ISSN (Online) 2747-0563
Volume 1 Tahun 2020

Otomatisasi Pewarnaan Citra Monokrom dengan Metode Generative Adversarial Network

Fajar Andhika Putra[1], Eva Yulia Puspaningrum[2*], Wahyu S.J Saputra[3]

[1] Informatika, Universitas Pembangunan Nasional "Veteran" Jawa Timur
[1]fajarhandika12@gmail.com
[3]wahyu.s.j.saputra.if@upnjatim.ac.id
*Corresponding author email: evapuspaningrum.if@upnjatim.ac.id

***Abstrak*— Sistem pewarnaan otomatis pada citra *grayscale* diperlukan agar perbaikan pada citra monokrom dapat dilakukan secara cepat dan tanpa keahlian khusus. Dalam pengimplementasian sistem ini terdapat beberapa kendala seperti domain warna pada satu objek bisa sangat beragam, dimana setiap benda bisa memiliki lebih dari 1 kemungkinan warna. Pada penelitian ini penulis menggunakan *Self-attention Generative Adversatial Network* (SAGAN) untuk melakukan pewarnaan otomatis pada *dataset Hackaton Blossom (Flower Classification)* yang bisa diakses pada situs resmi Kaggle, data ini berisikan citra 102 spesies bunga dengan jumlah data sebanyak 6500 data tes, 1000 data test dan 100 data validasi. SAGAN dipilih sebagai metode yang digunakan karena diklaim dapat menyelesaikan masalah *image-to-image translation* seperti *style transfer,* dan *image colorization.* Pada penelitian ini penulis berhasil mengimplementasikan dan melatih model SAGAN yang konvergen dengan memanfaatkan *spectral normalization*, selain itu penulis menambahkan *Structural Similarity Index Measure* (SSIM) *loss* untuk mengurangi munculnya *artifact* pada citra yang dihasilkan, model dilatih hingga 50 epoch dan berhasil memperoleh nilai Peek Signal-to-Noise Ratio (PSNR)diantara 51 dB hingga 73 dB dengan rata-rata nilai PNSR sebesar 61 dB.**

***Kata Kunci*— *Deep Learning*, SAGAN, Pewarnaan Otomatis, Citra Monokrom, *Spectral Normalization*.**

I. PENDAHULUAN

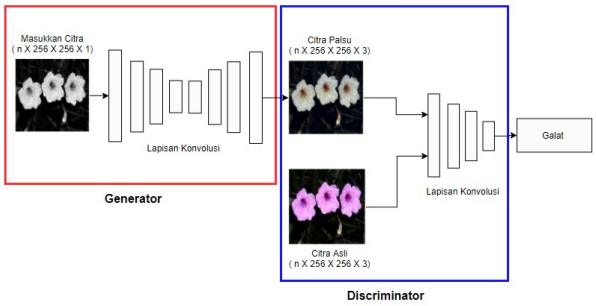
Citra Monokrom adalah sebuah citra yang hanya memiliki hitam atau putih pada sistem grafiknya[1]. Perbedaan warna pada citra monokrom ditentukan oleh derajat keabuan atau *grayscale*, biasanya nilai yang digunakan berada dalam rentang 0 sampai 1. Citra yang dihasilkan pada masa awal kemunculan kamera biasanya memiliki kualitas yang buruk dan terbatas pada citra monokrom sehingga sangat sulit untuk diinterpretasikan oleh mata manusia, untuk itu diperlukan perbaikan untuk memperoleh citra yang lebih baik agar citra lebih mudah untuk diinterpretasikan serta di analisis dengan menonjolkan ciri-ciri atau informasi tertentu pada citra [2]. Salah satu cara untuk memperbaiki kualitas citra adalah dengan mewarnai citra monokrom menjadi citra berwarna, umumnya cara yang dilakukan untuk mewarnai citra monokrom adalah dengan menggunakan perangkat lunak desain grafis, dimana kebanyakan prosesnya masih dilakukan secara manual sehingga membutuhkan waktu yang tidak sedikit serta keahlian khusus dalam mengoperasikan perangkat lunak yang tersedia.

