BatikGAN: A Generative Adversarial Network for Batik Creation

Wei-Ta Chu
National Cheng Kung University
Taiwan
wtchu@gs.ncku.edu.tw

Lin-Yu Ko
National Chung Cheng University
Taiwan
linyukotw@gmail.com

ABSTRACT

We propose a texture synthesis method based on generative adversarial networks, focusing on a cultural emblem, called Batik, of southeastern Asian countries. We propose a two-stage training approach to construct the network, first generating patches and then combining patches to generate the entire Batik image. Regular repetition and synthesis artifact removal are jointly considered to guide model training. In the evaluation, we show that the proposed generator fuses two Batik styles, removes blocking artifacts, and generates harmonious Batik images. Qualitative and quantitative evaluations are provided to show promising performance from several perspectives.

CCS CONCEPTS

• **Information systems → Image search**; • **Computing methodologies →** *Image representations.*

KEYWORDS

texture synthesis, image generation, generative adversarial network

ACM Reference Format:
Wei-Ta Chu and Lin-Yu Ko. 2020. BatikGAN: A Generative Adversarial Network for Batik Creation. In *Proceedings of the 2020 Joint Workshop on Multimedia Artworks Analysis and Attractiveness Computing in Multimedia (MMArt-ACM '20), June 8, 2020, Dublin, Ireland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3379173.3393710

1 INTRODUCTION

Texture synthesis has been studied for years because of its wide applications. Interestingly, different cultures have their unique texture elements, making specific texture symbolize different cultures. Batik, for example, presents texture patterns uniquely in southeastern Asia like Indonesia and Malaysia. Because of its mysterious and fascinating lines and color schemes, it was recognized as the cultural heritage of Indonesia in 2009. Recently, Batik has become widely-spread texture patterns for clothes design and decoration.

In the past, the goal of image style transfer is to transfer the source image into that with the style of the target image. However, we notice that it is very difficult for users to imagine how the image looks like when two styles are mixed. On the other hand, current texture synthesis methods focus on expanding a given patch with the consideration of continuity and geometric structure. There is no good way to generate a texture image jointly considering two patches with different styles. In this work, we propose a method somehow in-between image style transfer and texture synthesis, which takes Batik image generation as the target domain. Particularly, we propose a generative adversarial network (GAN) to create Batik images based on two patches with different styles.

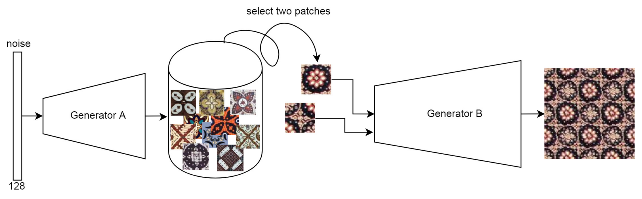

Figure 1: The proposed framework consists of two stages, i.e., patch generator and Batik image generator.

Fig. 1 illustrates the proposed two-stage framework. First, a patch generator is constructed to generate basic image patches. We adopt Wasserstein Generative Adversarial Networks with Gradient Penalty (WGAN-GP) [3] to build the patch generator. Second, a Batik image generator is constructed based on two generated patches selected by the user. The Batik image generator learns to place patches and aims to generate harmonious Batik images. The reason to divide the entire framework into two parts is that directly generating Batik images consisting of rich (high-variation task) and regularly located patches (low-variation task) is too challenging. Conceptually the two parts are somewhat contradictive. Dividing it into two separate parts would make each part focus on its task.

Our contributions are twofold:

- We explore regular-texture synthesis based on patches of two different texture styles. With two patches as input, the Batik image generator fuses styles of two patches, and generates a Batik image of a new style.
- This would be one of the first works focusing on generating images of a cultural emblem.

2 SYSTEM FRAMEWORK

2.1 Patch Generator

In this work, we adopt WGAN-GP [3] to implement the patch generator. The discriminator D learns to distinguish the generated fake image from the real image, and the generator G tries to fool the discriminator by generating good images. The WGAN-GP is constructed by finding the best settings via the following minimax game:

$$\min_G \max_D \mathcal{L}_{adv}. \tag{1}$$

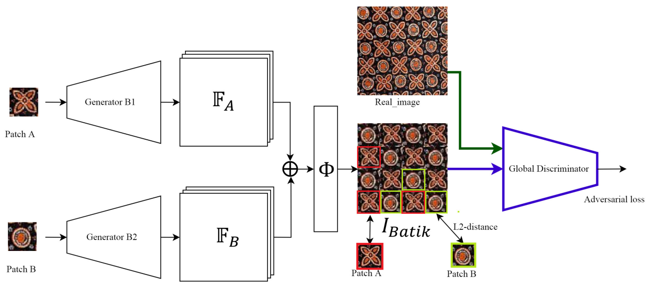

Figure 2: Illustration of the proposed BatikGAN.

