KERJA PRAKTIK - EF234603

Pengembangan Model AI NijiGAN menggunakan Contrasive Learning dan Persamaan Diferensiasi untuk Translasi Gambar Nyata ke Anime

Institut Teknologi Sepuluh Nopember
Jl. Teknik Kimia, Keputih, Kec. Sukolilo, Surabaya, Jawa Timur 60111
Periode: 1 Agustus 2024 - 30 November 2024

Oleh:

Farah Dhia Fadhila 5025211030
Dimas Prihady Setyawan 5025211184

Pembimbing Jurusan
Dr. Eng Darlis Herumurti, S.Kom, M.Kom
Pembimbing Lapangan
Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2024

KERJA PRAKTIK – EF234603

Pengembangan Model AI NijiGAN menggunakan Contrasive Learning dan Persamaan Diferensiasi untuk Translasi Gambar Nyata ke Anime

Institut Teknologi Sepuluh Nopember
Jl. Teknik Kimia, Keputih, Kec. Sukolilo, Surabaya, Jawa Timur 60111
Periode: 1 Agustus 2024 - 30 November 2024

Oleh:

Farah Dhia Fadhila 5025211030
Dimas Prihady Setyawan 5025211184

Pembimbing Jurusan
Dr. Eng Darlis Herumurti, S.Kom, M.Kom
Pembimbing Lapangan
Dr. Anny Yuniarti, S.Kom., M.Comp.Sc.

DEPARTEMEN TEKNIK INFORMATIKA
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2024

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN
KERJA PRAKTIK

Pengembangan Model AI NijiGAN menggunakan Contrasive Learning dan Persamaan Diferensiasi untuk Translasi Gambar Nyata ke Anime

Oleh:

Farah Dhia Fadhila	5025211030
Dimas Prihady Setyawan	5025211184

Disetujui oleh Pembimbing Kerja Praktik:

1. Dr. Eng Darlis Herumurti, S.Kom, M.Kom
   NIP. 197712172003121001

(Pembimbing Departemen)

2. Dr. Anny Yuniarti, S.Kom., M.Kom.
   NIP. 198106222005012002

(Pembimbing Lapangan)

[Halaman ini sengaja dikosongkan]

Pengembangan Model AI NijiGAN menggunakan Contrasive Learning dan Persamaan Diferensiasi untuk Translasi Gambar Nyata ke Anime

Nama Mahasiswa	: Farah Dhia Fadhila
NRP	: 5025211030
Nama Mahasiswa	: Dimas Prihady Setyawan
NRP	: 5025211184
Departemen	: Teknik Informatika FTEIC-ITS
Pembimbing Departemen	: Dr. Eng Darlis Herumurti, S.Kom, M.Kom
Pembimbing Lapangan	: Dr. Anny Yuniarti, S.Kom.,M.Kom

ABSTRAK

Generative AI telah merevolusi industri animasi, termasuk konversi gambar dunia nyata menjadi anime melalui translasi tanpa pasangan. Scenimefy, model berbasis contrastive learning*, mencapai hasil fidelitas tinggi namun terbatas oleh ketergantungan pada data berpasangan berkualitas rendah dan arsitektur berparameter tinggi. Penelitian ini memperkenalkan NijiGAN, model inovatif yang memanfaatkan Neural Ordinary Differential Equations (NeuralODEs) untuk transformasi berkelanjutan. NijiGAN menggunakan separuh parameter Scenimefy dengan data* pseudo-paired *untuk pelatihan, sehingga meningkatkan proses pelatihan dan menghilangkan kebutuhan data berkualitas rendah. Evaluasi menunjukkan NijiGAN menghasilkan gambar berkualitas lebih tinggi dengan Mean Opinion Score (MOS) 2,192, melampaui AnimeGAN (2,160), serta skor Frechet Inception Distance (FID) 58,71, lebih baik dari Scenimefy (60,32). Hasil ini menegaskan bahwa NijiGAN*

kompetitif sebagai alternatif state-of-the-art *dalam translasi gambar*.

Kata Kunci : Translasi Gambar ke Gambar, NeuralODEs, Pelatihan* Semi-supervised*, Generative Artificial Intelligence

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Allah SWT atas penyertaan dan karunia-Nya sehingga penulis dapat menyelesaikan salah satu kewajiban penulis sebagai mahasiswa Departemen Teknik Informatika ITS yaitu Kerja Praktik yang berjudul: Pengembangan Model AI NijiGAN menggunakan *Contrasive Learning* dan Persamaan Diferensiasi untuk Translasi Gambar Nyata ke Anime.

Penulis menyadari bahwa masih banyak kekurangan baik dalam melaksanakan kerja praktik maupun penyusunan buku laporan kerja praktik ini. Namun penulis berharap buku laporan ini dapat menambah wawasan pembaca dan dapat menjadi sumber referensi.

Melalui buku laporan ini penulis juga ingin menyampaikan rasa terima kasih kepada orang-orang yang telah membantu menyusun laporan kerja praktik baik secara langsung maupun tidak langsung antara lain:

1. Tuhan Yang Maha Esa atas berkat dan rahmat yang selalu diberikan oleh-Nya.
2. Kedua orang tua penulis
3. Bapak Dr. Eng Darlis Herumurti, S.Kom, M.Kom selaku dosen pembimbing kerja praktik sekaligus koordinator kerja praktik
4. Ibu Dr. Anny Yuniarti, S.Kom. selaku pembimbing lapangan selama kerja praktik berlangsung.
5. Segenap jajaran dosen dan tenaga pendidik ITS

Surabaya, 20 Desember 2024

Farah Dhia Fadila dan Dimas Prihady Setyawan

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Kemajuan ilmu pengetahuan dan teknologi (IPTEK), terutama di bidang *Artificial Intelligence* (AI), telah berkembang pesat dalam beberapa tahun terakhir. Salah satu inovasi yang menarik perhatian adalah kemampuan AI dalam melakukan translasi visual, yaitu mengubah gambar nyata menjadi gambar bergaya anime dengan tingkat akurasi dan detail yang luar biasa. Teknologi ini tidak hanya menciptakan peluang baru dalam industri kreatif, seperti animasi dan desain, tetapi juga menunjukkan bagaimana AI dapat merevolusi cara kita menciptakan dan menikmati karya seni.

Institut Teknologi Sepuluh Nopember (ITS) sebagai Perguruan Tinggi Negeri (PTN) yang berfokus pada pengembangan teknologi, terus menghasilkan inovasi melalui karya dosen dan mahasiswanya. Salah satu bidang riset yang menjadi perhatian adalah teknologi kecerdasan buatan (AI), termasuk penelitian terkait translasi gambar nyata ke gambar bergaya anime. Penelitian serupa sebelumnya telah dilakukan oleh Nanyang Technological University (NTU) dengan makalah bertajuk "*Scenimefy: Learning to Craft Anime Scene via Semi-Supervised Image-to-Image Translation*".

Model Scenimefy menggunakan teknik *contrastive learning* dan pelatihan *semi-supervised* untuk menerjemahkan gambar nyata ke gambar bergaya anime dengan tingkat akurasi yang tinggi. Namun, model ini memiliki beberapa kekurangan, seperti ketergantungan yang besar pada data berpasangan (*paired data*) yang disesuaikan dengan domain anime, yang sering kali menghasilkan data berkualitas rendah. Selain itu, jumlah

parameter yang besar membuat Scenimefy menjadi terlalu kompleks dan kurang efisien.

Untuk mengatasi kekurangan tersebut, ITS mengembangkan model bernama NijiGAN dalam program kerja praktik ini. Model ini dirancang dengan fokus pada efisiensi yang lebih tinggi, ukuran yang lebih kecil, dan kemampuan untuk bekerja secara *real-time*. NijiGAN menggunakan pendekatan *Neural Ordinary Differential Equations* (NeuralODEs) untuk melakukan translasi gambar tanpa pasangan (*unpaired image-to-image translation*) serta transfer gaya. Selain itu, NijiGAN memanfaatkan data *pseudo-paired* yang dihasilkan oleh Scenimefy untuk *supervised training*, sehingga dapat mengurangi ketergantungan pada data berpasangan berkualitas rendah.

Diharapkan, NijiGAN dapat menjadi dasar bagi penelitian lanjutan, baik oleh dosen maupun mahasiswa ITS atau pihak lain yang tertarik mengembangkan teknologi AI untuk translasi gambar nyata ke gambar bergaya anime dengan cara yang lebih efisien dan *real-time.*

1.2. Tujuan

Tujuan kerja praktik ini adalah menyelesaikan kewajiban nilai kerja praktik sebesar 4 sks dan mengembangkan model translasi gambar ke gambar dengan parameter yang lebih kecil dan efisiensi yang lebih tinggi dibandingkan model terdahulunya yakni Scenimefy.

1.3. Manfaat

Manfaat yang diperoleh dengan adanya pengembangan model NijiGAN ini antara lain penulis dapat memperoleh pengetahuan lebih lanjut terkait

Generative AI dan juga memperoleh pengalaman dalam mengembangkan riset model AI.

1.4. Rumusan Masalah

Rumusan masalah dari kerja praktik ini adalah sebagai berikut:

1. Bagaimana cara memperbaiki kompleksitas jumlah parameter yang tinggi pada model Scenimefy agar menjadi lebih efisien?
2. Bagaimana cara meningkatkan kualitas hasil translasi pada model Scenimefy yang terlalu bergantung pada data pasangan (*paired*) berkualitas rendah yang dihasilkan oleh model StyleGAN yang disesuaikan untuk domain anime?

1.5. Lokasi dan Waktu Kerja Praktik

Pengerjaan kerja praktik dilakukan secara luring di laboratorium KCKS lantai 11 tower 2 ITS dan daring melalui platform Zoom

Kerja praktik dilaksanakan mulai tanggal 1 Agustus 2024 hingga 30 November 2024. Kegiatan ini berlangsung setiap hari Selasa dan Kamis, pukul 16.00 hingga 20.00 WIB.

1.6. Metodologi Kerja Praktik

Metodologi dalam pembuatan buku kerja praktik meliputi :

1.6.1. Perumusan Masalah

Kami melakukan diskusi dengan Kevin Putra Santoso selaku kepala dari proyek KP dan Bu Anny Yuniarti sebagai dosen pembimbing lapangan. Pada saat rapat kami mendiskusikan bagaimana arsitektur dari NijiGAN akan dibangun berupa modul-modul dari *supervised training* dan *unsupervised training*.

1.6.2. Studi Literatur

Setelah mendapat gambaran bagaimana arsitektur NijiGAN, kami mempelajari bahasa dan beberapa pustaka yang digunakan, seperti Python, PyTorch, dan lain-lain. Selain itu, kamu juga dijelaskan alur bagaimana arsitektur mulai dari *supervised training* dan *unsupervised training* dari NijiGAN bekerja.

1.6.3. Analisis dan Perancangan Sistem

Arsitektur NijiGAN yang terdiri dari *supervised training* dan *unsupervised training*. *Supervised training* mempelajari cara membuat gambar anime berdasarkan hasil model Scenemify sementara *unsupervised training* berperan dalam mempelajari gaya dari anime Makoto Shinkai sementara.

