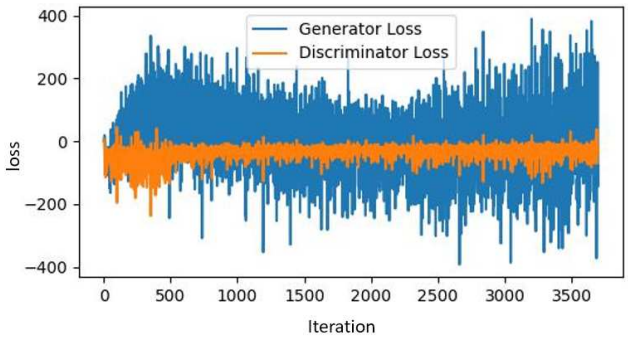

Fig. 5. D Loss and G Loss at epoch 100

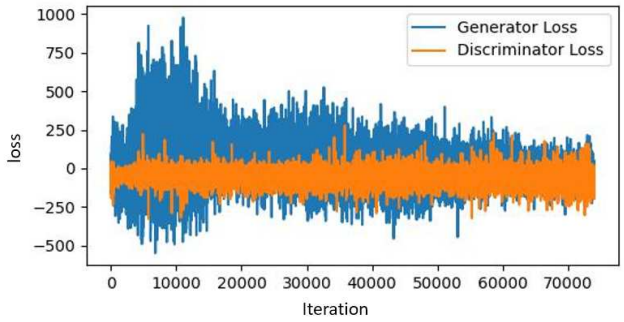

Fig. 6. D Loss and G Loss at epoch 2000

After the training ends, the generated batik pattern from The Batik300 dataset can be seen in Figure 7.

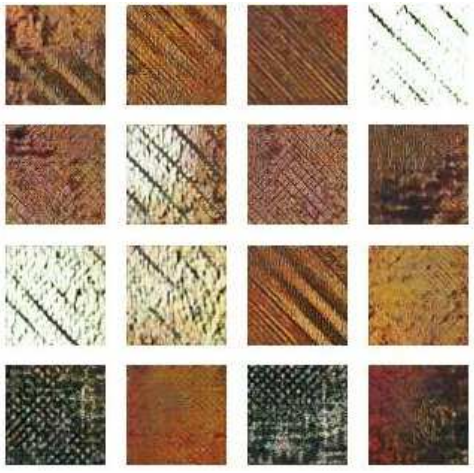

Fig. 7. Batik pattern generated from dataset Batik 300

B. Experiment with Batik Nitik Dataset

We did several experiments using The Batik Nitik dataset, and we found that the optimum training used epochs 850 and batch size 32. In Figure 8, we can see at epoch 50 that the Discriminator (D) and Generator (G) losses still far from converge. As shown in Figure 9, The loss starts converging near 850 epoch. Finally, the training result using Batik Nitik dataset can be seen in Figure 10, we can see that the pattern result generated with this dataset is almost identical with the training datasets.

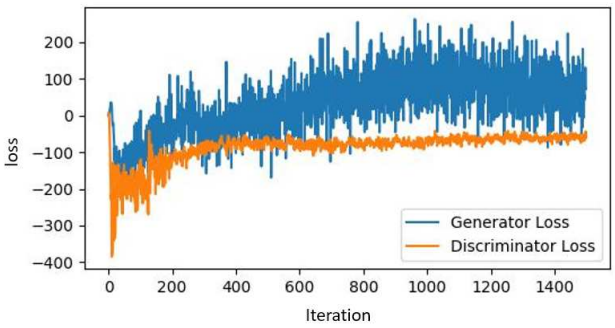

Fig. 8. D Loss and G Loss at epoch 50

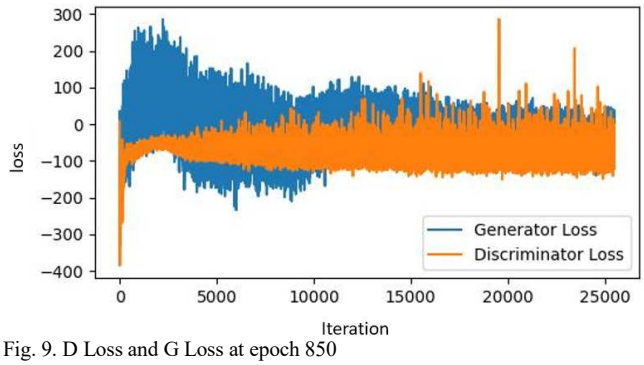

Fig. 9. D Loss and G Loss at epoch 850


Fig. 10. Batik pattern generated from dataset Batik Nitik 960

IV. Discussion

A synthetic batik pattern has been successfully generated using WGAN-GP. The results are promising, especially for experiments using Batik Nitik 960. In this case, the two datasets have quite fundamental differences. Batik Nitik 960 has many images, so the model can learn from many images during training. Apart from that, the background color of the batik is also the same, so there is no mixing of colors, which causes the resulting image to be less good. Lastly, Batik Nitik 960 is a collection of batik images with the same type of batik.