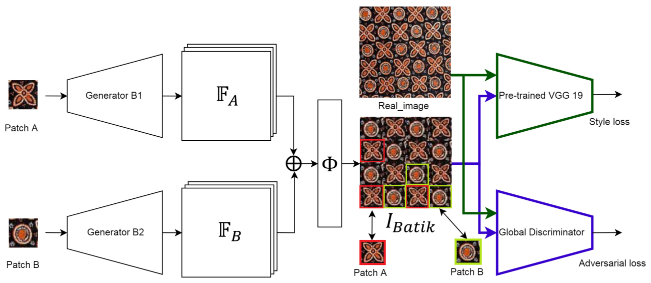

Figure 3: Illustration of the BatikGAN with style features (named as the BatikGAN_S model).

The loss function $\mathcal{L}_{adv}$ can be formulated as:

$$\mathcal{L}_{adv} = \mathbb{E}_{\tilde{\boldsymbol{x}} \backsim \mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x} \backsim \mathbb{P}_r}[D(\boldsymbol{x})]$$

$$+ \underbrace{\lambda \mathbb{E}_{\hat{\boldsymbol{x}} \backsim \mathbb{P}_{\hat{x}}}[(\| \nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}}) \|_2 - 1)^2]}_{gradient\ penalty} \tag{2}$$

where $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{\boldsymbol{x}} = G(\boldsymbol{z})$, $\boldsymbol{z} \backsim p(\boldsymbol{z})$. $\mathbb{P}_r$ is the data distribution. $\hat{x} \backsim \mathbb{P}_{\hat{x}}$ means sampling uniformly along straight lines between pairs of points sampled from a distribution. Detailed explanation please refer to [3].

By constructing the generator based on various Batik patches, we can make the generator capture characteristics of Batik patches and thus generate appropriate patches.

2.2 Batik Generator

To combine two Batik patches, we design a Batik generator that fuses styles of two patches and generates a new Batik image. A major challenge for this task is to make the generated Batik harmonious. We design a local feature loss to guide model training, making the generated image harmonious with fewer blocking artifacts.

2.2.1 BatikGAN. We first develop a baseline system called BatikGAN, as illustrated in Figure 2. Based on patch pairs $Patch_A$ and $Patch_B$ cut from Batik training images $\{\boldsymbol{x}\}$, the BatikGAN is constructed to include two generators G_{B1} and G_{B2}, and a discriminator D_{global}. Feeding $Patch_A$ and $Patch_B$ to G_{B1} and G_{B2}, respectively, feature maps $\mathbb{F}_A$ and $\mathbb{F}_B$ are generated by G_{B1} and G_{B2}. These two feature maps are stacked together as $\mathbb{F}_{AB}$. Then, the Batik image I_{Batik} is generated from $\mathbb{F}_{AB}$ through the fusion function Φ, i.e., $I_{Batik} = \Phi(\mathbb{F}_{AB})$. The fusion function is implemented by 3 convolutional layers.

The discriminator D_{global} aims to distinguish whether the generated image I_{Batik} is real or fake. Given the training data $\{\boldsymbol{x}, Patch_A, Patch_B\}$, the BatikGAN is constructed by finding the best settings via the following minimax game:

$$\min_{I_{Batik}} \max_{D_{global}} \mathcal{L}_{adv}^{(1)} + \mathcal{L}_{local\_l2}^{(1)}, \tag{3}$$

where the adversarial loss $\mathcal{L}_{adv}^{(1)}$ is given as

$$\begin{aligned} \mathcal{L}_{adv}^{(1)}(I_{Batik}, D_{global}) = \mathbb{E}_{\boldsymbol{x}}[\log D_{global}(\boldsymbol{x})] \\ + \mathbb{E}[\log(1 - D_{global}(I_{Batik}))]. \end{aligned} \tag{4}$$