Dari permasalahan tersebut dibutuhkan sebuah sistem yang dapat mewarnai citra monokrom secara otomatis sehingga tidak memerlukan keahlian khusus maupun waktu yang lama. Penelitian tentang pewarnaan citra monokrom menjadi citra berwarna telah dilakukan oleh Deshpande et al. pada tahun 2015 dengan menggunakan variasi dari LEARCH untuk menyeimbangkan akurasi *pixelwise* dan *spatial error*, penelitian ini menggunakan RMS pada *error matric*-nya dan memperolah nilai tertinggi 0,236%, dan mendapat 9,8% pada uji turing, kedua pengujian ini menggunan SUN *dataset* [3]. Selanjutnya penelitian serupa juga dilakukan oleh Zhang et al. pada tahun 2016 menggunakan metode Convolution Neural Network dan distribusi probabilitas dari tiap *pixel*, penelitian ini berhasil mendapat nilai sebesar 17,2% pada uji turing pada SUN *dataset* [4]. Pada tahun 2017 Cao et al. menggunakan *Wasserstein Generative Adversarial Network* (WGAN) untuk melakukan pewarnaan pada citra monokrom dan menguji modelnya menggunakan *dataset* LSUN *bedroom* dan berhasil memperoleh 62,6% pada uji turing [5]. Dari beberapa penelitian yang dipaparkan tersebut WGAN yang merupakan salah satu variasi dari GAN memiliki akurasi yang lebih baik jika dibandingkan dengan metode yang lain, sehingga pada penelitian ini, dipilih metode GAN karena akurasinya yang lebih baik. Penelitian ini menggunakan *Self-attention GAN* (SAGAN) yang merupakan salah satu variasi dari GAN selain WGAN, SAGAN dipilih karena pada penelitian Zhang et al. metode ini diklaim menghasilkan *inception-score* yang lebih baik serta proses pelatihannya yang lebih stabil dari WGAN [6], model pada penelitian ini diuji menggunakan *dataset* dengan nama *Hackaton Blossom (Flower Classification)* yang didapat dari situs resmi Kaggle[7].

II. METODOLOGI PENELETIAN

Model yang digunakan pada penlitian ini adalah *Self-attention Generative Adversarial Network* (SAGAN), dengan input berupa citra *grayscale*, model dapat menghasilkan citra berwarna berdasarkan pasangan citra *grayscale* dengan citra warna pada data latih.

ISSN (Online) 2747-0563
Volume 1 Tahun 2020



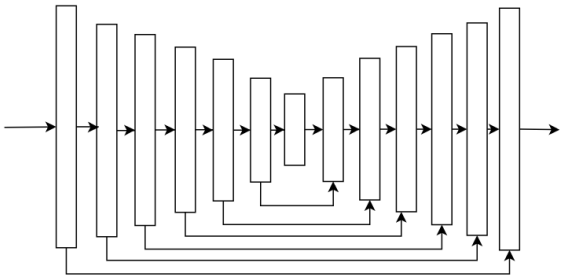
Gbr. 1 Ilustrasi SAGAN.

Pada *generator* kami menggunakan *feed-forward deep neural network*, *generator* menerima citra *grayscale* berukuran 128x128 piksel sebagai input dan mengeluarkan citra berwarna dengan resolusi yang sama. Sama seperti *generator*, kami menggunakan *feed-forward deep neural network* pada *discriminator* hanya saja masukannya berupa 2 citra berwarna kemudian mengklasifikasi masukan yang diterima dari *dataset* dan *generator,* dan menghasilkan galat yang nantinya akan digunakan pada proses propagasi balik.

A. Arsitektur Model

Sedikit berbeda dengan beberapa solusi yang diusulkan sebelumnya [8], [9] yang menggunakan endcoder-decoder tradisional dimana masukan diproses pada deretan lapisan *downsaple* hingga pada resolusi yang paling kecil, kemudian melewati lapisan *fully connected* baru kemudian melewati lapisan *upsample* untuk mengembalikan resolusi citra, pada arsitektur seperti ini dapat mengakibatkan hilangnya beberapa informasi yang penting pada citra ketika melewati tiap lapisan, padahal pada masalah pewarnaan citra *edge* merupakan informasi yang sangat penting dan diminimalisir kehilangannya ketika melewati lapisan konvolusi untuk menjaga kualitas citra, karena alasan tersebut pada penelitian ini kami menggunakan "U-net" pada *generator* seperti yang disarankan oleh isola et al. [10] selain itu kami juga menambahkan *self-attention* dan *spectral normalization* untuk mendapatkan hasil yang detail dan performa yang lebih stabil seperti yang diusulkan oleh Zhang et al [6].