1.6.4. Implementasi Sistem

Implementasi model NijiGAN menggunakan Supercomputer menggunakan bahasa Python dengan kerangka kerja Pytorch. Pada tahap ini kami menerapkan modul-modul pelatihan untuk *supervised training* dan *unsupervised training*.

1.6.5. Penguji dan Evaluasi

Setelah implementasi selesai, hasil dari NijiGAN dibandingkan secara penilaian kuantitatif dengan beberapa model pembanding berdasarkan nilai FID. Sementara penilaian semi-kualitatif menggunakan *Mean Opinion Score* (MOS) yang dinilai dari model NijiGAN dan pembanding berdasarkan pengamatan mata manusia.

1.6.6. Kesimpulan dan Saran

Setelah keseluruhan model dan uji selesai dilaksanakan, model NijiGAN siap untuk digunakan.

1.7. Sistematika Laporan

1.7.1. Bab I Pendahuluan

Bab ini berisi latar belakang, tujuan, manfaat, rumusan masalah, lokasi dan waktu kerja praktik, metodologi, dan sistematika laporan.

1.7.2. Bab II Profil Perusahaan

Bab ini berisi gambaran umum Institut Teknologi Sepuluh Nopember mulai dari profil dan lokasi perusahaan.

1.7.3. Bab III Tinjauan Pustaka

Bab ini berisi dasar teori dari teknologi yang digunakan dalam menyelesaikan proyek kerja praktik.

1.7.4. Bab IV Analisis dan Perancangan Infrastruktur Sistem

Bab ini berisi mengenai tahap analisis sistem aplikasi dalam menyelesaikan proyek kerja praktik.

1.7.5. Bab V Implementasi Sistem

Bab ini berisi uraian tahap - tahap yang dilakukan untuk proses implementasi aplikasi.

1.7.6. Bab VI Pengujian dan Evaluasi

Bab ini berisi hasil uji coba dan evaluasi dari aplikasi yang telah dikembangkan selama pelaksanaan kerja praktik.

1.7.7. Bab VII Kesimpulan dan Saran

Bab ini berisi kesimpulan dan saran yang didapat dari proses pelaksanaan kerja praktik.

[Halaman ini sengaja dikosongkan]

BAB II
PROFIL PERUSAHAAN

2.1. Profil Perusahaan

Institut Teknologi Sepuluh Nopember atau disingkat dengan ITS merupakan Perguruan Tinggi Negeri Berbadan Hukum (PTNBH) menurut Peraturan Pemerintah No. 83 tahun 2014 yang menjadi rujukan dalam pendidikan, penelitian dan pengabdian masyarakat serta pengembangan inovasi terutama dalam bidang industri dan kelautan. ITS berdiri atas usulan dr. Angka Nitisastro dalam Lustrum I PII Jawa Timur 17 Agustus 1957 agar mendirikan Yayasan Perguruan Tinggi Teknik (YPTT) di Surabaya. ITS baru disahkan oleh Presiden Soekarno pada 10 November 1957 dengan nama Perguruan Teknik Sepuluh Nopember.

Awal mula pendirian ITS ditujukan untuk mengembangkan tenaga insinyur dan memanfaatkan kekayaan hasil alam yang belum maksimal. Sehingga pada saat itu Teknik Sipil dan Teknik Mesin menjadi departemen awal yang ada di ITS. Kemudian pada tanggal 3 November 1960 barulah nama Institut Teknologi Sepuluh Nopember disahkan dan mendapatkan status sebagai Perguruan Tinggi Negeri dari SK Menteri Pendidikan Pengajar dan Kebudayaan No. 93367/UU. Departemen ITS juga mulai bertambah menjadi lima yaitu Teknik Sipil, Teknik Mesin, Teknik Kimia, Teknik Elektro, dan Teknik Perkapalan.

Selain program sarjana S1, ITS juga memiliki program vokasi, magister, doktor, internasional, profesi insinyur, dan kuliah terapan. PTN ini juga memiliki 9 fakultas dan puluhan program studi yang kebanyakan berfokus pada bidang teknologi dan ilmu teknik. Fakultas yang ada di ITS diantaranya,

1. Fakultas Teknik Sipil, Perencanaan, dan Kebumian
2. Fakultas Desain Kreatif dan Binis Digital
3. Fakultas Vokasi
4. Fakultas Sains dan Analitika Data
5. Fakultas Teknologi Kelautan
6. Sekolah Interdisiplin Manajemen dan Teknologi
7. Fakultas Industri dan Rekayasa Sistem
8. Fakultas Teknologi Elektro dan Informatika Cerdas
9. Fakultas Kedokteran dan Kesehatan

2.2. Lokasi

Jl. Teknik Kimia, Keputih, Kec. Sukolilo, Surabaya, Jawa Timur 60111.

BAB III
TINJAUAN PUSTAKA

3.1. Kecerdasan Buatan Generatif (GenAI)

GenAI merujuk pada sistem kecerdasan buatan yang memiliki kemampuan untuk menciptakan teks, gambar, atau bentuk media lainnya melalui pemanfaatan model generatif (Pinaya dkk., t.t.). Model-model ini mempelajari pola dan struktur dalam data pelatihan mereka, kemudian menghasilkan data baru yang memiliki karakteristik serupa. GenAI mencakup berbagai jenis, yang masing-masing disesuaikan untuk tugas-tugas atau bentuk media tertentu. Beberapa jenis model yang banyak dikenal antara lain: Generative Adversarial Networks (GANs), Model Berbasis Transformer (TRMs), *Variational Autoencoders* (VAEs), dan model difusi (DMs)(Singh Sengar dkk., 2024).

3.2. Generative Adversarial Networks

Generative Adversarial Networks (GANs) adalah salah satu inovasi signifikan dalam bidang pembelajaran mesin, yang pertama kali diperkenalkan oleh Ian Goodfellow, dkk. pada tahun 2014 (Goodfellow dkk., 2020). Secara konsep, GANs terdiri dari dua *neural networks* yang saling berkompetisi dalam suatu kerangka permainan *zero-sum*. Dua jaringan ini adalah generator dan diskriminator, yang masing-masing memiliki peran yang berlawanan tetapi saling melengkapi. Pada GANs, generator bertujuan untuk menghasilkan data yang sedekat mungkin dengan data asli yang ada dalam dataset pelatihan, sedangkan diskriminator bertugas untuk membedakan antara data yang dihasilkan oleh generator dengan data asli. Secara matematis, fungsi obyektif yang digunakan dalam GANs didefinisikan sebagai berikut.

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1 - D\left(G(z)\right)\right)\right] \quad (1)$$

dengan $p_{data}(x)$ adalah distribusi data asli, sementara $p_z(\mathrm{z})$ adalah distribusi laten (biasanya distribusi normal) yang digunakan sebagai input untuk generator G. Diskriminator D(x) mengeluarkan probabilitas antara 0 dan 1 yang menunjukan kemungkinan suatu sampel x berasal dari data asli (Heusel dkk., t.t.).

Secara intuitif, generator belajar untuk menipu diskriminator, sementara diskriminator terus mengasah kemampuannya untuk mendeteksi data palsu. Proses ini berlanjut secara iteratif hingga mencapai titik keseimbangan, dimana data yang dihasilkan oleh generator tidak lagi dapat dibedakan dari data asli oleh diskriminator. Namun, mencapai titik keseimbangan ini dalam praktiknya tidak selalu mudah, dan pelatihan GANs sering kali rentan terhadap beberapa masalah seperti mode *collapse* dan ketidakstabilan(Heusel dkk., t.t.).

3.3. Translasi Gambar-ke-gambar Tak Berpasangan

Istilah ini merujuk pada tugas kompleks dalam *machine learning* yang melibatkan transformasi gambar dari satu domain ke domain yang lain tanpa menggunakan pasangan data yang sesuai antara kedua domain tersebut. Dalam konteks ini, model yang paling umum digunakan adalah *Cycle-consistent* Generative Adversarial Networks (CycleGAN) yang diperkenalkan oleh Zhu, dkk. Pada tahun 2017 (Zhu dkk., t.t.). CycleGAN menggunakan dua generator dan dua diskriminator untuk melakukan tugas ini dengan dukungan siklus konsistensi (cycle consistency), yang memastikan bahwa transformasi bolak-balik antara

kedua domain menghasilkan gambar yang mirip dengan gambar asli secara konsisten.

Secara matematis, dua generator $\mathrm{G}: \mathrm{A} \rightarrow \mathrm{B}$ dan $\mathrm{F}: \mathrm{B} \rightarrow \mathrm{A}$ masing-masing didefinisikan sebagai fungsi yang memetakan domain A ke B dan B ke A, sedangkan dua diskriminator D_B dan D_A masing-masing didefinisikan sebagai fungsi yang membedakan data asli pada domain B terhadap data sintesisnya, dan data domain A terhadap data sintesisnya. Tujuan dari generator G adalah menghasilkan gambar dalam domain B dari gambar di domain A sedemikian rupa sehingga D_B tidak dapat membedakannya dari gambar asli di domain B. Demikian pula, generator F bertujuan untuk melakukan hal yang sama dari domain B ke domain A. Kedua komponen ini bekerja di bawah sebuah fungsi obyektif:

$$\begin{aligned}\mathcal{L}_{GAN}(G, D_B, A, B) &= \mathbb{E}_{b \sim p_{data}(b)}[\log D_B(b)] \\ &+ \mathbb{E}_{a \sim p_{data}(a)}\left[\log\left(1 - D_B\big(G(a)\big)\right)\right]\end{aligned} \tag{2}$$

Fungsi ini bekerja dengan cara yang serupa seperti fungsi obyektif GAN standar, dimana G berusaha untuk menghasilkan gambar $\mathrm{G}(a)$ yang terlihat seperti gambar dalam domain B, dan D_B mencoba membedakan antara gambar yang dihasilkan $\mathrm{G}(a)$ dan gambar asli b. Siklus konsistensi secara matematis didefinisikan:

$$\begin{aligned}\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{a \sim p_{data}(a)}&\left[\left\|F\big(G(a)\big) - a\right\|_1\right] \\ &+ \mathbb{E}_{b \sim p_{data}(b)}\left[\left\|G\big(F(b)\big) - b\right\|_1\right]\end{aligned} \tag{3}$$

dengan $\|\cdot\|_1$ menunjukan norma L1 yang digunakan untuk menghitung selisih antara nilai piksel gambar asli

dengan gambar yang direkonstruksi setelah dua kali transformasi. Secara keseluruhan, fungsi obyektif total mendefinisikan optimasi parameter dari CycleGAN untuk meminimalisir nilai loss yang dihasilkan:

$$\begin{aligned}\mathcal{L}(G, F, D_A, D_B) = \mathcal{L}_{GAN}(G, D_B, A, B) \\ + \mathcal{L}_{GAN}(F, D_A, B, A) \\ + \lambda \mathcal{L}_{cyc}(G, F)\end{aligned} \quad (4)$$

dengan λ adalah hyperparameter yang menyeimbangkan pentingnya antara loss GAN dan *cycle-consistency loss* (Zhu dkk., t.t.).