For $\mathcal{L}_{local\_l2}^{(1)}$, we notice that two patches are usually interlaced in a Batik image. We thus design a scheme to enforce repetition. Let us equally divide a generated images into 16 blocks. The blocks from left to right, top to bottom, are denoted as $block_1$, $block_2$, $block_3$, ..., and $block_{16}$, respectively. The loss considering repetition and local information is designed as :

$$\mathcal{L}_{local\_l2}^{(1)} = \sum_{y \in Y} \sqrt{(Patch_A - Block_y)^2} + \sum_{z \in Z} \sqrt{(Patch_B - Block_z)^2}. \tag{5}$$

where $Y = \{1, 3, 6, 8, 9, 11, 14, 16\}$ is a set of indices indicating a part of the blocks, and $Z = \{2, 4, 5, 7, 10, 12, 13, 15\}$ is also a set of indices indicating another part of blocks. Notice that the indices in Y and Z are interlaced.

2.2.2 BatikGAN with Style Features. Although the BatikGAN enables fair generation results, color and style attributes of generation results are usually not satisfactory. Motivated by [1][8][11], we want to further consider style features extracted by the VGG-19 model pre-trained on the ImageNet dataset, as shown in Figure 3. The VGG-19 model consists of sixteen convolutional layers and three fully-connected layers. The convolutional layers are named as conv1_1, conv1_2, conv2_1, conv2_2, and so on. The conv3_1 layer, for example, is the 5th convolutional layer that just follows the second pooling layer. In the VGG-19 model, ReLU is used as the activation function. Let relu*i_j* denote the result of conv*i_j* with ReLU activation. We compute Gram matrices between the feature maps output by the relu1_1, relu2_1, relu3_1, relu4_1, and relu5_1 layers, respectively. Then, we compute the L2 distances between the corresponding Gram matrices of the generated image and the ground truth, and sum up the corresponding distances as style loss $\mathcal{L}_{style}^{(2)}$. The weights to integrate distances from relu1_1, relu2_1, relu3_1, relu4_1, and relu5_1 are set to 0.244, 0.061, 0.015, 0.004, and 0.004, respectively. More specifically, they are given by $1000/(64 \times 64)$, $1000/(128 \times 128)$, $1000/(256 \times 256)$, $1000/(512 \times 512)$, and $1000/(512 \times 512)$, as mentioned in [11].

Finally, the overall loss function $\mathcal{L}_{BatikGAN\_S}^{(2)}$ of the BatikGAN with style features can be written as:

$$\mathcal{L}_{BatikGAN\_S}^{(2)} = \mathcal{L}_{adv}^{(2)} + \mathcal{L}_{local\_l2}^{(2)} + \mathcal{L}_{style}^{(2)}, \tag{6}$$

where the adversarial loss $\mathcal{L}_{adv}^{(2)}$ and the patch L2 distance $\mathcal{L}_{local\_l2}^{(2)}$ are defined as the same way in Sec. 2.2.1.

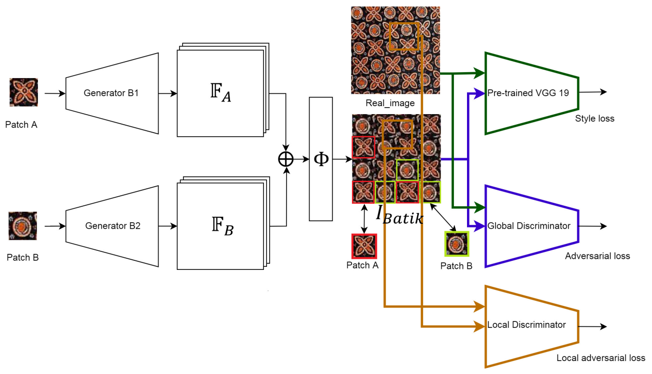

Figure 4: Illustration of BatikGAN model with style and local feature (BatikGAN_SL).

2.2.3 *BatikGAN with Style and Local Features.* One problem of the BatikGAN_S model is that it relies too much on the patch-based L2 distance. Although it makes generated patches well interlaced, it yields obvious blocking artifacts. The generated images usually look like a simple repetition of two patches. Motivated by [10], we design a local discriminator D_{local} to judge details of generated images. The local discriminator, in contrast to the global discriminator D_{global} mentioned earlier, only checks small local regions. We randomly sample m cropped region of size $s \times s$ from the generated image I_{Batik} and the ground-truth image $\boldsymbol{x}$, and then check them by the discriminator. Notice that one sampled region may cover pixels from multiple blocks defined in Eqn. (5). The local discriminator D_{local} is thus designed to check artifacts at block boundaries.