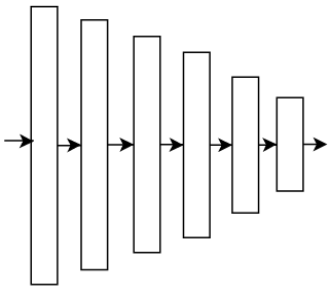
Berikut merupakan arsitektur yang kami gunakan pada penelitian ini, dimana *Ck* merupakan lapisan dengan Konvolusi-*Batch Normalization*-LeakyReLU dan *k* merupakan *filter* pada lapisan, *CDk* merupakan lapisan dengan Konvolusi-*Dropout*-ReLU (*dropout rate* = 50%). Pada *generator* semua lapisan menggunakan 4x4 *spatial filter* dan *stride* 2.



Gbr. 2 Ilustrasi arsitektur pada generator.

Pada *generator* arsitektur terbagi menjadi 2 yaitu *encoder* dan *decoder* dimana pada bagian *encoder* arsitektur yang digunakan adalah *C64-C128-C256-C512-C512-C512-C512,* kemudian pada bagian *decoder* arsitektur yang digunakan adalah *CD512-CD512-CD512-CD256-CD128-CD64*, selain itu terdapat *skip connection* pada lapisan ke *n* pada *encoder* dan *n – 1* pada *decoder* sehingga *channel* pada *decoder* akan berubah menjadi 1024-1024-1024-512-256-128 arsitektur ini diusulkan oleh Zhang et al. [10], sebagai tambahan pada lapisan pertama *encoder* tidak digunakan *batch normalization* dan selain lapisan pertama dan kedua pada *decoder* kami tidak menggunakan *dropout*. Pada lapisan *n* hingga lapisan *n – 1* kami menggunakan *self-attention*, serta pada setiap lapisan konvolusi kami menggunakan *spectral normalization.* Pada LeakyReLU kami menggunakan alpha sebesar 0,2.

Arsitektur pada *discriminator* merupakan arsitektur umum yang digunakan untuk menyelesaikan masalah klasifikasi, sama seperti *generator* pada *discriminator* kami juga menggunakan *spectral normalization* pada setiap lapisan konvolusi dan *self-attention* pada lapisan *n* hingga lapisan *n–1* berikut merupakan arsitektur *discriminator* yang kami gunakan *CD64-CD128-C256-C512-C512* kemudian di akhir lapisan kami menggunakan 1x1 lapisan konvolusi untuk memetakan keluaran menjadi 1 dimensi dan diikuti dengan sigmoid sebagai fungsi aktivasinya, LeakyReLU pada arsitektur ini menggukan alpha sebesar 0,2.



Gbr. 3 Ilustrasi arsitektur pada discriminator.

B. Loss Function

Dengan *G* merupakan *generator, D* merupakan *discriminator, x* merupakan citra sesungguhnya, *y* merupakan citra *grayscale, loss function* pada GAN dapat dirumuskan sebagai berikut

$$L_{GAN}(G,D) = E_x[\log D(x)] + E_y\left[\log\left(1 - D\big(G(y)\big)\right)\right] \quad (1)$$

Pada penelitian sebelumnya ditemukan bahwa dengan menambahkan *loss function* yang lebih tradisional dapat meningkatkan hasil yang didapatkan, seperti L1 [10], dalam beberapa kasus lain hasil keluaran yang dihasilkan terdapat *noise* yang biasa disebut dengan *artifact* [11], untuk itu kami menambahkan SSIM, dengan μ_x merupakan rata-rata x, μ_y merupakan rata-rata y, σ_x merupakan varian dari x, σ_y merupakan varian dari y dan c_1, c_2 merupakan variabel untuk menstabilkan pembagian, maka fungsi SSIM dapat dirumuskan sebagai berikut

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} (2)$$

Sehingga *loss funtion* yang kami gunakan adalah

$$G^* = \arg\min_G \max_D L_{GAN}(G,D) + \alpha SSIM(x,y) (3)$$

Dimana α merupakan bilangan antara 0 – 100, dan x adalah data sesungguhnya. Semakin tinggi nilai α maka keluaran yang dihasilkan akan semakin mendekati data sesungguhnya. Pada penelitian ini penulis menggunakan α sebesar 100

III. HASIL DAN PEMBAHASAN

A. *Dataset*

Dataset yang kami gunakan pada penelitian ini merupakan data yang penulis dapatkan dari situs Kaggle [7], yang dibuat oleh pengguna bernama Spaic Hackathoners data ini berisi citra bunga dari 102 spesies yang berbeda dengan total data sekitar 6500 data dimana data tes sekitar 1000 data serta sekitar 100 data validasi, pada data latih terdapat sekitar 50 data tiap kelasnya.