Meskipun dengan kemampuan CycleGAN yang mengagumkan dalam beberapa tugas, penelitian yang dilakukan oleh Park, dkk. (Park dkk., 2020) menunjukkan bahwa siklus konsistensi mengasumsikan bahwa hubungan antara dua domain adalah bijeksi, yang sering kali terlalu membatasi. Untuk mengatasi hal tersebut, Park mengusulkan pendekatan teknik self-supervised learning dengan pembelajaran kontrastif yang dapat mengurangi kompleksitas dari CycleGAN dan juga meningkatkan performanya.

3.4. Frechet Inception Distance (FID)

Fréchet Inception Distance (FID) adalah metrik yang digunakan untuk mengevaluasi kualitas gambar yang dihasilkan oleh model generatif, seperti Generative Adversarial Networks (GANs). FID mengukur kesamaan antara distribusi fitur dari gambar yang dihasilkan dengan distribusi fitur dari gambar asli (*ground truth*). Untuk menghitung FID, pertama-tama gambar asli dan gambar yang dihasilkan dilewatkan melalui model pralatih, biasanya jaringan Inception, untuk menghasilkan representasi fitur pada lapisan tertentu.

Setelah mendapatkan representasi fitur, FID menghitung jarak antara dua distribusi Gaussian yang diaproksimasi dari fitur-fitur ini. Jarak *Fréchet* antara dua distribusi Gaussian, $\mathcal{N}(\mu_1, \Sigma_1)$ dan $\mathcal{N}(\mu_2, \Sigma_2)$, dihitung dengan rumus berikut:

$$FID = \|\mu_1 - \mu_2\|^2 + \mathrm{Tr}\left(\Sigma_1 + \Sigma_2 - 2\sqrt{\Sigma_1\Sigma_2}\right) \qquad (5)$$

dengan μ_1 dan μ_2 adalah vektor rata-rata dari fitur, dan Σ_1 dan Σ_2 adalah matriks kovarians. Nilai FID yang lebih rendah menunjukkan bahwa gambar yang dihasilkan lebih mirip dengan gambar asli, baik dalam hal kualitas visual maupun distribusi statistik. FID menjadi standar dalam evaluasi model generatif karena mampu menangkap perbedaan visual dan statistik antara gambar asli dan hasil yang dihasilkan, yang tidak dapat diukur dengan sederhana menggunakan metrik seperti Mean Squared Error atau Peak Signal-to-Noise Ratio (PSNR).

[Halaman ini sengaja dikosongkan]

BAB IV
ANALISIS DAN PERANCANGAN INFRASTRUKTUR SISTEM

4.1. Analisis Sistem

Pada bab ini akan dijelaskan mengenai tahapan dalam membangun infrastruktur model dari NijiGAN yaitu analisis dari infrastruktur model yang akan dibangun. Hal tersebut dijelaskan ke dalam dua bagian, definisi umum aplikasi dan analisis kebutuhan.

4.1.1. Definisi Umum Aplikasi

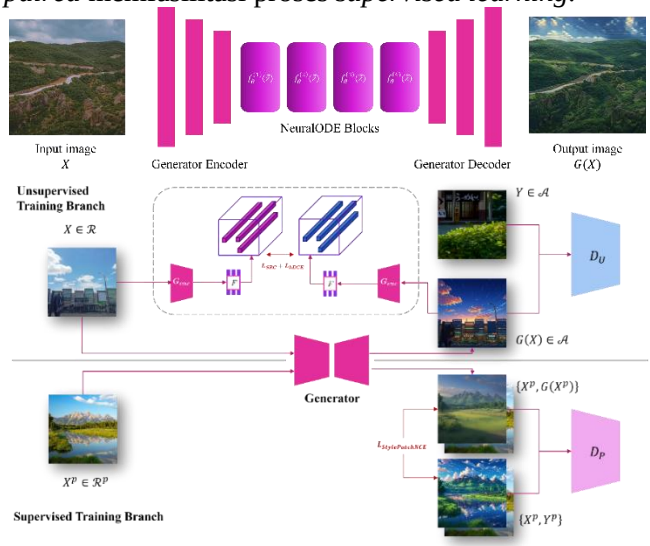
Secara umum, model NijiGAN merupakan model yang dapat mentranslasikan gambar dari domain asli menjadi domain anime. Model ini memiliki dua cabang training yaitu *supervised training* berperan dalam menggambar gaya anime menggunakan *pseudo-paired* yang dibuat dari hasil Scenemify dan *unsupervised training* yang berperan untuk mentransferkan gaya Makoto Shinkai.

4.2. Perancangan Infrastruktur Sistem

4.2.1. Desain Sistem

Desain sistem dari model NijiGAN bertujuan mengubah gambar dunia nyata dari domain input agar terlihat seperti gambar anime dari domain output. Untuk mencapai hal ini, kami mempersiapkan beberapa jenis dataset berpasangan: data *pseudo-paired* $\{(x_p, y_p) \mid x_p \in X^p, y_p \in Y^p\}$ dan data *unpaired* $\{(x, y) \mid x \in X, y \in Y\}$. Data *pseudo-paired* dihasilkan dengan menerjemahkan koleksi gambar dunia nyata menjadi anime menggunakan model Scenimefy yang telah dilatih sebelumnya, untuk memastikan konsistensi struktur dan gaya anime yang lebih baik, terutama

terkait dengan referensi gaya anime. Tujuan dari data *pseudo-paired* adalah untuk memberikan informasi tentang target dari gambar yang diterjemahkan sehingga generator dapat mengikuti karakteristik dari *pseudo-target*nya, yaitu gaya anime. Dalam hal ini, Y^p memiliki tekstur dan fitur semantik yang menyerupai anime, dan fitur-fitur ini akan digunakan dalam proses pembelajaran generator. Dengan kata lain, data *pseudo-paired* memfasilitasi proses *supervised learning*.



Gambar 5.1 Desain Arsitektur NijiGAN

Pada desain model NijiGAN, Kami mengusulkan penggunaan Neural ODEs dalam generator sebagai pengganti generator CUT, yang mengimplementasikan blok ResNet. Melalui penggunaan Neural ODEs, kami menerapkan metode dinamika kontinu selama proses pelatihan.

Sementara itu, data *unpaired* bertujuan untuk memberikan informasi tentang gaya lukisan referensi.

Dalam hal ini, lukisan oleh Makoto Shinkai diadopsi sebagai referensi utama. (Jiang dkk., 2023)

Neural ODEs memungkinkan model untuk menangkap dinamika yang kompleks dalam data gambar dengan lebih baik, yang sangat penting dalam konteks generasi gambar dan pemrosesan citra. Dengan pendekatan ini, generator dapat belajar dari pola temporal dan spasial yang ada dalam data, sehingga menghasilkan gambar yang lebih realistis dan sesuai dengan gaya yang diinginkan.

4.2.2. Neural Ordinary Differential Equations di CUT

CUT secara umum terdiri dari tiga komponen: *encoder*, *bottleneck*, dan *decoder*. *Encoder* berfungsi untuk mengekstrak fitur penting dari gambar input di domain X atau X^p. Dalam arsitektur CUT, *encoder* menggunakan serangkaian lapisan konvolusi untuk mengubah gambar input menjadi representasi laten yang lebih abstrak. Representasi ini menangkap informasi semantik dan tekstur yang esensial yang diperlukan untuk proses translasi gaya. *Bottleneck* kemudian mengambil representasi laten dari *encoder*. Dalam CUT, *bottleneck* memainkan peran penting dalam memastikan bahwa fitur yang diekstrak dari domain input X atau X^p dapat dipetakan secara efektif ke domain output Y atau Y^p sebuah representasi diskriminatif yang efektif untuk tugas translasi tanpa memerlukan data berpasangan secara langsung. *Decoder* bertanggung jawab untuk merekonstruksi gambar dari representasi laten yang diproses oleh *bottleneck*. *Decoder* mengubah fitur-fitur ini menjadi gambar keluaran di domain Y atau Y^p, dalam hal ini, gaya anime. Proses decoding harus memastikan bahwa konten asli dari gambar input tetap terjaga sementara gaya visualnya diubah sesuai dengan target.

Dalam NijiGAN, kami menerapkan Neural ODEs untuk menggantikan lapisan ResNet diskrit di bottleneck sehingga evolusi status di setiap lapisan jaringan saraf dapat berlangsung secara dinamis. Neural ODEs adalah persamaan diferensial yang memanfaatkan jaringan saraf untuk memparameterisasi ruang vektor. Secara umum, Neural ODEs didefinisikan oleh formula berikut:

$$\frac{dh}{dt}(t) = f_\theta\big(t, h(t)\big); h(0) = h_0 \tag{6}$$

di mana $f_\theta\big(t, h(t)\big)$ adalah sebuah lapisan yang terdiri dari jaringan saraf konvolusional, fungsi aktivasi, dan *batch normalization*. Formula ini diperoleh melalui pemodelan ResNet:

$$h_{t+1} = h_t + f(h_t, \theta_t), \tag{7}$$

di mana h_t mewakili status tersembunyi dari jaringan saraf pada lapisan t. Perlu dicatat bahwa status diperbarui secara diskrit dengan menambahkan output dari fungsi yang bergantung pada status tersembunyi dan parameter θ_t. Jika setiap perubahan status didefinisikan dengan variabel ukuran langkah Δt, maka persamaan di atas akan menyerupai metode Euler, sebuah teknik numerik untuk menyelesaikan persamaan diferensial:

$$h_{t+1} = h_t + \Delta h = h_t + \Delta t \cdot f(h_t, \theta_t) \tag{8}$$

Di sini, $f(h_t, \theta_t)$ adalah fungsi yang mendefinisikan bagaimana status tersembunyi berubah menggunakan parameter θ_t. Karena $\Delta h = h_{t+1} - h_t = \Delta t \cdot f(h_t, \theta_t)$, kita dapat menuliskannya sebagai

$$\frac{h_{t+1} - h_t}{\Delta t} = f(h(t), \theta_t) \quad (9)$$

Metode Euler memperkirakan solusi dari ODE dengan mendiskretkan waktu menjadi langkah-langkah kecil Δt. Jika Δt menjadi sangat kecil, pembaruan dari h_{t+1}, yang awalnya tampak diskrit karena memiliki segmen waktu diskrit $[t_0, t_1, t_2, \dots]$ dengan $t_{k+1} = t_k + \Delta t$, akan tampak kontinu, membentuk interval $[t_0, t_k]$. Dengan kata lain, batasnya mendekati nol, sehingga status tersembunyi akan memiliki dinamika fungsi yang bergantung pada waktu:

$$\lim_{\Delta t \to 0} \frac{h(t + \Delta t) - h(t)}{\Delta t} = \frac{dh}{dt}(t) \quad (10)$$

Dengan demikian, persamaan diferensial dalam persamaan 1 terbentuk, dan pemodelan lapisan dalam jaringan saraf kini didefinisikan secara kontinu. Jika ada rentang yang menggambarkan kedalaman jaringan saraf ini, katakanlah $[t_0, t_1]$, maka solusi untuk persamaan diferensial ini adalah

$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} f_\theta\big(t, h(t)\big) dt \quad (11)$$

Untuk menyelesaikan persamaan diferensial ini, teknik numerik diperlukan, misalnya metode Runge-Kutta. Namun, NijiGAN menerapkan metode Dormand-Prince (RKDP), sebuah keluarga dari metode *Runge-Kutta*.(Yan dkk., 2019) (Chen dkk., 2018) Pembelajaran *semi-supervised* diterapkan untuk memetakan gambar x dari domain dunia nyata R ke gambar y di domain anime A dengan m emanfaatkan dataset *pseudo-paired* $P =$

$\left\{x_p^{(i)}, y_p^{(i)}\right\}_{i=1}^{N}$. Secara umum, proses pelatihan ini menggunakan dua cabang: cabang *supervised learning* dan *unsupervised learning*.