The local discriminator D_{local} learns to recognize a pair of regions as positive examples or negative examples. If the discriminator takes the pair from the same image, $D_{local}(\cdot) = 1$; otherwise $D_{local}(\cdot) = 0$. Let $C_i(\boldsymbol{x})$ be a cropped region of size $s \times s$ from the ground truth image $\boldsymbol{x}$. Given m pairs of cropped images $(C_1(\boldsymbol{x}), C_1(I_{Batik})), (C_2(\boldsymbol{x}), C_2(I_{Batik})), ..., (C_m(\boldsymbol{x}), C_m(I_{Batik}))$, we define the local adversarial loss $\mathcal{L}^{(3)}_{adv'}$ as follows:

$$\mathcal{L}^{(3)}_{adv'} = -\sum_m (D_{local}(C_m(\boldsymbol{x}), C_m(I_{Batik})) - 1)^2 . \quad (7)$$

We also encourage that m pairs of cropped images should be generated with the same style. The idea is similar to [8]. By computing local style loss $\mathcal{L}^{(3)}_{local\_style}$, the generator learns to generate a Batik image without blocking artifacts, as shown in Figure 4. Finally, the BatikGAN with style and local features (named as BatikGAN_SL) can be constructed based on the loss $\mathcal{L}^{(3)}_{BatikGAN\_SL}$:

$$\mathcal{L}^{(3)}_{BatikGAN\_SL} = \underbrace{\mathcal{L}^{(3)}_{adv} + \mathcal{L}^{(3)}_{local\_l2} + \mathcal{L}^{(3)}_{style}}_{Global feature} + \underbrace{\mathcal{L}^{(3)}_{adv'} + \mathcal{L}^{(3)}_{local\_style}}_{Local feature}, \quad (8)$$

where the adversarial loss $\mathcal{L}^{(3)}_{adv}$ and the patch L2 distance $\mathcal{L}^{(3)}_{local\_l2}$ are defined as the same way in Sec. 2.2.1, and the global style loss $\mathcal{L}^{(3)}_{style}$ is defined in Sec. 2.2.2.

With this system, users are allowed to generate Batik patches by the patch generator, and then choose two favorite patches as the input to generate Batik with mixed styles.

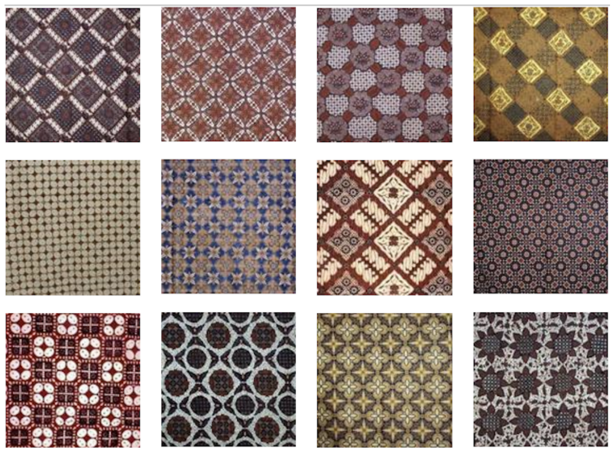

Figure 5: Sample Batik images obtained from the Machine Learning and Computer Vision (MLCV) Lab, Faculty of Computer Science, Universitas Indonesia [4].

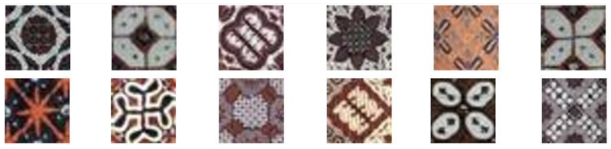

Figure 6: Sample patches cut from real Batik images.

3 EVALUATION

3.1 Dataset

We adopt the dataset proposed in [4] for evaluation. Batik images of six styles are included, including Ceplok, Kawung, Lereng, Mix motif, Nitik, and Parang. We choose 163 Batik images of regular styles and resize them into 128×128. After horizontal flipping and vertical flipping, we augment the data to 652 Batik images. Some sample images are shown in Figure 5. A Batik image is usually constituted by two basic patches.