Agar sesuai dengan model yang kami buat pada penelitian ini, citra harus di *resize* menjadi ukuran 128x128 piksel kemudian pada data latih dilakukan augmentasi data dengan cara melakukan *flip* baik secara horizontal maupun vertikal pada data acak untuk menambah jumlah data dengan peluang 50% pada setiap citra.

Gbr. 4 Keluaran dari augmentasi data.

Pada penelitian ini model yang dibuat membutuhkan pasangan data citra *grayscale* dengan citra berwarna, untuk itu data harus diubah menjadi 2 citra, citra *grayscale* yang akan dijadikan masukan pada *generator* dan citra berwarna yang akan menjadi masukan pada *discriminator* bersama dengan citra keluaran dari *generator*.

Untuk mendapatkan citra *grayscale* citra terlebih dahulu diubah kedalam bentuk LAB *colorspace*, berbeda dengan RGB yang membagi warna menjadi merah-hijau-biru LAB membagi warna menjadi *Lightness-a-b* dimana *a* adalah nilai dari warna merah-hijau dan *b* adalah nilai dari warna biru-kuning sementara *L* atau *Lightness* merupakan derajat keabuan atau *grayscale* dari citra, nilai inilah yang nantinya akan menjadi masukan pada *generator*, yang kemudian *generator* akan memprediksi nilai a-b dari citra kemudian nilai *L* (masukan pada *generator*) dan *a-b* (keluaran dari *generator*) akan digabungkan kembali agar menjadi citra berwarna yang nantinya akan menjadi masukan pada *discriminator* bersama dengan citra sesungguhnya.

Gbr. 5 Citra kiri merupakan citra sesungguhnya, dan citra kanan merupakan nilai L dari hasil konversi citra RGB ke-LAB.

Proses terakhir sebelum cintra menjadi masukan pada model adalah melakukan normalisasi pada citra *grayscale* yang didapatkan dari proses sebelumnya, hal ini dimaksudkan agar komputasi yang dilakukan tidak terlalu besar. Nilai setiap piksel setelah dilakukan normalisasi adalah antara 0 sampai 1.

Sedikit berbeda dengan data latih, pada data tes dan validasi kami tidak melakukan augmentasi data, dan hanya mengubah ukuran gambar menjadi 128x128 piksel, kemudian mengubah citra menjadi *grayscale* dan terakhir citra dinormalisasi.

B. *Evaluasi*

Proses evaluasi pada masalah pewarnaan sebenarnya merupakan proses yang sederhana dan langsung, namun proses ini sangat subjektif sehingga tidak terlalu akurat, untuk itu kami mengevaluasi model yang telah dibuat secara kuantitatif menggunakan *Peak Signal-to-Noise Ratio* atau PSNR. PSNR adalah besaran yang mewakili rasio antara kekuatan maksimum sinyal dan kekuatan gangguan, karena banyak sinyal yang memiliki rentang yang sangat bervariasi dan dinamis PSNR biasanya dinyatakan dalam satuan desibel (*dB*).

PSNR digunakan untuk menghitung atau memperkirakan kualisas citra palsu (keluaran dari *generator*) dengan citra asli [12], diberikan citra u_0 yang merupakan citra asli berwarna dengan ukuran m x n dan citra hasil keluaran pewarnaan *u,* PSNR dapat dirumuskan sebagai berikut :

$$PSNR = 10\times \log\left(\frac{3mn(MAX^2)}{\sum_{RGB}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[u(i,j) - u_0(i,j)]^2}\right) (4)$$

Dimana *MAX* merupakan kemungkinan nilai tertinggi pada piksel (contoh 255 pada citra 8bit standar), dan $\sum_{RGB}$ merupakan penjumlahan nilai piksel pada *channel* merah, hijau dan biru, semakin tinggi nilai PSNR berarti semakin tinggi pula kualitas citra yang dihasilkan.

