KERJA PRAKTIK - IF184801

Pengembangan NijiGAN: Mengubah Apa yang Anda Lihat Menjadi Anime dengan Pembelajaran Semi-Terawasi Kontrasif dan Persamaan Diferensial Biasa Neural

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

Oleh:

Adam Haidar Azizi 5025211114

Pembimbing Jurusan
Hadziq Fabroyir, S.Kom., Ph.D.
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 - IF184801

Pengembangan NijiGAN: Mengubah Apa yang Anda Lihat Menjadi Anime dengan Pembelajaran Semi-Terawasi Kontrasif dan Persamaan Diferensial Biasa Neural

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

Oleh:

Adam Haidar Azizi 5025211114

Pembimbing Jurusan
Hadziq Fabroyir, S.Kom., Ph.D.
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

DAFTAR ISI iv

DAFTAR GAMBAR viii

DAFTAR TABEL x

LEMBAR PENGESAHAN xii

KATA PENGANTAR xvii

BAB I PENDAHULUAN 1

1.1. Latar Belakang 1

1.2. Tujuan 3

1.3. Manfaat 3

1.4. Rumusan Masalah 3

1.5. Lokasi dan Waktu Kerja Praktik 4

1.6. Metodologi Kerja Praktik 4

1.6.1. Perumusan Masalah 4

1.6.2. Studi Literatur 4

1.6.3. Analisis dan Perancangan Sistem 4

1.6.4. Implementasi Sistem 5

1.6.5. Penguji dan Evaluasi 5

1.6.6. Kesimpulan dan Saran 5

1.7. Sistematika Laporan 5

1.7.1. Bab I Pendahuluan 5

1.7.2. Bab II Profil Perusahaan 5

1.7.3. Bab III Tinjauan Pustaka 5

1.7.4. Bab IV Analisis dan Perancangan Infrastruktur Sistem 5

1.7.5. Bab V Implementasi Sistem 6

1.7.6. Bab VI Pengujian dan Evaluasi 6

1.7.7. Bab VII Kesimpulan dan Saran 6

1.7.8. Bab VI Pengujian dan Evaluasi 6

1.7.9. Bab VII Kesimpulan dan Saran 6

BAB II PROFIL PERUSAHAAN 8

2.1. Profil Perusahaan 8

2.2. Lokasi 9

BAB III TINJAUAN PUSTAKA 11

3.1. Scenemify 11

3.3. Constrative Learning 12

3.4. Frechet Inception Distance (FID) 13

BAB IV ANALISIS DAN PERANCANGAN INFRASTRUKTUR SISTEM 15

4.1. Analisis Sistem 15

4.1.1. Definisi Umum Aplikasi 15

4.2. Perancangan Infrastruktur Sistem 15

4.2.1. Persamaan NeuralODE Pada CUT 17

4.2.2. Supervised Learning 19

4.2.3. Unsupervised Learning 21

4.2.4. Proses Training NijiGAN 23

BAB V IMPLEMENTASI SISTEM 25
5.1. Implementasi Modul Pada Model 25
5.1.1. StyleEncoder 26
5.1.3. Lpips 30
5.1.4. VQI2I_AdaIN 35
5.2. Modul Testing NijiGAN 41
BAB VI PENGUJIAN DAN EVALUASI 50
6.1. Tujuan Pengujian dan Evaluasi Model 50
6.2. Metrik Evaluasi Model 50
6.3. Evaluasi Kuantitatif 51
6.4. Evaluasi Kualitatif 53
6.5. Studi Ablasi 54
BAB VII KESIMPULAN DAN SARAN 56
7.1. Kesimpulan 56
7.2. Saran 56
DAFTAR PUSTAKA 58
BIODATA PENULIS I 60

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

Gambar 4. 1 Arsitektur NijiGAN 16
Gambar 4. 2 Fase Supervised Learning 20
Gambar 4. 3 Fase Unsupervised Learning 21
Gambar 6. 1 Perbandingan kualitas gambar antara NijiGAN dan beberapa model dasar. 53
Gambar 6. 2 Studi Ablasi Terhadap Pengunaan Komponen NijiGAN 54

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

Tabel 5. 1 Implementasi StyleEncoder 27
Tabel 5. 2 Implementasi ContentEncoder..... 30
Tabel 5. 3 Konstruktor Utama LPIPS 32
Tabel 5. 4 Implementasi Fungsi make_layers..... 33
Tabel 5. 5 Implementasi Fungsi Propagasi Maju LPIPS 35
Tabel 5. 6 Implementasi VQI2I_adaIN..... 41
Tabel 5. 7 Modul Testing NijiGAN 44
Tabel 5. 8 Deployment CartoonGAN 48