The patch generator is trained based on 1,304 patches cut from the Batik images. We set the patch size as 32×32. Figure 6 shows some patches.

3.2 Training Details

To train the patch generator, the mini-batch size is set as 32, and the model is trained for 200,000 iterations. We randomly initialize parameters of the generator and the discriminator. The Adam optimizer is adopted to find good model parameters with momentum 0.5, and the learning rate is set as 0.0001. The gradient penalty λ is set as 10, according to [3].

To train the BatikGAN_SL model, the weights for $\mathcal{L}^{(3)}_{adv}$, $\mathcal{L}^{(3)}_{local\_l2}$, $\mathcal{L}^{(3)}_{style}$, $\mathcal{L}^{(3)}_{adv'}$, and $\mathcal{L}^{(3)}_{local\_style}$ are set as 1, 1, 10, 100, and 1, respectively. The Adam optimizer is adopted to find good model parameters with momentum 0.5. The learning rate for the generator is set as 0.001 and that for the discriminator is set as 0.0001. This model is trained for 1,000 epochs.

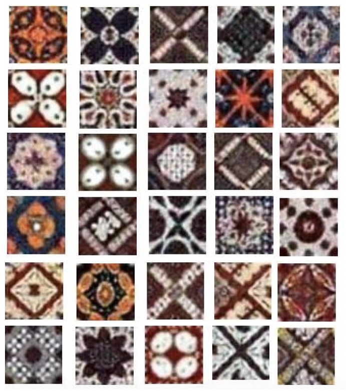

Figure 7: Some patches generated by the patch generation model (best viewed in color).

In the training process, we first minimize the losses $\mathcal{L}^{(3)}_{adv}$, $\mathcal{L}^{(3)}_{local\_l2}$, and $\mathcal{L}^{(3)}_{style}$ in $\mathcal{L}^{(3)}_{BatikGAN\_SL}$ for 50 epochs. We then add local feature losses $\mathcal{L}^{(3)}_{adv'}$ and $\mathcal{L}^{(3)}_{local\_style}$ to $\mathcal{L}^{(3)}_{BatikGAN\_SL}$, and train the local discriminator to distinguish cropped regions of size 48×48 for 950 epochs. All the weights and training settings for the BatikGAN and BatikGAN_S are quite the same.

3.3 Patch Generation Results

Some sample generated patches are shown in Figure 7. Let $Patch(x, y)$ denote the patch at the row x and the column y. $Patch(2, 1)$ and $Patch(3, 2)$ are similar but are in different color. $Patch(5, 4)$ and $Patch(6, 4)$ are also similar but in different color and with different lines. This shows the patch generator has the ability to create a variety of patches.

3.4 Batik Image Generation Results

Figure 8 shows sample generated Batik images. As can be seen, the BatikGAN_SL model not only learns repetition of two patches but also yields harmonious results. To compare our model with simple repetition, Figure 9 shows the visual comparison. The proposed BatikGAN_SL model generates Batik images with fused styles, with more diverse and rich presentation than simple repetition.

3.5 Performance Comparison

Given two patches, we compare Batik images generated by the three proposed models in Figure 10. In BatikGAN's results, the generator learns how to place two patches, but the generated results are noisy. By considering the global style feature, the BatikGAN_S learns fusing two styles and generates clearer results. Finally, by further considering local features, the BatikGAN_SL learns to generate Batik images with fewer blocking artifacts.

We can quantitatively evaluate quality of generation results based on Fréchet Inception Distance (FID) [5]. FID measures the distance between the real distribution and the generated distribution. It is calculated as follows: First, embedding the generated images into a feature space by Inception Net [9]. Second, build a continuous multivariate Gaussian for these embeddings and compute the mean and covariance. The aforementioned two processes are also applied to ground truth images. Third, calculate the Fréchet distance between these two Gaussians:

$$FID(x, g) = \| \mu_x - \mu_g \|_2^2 + Tr(\Sigma_x + \Sigma_g - 2(\Sigma_x \Sigma_g)^{1/2}), \quad (9)$$

where (μ_g, Σ_g) and (μ_x, Σ_x) are the mean and covariance of embeddings from the generated distribution and the real data distribution, respectively.

Table 1 shows performance comparison between three models, in terms of FIDs. As can be seen, the BatikGAN_SL's results yield the smallest FID.