C. *Ujicoba*

Percobaan dilakukan pada Kaggle kernel dengan sistem operasi Linux, Intel(R) Xeon(R) @ 2,00 GHz, dengan RAM 13 GB, GPU 16 GB dengan bahasa pemrograman Python, serta *Deep learning framework Tensorflow.Keras*.

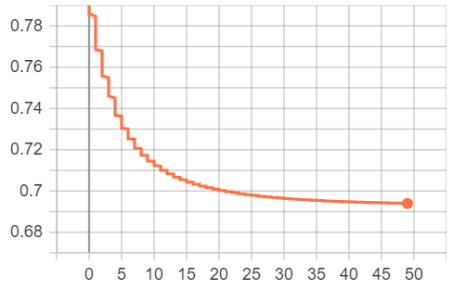
Model dilatih menggunakan *mini-batch SGD* selama 50 *epoch* dengan ukuran *batch* 4 pada citra berukuran 128×128, pada *optimizer* kami menggunakan *Adam optimizer* dengan *learning rate* sebesar $1e^{-4}$ pada *generator* dan $4e^{-4}$ pada

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

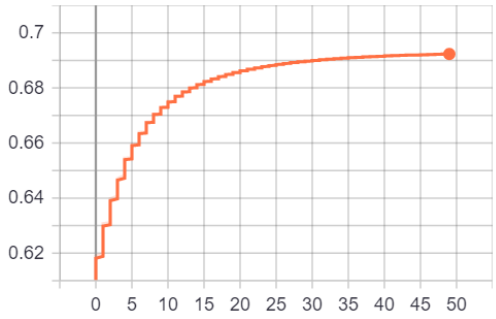
discriminator dengan beta_1 sebesar 0,0 dan beta_2 sebesar 0,9 pada *generator* maupun *discriminator*. Waktu yang diperlukan untuk menyelesaikan proses latih adalah 24.816 detik atau sekitar 7 jam.

D. Hasil

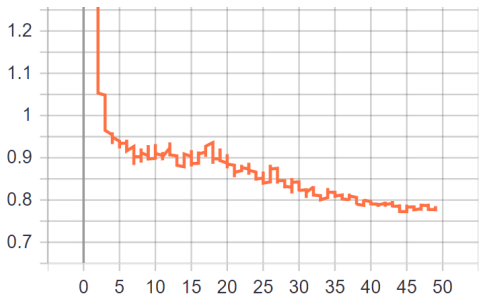
Selama proses penelitian kami melakukan pelacakan pada *generator loss, dicriminator loss,* dan *SSIM loss* untuk mengetahui perkembangan model selama proses pelatihan berlangsung, berikut grafik dari ke 3 *loss function* tersebut



Gbr. 6 Grafik *discriminator loss.*



Gbr. 7 Grafik *generator loss.*



Gbr. 8 Grafik *SSIM loss.*

Seperti yang terlihat pada Gbr. 6, Gbr. 7 menginterpretasi galat pada GAN tidak seperti model regresi maupun klasifikasi biasa, pada GAN nilai *discriminator loss* dan *generator loss* tidak boleh saling mengungguli yang lain karena akan mengakibatkan model tidak konvergen. Pada kasus ini nilai yang ideal adalah sekitar 0,69 yang merupakan hasil dari log(2), nilai ini menandakan bahwa *discriminator* tidak yakin diantara 2 pilihan.