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN
KERJA PRAKTIK

Pengembangan NijiGAN: Mengubah Apa yang Anda Lihat Menjadi Anime dengan Pembelajaran Semi-Terawasi Kontrasif dan Persamaan Diferensial Biasa Neural

Oleh:

Adam Haidar Azizi 5025211114

Disetujui oleh Pembimbing Kerja Praktik:

1. Hadziq Fabroyir, S.Kom., Ph.D.
   NIP. 198602272019031006

(Pembimbing Departemen)

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

(Pembimbing Lapangan)

[Halaman ini sengaja dikosongkan]

Pengembangan NijiGAN: Mengubah Apa yang Anda Lihat Menjadi Anime dengan Pembelajaran Semi-Terawasi Kontrasif dan Persamaan Diferensial Biasa Neural

Nama Mahasiswa	: Adam Haidar Azizi
NRP	: 5025211114
Departemen	: Teknik Informatika FTEIC-ITS
Pembimbing Departemen	: Hadziq Fabroyir, S.Kom., Ph.D.
Pembimbing Lapangan	: Dr. Anny Yuniarti, S.Kom.,M.Kom

ABSTRAK

Kecerdasan Buatan Generatif (Generative AI) telah mengubah industri animasi. Beberapa model telah dikembangkan untuk terjemahan gambar-ke-gambar, khususnya yang berfokus pada mengubah gambar dunia nyata menjadi anime melalui terjemahan tanpa pasangan (unpaired translation). Scenimefy, sebuah pendekatan penting yang memanfaatkan pembelajaran kontras (contrastive learning), berhasil mencapai terjemahan adegan anime dengan fidelitas tinggi dengan mengatasi keterbatasan data berpasangan melalui pelatihan semi-supervised. Namun, Scenimefy menghadapi keterbatasan karena bergantung pada data berpasangan dari StyleGAN yang telah disesuaikan dalam domain anime, yang sering kali menghasilkan dataset berkualitas rendah. Selain itu, arsitektur Scenimefy yang memiliki banyak parameter memberikan peluang untuk optimisasi komputasi. Penelitian ini memperkenalkan NijiGAN, sebuah model baru yang mengintegrasikan Neural Ordinary Differential Equations (NeuralODEs), yang menawarkan keuntungan unik dalam

pemodelan transformasi kontinu dibandingkan dengan jaringan residual tradisional. NijiGAN berhasil mengubah adegan dunia nyata menjadi visual anime dengan fidelitas tinggi menggunakan setengah dari parameter Scenimefy. Model ini menggunakan data pseudo-paired yang dihasilkan melalui Scenimefy untuk pelatihan terawasi, menghilangkan ketergantungan pada data berpasangan berkualitas rendah dan meningkatkan proses pelatihan.

Kata Kunci : Tranlasi Gambar-ke-Gambar, NeuralODEs , Semi-Supervised Learning, Kecerdasan Buatan Generatif

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji dan syukur penulis panjatkan kepada Allah SWT atas limpahan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan salah satu kewajiban sebagai mahasiswa Departemen Teknik Informatika ITS, yaitu Kerja Praktik dengan judul: *Pengembangan NijiGAN: Mengubah Apa yang Anda Lihat Menjadi Anime dengan Pembelajaran Semi-Terawasi Kontrasif dan Persamaan Diferensial Biasa Neural*

.

Penulis menyadari bahwa terdapat banyak kekurangan baik dalam pelaksanaan kerja praktik maupun dalam penyusunan laporan ini. Meski demikian, penulis berharap laporan ini dapat memberikan wawasan tambahan bagi pembaca dan menjadi sumber referensi yang bermanfaat.

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 Hadziq Fabroyir, S.Kom., Ph.D. selaku dosen pembimbing kerja praktik sekaligus koordinator kerja praktik
4. Ibu Dr. Anny Yuniarti, S.Kom. selaku pembimbing lapangan selama kerja praktik berlangsung.
5. Teman-teman yang telah terlibat dalam pengerjaan dan pengembangan model NijiGAN dalam KP ini.