Table 1: Performance comparison between the BatikGAN, the BatikGAN_S, and the BatikGAN_SL models.

	BatikGAN	BatikGAN_S	BatikGAN_SL
FID	168.42	163.92	**137.87**

Table 2: Performance comparison between [2], [6], [7], and the BatikGAN_SL model.

	[2]	[6]	[7]	BatikGAN_SL
FID	186.65	224.96	199.20	**137.87**

3.6 Comparison with Other Methods

Some methods have been proposed to interpolate two styles. The method of [2] can be used to synthesize a texture in-between two styles. Let G_1 and G_2 denote as Gram matrices of two textures. A new texture can be generated with the combination $\alpha \times G_1 + (1 - \alpha) \times G_2$, where $0 \leq \alpha \leq 1$ is the integration factor. We first make simple repetition based on $Patch_A$ and $Patch_B$ and generate texture images $Texture_A$ and $Texture_B$, respectively. We then set $\alpha = 0.5$ and use the method in [2], [6], and [7] to fuse two texture styles. We show performance comparison between [2], [6], [7], and our method in Figure 11. In the result, although [2], [6], and [7] fuses two patch styles, content of patches is not well maintained, and the generated image looks a bit messy. The results of our generated images show continuous lines and clear styles. Table 2 shows comparison between [2], [6], [7], and the Batik_SL model in terms of FID.

3.7 Texture Synthesis

We also use our model to synthesize new textures based on a texture dataset. We randomly crop 32×32 patches from the collected texture dataset, and then feed two patches to the BatikGAN_SL model to generate new textures. Figure 12 shows sample results. These results show that the proposed model is generic to generate different styles of texture images other than Batik images.

3.8 User Study

We study how users judge the generated Batik images. This survey was done by collecting replies from 209 subjects with diverse background. We randomly show nine pairs of images to each of them. For each pair, one is from the Batik dataset and another is generated by the BatikGAN_SL model. The question asked was:

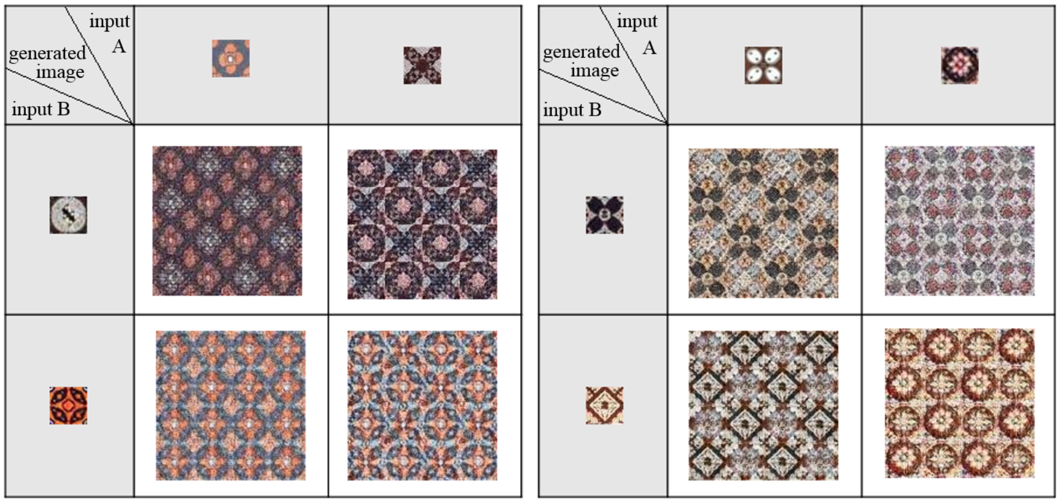

Figure 8: Batik images generated by the BatikGAN_SL model. The generated images not only show repetition of two patches, but also present fused styles based on the two given patches (best viewed in color).

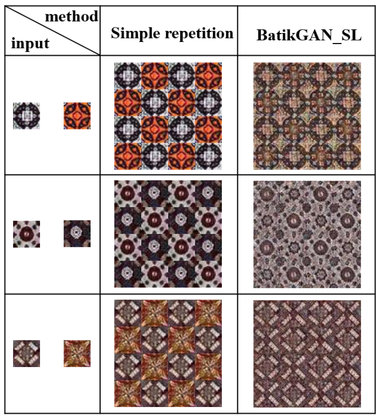

Figure 9: Comparison between the generated Batik images and simple repetition (best viewed in color).