Cabang *supervised* diimplementasikan untuk memungkinkan model belajar dari fitur-fitur yang diekstrak dari dataset *pseudo-paired* yang dihasilkan secara sintetis. Fitur-fitur yang diekstrak ini akan digunakan dalam proses pelatihan dan dalam pemetaan fitur untuk stylisasi adegan. Cabang *supervised* ini didasarkan pada kerangka kerja conditional GAN, yang dilatih menggunakan *loss adversarial* bersyarat sebagai berikut:

$$\mathcal{L}_{c\mathcal{GAN}}(G, D_P) = E_{y^p, x^p}[log D_P(y^p, x^p)] + E_{x^p}\left[log\left(1 - D_P\left(x^p, G(x^p)\right)\right)\right] \quad (12)$$

Di mana diskriminator *patch* D_P digunakan untuk membedakan antara $\{(y^p, x^p)\}$ dan $\{(G(x^p), x^p)\}$, dan $(\cdot,\cdot)$ menunjukkan operasi konkatenasi.

Dalam proses *supervised training* ini, alih-alih menggunakan *ground truth* dari domain target Y, kami memanfaatkan *pseudo-ground truth* Y^p. Kami menerapkan pembelajaran kontras *patch-wise* untuk membantu model belajar kesamaan lokal yang kuat dan fokus pada detail halus. Pendekatan ini memetakan patch dari gambar yang dihasilkan ke patch yang sesuai dalam *pseudo-ground truth*, memberikan supervisi yang kuat. Pemetaan ini memanfaatkan *patch* yang serupa antara gambar yang diterjemahkan dan *pseudo-ground truth*-nya, bukan bergantung pada *patch* yang identik. *Patch* yang terletak di posisi yang sama harus disematkan lebih dekat satu sama lain, sementara yang berada di lokasi berbeda harus dijaga lebih jauh.

Dengan menggunakan *loss StylePatchNCE*, kami membagi generator menjadi dua komponen, *encoder* G_{enc} dan *decoder* G_{dec}. (Jiang dkk., 2023) Fitur dari G_{enc} memfasilitasi translasi gambar dengan menyelaraskan patch yang cocok dengan gambar input. Fitur-fitur ini kemudian dimasukkan ke dalam jaringan MLP dua lapisan yang dapat dilatih *F* untuk memperoleh fitur patch tersemat. Untuk menangkap tekstur anime pada tingkat granularitas yang berbeda, kami mengekstrak fitur multi-skala dari total *L* lapisan di G_{enc}. *StylePatchNCE* didefinisikan sebagai berikut(Jiang dkk., 2023):

$$L_{StylePatchNCE}(G, F, Y^p) = \sum_{l=1}^{L} \sum_{i \neq j} L_{patch}^{style}\left(\widetilde{v_l^i}, v_l^i, v_l^j\right) \quad (13)$$

Di mana $\widetilde{v_l^i}$ dan v_l^i adalah patch tersemat lapisan ke-l pada lokasi i dari $G(x^p)$ dan y^p, masing-masing, dan L_{patch}^{style} didefinisikan sebagai berikut:

$$L_{patch}^{style} = -\log\left[\frac{\exp(v \cdot v^+)}{\exp(v \cdot v^+) + \sum_{i=1}^{N} \exp\left(v \cdot v_i^-\right)}\right] \quad (14)$$

L_{patch}^{style} memungkinkan model untuk mempertahankan struktur gambar asli selama translasi ke domain yang berbeda dengan memaksimalkan kesamaan antara pasangan data relevan $\exp\{(v \cdot v^+)$ dan meminimalkan kesamaan antara pasangan tidak relevan $\exp\{(v \cdot v_i^-)$. (Jiang dkk., 2023) Secara keseluruhan,

total *loss* untuk cabang *supervised training* dirumuskan sebagai:

$$L_{sup} = L_{cGAN}(G, D_P) + \lambda_{style} L_{StylePatchNCE}(G, F, Y^p) \quad (15)$$

di mana λ_{style} adalah bobot untuk *loss StylePatchNCE*. *Unsupervised branch* dirancang untuk mereplikasi gaya visual dari dataset anime Makoto Shinkai di domain A. Dalam cabang ini, tujuan adalah untuk mengadaptasi gambar *anime* ke dalam gaya Makoto Shinkai sambil mempertahankan struktur aslinya. Di dunia nyata, *patch* gambar dari berbagai lokasi memiliki hubungan semantik yang sangat heterogen, sehingga sulit untuk menemukan *patch* yang cocok untuk pelatihan. Namun, hubungan semantik yang beragam ini harus dipertimbangkan untuk mencapai translasi gambar yang sesuai dengan konten.(Jung dkk., 2022) Untuk mengatasi hal ini, *loss* konsistensi hubungan semantik L_{SRC} diterapkan untuk meminimalkan Jensen-Shannon Divergence (JSD) antara distribusi kesamaan *patch* di domain x dan $G(x)(Jung\ dkk., 2022)$:

$$L_{SRC} = \text{JSD}\left(G_{enc}(x) \mid G_{enc}\big(G(x)\big) \right) \quad (16)$$

Selain itu, loss kontras negatif keras L_{hDCE} diperlukan untuk secara progresif meningkatkan kemampuan mengidentifikasi negatif keras dengan merancang distribusi negatif keras dan membangun koneksi dengan hubungan kesamaan(Jung dkk., 2022):

$$L_{hDCE} = E_{z,w\sim p_{ZW}}\left[-\log\left(\frac{\exp(w^T z/\tau)}{\left(NE_{z^-\sim q_{z^-}}[\exp(w^T z^-/\tau)]\right.}\right.\right. \quad (17)$$

di mana N adalah jumlah negatif dan τ adalah parameter suhu untuk setiap pasangan positif $(z, w) \sim p_{ZW}$ dengan z menjadi vektor tersemat dari patch Z dalam gambar *anime* sintetis dan w adalah vektor tersemat dari *patch* W dalam gambar referensi gaya *anime*. Dengan demikian, total *loss* untuk cabang *unsupervised training* dirumuskan sebagai:

$$L_{unsup} = L_{GAN}(G, D_U) + \lambda_{SRC} L_{SRC} + \lambda_{hDCE} L_{hDCE} \quad (18)$$

di mana D_U adalah diskriminator untuk *unsupervised training*, λ_{SRC} adalah bobot untuk loss konsistensi hubungan semantik, dan λ_{hDCE} adalah bobot untuk *loss* kontras negatif keras.

4.2.3. Pelatihan Keseluruhan

Optimizer Adam digunakan untuk mengoptimalkan parameter NijiGAN berdasarkan fungsi *loss* L_{i2i} selama fase *backpropagation* dari pelatihan. Skema pelatihan keseluruhan terdiri dari 25 epoch, dengan masing-masing *epoch* memiliki 24.000 langkah. Sepuluh *epoch* pertama berfungsi sebagai *epoch* pemanasan. Setelah 10 *epoch* awal, bobot *supervised loss*, $\lambda_{sup}(t)$, diterapkan secara bertahap seiring dengan kemajuan *epoch*. Total *loss* untuk satu *epoch* dirumuskan sebagai:

$$L_{i2i} = L_{unsup} + \lambda_{sup} L_{sup} \quad (19)$$

Dalam rumus ini, L_{unsup} mewakili *loss* dari cabang *unsupervised*, sementara L_{sup} mewakili *loss* dari cabang *supervised*. Dengan pendekatan ini, model diharapkan dapat belajar dengan lebih baik dari kedua jenis data, memanfaatkan informasi yang diperoleh dari dataset *pseudo-paired* dan *unpaired* untuk meningkatkan kualitas hasil generasi gambar *anime*.

BAB V
IMPLEMENTASI SISTEM

Bab ini membahas tentang implementasi dari model yang kami buat. Implementasi ini akan dibagi ke dalam beberapa bagian, yaitu bagian pembuatan validasi model, *deployment* hasil model pembanding, modul Mean Opinion Score (MOS),

5.1 Validasi Model

Validasi model mencakup beberapa fungsi yang memiliki peranan penting dalam memantau kemajuan pelatihan. Fungsi-fungsi tersebut meliputi *generate_validation_images*, *save_validation_images*, *validate*, *save_checkpoint*, dan *load_checkpoint*, yang diterapkan dalam kelas trainer untuk melatih model.

Fungsi *generate_validation_images* yang tercantum pada Kode 5.1 bertujuan untuk menghasilkan gambar-gambar yang digunakan dalam visualisasi validasi selama proses pelatihan model. Fungsi ini dimulai dengan mengatur model generator (*self.gen*) ke mode evaluasi menggunakan *self.gen.eval()*, yang penting untuk memastikan bahwa model beroperasi dalam mode inferensi tanpa pembaruan parameter dan gradien. Selanjutnya, fungsi ini melakukan pengkodean gaya untuk tiga jenis gambar, yaitu *x_p* (label 1), *y_p* (label 0), dan *y* (label -1), menghasilkan representasi gaya (*s_xp*, *s_yp*, dan *s_r*). Pencampuran gaya dilakukan dengan menggabungkan *s_yp* dan *s_r* untuk menghasilkan gaya campuran (*s_yr*). Gambar palsu kemudian dihasilkan menggunakan metode generator, termasuk *fake_xp*, *fake_yp*, dan *fake_yp_mixed*, yang masing-masing menggunakan gaya yang sesuai. Selain itu, fungsi ini juga menghasilkan terjemahan *unsupervised* (*fake_y*) dari input *x* dan melakukan

rekonstruksi terhadap gambar asli dengan menghasilkan *rec_xp*, *rec_yp*, dan *rec_y*. Akhirnya, fungsi ini mengembalikan hasil dalam bentuk dictionary terstruktur, membagi hasil menjadi kategori *supervised* dan *unsupervised*, yang mencakup gambar asli, gambar palsu, rekonstruksi, serta kehilangan kuantifikasi (*qloss*) terkait. Dengan demikian, fungsi ini tidak hanya berfungsi untuk memvisualisasikan hasil selama pelatihan tetapi juga memberikan wawasan tentang efektivitas model dalam memahami dan mereproduksi berbagai gaya serta konten dari data inputnya.

```
generate_validation_images(self, x_p, y_p, x, y):
 """Generate images for validation visualization"""
 self.gen.eval()
 with torch.no_grad():
    # Generate supervised translations
    s_xp = self.gen.encode_style(x_p, label=1)
    s_yp = self.gen.encode_style(y_p, label=0)
    s_r = self.gen.encode_style(y, label=-1)
    s_yr = self.gen.style_mix(s_yp, s_r)

    fake_xp, _, _ = self.gen(y_p, label=0, cross=True,
s_given=s_xp)
    fake_yp, _, _ = self.gen(x_p, label=1, cross=True,
s_given=s_yp)
    fake_yp_mixed, _, _ = self.gen(x_p, label=1,
cross=True, s_given=s_yr)

    # Generate unsupervised translations
    fake_y, _, _ = self.gen(x, label=1, cross=True,
s_given=s_r)

    # Generate reconstructions
    rec_xp, qloss_xp, _ = self.gen(x_p, label=1,
cross=False)
```

```
    rec_yp, qloss_yp, _ = self.gen(y_p, label=0,
cross=False)
    rec_y, qloss_y, _ = self.gen(y, label=-1, cross=False)

 return {
    'supervised': {
       'x_p': x_p, 'y_p': y_p,
       'fake_xp': fake_xp, 'fake_yp': fake_yp,
       'fake_yp_mixed': fake_yp_mixed,
       'rec_xp': rec_xp, 'rec_yp': rec_yp,
       's_xp':s_xp, 's_yp':s_yp, 's_yr':s_yr, 's_r':s_r,
       'qloss_xp':qloss_xp, 'qloss_yp':qloss_yp
    },
    'unsupervised': {
       'x': x, 'y': y,
       'fake_y': fake_y,
       'rec_y': rec_y,
       'qloss_y':qloss_y
    }
 }
```