In contrast to the Batik Nitik 960 dataset, the Batik 300 dataset has background colors and patterns that are very different from each other, also the total image for training is not so much. This causes the generator model does not have enough resources to learned and generate data to beat the Discriminator model.

V. Conclusion and Future Research

This paper concludes that WGAN-GP can generate batik patterns using both datasets, but with notes that the generated image will look better when the amount of datasets is enough.

As a suggestion for further research. First, we are encouraged to tuning the hyperparameter when training using Batik300. Second, Batik300 dataset needs to be enlarged three or four times. It would be good if the pattern is classified to several types of batik.

References

[1] "Batik Indonesia." [Online]. Available: https://warisanbudaya.kemdikbud.go.id/?newdetail&detailTetap=71

[2] Romanti, "Menilik Sejarah Batik, Salah Satu Duta Budaya Indonesia." [Online]. Available: https://itjen.kemdikbud.go.id/web/menilik-sejarah-batik-salah-satu-duta-budaya-indonesia/

[3] Y. Yuan, Y. Ito, and K. Nakano, "Art Font Image Generation with Conditional Generative Adversarial Networks," in *2020 Eighth International Symposium on Computing and Networking Workshops (CANDARW)*, Naha, Japan: IEEE, Nov. 2020, pp. 151–156. doi: 10.1109/CANDARW51189.2020.00039.

[4] S. S. Nasrin and R. I. Rasel, "HennaGAN: Henna Art Design Generation using Deep Convolutional Generative Adversarial Network (DCGAN)," in *2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE)*, Bhubaneswar, India: IEEE, Dec. 2020, pp. 196–199. doi: 10.1109/WIECON-ECE52138.2020.9398005.

[5] N. Matsumura, H. Tokura, Y. Kuroda, Y. Ito, and K. Nakano, "Tile Art Image Generation Using Conditional Generative Adversarial Networks," in *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*, Takayama: IEEE, Nov. 2018, pp. 209–215. doi: 10.1109/CANDARW.2018.00047.

[6] Z. Chen, L. Chen, Z. Zhao, and Y. Wang, "AI Illustrator: Art Illustration Generation Based on Generative Adversarial Network," in *2020 IEEE 5th International Conference on Image, Vision and Computing (ICIVC)*, Beijing, China: IEEE, Jul. 2020, pp. 155–159. doi: 10.1109/ICIVC50857.2020.9177494.

[7] T. Tirtawan, E. K. Susanto, P. C. S. W. Lukman Zaman, and Y. Kristian, "Batik Clothes Auto-Fashion using Conditional Generative Adversarial Network and U-Net," in *2021 3rd East Indonesia Conference on Computer and Information Technology (EIConCIT)*, Surabaya, Indonesia: IEEE, Apr. 2021, pp. 145–150. doi: 10.1109/EIConCIT50028.2021.9431867.

[8] R. R. Isnanto, A. Hidayatno, and A. A. Zahra, "Fractal Batik Motifs Generation Using Variations of Parameters in Julia Set Function," in *2020 8th International Conference on Information and Communication Technology (ICoICT)*, Yogyakarta, Indonesia: IEEE, Jun. 2020, pp. 1–6. doi: 10.1109/ICoICT49345.2020.9166282.

[9] A. E. Minarno, Moch. C. Mustaqim, Y. Azhar, W. A. Kusuma, and Y. Munarko, "Deep Convolutional Generative Adversarial Network Application in Batik Pattern Generator," in *2021 9th International Conference on Information and Communication Technology (ICoICT)*, Yogyakarta, Indonesia: IEEE, Aug. 2021, pp. 54–59. doi: 10.1109/ICoICT52021.2021.9527514.

[10] A. Minarno and N. Suciati, "Batik 300." Jan. 24, 2022. doi: 10.17632/vz7pzt2grf.1.

[11] A. E. Minarno, H. A. Nugroho, and I. Soesanti, "Batik Nitik 960." Jan. 23, 2023. doi: 10.17632/sgh484jxzy.3.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, and S. Ozair, "Generative Adversarial Nets," Jun. 2014.

[13] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein GAN," 2017, doi: 10.48550/ARXIV.1701.07875.

[14] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," 2017, doi: 10.48550/ARXIV.1704.00028.