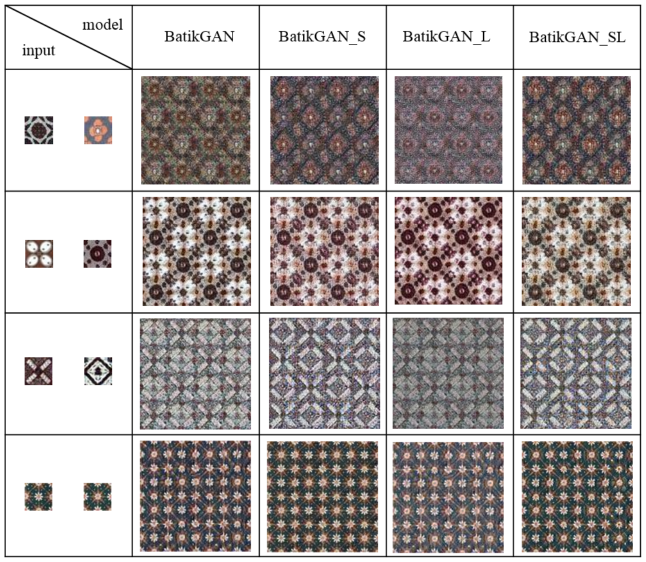

Figure 10: Batik images generated by the BatikGAN, the BatikGAN_S, BatikGAN_L, and the BatikGAN_SL models (best viewed in color).

Which looks visually better? For this question, 39% of subjects think that the images generated by the BatikGAN_SL model are better, 41% think the true Batik images are better, and 20% think that they are similar. These ratios show that the quality of generated Batik images is comparable to real Batik images.

We also compare the proposed three models, i.e., BatikGAN, BatikGAN_S, and BatikGAN_SL. We show three images generated by the three models, respectively, to each subject at the same time. Overall, 27%, 28%, and 45% of the subjects think that images generated by BatikGAN, BatikGAN_S, and BatikGAN_SL are the best, respectively. This result shows that images generated by the BatikGAN_SL model are more appreciated by subjects.

A statistical study is done to see if there is a statistical difference between the real Batik images and the images generated by the BatikGAN_SL model. The question asked is: *According to your instinct, which is better?* The voting results are shown in Table 3. In this table, numbers mean how many people vote to real Batik images or generated image. Over nine pairs of comparison, the mean number of votes to real Batik images is 80.78, and the standard deviation is 24.62. The mean number of votes to images generated by the BatikGAN_SL model is 84.67, and the standard deviation is 20.24. At $\alpha = 0.05$, we explore whether there exists a statistical difference in means of votes. Let μ_1 and μ_2 denote the mean votes to real Batik images and generated images, respectively.

Step 1 State the hypotheses and identify the claim.

$$H_0 : \mu_1 = \mu_2 \text{ (claim)} \quad \text{and} \quad H_1 : \mu_1 \neq \mu_2. \tag{10}$$

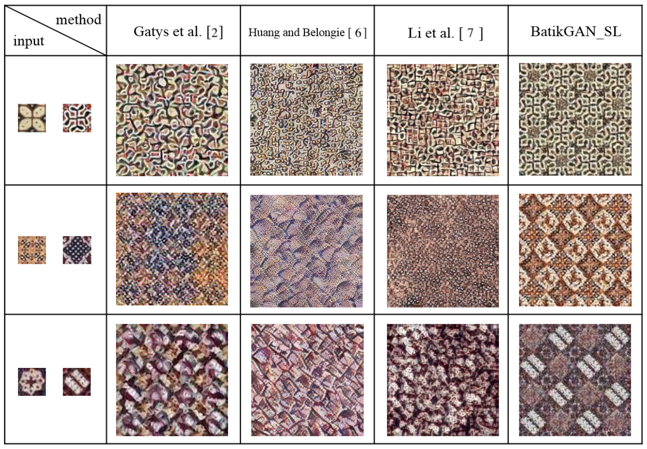

Figure 11: Comparison between [2], [6], [7], and the BatikGAN_SL model (best viewed in color).