Surabaya, 20 Desember 2024

Adam Haidar Azizi

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1. Latar Belakang

Kemajuan pesat dalam Kecerdasan Buatan Generatif (GenAI) telah mendorong berbagai terobosan di industri kreatif. Di antara berbagai model generatif, *Generative Adversarial Networks* (GANs) menonjol sebagai salah satu pendekatan yang paling berpengaruh, sehingga memungkinkan sistem untuk menghasilkan data baru yang mendekati distribusi data pelatihan. Implementasi GANs telah merambah berbagai domain, termasuk sintesis gambar, musik, dan teks, dengan hasil yang secara substansial menyerupai kreasi manusia. Meskipun demikian, penerapan GANs pada tugas-tugas spesifik, seperti *style transfer* antar domain yang berbeda secara visual, masih menghadapi tantangan signifikan. Salah satu tantangan yang paling kompleks adalah *unpaired image-to-image translation* di mana model harus belajar mentransfer gaya antar domain tanpa pasangan data yang eksplisit. Transfer gaya dari gambar pemandangan nyata ke gaya anime adalah contoh nyata dari tantangan ini, mengingat perbedaan struktural dan tekstural yang substansial antara kedua domain tersebut .

Pendekatan terkini seperti CycleGAN mencoba mengatasi tantangan ini melalui mekanisme siklus konsistensi, yang memastikan bahwa gambar yang ditransformasikan dapat dikembalikan ke domain asalnya tanpa kehilangan informasi penting. Namun, pendekatan ini memiliki

keterbatasan, terutama karena asumsi bijeksi yang sering kali tidak sesuai dengan kenyataan. Selain itu, ketersediaan dataset berkualitas tinggi untuk pelatihan model sering kali menjadi kendala utama, menghasilkan *output* yang suboptimal dari segi kualitas visual dan konsistensi gaya. Untuk menjawab tantangan-tantangan tersebut, Jiang et al. memperkenalkan Scenimefy , model yang dirancang untuk mentransfer gaya anime ke gambar pemandangan nyata dengan lebih akurat. Dengan memanfaatkan pendekatan *semi-supervised learning*, Scenimefy menggabungkan beberapa fase dalam implementasinya, seperti sintesis data pasangan semu, segmentasi semantik untuk seleksi data, dan pelatihan *semi-supervised*. Meskipun demikian, Scenimefy masih menghadapi kesulitan dalam mengurangi artefak visual dan kesenjangan domain antara gambar nyata dan anime.

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 2 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 dan Bu Anny Yuniarti. Pada saat rapat kami mendiskusikan bagaimana arsitektur dari NijiGAN akan dibangun berupa modul-modul dari *unsupervised training* dan *supervised 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 *unsupervised training* dan *supervised training* dari NijiGAN bekerja.

1.6.3. Analisis dan Perancangan Sistem

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

1.6.4. Implementasi Sistem

Implementasi model NijiGAN 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.

1.7.8. Bab VI Pengujian dan Evaluasi

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

1.7.9. 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.

[Halaman ini sengaja dikosongkan]

BAB III
TINJAUAN PUSTAKA

3.1. Scenemify

Scenimefy mengatasi tantangan dalam mengubah gambar dunia nyata menjadi gambar bergaya anime dengan menangani kompleksitas unik gaya anime, keterbatasan dataset berkualitas tinggi, dan kesenjangan besar antara visual dunia nyata dan animasi. Untuk mengatasi masalah ini, *Scenimefy* menggabungkan *supervised learning* dan *unsupervised learning*, yang membantu model tetap konsisten dengan memanfaatkan data *pseudo-paired* dan *unpaired* yang mengandung referensi gaya anime. Prosesnya melibatkan pembuatan data *pseudo-paired*, penyaringan data tersebut berdasarkan segmentasi semantik (*semantic segmentation*), dan pelatihan semi-terawasi (*semi-supervised training*). Data *pseudo-paired* berfungsi untuk membantu pelatihan terawasi, sehingga model dapat mempelajari cara mempertahankan detail penting selama proses translasi. Segmentasi semantik digunakan untuk menyaring gambar berkualitas rendah yang tidak sesuai dengan struktur yang dibutuhkan. Pendekatan ini memungkinkan model untuk memahami baik struktur gambar dunia nyata maupun gaya anime yang khas. Pada tahap awal, model lebih bergantung pada pelatihan yang dipandu (*guided training*), tetapi seiring waktu, model secara bertahap mampu menangani tugas tersebut dengan lebih sedikit pengawasan [1].