Kode 5.1 *Validation Images*

Berikutnya terdapat fungsi *save_validation_images* yang tercantum pada Kode 5.2 bertujuan untuk menyimpan gambar-gambar validasi dalam bentuk *grid*. Fungsi ini dimulai dengan membuat nama file unik berdasarkan *epoch*, *iterasi*, dan *timestamp* saat ini. Selanjutnya, fungsi ini membuat *grid* untuk dua kategori: *supervised* dan *unsupervised*. Grid *supervised* menggabungkan gambar asli, gambar palsu, dan rekonstruksi menggunakan *torch.cat*, sementara grid *unsupervised* menggabungkan gambar asli dan rekonstruksi dari data *unsupervised*. Setelah kedua grid dibuat, fungsi ini menyimpan masing-masing *grid* ke dalam direktori yang ditentukan dengan nama file yang sesuai,

menggunakan *save_image* untuk memastikan gambar dinormalisasi ke rentang [0, 1]. Dengan demikian, fungsi ini berperan penting dalam mendokumentasikan hasil validasi model selama pelatihan secara terstruktur.

```
def save_validation_images(self, images, epoch, iteration):
    """Save validation image grids"""
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f'epoch_{epoch}_iter_{iteration}_{timestamp}'

    # Create supervised grid
    supervised_grid = torch.cat([
        images['supervised']['x_p'],
        images['supervised']['y_p'],
        images['supervised']['fake_xp'],
        images['supervised']['fake_yp'],
        images['supervised']['fake_yp_mixed'],
        images['supervised']['rec_xp'],
        images['supervised']['rec_yp']
    ], dim=0)

    # Create unsupervised grid
    unsupervised_grid = torch.cat([
        images['unsupervised']['x'],
        images['unsupervised']['y'],
        images['unsupervised']['fake_y'],
        images['unsupervised']['rec_y']
    ], dim=0)

    # Save grids
    save_image(
        supervised_grid,
        os.path.join(self.image_dir, f'{filename}_supervised.png'),
```

```
            nrow=images['supervised']['x_p'].size(0),
            normalize=True
        )
        save_image(
            unsupervised_grid,
            os.path.join(self.image_dir,
f'{filename}_unsupervised.png'),
          nrow=images['unsupervised']['x'].size(0),
            normalize=True
)
```

Kode 5.2 *Save Validation Images*

Fungsi *validate* yang tercantum pada Kode 5.3 bertujuan untuk menjalankan langkah validasi jika *validation loader* tersedia. Jika tidak ada *validation loader*, fungsi ini akan mengembalikan nilai *None*. Ketika fungsi dijalankan, ia mengambil satu batch data validasi dari *validation loader* dan memindahkan data tersebut ke perangkat yang sesuai (CPU atau GPU). Selanjutnya, fungsi ini menghasilkan gambar validasi dan menghitung kerugian (*loss*) yang terkait. Gambar disimpan setiap 10 iterasi untuk mengurangi frekuensi penyimpanan. Dalam proses perhitungan kerugian, fungsi ini menggunakan mode non-gradien untuk menghitung kerugian *supervised* dan *unsupervised* dengan memanggil fungsi *calculate_supervised_loss* dan *calculate_unsupervised_loss*, masing-masing. Kerugian total dihitung dengan menggabungkan kerugian *supervised* dan *unsupervised* menggunakan faktor pengali *lambda_sup*, yang dihitung berdasarkan epoch saat ini. Jika terjadi kesalahan selama proses validasi, fungsi ini akan menangkapnya dan mencetak pesan kesalahan, serta mengembalikan nilai *None*. Maka dari itu, fungsi ini berperan penting dalam mengevaluasi kinerja model selama pelatihan dengan cara yang terstruktur dan sistematis.

```
def validate(self, epoch, iteration):
    """Run validation step if validation loader is
available"""
    if self.val_loader is None:
        return None, None, None

    try:
        # Get validation batch
        val_batch = next(iter(self.val_loader))
        x_p, y_p, x, y = val_batch.values()
        x_p, y_p = x_p.to(self.device),
y_p.to(self.device)
        x, y = x.to(self.device), y.to(self.device)

        # Generate validation images and calculate
losses
        val_images =
self.generate_validation_images(x_p, y_p, x, y)
        if iteration % 10 == 0:  # Save images less
frequently
            self.save_validation_images(val_images,
epoch, iteration)

        # Calculate validation losses
        with torch.no_grad():
            lambda_sup =
torch.cos(torch.tensor((torch.pi*(epoch -
1)/(self.epoch_end*2)))).to(self.device)

            print("val supervised calculating")
            # print("Validation Images Structure:",
val_images)
            l_supervised =
calculate_supervised_loss(
                self.gen, self.F, x_p, y_p,
```

```
                    val_images['supervised']['fake_xp'],
                    val_images['supervised']['fake_yp'],

    val_images['supervised']['fake_yp_mixed'],
                    val_images['supervised']['rec_xp'],
                    val_images['supervised']['rec_yp'],
                    val_images['supervised']['s_xp'],
                    val_images['supervised']['s_yp'],
                    val_images['supervised']['s_yr'],
                    val_images['supervised']['s_r'],
                    val_images['supervised']['qloss_xp'],
                    val_images['supervised']['qloss_yp'],
                )
                print("val unsupervised calculating")
                l_unsupervised =
    calculate_unsupervised_loss(
                    self.gen, self.F, x, y,
                    val_images['unsupervised']['fake_y'],
                    val_images['unsupervised']['rec_y'],
                    val_images['unsupervised']['qloss_y'],
                    self.n_patches, epoch
                )

                total_loss =
    calculate_total_loss(l_supervised, l_unsupervised,
    lambda_sup)

            return l_supervised.item(),
    l_unsupervised.item(), total_loss.item()

        except Exception as e:
            print(f"Validation error: {str(e)}")
            return None, None, None
```

Kode 5.3 Validate

Fungsi *save_checkpoint* yang tercantum pada Kode 5.4 bertujuan untuk menyimpan titik pemulihan (*checkpoint*) model beserta metrik pengujian tambahan jika tersedia. Fungsi ini dimulai dengan menentukan jalur penyimpanan untuk checkpoint berdasarkan direktori penyimpanan (*self.save_dir*) dan nomor *epoch* saat ini. Sebuah dictionary bernama *checkpoint* kemudian dibuat untuk menyimpan informasi penting, termasuk *epoch*, iterasi, dan status dari berbagai komponen model seperti generator, ekstraktor fitur, dan semua *optimizer* yang digunakan dalam pelatihan. Jika metrik pengujian disediakan, informasi tersebut juga akan dimasukkan ke dalam *dictionary checkpoint*. Setelah semua informasi terkumpul, fungsi ini menggunakan *torch.save* untuk menyimpan checkpoint ke dalam file dengan nama yang telah ditentukan. Terakhir, fungsi ini mencetak pesan konfirmasi yang menunjukkan bahwa *checkpoint* telah berhasil disimpan. Dapat disimpulkan, fungsi ini berperan penting dalam memastikan bahwa kemajuan pelatihan dapat dipulihkan di lain waktu, memberikan fleksibilitas dan keamanan dalam proses pengembangan model.

```
def save_checkpoint(self, epoch, iteration, metrics=None):
    """Save model checkpoint with additional testing
metrics"""
    checkpoint_path = os.path.join(self.save_dir,
f'checkpoint_epoch_{epoch}.pth')
    checkpoint = {
        'epoch': epoch,
        'iteration': iteration,
        'generator_state_dict': self.gen.state_dict(),
        'feature_extractor_state_dict': self.F.state_dict(),
        'optimizer_AE_state_dict':
self.optimizer_AE.state_dict(),
```

```
            'optimizer_F_state_dict':
self.optimizer_F.state_dict() if self.optimizer_F else None,
            'optimizer_Dp_1_state_dict':
self.optimizer_Dp_1.state_dict(),
            'optimizer_Dp_2_state_dict':
self.optimizer_Dp_2.state_dict(),
            'optimizer_Du_state_dict':
self.optimizer_Du.state_dict(),
        }

        # Include test metrics if available
        if metrics is not None:
            checkpoint['test_metrics'] = metrics

        torch.save(checkpoint, checkpoint_path)
        print(f"Checkpoint saved: {checkpoint_path}")
```

Kode 5.4 Save Checkpoint

Fungsi *load_checkpoint* yang tercantum pada Kode 5.5 bertujuan untuk memuat titik pemulihan (*checkpoint*) model dari *file* yang telah disimpan sebelumnya. Fungsi ini dimulai dengan memuat *checkpoint* menggunakan *torch.load*, yang mengembalikan semua informasi yang tersimpan, termasuk status model dan *optimizer*, ke dalam perangkat yang sesuai (CPU atau GPU). Setelah *checkpoint* dimuat, fungsi ini memperbarui status dari berbagai komponen model, seperti generator dan ekstraktor fitur, dengan menggunakan metode *load_state_dict*. Hal ini juga berlaku untuk semua *optimizer* yang digunakan dalam pelatihan; jika *optimizer* tertentu tidak tersedia dalam *checkpoint*, fungsi ini akan melewatkannya. Setelah semua status berhasil dimuat, fungsi ini mengembalikan nilai *epoch* dan iterasi terakhir yang disimpan dalam *checkpoint*. Dengan demikian, fungsi ini sangat penting untuk melanjutkan pelatihan model dari titik tertentu,

memungkinkan pengguna untuk menghemat waktu dan sumber daya dengan tidak perlu memulai pelatihan dari awal.

```
def load_checkpoint(self, checkpoint_path):
    """Load model checkpoint"""
    checkpoint = torch.load(checkpoint_path,
map_location=self.device)

    self.gen.load_state_dict(checkpoint['generator_state_
dict'])
    self.F.load_state_dict(checkpoint['feature_extractor_
state_dict'])
    self.optimizer_AE.load_state_dict(checkpoint['optim
izer_AE_state_dict'])
    if checkpoint['optimizer_F_state_dict'] is not None:
        self.optimizer_F.load_state_dict(checkpoint['optim
izer_F_state_dict'])
    self.optimizer_Dp_1.load_state_dict(checkpoint['opti
mizer_Dp_1_state_dict'])
    self.optimizer_Dp_2.load_state_dict(checkpoint['opti
mizer_Dp_2_state_dict'])
    self.optimizer_Du.load_state_dict(checkpoint['optimi
zer_Du_state_dict'])

    return checkpoint['epoch'], checkpoint['iteration']
```