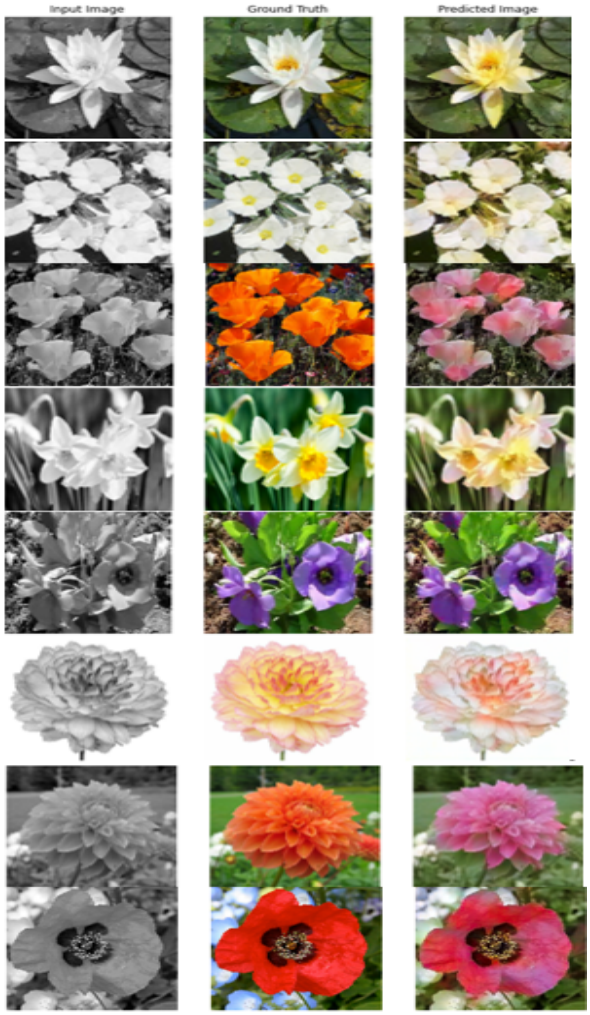
Pada Gbr. 8 terlihat bahwa seiring dengan bertambahnya *epoch* nilai *SSIM loss* semakin berkurang, hal ini menunjukkan bahwa selama proses pelatihan kualitas citra yang dihasilkan oleh *generator* semakin bagus.

TABEL I
PERBANDINGAN NILAI PSNR

Model	PSNR Min	PSNR Max	PSNR AVG
Pix2pix	51,3	71,3	60
Model Kami	51,75	73,64	61

Nilai PSNR yang didapatkan pada data uji ketika *epoch* ke 50 berkisar antara 51,75 dB hingga 73,6 dB dengan rata-rata nilai PSNR sekitar 61 dB, hasil ini menunjukkan sedikit peningkatan dari model pix2pix yang diusulkan oleh Isola et al. yang mendapat nilai PSNR antara 51,3 dB hingga 71,3 dB dengan rata-rata nilai PSNR sekitar 60 db, hanya saja pada pix2pix nilai *generator loss* dan *discriminator loss* selama proses pelatihan tidak stabil, serta terkadang model menjadi tidak konvergen.

Berikut ini merupakan beberapa citra yang dihasilkan oleh *generator*:



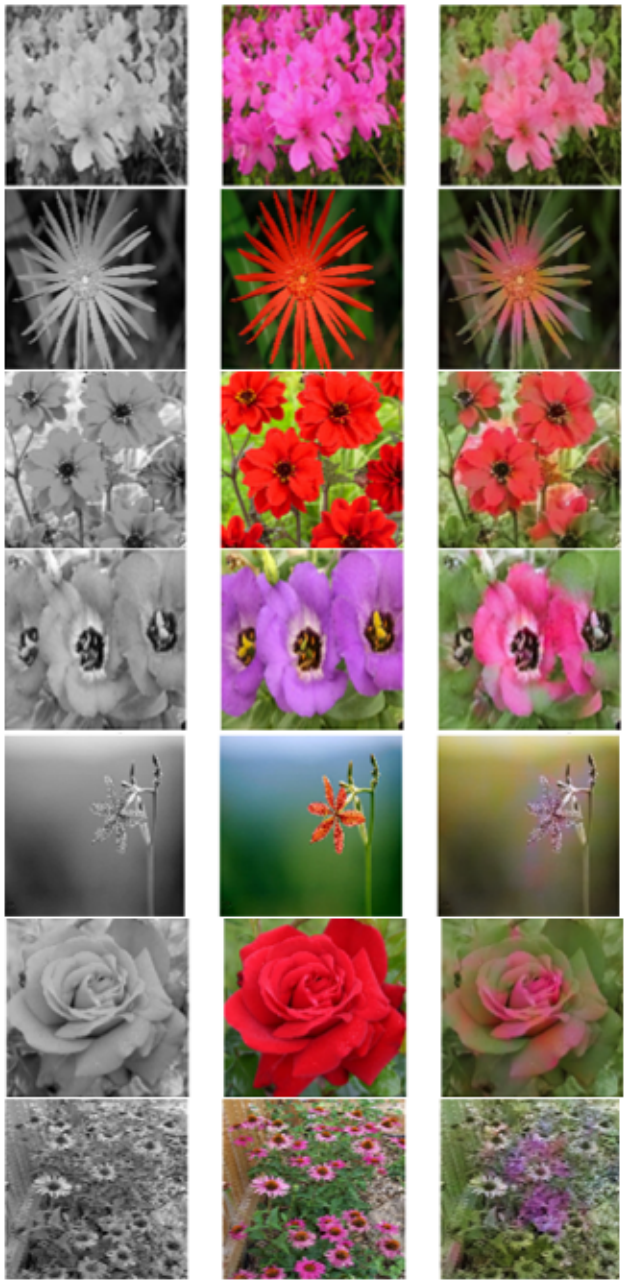
Gbr. 9 Keluaran yang berhasil, berurutan dari kiri adalah: citra masukan, citra asli, citra keluaran.