3.2. Neural Ordinary Differential Equations

NeuralODEs memandang setiap lapisan jaringan saraf sebagai transformasi yang berjalan secara kontinu, bukan sebagai serangkaian lapisan diskrit seperti pada *ResNets*. Pendekatan ini tidak hanya mengurangi kompleksitas model tetapi juga, dalam beberapa kasus,

menghasilkan keluaran yang lebih halus dan konsisten. Dengan memahami dinamika yang berjalan secara kontinu, *NeuralODEs* memungkinkan model untuk melakukan generalisasi dengan lebih baik, terutama pada tugas seperti transformasi gambar, di mana kehalusan dan detail yang presisi sangat penting [2]. Pada *NijiGAN*, *NeuralODEs* digunakan untuk menggantikan bagian bottleneck *ResNet* pada *Scenimefy* yang mengadopsi model *CUT* tradisional. Hal ini bertujuan untuk menyederhanakan model tanpa mengorbankan kualitas maupun detail, terutama saat menerjemahkan adegan yang kompleks dari lanskap dunia nyata menjadi gambar bergaya anime.

3.3. Constrative Learning

Contrastive learning telah menjadi metode yang populer untuk mencapai transfer gaya tanpa memerlukan dataset berpasangan [3][4][5]. Metode seperti *CUT* (*Contrastive Unpaired Translation*) memanfaatkan *contrastive learning* untuk mencocokkan potongan-potongan (*patches*) dari gambar yang telah diterjemahkan dengan domain aslinya, sambil memaksimalkan perbedaan dengan gambar lainnya. Pendekatan ini menjaga konsistensi antara domain yang tidak berpasangan. *CUT* telah menunjukkan hasil yang mengesankan dalam tugas-tugas *image-to-image translation*, dengan kemampuan mempertahankan fitur utama sambil mengadaptasi gaya [1]. NijiGAN mengembangkan pendekatan ini lebih lanjut dengan mengintegrasikan fungsi *contrastive loss* untuk meningkatkan konsistensi gaya anime, memastikan bahwa gambar yang diterjemahkan tetap mempertahankan detail dan nuansa khas dari referensi gaya anime. Dengan menyempurnakan teknik-teknik ini dalam kerangka kerja semi-terawasi (*semi-supervised framework*), NijiGAN

bertujuan untuk menjembatani kesenjangan antara domain gambar dunia nyata dan anime, sekaligus meminimalkan artefak pada hasil akhir.

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 *pretrained*, 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:

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

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 detail arsitektur.

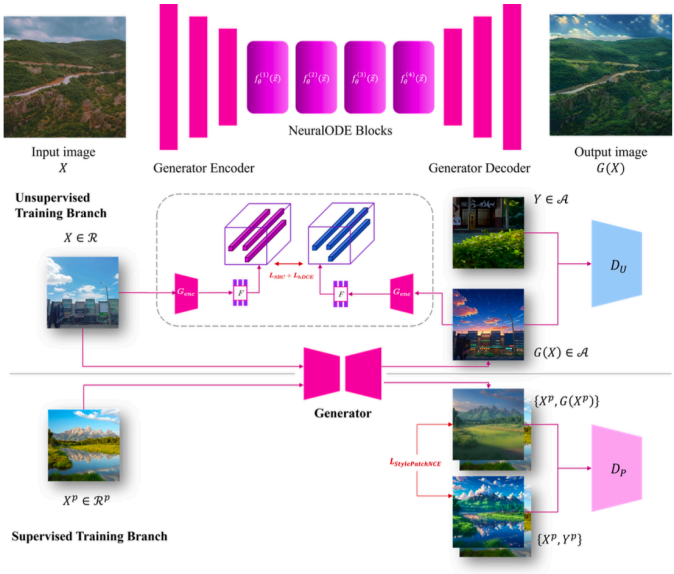
4.1.1. Definisi Umum Aplikasi

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

4.2. Perancangan Infrastruktur Sistem

NijiGAN adalah sebuah model generatif yang dirancang untuk mengubah gambar dunia nyata menjadi gambar bergaya anime dengan kualitas tinggi. Model ini memanfaatkan inovasi terbaru dalam kecerdasan buatan, yaitu *Neural Ordinary Differential Equations* (NeuralODEs), yang menggantikan pendekatan tradisional menggunakan blok *ResNet* pada model generatif lainnya. Tujuan utama NijiGAN adalah untuk menghasilkan gambar dengan detail yang lebih halus dan gaya anime yang lebih konsisten, sekaligus mengurangi

kompleksitas model yang biasanya memerlukan banyak parameter dan memori.