Kode 5.5 Load Checkpoint

Bagian kode yang tercantum pada Kode 5.6 menunjukkan implementasi dari proses validasi dalam kelas pelatih (*trainer*). Dalam setiap iterasi pelatihan, jika iterasi saat ini memenuhi kriteria frekuensi validasi yang ditentukan (*self.validation_frequency*), fungsi *validate* akan dipanggil untuk menjalankan langkah validasi. Hasil dari validasi ini disimpan dalam variabel *val_results*. Fungsi ini memeriksa apakah semua nilai dalam *val_results* tidak bernilai *None*, yang menandakan

bahwa proses validasi berhasil dilakukan. Jika semua hasil validasi tersedia, kode ini kemudian mengekstrak kerugian *supervised*, *unsupervised*, dan total dari hasil tersebut. Selanjutnya, informasi mengenai kerugian validasi dicetak ke konsol dengan format yang rapi, menunjukkan nilai kerugian untuk masing-masing kategori. Dengan demikian, bagian kode ini berfungsi untuk memberikan umpan balik langsung tentang kinerja model selama pelatihan, memungkinkan pengembang untuk memantau kemajuan dan efektivitas model secara *real-time*.

```
#----The Rest of The Trainer Class----
# Run validation if available
                if i % self.validation_frequency == 0:
                    val_results = self.validate(epoch, i)
                    if all(v is not None for v in val_results):
                        val_supervised, val_unsupervised,
val_total = val_results
                        print(f"\nValidation Losses - Supervised:
{val_supervised:.4f}, "
                              f"Unsupervised:
{val_unsupervised:.4f}, "
                              f"Total: {val_total:.4f}")
```

Kode 5.6 Validation Implementation

5.2 Deployment Hasil Model Pembanding AnimeGAN

Pada bagian kode yang tercantum, yaitu Kode 5.7 menunjukkan langkah-langkah untuk melakukan *deployment* model pembanding, yaitu AnimeGANv3. Pertama, kode ini mengkloning repositori AnimeGANv3 dari GitHub dan mengunduh model ONNX yang diperlukan untuk inferensi. Selanjutnya, dua URL dataset dari Kaggle disiapkan untuk diunduh, yaitu dataset Shinkai dan CycleGAN. Kode ini menggunakan *od.download* untuk mengunduh kedua dataset

tersebut ke jalur penyimpanan yang ditentukan. Setelah proses pengunduhan selesai, pesan konfirmasi dicetak untuk menunjukkan bahwa dataset telah berhasil diunduh. Terakhir, kode ini menjalankan skrip Python yang melakukan inferensi menggunakan model ONNX yang telah diunduh, dengan menentukan jalur input dan output untuk gambar yang akan diproses. Dapat disimpulkan, bagian kode ini berfungsi sebagai pengantar untuk menyiapkan semua komponen yang diperlukan untuk melakukan inferensi dengan model AnimeGANv3.

```
import opendatasets as od

!git clone https://github.com/TachibanaYoshino/AnimeGANv3
!wget
https://github.com/TachibanaYoshino/AnimeGANv3/releases/d
ownload/v1.1.0/AnimeGANv3_Shinkai_37.onnx

shinkai_url = 'https://www.kaggle.com/datasets/
kevinputrasantoso/shinkai-set'

cyclegan_url = 'https://www.kaggle.com/datasets/
kevinputrasantoso/dataset-cyclegan-uji'
od.download(shinkai_url, download_path)
od.download(cyclegan_url, download_path)

print("Dataset downloaded successfully!")

!python ./AnimeGANv3/deploy/test_by_onnx.py -i
"./DATASET/dataset-cyclegan-uji/Dataset CycleGAN UJI" -
o ./DATASET/cycleanimegan -
m ./AnimeGANv3_Shinkai_37.onnx
```

Kode 5.7 Deployment model pembanding, AnimeGAN

5.3 Deployment Hasil Model Pembanding Scenimefy

Bagian kode yang tercantum pada Kode 5.8 berfungsi untuk *deployment* model pembanding, yaitu Scenimefy. Pertama, kode ini mendefinisikan URL untuk dua dataset dari Kaggle, yaitu Shinkai dan CycleGAN, serta menentukan jalur penyimpanan untuk dataset tersebut. Menggunakan pustaka *opendatasets*, kode ini mengunduh kedua dataset ke dalam direktori yang telah ditentukan. Setelah proses pengunduhan selesai, pesan konfirmasi dicetak untuk menunjukkan bahwa dataset telah berhasil diunduh.

Selanjutnya, kode ini mengunduh model *pretrained* bernama Shinkai_net_G.pth dari repositori GitHub dan menyimpannya di direktori yang sesuai. Kode kemudian berpindah ke direktori Scenimefy dan menyalin dataset CycleGAN ke lokasi yang diperlukan dalam struktur folder proyek. Terakhir, kode ini menjalankan skrip Python *test.py* dengan berbagai parameter untuk melakukan inferensi menggunakan model Scenimefy, termasuk menentukan jalur data, nama model, mode pengujian, dan direktori untuk menyimpan hasil serta checkpoint. Maka dari itu, bagian kode ini menyusun semua langkah yang diperlukan untuk menyiapkan dan menjalankan inferensi pada dataset menggunakan model yang telah dilatih sebelumnya.

```
import opendatasets as od

shinkai_url = 'https://www.kaggle.com/datasets/
kevinputrasantoso/shinkai-set'

cyclegan_url = 'https://www.kaggle.com/datasets/
kevinputrasantoso/dataset-cyclegan-uji'

download_path = './DATASET/'
```

```
# Download the dataset
od.download(shinkai_url, download_path)
od.download(cyclegan_url, download_path)

print("Dataset downloaded successfully!")

!wget https://github.com/Yuxinn-
J/Scenimefy/releases/download/
v0.1.0/Shinkai_net_G.pth -P
Semi_translation/pretrained_models
/shinkai-test/

!cd /kaggle/working/Scenimefy

!cp -r '/kaggle/working/DATASET/dataset-cyclegan-
uji/Dataset CycleGAN UJI'
/kaggle/working/Scenimefy/Semi_translation/datasets/Sample/t
estA

!python /kaggle/working/Scenimefy/Semi_translation/test.py --
dataroot
'/kaggle/working/Scenimefy/Semi_translation/datasets/Sample'
--name shinkai-test --CUT_mode CUT  --model cut --phase
test --epoch Shinkai --preprocess none --results_dir
/kaggle/working/results/cyclegan --checkpoints_dir
/kaggle/working/Scenimefy/Semi_translation/pretrained_model
s
```

Kode 5.8 Deployment model pembanding, Scenimefy

5.4 Hasil Modul Mean Opinion Score

Bagian kode yang tercantum pada Kode 5.9 berfungsi untuk mengambil gambar dari direktori tertentu dan mengembalikannya dalam bentuk respon JSON. Kode ini dimulai dengan mengekspor

fungsi *GET* yang akan dipanggil saat permintaan *GET* dilakukan. Di dalam fungsi ini, direktori dasar untuk gambar ditentukan dengan menggunakan *path.resolve* untuk menunjuk ke folder *public/images*. Fungsi *getImagesFromDirectory* didefinisikan untuk membaca nama file gambar dari direktori yang diberikan, menggabungkan jalur dasar dengan nama direktori, dan mengembalikan *array* yang berisi jalur relatif untuk setiap gambar. Kode ini kemudian memanggil fungsi tersebut untuk empat kategori gambar: *animegan*, *cartoongan*, *scenemify*, dan *nijigan*, mengumpulkan semua jalur gambar ke dalam satu *array* bernama *allImages*. Selanjutnya, kode ini menerapkan algoritma pengacakan *Fisher-Yates* melalui fungsi *shuffleArray*, yang mengacak urutan gambar dalam *array*. Setelah itu, sepuluh gambar acak dipilih dari *array* yang telah diacak menggunakan metode *slice*. Akhirnya, fungsi ini mengembalikan respons JSON yang berisi sepuluh gambar terpilih, dengan pengaturan *header* untuk mencegah *caching*.

Jika terjadi kesalahan selama proses pengambilan gambar, pesan kesalahan dicetak ke konsol, dan respons JSON dengan status 500 dikembalikan, menunjukkan bahwa pemuatan gambar gagal. Dengan demikian, bagian kode ini memberikan cara yang efisien untuk mengambil dan mengacak gambar dari berbagai kategori untuk digunakan dalam aplikasi *web*.

```
export const dynamic = "force-dynamic";

import { NextResponse } from "next/server";
import fs from "fs";
import path from "path";

export async function GET() {
  try {
    const baseDir = path.resolve("./public/images");
```

```
  // Fungsi untuk mengambil gambar dari direktori tertentu
  const getImagesFromDirectory = (directory: string): string[]
=> {
    const dirPath = path.join(baseDir, directory);
    const files = fs.readdirSync(dirPath);
    return files.map((file) => `/images/${directory}/${file}`);
  };

  // Mengambil gambar dari masing-masing kategori
  const animeganImages =
getImagesFromDirectory("animegan");
  const cartoonganImages =
getImagesFromDirectory("cartoongan");
  const scenemifyImages =
getImagesFromDirectory("scenemify");
  const nijiganImages = getImagesFromDirectory("nijigan");

  // Gabungkan semua gambar dari setiap kategori
  const allImages = [
    ...animeganImages,
    ...cartoonganImages,
    ...scenemifyImages,
    ...nijiganImages,
  ];

  // Fungsi pengacakan Fisher-Yates
  const shuffleArray = (array: string[]): string[] => {
    for (let i = array.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [array[i], array[j]] = [array[j], array[i]];
    }
    return array;
  };
```

```
    // Pengacakan seluruh gambar
    const shuffledImages = shuffleArray(allImages);

    // Pilih 10 gambar acak
    const selectedImages = shuffledImages.slice(0, 10);

    // Respon dengan array gambar yang telah dipilih
    return NextResponse.json(selectedImages, {
      headers: {
        "Cache-Control": "no-store",
      },
    });
  } catch (error) {
    console.error("Error fetching images:", error);
    return NextResponse.json(
      { error: "Failed to load images" },
      { status: 500 }
    );
  }
}
```

Kode 5.9 *Images Route*

Bagian kode yang tercantum pada Kode 5.10 berfungsi untuk mengirim data ke *Google Sheets* melalui permintaan *POST*. Kode ini dimulai dengan mendefinisikan tipe data *SheetForm*, yang mencakup nomor, nama *file*, direktori, dan *rating*. Fungsi *POST* kemudian diekspor untuk menangani permintaan yang masuk. Di dalam fungsi ini, data dari permintaan diambil dan diuraikan menjadi array objek *SheetForm*. Selanjutnya, kode ini menyiapkan autentikasi untuk *Google Sheets* menggunakan *google.auth.GoogleAuth*, di mana kredensial diambil dari variabel lingkungan yang telah disiapkan sebelumnya. Setelah autentikasi berhasil, objek *sheets* dibuat untuk berinteraksi dengan API *Google Sheets*.