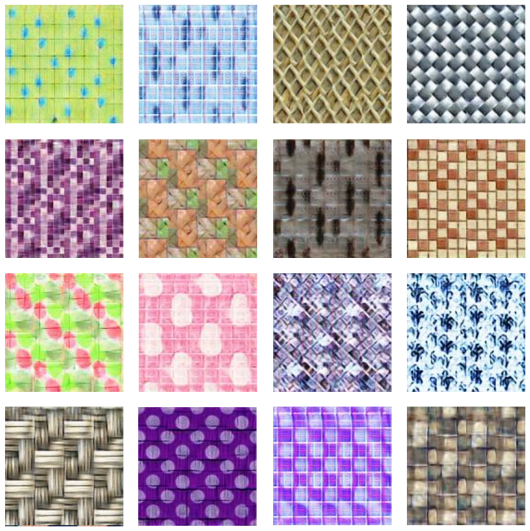

Figure 12: Sample texture synthesis results by the BatikGAN_SL model.

Step 2 Find the critical value. Since the test is two-tailed and $\alpha = 0.05$, the degrees of freedom is $9 - 1 = 8$. According to the statistical t-distribution table, the critical values are +2.306 and −2.306.
Step 3 Compute the test value.

$$t = (\bar{X_1} - \bar{X_2})/\sqrt{s_1^2/n_1 + s_2^2/n_2} = 0.366. \quad (11)$$

Step 4 Make the decision. Do not reject the null hypothesis since the test value 0.366 is less than the critical value 2.306.
Step 5 Summarize the results. There is not enough evidence to reject the claim that the mean numbers of votes for real Batik images and that generated by the BatikGAN_SL model are the same.

Table 3: Statistics of subjects' votes.

	pair 1	pair 2	pair 3	pair 4	pair 5	pair 6	pair 7	pair 8	pair 9
Real Batik images	103	118	81	48	61	65	73	112	66
Generated images	81	57	68	119	103	81	104	67	82
Same	25	34	60	42	45	63	32	30	61

4 CONCLUSION

We have presented a regular-texture synthesis method based on generative adversarial networks. With two patches as input, the generator fuses styles of two patches, and generates a Batik image of new style. By considering features progressively, the generator learns how to fuse two styles, removes blocking artifacts, and generates harmonious Batik images. We do a comprehensive user study and show promising results. Although the BatikGNA_SL model successfully generates harmonious Batik images with fused styles, the generated images are still not controllable. In the future, we will try to investigate attributes of Batik images, and develop an attribute-guided Batik image generator.

5 ACKNOWLEDGEMENT.

This work was partially supported by the Ministry of Science and Technology, Taiwan, under the grant 108-2221-E-006-227-MY3, 107-2221-E-006-239-MY2, 107-2923-E-194-003-MY3, 107-2627-H-155-001, and 107-2218-E-002-055.

REFERENCES

[1] V. Dumoulin, J. Shlens, and M. Kudlur. 2017. A learned representation for artistic style. In *Proc. of ICLR.*
[2] L.A. Gatys, A.S. Ecker, and M. Bethge. 2015. Texture synthesis using convolutional neural networks. In *Proc. of NIPS.* 262–270.
[3] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A.C. Courville. 2017. Improved training of wasserstein gans. In *Proc. of NIPS.* 5767–5777.
[4] Y. Gultom, A.M. Arymurthy, and R.J. Masikome. 2018. Batik Classification using Deep Convolutional Network Transfer Learning. *Jurnal Ilmu Komputer dan Informasi* 11, 2 (2018), 59–66.
[5] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proc. of NIPS.* 6626–6637.
[6] X. Huang and S. Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proc. of the ICCV.* 1501–1510.
[7] Y. Li, C. Fang, Z. Yang, J.and Wang, X. Lu, and M.-H. Yang. 2017. Universal style transfer via feature transforms. In *Proc. of NIPS.* 386–396.
[8] F. Luan, S. Paris, E. Shechtman, and K. Bala. 2018. Deep painterly harmonization. *Computer Graphics Forum* 37, 4 (2018), 95–106.
[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. 2015. Going deeper with convolutions. In *Proc. of CVPR.* 1–9.
[10] W. Xian, P. Sangkloy, V. Agrawal, A. Raj, J. Lu, C. Fang, F. Yu, and J. Hays. 2018. Texturegan: Controlling deep image synthesis with texture patches. In *Proc. of CVPR.* 8456–8465.
[11] Y. Zhou, Z. Zhu, X. Bai, D. Lischinski, D. Cohen-Or, and H. Huang. 2018. Non-stationary texture synthesis by adversarial expansion. In *Proc. of SIGGRAPH.*