Gambar 4. 1 Arsitektur NijiGAN

Arsitektur NijiGAN terdiri dari beberapa komponen utama yang bekerja bersama untuk mencapai hasil akhir yang optimal. Proses dimulai dengan *encoder* yang mengekstraksi fitur penting dari gambar *input*, lalu dilanjutkan dengan *bottleneck* yang menggunakan NeuralODEs untuk memproses data secara kontinu, dan akhirnya menghasilkan gambar anime melalui *decoder*. Dalam rangka meningkatkan kualitas hasil, NijiGAN memanfaatkan data *pseudo-paired* dan *unpaired* dalam pelatihannya, yang memungkinkan model belajar dari kedua jenis data ini tanpa memerlukan pasangan gambar yang sepenuhnya sesuai.

4.2.1. Persamaan NeuralODE Pada CUT

Secara umum, arsitektur *CUT (Contrastive Unpaired Translation)* terdiri dari tiga komponen utama: *encoder*, *bottleneck*, dan *decoder*.

Encoder bertugas mengekstrak fitur penting dari gambar *input* yang berada pada domain X atau X_p. Dalam arsitektur *CUT*, *encoder* menggunakan sejumlah *lapisan konvolusi* untuk mengubah gambar *input* menjadi representasi laten yang lebih abstrak. Representasi laten ini menyimpan informasi semantik dan tekstur utama yang diperlukan untuk proses *style transfer*.

Bottleneck menjadi elemen kunci dalam memastikan bahwa fitur-fitur yang diekstrak dari domain *input* (X atau X_p) dapat dipetakan secara efektif ke domain *output* (Y atau Y_p). *Bottleneck* ini berfungsi sebagai representasi diskriminatif yang efisien untuk tugas translasi, tanpa membutuhkan data pasangan langsung (*directly paired data*). Dalam konteks *NijiGAN*, bagian ini dioptimalkan dengan mengganti lapisan *ResNet* diskret yang ada di model sebelumnya (*CUT/Scenimefy*) dengan *NeuralODEs*. *NeuralODE* memungkinkan transformasi *state* di setiap lapisan *neural network* dilakukan secara kontinu, sehingga menghasilkan proses pemetaan yang lebih halus dan efisien.

Decoder bertanggung jawab untuk merekonstruksi gambar dari representasi laten yang telah diproses oleh *bottleneck*. Proses *decoding* ini bertujuan untuk mengubah fitur laten menjadi gambar *output* di domain Y atau Y_p, yang dalam kasus ini bergaya anime. Selama proses *decoding*, model harus memastikan bahwa

konten asli dari gambar *input* tetap terjaga, meskipun gaya visualnya diubah sesuai dengan domain target.

Secara spesifik, inovasi pada *NijiGAN* terletak pada penggunaan *NeuralODEs* di bagian *bottleneck*. Pendekatan ini menggantikan lapisan *ResNet* tradisional dengan metode dinamika kontinu, yang memungkinkan perubahan *state* berlangsung secara lebih dinamis dibandingkan pembaruan diskret. Hal ini tidak hanya meningkatkan kualitas visual hasil translasi, tetapi juga mengurangi kompleksitas parameter model secara keseluruhan.

Dalam NijiGAN, kami mengganti lapisan ResNet diskret di *bottleneck* dengan NeuralODEs untuk memungkinkan evolusi dinamis dari status di seluruh lapisan jaringan neural. NeuralODEs menggunakan jaringan neural untuk memparameterisasi medan vektor dan umumnya dijelaskan oleh persamaan diferensial berikut:

$$\frac{dh}{dt}(t) = f_\omega\big(t, h(t)\big); \quad h(0) = h_0$$

Di mana $f_\omega\big(t, h(t)\big)$ terdiri dari jaringan neural konvolusi, fungsi aktivasi, dan normalisasi batch. Persamaan ini memodelkan dinamika ResNet sebagai berikut:

$$h_{t+1} = h_t + f(h_t, \omega_t)$$

Di sini, h_t merepresentasikan status tersembunyi dari jaringan neural pada lapisan t. Status diperbarui secara diskret dengan menambahkan *output* fungsi $f(h_t, \omega_t)$. Dengan memperkenalkan variabel ukuran langkah Δt, persamaan ini menyerupai

metode Euler untuk menyelesaikan persamaan diferensial:

$$h_{t+1} = h_t + \Delta t \cdot f(h_t, \omega_t)$$

Jika Δt mendekati nol, status tersembunyi bertransisi secara kontinu, membentuk dasar persamaan diferensial:

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

Solusi dari persamaan diferensial ini pada rentang $[t_0, t_1]$ diberikan oleh:

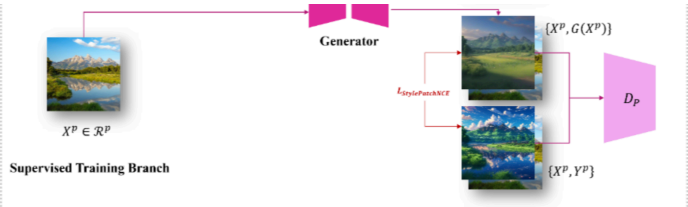
$$h(t_1) = h(t_0) + \int_{t_0}^{t_1} f_\omega \left(t, h(t)\right) dt$$

Untuk menyelesaikan persamaan ini, metode numerik seperti Runge-Kutta atau Dormand-Prince (RKDP) digunakan. NijiGAN menggunakan RKDP untuk efisiensi pelatihan. Pembelajaran semi-terawasi digunakan untuk memetakan gambar x dari domain dunia nyata R ke domain anime A menggunakan dataset *pseudo-paired* $P = \{x_p^{(i)}, y_p^{(i)}\}_{i=1}^{N}$. Proses pelatihan melibatkan cabang terawasi dan tidak terawasi.

4.2.2. Supervised Learning

Dalam proses *supervised learning* model dilatih untuk mempelajari fitur yang diekstrak dari *pseudo-paired* dataset yang dihasilkan secara sintetis. Dataset ini terdiri dari pasangan gambar dunia nyata dan gambar anime yang dihasilkan dengan menggunakan model seperti Scenimefy. Fitur-fitur yang diekstrak dari dataset ini digunakan dalam

proses pelatihan dan pemetaan untuk scene stylization (perubahan gaya gambar).



Gambar 4. 2 Fase Supervised Learning

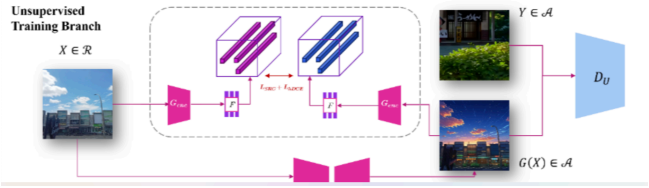
Model menggunakan *framework* conditional GAN untuk melakukan pembelajaran pada tahap *supervised learing*. Ada dua komponen utama dalam proses ini yaitu *Patch Discriminator* dan *Adversial Loss.* Pada *Patch Discriminator* model membedakan antara pasangan data asli (gambar anime dan dunia nyata) dan gambar yang dihasilkan oleh model. Discriminator ini berfungsi untuk memberi *feedback* kepada model tentang seberapa baik gambar yang dihasilkan mirip dengan gambar target (anime). Pada bagian *Adversial Loss,* model belajar dengan cara mengevaluasi hasil yang dihasilkan menggunakan *adversarial loss,* yang bertujuan meminimalkan perbedaan antara gambar yang dihasilkan dan gambar asli yang sesuai dengan referensi gaya anime.

Sebagai pengganti penggunaan *ground truth* dari domain target (gambar anime asli), bagian ini memanfaatkan *pseudo-ground truth* yang dihasilkan secara otomatis. *Pseudo-ground truth* ini terdiri atas gambar-gambar anime yang dibuat secara sintetis menggunakan dataset yang telah tersedia. Model dilatih untuk mempelajari kesamaan lokal antar *patch* gambar. Dalam proses ini, gambar yang dihasilkan oleh model dibandingkan dengan *patch* yang relevan pada *pseudo-*

ground truth. Pendekatan ini dirancang untuk menjaga konsistensi tekstur dan gaya pada gambar anime yang dihasilkan, tanpa harus bergantung pada *patch* yang identik. Pendekatan ini disebut dengan *patch-wise contrastive learning*. *cGAN Loss* bertugas untuk memastikan gambar yang dihasilkan memiliki kesamaan dengan gambar referensi (anime) dan membedakan gambar asli dari gambar yang dihasilkan. *StylePatchNCE Loss* mengatur kesamaan gaya dan tekstur antara gambar yang dihasilkan dengan *pseudo-ground truth* pada tingkat *patch*.