Fungsi ini kemudian menggunakan metode *sheets.spreadsheets.values.append* untuk menambahkan nilai-nilai baru ke *spreadsheet* yang ditentukan oleh ID *spreadsheet* yang juga disimpan dalam variabel lingkungan. Nilai-nilai tersebut diambil dari body permintaan dan dipetakan ke dalam format yang sesuai untuk dimasukkan ke dalam *spreadsheet*. Jika operasi berhasil, fungsi ini mengembalikan respon JSON yang menunjukkan keberhasilan dan data respon dari API. Namun, jika terjadi kesalahan selama proses, pesan kesalahan dicetak ke konsol, dan respons JSON dengan status 500 dikembalikan, menunjukkan bahwa terjadi kesalahan internal pada server. Dengan demikian, bagian kode ini memberikan cara yang efisien untuk mengintegrasikan aplikasi dengan *Google Sheets* sebagai *backend* untuk menyimpan data.

```
import { NextRequest, NextResponse } from 'next/server';
import { google } from 'googleapis';

type SheetForm = {
  no: number;
  filename: string;
  directory: string;
  rating: number;
};

// Named export for POST method
export async function POST(req: NextRequest) {
  const body: SheetForm[] = await req.json(); // Extract body from the request

  try {
    // Prepare Google Sheets authentication
    const auth = new google.auth.GoogleAuth({
      credentials: {
```

```
      client_email: process.env.GOOGLE_CLIENT_EMAIL,
                                                    private_key:
process.env.GOOGLE_PRIVATE_KEY?.replace(/\\n/g, '\n'),
    },
    scopes: [
     'https://www.googleapis.com/auth/spreadsheets',
     'https://www.googleapis.com/auth/drive',
     'https://www.googleapis.com/auth/drive.file',
    ],
  });

  const sheets = google.sheets({ version: 'v4', auth });

  // Append values to the spreadsheet
  const response = await sheets.spreadsheets.values.append({
    spreadsheetId: process.env.GOOGLE_SHEET_ID,
     range: 'B:D', // Adjusted range for filename, directory, and
rating
    valueInputOption: 'USER_ENTERED',
    requestBody: {
     values: body.map((item) => [item.filename, item.directory,
item.rating]),
    },
  });

      return NextResponse.json({ success: true, response:
response.data });
 } catch (error) {
  console.error(error);
  return NextResponse.json({ error: 'Internal Server Error' }, {
status: 500 });
 }
}
```

Kode 5.10 Submit Route

Bagian kode yang tercantum pada Kode 5.11 berfungsi sebagai logika halaman *frontend* untuk aplikasi yang memungkinkan pengguna memberikan *rating* pada gambar. Kode ini dimulai dengan mendeklarasikan beberapa state menggunakan *useState*, termasuk *imageUrls* untuk menyimpan URL gambar, *ratings* untuk menyimpan rating yang diberikan pengguna, *currentIndex* untuk melacak indeks gambar saat ini, serta *loading* dan *isFinished* untuk mengelola status pemrosesan dan penyelesaian pengiriman data. Kode ini juga mengimpor *hook useToast* untuk menampilkan notifikasi kepada pengguna. Dalam efek samping (*useEffect*), fungsi *fetchImages* dipanggil untuk mengambil gambar dari API. Fungsi ini melakukan permintaan *GET* ke endpoint */api/images*, dan jika berhasil, URL gambar disimpan dalam state *imageUrls*, sementara semua rating diinisialisasi dengan nilai "1". Jika terjadi kesalahan selama pengambilan gambar, pesan kesalahan dicetak ke konsol.

Fungsi *handleRatingChange* digunakan untuk memperbarui rating gambar berdasarkan input pengguna. Fungsi *handleNext* dan *handlePrevious* memungkinkan pengguna untuk menavigasi antara gambar dengan mengubah nilai *currentIndex*. Fungsi *handleFinish* menangani pengiriman data *rating* ke *server*. Setelah mengumpulkan data dalam format yang sesuai, fungsi ini melakukan permintaan *POST* ke *endpoint /api/submit*. Jika pengiriman berhasil, notifikasi sukses ditampilkan menggunakan *toast*, dan halaman akan dimuat ulang setelah beberapa detik untuk memberikan umpan balik kepada pengguna. Jika terjadi kesalahan saat pengiriman, pesan kesalahan ditampilkan melalui *toast*. Maka dari itu, bagian kode ini menyediakan antarmuka interaktif bagi pengguna untuk memberikan rating pada gambar dan mengelola proses pengiriman data dengan efisien.

```
"use client";

export const fetchCache = "force-no-store";

import { useState, useEffect } from "react";
import { useToast } from "@/hooks/use-toast";

export default function Home() {
  const [imageUrls, setImageUrls] = useState<string[]>([]);
  const [ratings, setRatings] = useState<string[]>([]);
  const [currentIndex, setCurrentIndex] = useState(0); // Track
current card index
  const [loading, setLoading] = useState(false);
  const [isFinished, setIsFinished] = useState(false); // Track if
submission is completed
  const { toast } = useToast();

  useEffect(() => {
    const fetchImages = async () => {
      try {
        const response = await
fetch(`/api/images?ts=${Date.now()}`);

        if (!response.ok) {
          throw new Error("Failed to fetch images");
        }

        const images = await response.json();
        setImageUrls(images);
        setRatings(Array(images.length).fill("1"));
      } catch (error) {
        console.error("Error fetching images:", error);
      }
    };
```

```
    fetchImages();
  }, []);

  const handleRatingChange = (index: number, rating: string)
=> {
    const updatedRatings = [...ratings];
    updatedRatings[index] = rating;
    setRatings(updatedRatings);
  };

  const handleNext = () => {
    if (currentIndex < imageUrls.length - 1) {
      setCurrentIndex((prev) => prev + 1);
    }
  };

  const handlePrevious = () => {
    if (currentIndex > 0) {
      setCurrentIndex((prev) => prev - 1);
    }
  };

  const handleFinish = async () => {
    setLoading(true);
    const formData = imageUrls.map((url, index) => {
      const parts = url.split("/");
      const filename = parts[parts.length - 1];
      const directory = parts[parts.length - 2];

      return {
        filename,
        directory,
        rating: ratings[index],
      };
    });
```

```
    try {
      const response = await fetch("/api/submit", {
        method: "POST",
        headers: {
          Accept: "application/json",
          "Content-Type": "application/json",
        },
        body: JSON.stringify(formData),
      });

      if (response.ok) {
        toast({
          className: "bg-primary text-white",
          title: "Data Submitted! This window will reload in a
moment.",
          description: "Your data has been successfully sent!
You're all done!",
        });
        setIsFinished(true); // Mark as finished after successful
submission

        // Reload the page after a delay to allow toast display
        setTimeout(() => window.location.reload(), 4000);
      } else {
        toast({
          className: "bg-red-600 text-white",
          title: "Submission Error",
          description: "An error occurred during submission.",
        });
      }
    } catch (error) {
      console.error("Error submitting form:", error);
      toast({
        className: "bg-red-600 text-white",
```

```
        title: "Network Error",
        description: "An error occurred while submitting your
data.",
      });
    } finally {
      setLoading(false);
    }
  };

  // The rest of the UI will be handled in the appropriate
component.
}
```

Kode 5.11 Frontend page logic

5.5 Hasil FID Model Pembanding dan NijiGAN

Bagian kode yang tercantum pada Kode 5.12 bertujuan untuk menghitung nilai FID antara hasil model NijiGAN dan dataset Shinkai, serta membandingkannya dengan model pembanding seperti AnimeGAN dan Scenemify. Proses dimulai dengan mendefinisikan direktori untuk gambar asli dari dataset Shinkai, gambar nyata dari dataset CycleGAN, dan gambar hasil generasi dari NijiGAN. Kode ini kemudian memuat model Inception v3 yang telah dilatih sebelumnya, menghapus lapisan klasifikasi, dan mengatur model untuk beroperasi dalam mode evaluasi. Fungsi *preprocess_image* disiapkan untuk melakukan pra-pemrosesan pada gambar dengan mengubah ukuran menjadi 299x299 piksel ukuran input yang diperlukan oleh Inception v3 dan menormalkan nilai piksel. Fungsi *extract_features* digunakan untuk mengekstrak fitur dari gambar yang telah diproses, mengumpulkan hasilnya dalam array numpy.

Selanjutnya, fungsi *calculate_stats* menghitung rata-rata dan kovarians dari fitur yang diekstrak, yang diperlukan untuk perhitungan FID. Fungsi *calculate_fid* menghitung nilai FID berdasarkan statistik rata-rata dan kovarians dari gambar nyata dan yang dihasilkan. Nilai FID dihitung dengan menggunakan rumus yang melibatkan perbedaan antara dua rata-rata dan jejak dari produk kovarians. Setelah fitur diekstrak dari gambar hasil generasi NijiGAN dan gambar asli *Shinkai*, fungsi *calculate_fid* dipanggil untuk menghitung nilai FID antara kedua set fitur tersebut. Hasilnya dicetak ke konsol, memberikan gambaran tentang seberapa mirip hasil generasi model NijiGAN dengan dataset Shinkai. Nilai FID yang lebih rendah menunjukkan bahwa gambar yang dihasilkan lebih mirip dengan gambar nyata, sehingga memberikan indikasi kualitas dari model generatif yang digunakan.

```
import torch
from torchvision.models import inception_v3
from torchvision import transforms
from PIL import Image
import numpy as np
from scipy.linalg import sqrtm

shinkai_dir = "/kaggle/working/DATASET/shinkai-set/day"
real_dir = "/kaggle/working/DATASET/dataset-cyclegan-
uji/Dataset CycleGAN UJI"
trans_dir = "/kaggle/working/results/cyclegan/shinkai-
test/test_Shinkai/images/fake_B"
```

```
shinkai_items = os.listdir(shinkai_dir)
real_items = os.listdir(real_dir)[:len(shinkai_items)]
trans_items = os.listdir(trans_dir)[:len(shinkai_items)]

# Load Inception v3 model
model = inception_v3(pretrained=True,
transform_input=False)
model.fc = torch.nn.Identity()  # Remove the classification
layer
model.to("cuda")
model.eval()

# Preprocessing function for images
def preprocess_image(image_path):
    transform = transforms.Compose([
        transforms.Resize((299, 299)),  # Inception v3 input size
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225])
    ])
    image = Image.open(image_path).convert('RGB')
    return transform(image).unsqueeze(0)

# Extract features
def extract_features(image_paths):
    features = []
    with torch.no_grad():
        for path in image_paths:
            image = preprocess_image(path)
            feature =
model(image.to("cuda")).squeeze().cpu().numpy()
            features.append(feature)
    return np.array(features)
```

```
# Calculate mean and covariance
def calculate_stats(features):
    mean = np.mean(features, axis=0)
    covariance = np.cov(features, rowvar=False)
    return mean, covariance

# FID calculation
def calculate_fid(real_features, generated_features):
    mu1, sigma1 = calculate_stats(real_features)
    mu2, sigma2 = calculate_stats(generated_features)
    diff = mu1 - mu2

    # Compute sqrt of product of covariances
    covmean, _ = sqrtm(sigma1.dot(sigma2), disp=False)
    if np.iscomplexobj(covmean):
        covmean = covmean.real

    fid = np.sum(diff2) + np.trace(sigma1 + sigma2 - 2 *
covmean)
    return fid

# Extract features
generated_features = extract_features(trans_items)
shinkai_features = extract_features(shinkai_items)

# Calculate FID
fid_score_trans_shinkai = calculate_fid(generated_features,
shinkai_features)
print(f"FID NijiGAN-Shinkai: {fid_score_trans_shinkai}")
```

Kode 5.12 Perhitungan FID pada hasil model pembanding dan NijiGAN

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini menjelaskan tahap uji coba terhadap model AI NijiGAN. Pengujian dilakukan untuk mengevaluasi performa dalam translasi gambar ke gambar yang dilakukan oleh model AI NijiGAN.