Seperti yang terlihat pada Gbr. 9 *generator* dapat menghasilkan citra berwarna dari citra *grayscale*, walaupun

ISSN (Online) 2747-0563
Volume 1 Tahun 2020

beberapa warna tidak sesuai dengan citra aslinya namun citra keluaran *generator* masih dapat dikenali dan realistis sehingga manusia masih bisa membedakan jenis bunga tersebut.

Selain citra yang sukses, masih terdapat banyak citra masukan yang tidak berhasil diwarnai oleh *generator* berikut beberapa citra yang gagal diwarnai oleh *generator*



Gbr. 10 Keluaran yang gagal, berurutan dari kiri adalah: citra masukan, citra asli, citra keluaran.

Seperti yang terlihat pada Gbr. 10 selama pengujian terdapat beberapa kasus dimana citra tidak sepenuhnya diwarnai dan beberapa citra yang memiliki tonal kecoklatan, terdapat pula beberapa keluaran citra yang masih menjadi citra monokrom.

IV. KESIMPULAN

Model yang digunakan pada penelitian ini dapat melakukan pewarnaan secara otomatis hingga akurasi sebesar 61 dB jika menggukan PSNR untuk validasi, meskipun beberapa warna pada citra yang dihasilkan oleh *generator* berbeda dengan citra asli namun masih dapat dikenali dan realistis, sehingga manusia masih dapat membedakan jenis bunga tersebut. Pengembangan yang dapat dilakukan dengan topik yang serupa adalah dengan menambah data yang ada ataupun menggunakan data yang berbeda, serta melakukan beberapa penyetelan *hyperparameter* dan arsitektur yang ada agar tercipta model yang lebih baik lagi.

UCAPAN TERIMA KASIH

Kami ucapkan terima kasih kepada pihak-pihak yang ikut berkontribusi serta dukungan selama proses pengerjaan makalah ini, sehingga makalah ini dapat diselesaikan tepat waktu.

REFERENSI

[1] SRI SUCI GIANA, “No Title,” Universitas Islam Negeri Sultan Syarif Kasim, 2014.
[2] J. C. Russ, *The Image Processing Handbook, Sixth Edition*, 6th ed. CRC Press, 2011.
[3] A. Deshpande, J. Rock, and D. Forsyth, “Learning Large-Scale Automatic Image Colorization,” Dec. 2015.
[4] R. Zhang, P. Isola, and A. A. Efros, “Colorful Image Colorization.” 2016.
[5] Y. Cao, Z. Zhou, W. Zhang, and Y. Yu, “Unsupervised Diverse Colorization via Generative Adversarial Networks.” 2017.
[6] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, “Self-Attention Generative Adversarial Networks.” 2019.
[7] S. HACKATHONERS, “Hackathon Blossom (Flower Classification),” 2019. https://www.kaggle.com/spaics/hackathon-blossom-flower-classification.
[8] P. Sangkloy, J. Lu, C. Fang, F. Yu, and J. Hays, “Scribbler: Controlling Deep Image Synthesis with Sketch and Color.” 2016.
[9] X. Wang and A. Gupta, “Generative Image Modeling using Style and Structure Adversarial Networks.” 2016.
[10] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks.” 2018.
[11] L. Galteri, L. Seidenari, M. Bertini, and A. Del Bimbo, “Deep Generative Adversarial Compression Artifact Removal.” 2017.
[12] D. Seo, Y. Kim, Y. D. Eo, and W. Park, “Learning-Based Colorization of Grayscale Aerial Images Using Random Forest Regression,” *Appl. Sci.*, vol. 8, p. 1269, 2018, doi: 10.3390/app8081269.