Bobot (*weight*) ditetapkan untuk masing-masing *loss function* untuk mengontrol kontribusi keduanya terhadap total *loss,* yang akan digunakan untuk mengoptimalkan model selama pelatihan.

4.2.3. Unsupervised Learning



Gambar 4. 3 Fase Unsupervised Learning

Komponen *unsupervised branch* pada NijiGAN dirancang untuk menyesuaikan gambar anime dengan gaya visual dari dataset anime Makoto Shinkai di domain A. Tujuan utama dari proses ini adalah mentransformasikan gambar anime agar memiliki gaya visual yang menyerupai karya anime Makoto Shinkai, sekaligus mempertahankan struktur asli dari gambar tersebut.

Dalam praktiknya, *patch* gambar dari berbagai lokasi memiliki hubungan semantik yang sangat heterogen, sehingga menimbulkan tantangan besar dalam menemukan pasangan *patch* yang tepat untuk proses pelatihan. Namun, keragaman hubungan semantik ini harus tetap dipertimbangkan untuk mencapai *image translation* yang sesuai dengan konten dari gambar yang diolah.

Untuk mengatasi tantangan ini, digunakan *Semantic Relation Consistency Loss* (SRC), yang dirancang untuk mengurangi *Jensen-Shannon Divergence* (JSD) antara distribusi kemiripan *patch* pada domain asli dan domain yang dihasilkan oleh model. *Semantic Relation Consistency Loss* bertujuan untuk meminimalkan perbedaan distribusi kemiripan *patch* antara hasil dari *encoder* gambar asli dan gambar yang telah diproses oleh model. Pendekatan ini memastikan bahwa gambar yang dihasilkan tetap menjaga konsistensi semantik baik dari segi konten maupun struktur.

Selain itu, untuk meningkatkan kemampuan model dalam mengidentifikasi *hard negatives*, digunakan *hard negative contrastive loss*. *Loss* ini bertujuan untuk memperbaiki kemampuan model dalam membedakan antara pasangan positif yang relevan dan pasangan negatif yang tidak relevan. *Hard Negative Contrastive Loss* (hDCE) mengoptimalkan model untuk lebih sensitif terhadap negatif yang keras (hard negatives)**,** yaitu pasangan yang tidak sesuai atau sulit dibedakan oleh model. Proses ini membantu dalam memperbaiki hubungan semantik yang lebih

halus antara gambar yang dihasilkan dan gambar referensi.

Secara keseluruhan, *loss* yang diterapkan pada bagian *unsupervised training* terdiri dari tiga komponen utama: *GAN Loss*, yaitu *loss* standar pada Generative Adversarial Network (GAN) untuk memastikan kualitas gambar yang dihasilkan; *Semantic Relation Consistency Loss* (SRC), yang bertujuan mengurangi perbedaan distribusi antara *patch* gambar yang dihasilkan dan gambar asli untuk menjaga konsistensi semantik; dan *Hard Negative Contrastive Loss* (hDCE), yang membantu model mempelajari perbedaan pada pasangan negatif yang sulit, sehingga meningkatkan keakuratan model.

4.2.4. Proses Training NijiGAN

Pelatihan model NijiGAN secara keseluruhan menggabungkan dua bagian: *supervised* dan *unsupervised.* Untuk memaksimalkan hasil, NijiGAN dilatih menggunakan *optimizer* Adam yang mengoptimalkan parameter model berdasarkan total *loss* i2i selama fase *backpropagation*. Proses pelatihan terdiri dari 25 epoch, dengan setiap epoch memiliki 24.000 langkah. Pada 10 epoch pertama, model menjalani fase *warm-up*, di mana parameter tertentu belum diterapkan sepenuhnya. Setelah 10 *epoch* pertama*, supervised loss weight* lambda_{sup}(t) mulai diterapkan secara bertahap sesuai dengan kemajuan *epoch,* dengan semakin besar bobot *loss* ini pada *epoch* yang lebih tinggi.

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI SISTEM

Bab ini membahas tentang implementasi dari bagian model yang kami buat. Implementasi ini akan dibagi ke dalam beberapa bagian, yaitu bagian pembuatan style_encoder, content_encoder, vgg19 lpips, vqi2iadain, *deployment* hasil model pembanding, modul testing.