6.1. Tujuan Pengujian

Pengujian dilakukan terhadap model AI NijiGAN guna mengevaluasi kemampuan dalam translasi gambar ke gambar yang dilakukan oleh model AI NijiGAN.

6.2. Kriteria Pengujian

Penilaian atas pencapaian tujuan pengujian didapatkan dengan memperhatikan beberapa hasil yang diharapkan berikut :

1. FID Score : Model AI NijiGAN menghasilkan nilai Frechet Inception Distance (FID) yang rendah, menandakan kualitas gambar yang dihasilkan lebih mirip dengan gambar referensi (gambar anime).
2. MOS Score : Model AI NijiGAN memperoleh nilai Mean Opinion Score (MOS) yang tinggi, menunjukkan bahwa gambar hasil translasi dinilai menarik, realistis, dan sesuai dengan ekspektasi manusia.
3. Model parameter dan memori penggunaan : Model AI NijiGAN menggunakan parameter dengan jumlah sedikit dan juga memakan memori yang rendah menujukkan Model AI NijiGAN relatif lebih ringan dengan kualitas gambar yang baik.

6.3. Skenario Pengujian

Skenario pengujian dilakukan melalui beberapa langkah sebagai berikut:

I. Pengukuran Frechet Inception Distance (FID):
   1. Jalankan model NijiGAN dan model pembanding menggunakan dataset uji.
   2. Hitung FID dari gambar yang telah ditranslasi oleh model NijiGAN dan model pembanding. Perhitungan ini dilakukan untuk menghitung kesamaan distribusi fitur antara gambar yang ditranslasi dengan gambar target (gambar anime).
   3. Catat nilai FID yang dihasilkan.

II. Survey Mean Opinion Score (MOS):
   1. Jalankan model NijiGAN dan model pembanding menggunakan dataset uji.
   2. Sebarkan survei kepada 30 responden dengan kriteria berikut:
      a. Responden diminta menilai gambar berdasarkan preferensi responden.
      b. Skala penilaian antara 1 (buruk) hingga 5 (sangat baik).
   3. Kumpulkan dan hitung rata-rata skor MOS berdasarkan hasil survei.

III. Model parameter dan memori
   1. Lakukan pelatihan model NijiGAN dengan dataset latih.
   2. Catat penggunaan parameter dan total penggunaan memori pada saat proses pelatihan sudah selesai.

6.4. Evaluasi Pengujian

Hasil pengujian dilakukan terhadap skor FID dan MOS. Tabel 6.1 di bawah ini menjelaskan evaluasi metrik dari hasil uji coba terhadap model AI NijiGAN. Tabel 6.2 menampilkan perbandingan total parameter dan memori yang digunakan oleh model AI NijiGAN dan model pembanding Scenimefy.

Tabel 6.1 Hasil FID dan MOS model AI NijiGAN dan model pembanding

Metode	AnimeGAN	Scenimefy	**NijiGAN**
FID	56.88	60.32	**58.71**
MOS	2.160	2.232	**2.192**

Tabel 6.2 Total memori dan parameter yang digunakan oleh model AI NijiGAN dan model pembanding Scenimefy

Model	Total Memori (GB)	Total Parameter (Juta)
NijiGAN	9.694	5.477
Scenimefy	9.718	11.378

6.5 Hasil dan Pembahasan

Kami juga melakukan perbandingan secara kualitatif terhadap gambar yang dihasilkan oleh model AI kami, NijiGAN, dengan beberapa model pembanding yakni AnimeGAN dan Scenimefy. Berikut adalah komparasi gambar yang telah ditranslasi model NijiGAN dan model pembandingnya.

Gambar 6.1 Hasil gambar yang telah ditranslasi oleh model AI NijiGAN dan beberapa model pembanding.

Pada Gambar 6.1 terlihat bahwa model AI NijiGAN sudah bisa mentranslasi gambar dunia nyata ke gambar domain anime dengan hasil yang kompetitif dengan model pembandingnya yakni AnimeGAN dan Scenimefy. Melalui beberapa sampel, NijiGAN telah berhasil mencapai representasi konten yang seimbang antara gambar asli dengan gambar yang telah ditranslasi. NijiGAN berhasil menghasilkan gambar dengan struktur yang halus dan menyerupai lukisan anime.

Sebaliknya, AnimeGAN menggunakan *anime-specific loss* yang dibuat secara manual, seperti *edge-smoothing loss*, untuk mencoba menciptakan tepi yang tajam, yang pada akhirnya membatasi tingkat transfer gaya. Selain itu, AnimeGAN menghasilkan output yang lebih menyerupai abstraksi tekstur

dengan tingkat stilisasi yang lemah, hampir tidak menampilkan karakteristik anime.

Sementara itu, Scenimefy berhasil melakukan transfer gaya yang efektif pada beberapa sampel, tetapi menampilkan artefak yang mencolok, termasuk konversi siang ke malam yang salah dan elemen tekstur yang menyebabkan gambar menjadi kabur di beberapa area. Di sisi lain, NijiGAN secara konsisten memberikan hasil yang stabil dan mulus.

Gambar 6.2 Perbandingan kualitas gambar yang dihasilkan oleh NijiGAN dan Scenimefy.

Selain itu, NijiGAN berhasil mengoptimalkan penggunaan memori dengan implementasi NeuralODE, yang menggunakan dinamika kontinu selama proses pelatihan. Penggunaan NeuralODE sebagai pengganti ResNet terbukti menjadi solusi efektif untuk mengoptimalkan jumlah parameter dalam model sambil tetap mempertahankan kualitas tinggi dalam translasi gambar. Hal ini terlihat dari jumlah parameter NijiGAN yang

hanya mencapai 5,4 juta dengan kebutuhan penyimpanan sebesar 20 MB, namun tetap mampu menghasilkan kualitas gambar yang kompetitif dibandingkan dengan Scenimefy.

Lebih lanjut, NijiGAN secara efektif mengurangi kemunculan artefak pada gambar yang dihasilkan. Seperti yang ditunjukkan pada Gambar 5, NijiGAN berhasil menghilangkan artefak berbentuk kotak-kotak (*checkered artifact*), sehingga menghasilkan gambar yang sangat halus.

BAB VII
KESIMPULAN DAN SARAN

7.1. Kesimpulan

Kesimpulan yang didapat setelah melakukan pengembangan model AI NijiGAN antara lain adalah:

i. Model NijiGAN untuk melakukan translasi gambar nyata ke gambar anime berhasil dikembangkan dengan memperbaiki keterbatasan model acuan, yaitu Scenimefy, melalui penerapan Neural Ordinary Differential Equations (NeuralODEs).

ii. Model NijiGAN mampu melakukan translasi gambar dengan jumlah parameter yang lebih kecil dan efisiensi yang lebih tinggi dibandingkan dengan Scenimefy.

iii. NijiGAN juga dapat mengurangi ketergantungan pada data pasangan (*paired*) berkualitas rendah dengan memanfaatkan data *pseudo-paired* untuk pelatihan, sehingga menghasilkan kualitas translasi yang lebih baik

7.2. Saran

Berdasarkan kesimpulan sebelumnya, harapan kami semoga model NijiGAN yang telah kami kembangkan dapat berguna dan bermanfaat ITS. Dengan seiring berkembangnya teknologi model ini dapat dikembangkan dengan lebih baik lagi oleh para peneliti baik dari ITS maupun luar ITS. Kami juga berharap dan menerima masukkan maupun saran yang membangun

mengenai kinerja kami selama program kerja praktik ini sebagai salah satu cara untuk perbaikan kami kedepannya.

DAFTAR PUSTAKA

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2018). *Neural Ordinary Differential Equations*. http://arxiv.org/abs/1806.07366

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144. https://doi.org/10.1145/3422622

Heusel, M., Ramsauer, H., … T. U.-… neural information, & 2017, undefined. (t.t.). Gans trained by a two time-scale update rule converge to a local nash equilibrium. *proceedings.neurips.cc*. Diambil 20 Desember 2024, dari https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html

Jiang, Y., Jiang, L., Yang, S., & Loy, C. C. (2023). *Scenimefy: Learning to Craft Anime Scene via Semi-Supervised Image-to-Image Translation*. http://arxiv.org/abs/2308.12968

Jung, C., Kwon, G., & Ye, J. C. (2022). *Exploring Patch-wise Semantic Relation for Contrastive Learning in Image-to-Image Translation Tasks*. http://arxiv.org/abs/2203.01532

Park, T., Efros, A. A., Zhang, R., & Zhu, J. Y. (2020). Contrastive Learning for Unpaired Image-to-Image Translation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12354 LNCS*, 319–345. https://doi.org/10.1007/978-3-030-58545-7_19

Pinaya, W., Graham, M., … E. K. preprint arXiv, & 2023, undefined. (t.t.). Generative ai for medical imaging: extending the monai framework. *arxiv.org*. Diambil 20 Desember 2024, dari https://arxiv.org/abs/2307.15208

Singh Sengar, S., Affan, ·, Hasan, B., Kumar, S., Carroll, F., & Hasan, A. Bin. (2024). Generative artificial intelligence: a systematic review and applications. *Springer*. https://doi.org/10.1007/s11042-024-20016-1

Yan, H., Du, J., Tan, V. Y. F., & Feng, J. (2019). *On Robustness of Neural Ordinary Differential Equations*. http://arxiv.org/abs/1910.05513

Zhu, J., Park, T., Isola, P., IEEE, A. E.-P. of the, & 2017, undefined. (t.t.). Unpaired image-to-image translation using cycle-consistent adversarial networks. *openaccess.thecvf.com*. Diambil 20 Desember 2024, dari http://openaccess.thecvf.com/content_iccv_2017/html/Zhu_Unpaired_Image-To-Image_Translation_ICCV_2017_paper.html

BIODATA PENULIS I

Nama	: Farah Dhia Fadhila
Tempat, Tanggal Lahir	: Jakarta, 19 Mei 2003
Jenis Kelamin	: Perempuan
Telepon	: +6287870091903
Email	: 5025211030@student.its.ac.id

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC , ITS
Angkatan	: 2021
Semester	: 7 (Tujuh)

BIODATA PENULIS II

Nama	: Dimas Prihady Setyawan
Tempat, Tanggal Lahir	: Jakarta, 30 Agustus 2003
Jenis Kelamin	: Laki-laki
Telepon	: +6281317491256
Email	: 5025211184@student.its.ac.id

AKADEMIS

Kuliah	: Departemen Teknik Informatika – FTEIC, ITS
Angkatan	: 2021
Semester	: 7 (Tujuh)