5.1. Implementasi Modul Pada Model

Dalam penelitian ini, kontribusi saya terletak pada pengembangan dan implementasi beberapa komponen model yang krusial untuk meningkatkan kualitas dan fleksibilitas dalam transfer gaya dan manipulasi gambar. Secara khusus, saya merancang StyleEncoder dan ContentEncoder yang masing-masing bertanggung jawab untuk mengekstraksi representasi gaya dan konten dari gambar. StyleEncoder difokuskan untuk menangkap informasi gaya visual, seperti tekstur dan warna, sedangkan ContentEncoder dirancang untuk menangkap elemen struktural dan konten mendasar dari gambar. Selain itu, saya juga mengintegrasikan LPIPS (*Learned Perceptual Image Patch Similarity*), sebuah metrik yang mengukur kesamaan antara gambar berdasarkan persepsi manusia, untuk meningkatkan evaluasi kesamaan gaya yang dihasilkan oleh model. Kontribusi signifikan lainnya adalah penerapan VQI2I_AdaIN, yang menggabungkan kuantisasi vektor dengan *Adaptive Instance Normalization* (AdaIN), memungkinkan transfer gaya yang lebih halus dan realistis antara gambar.

5.1.1. StyleEncoder

StyleEncoder berfungsi untuk mengekstraksi atribut gaya dari gambar, menghasilkan representasi gaya yang terkompresi. Proses dimulai dengan penerapan blok konvolusi awal, di mana gambar *input* melewati lapisan konvolusi dengan *padding* reflektif untuk menangkap pola global. Lapisan konvolusi ini diikuti oleh fungsi aktivasi ReLU untuk menghasilkan fitur non-linear. Setelah itu, blok penurunan resolusi (*downsampling*) secara bertahap mengurangi dimensi gambar dengan faktor dua pada setiap tahap, menghasilkan fitur yang lebih abstrak dan efisien untuk diproses lebih lanjut. Setelah melewati tahap konvolusi, fitur yang dihasilkan diratakan menggunakan *adaptive average pooling*, menghasilkan representasi gaya dengan ukuran tetap. Proses ini kemudian diakhiri dengan lapisan konvolusi tambahan yang memetakan representasi gaya ke dalam ruang dimensi gaya yang lebih spesifik. Untuk implementasi lengkap dapat dilihat pada tabel ini.

```
class StyleEncoder(nn.Module):
    def __init__(self, in_channels, hidden_dimensions,
                 style_dimensions, n_downsampling):
        super(StyleEncoder, self).__init__()
        self.use_bias = True
        self.in_channels = in_channels
        self.hidden_dimensions = hidden_dimensions
        self.style_dimensions = style_dimensions
        self.n_downsampling = n_downsampling
        self.block1 = nn.Sequential(
            nn.ReflectionPad2d(3),
            nn.Conv2d(in_channels=in_channels,
```

```
out_channels=hidden_dimensions,     kernel_size=7,     stride=1,
bias=self.use_bias),
            nn.ReLU()
        )
        self.block2 = nn.Sequential(
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels=hidden_dimensions,
out_channels=hidden_dimensions*2,    kernel_size=4,    stride=2,
bias=self.use_bias),
            nn.ReLU(),
            nn.ReflectionPad2d(1),
            nn.Conv2d(in_channels=hidden_dimensions*2,
out_channels=hidden_dimensions*4,    kernel_size=4,    stride=2,
bias=self.use_bias),
            nn.ReLU()
        )
        self.block3 = []

        for i in range(self.n_downsampling - 2):
            self.block3 += [
                nn.ReflectionPad2d(1),
                nn.Conv2d(in_channels=hidden_dimensions*4,
out_channels=hidden_dimensions*4,    kernel_size=4,    stride=2,
bias=self.use_bias),
                nn.ReLU()
            ]

        self.block3 = nn.Sequential(*self.block3)

        self.global_pooling = nn.AdaptiveAvgPool2d(1)
        self.ffn         =         nn.Conv2d(hidden_dimensions*4,
self.style_dimensions, kernel_size=1, stride=1, padding=0)

    def forward(self, x):
        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.global_pooling(x)
        x = self.ffn(x)
        return x
```

Tabel 5. 1 Implementasi StyleEncoder

5.1.2. ContentEncoder

ContentEncoder, berfokus untuk menangkap konten atau struktur utama dari gambar, dengan mempertahankan detail spasial yang penting. Proses dimulai dengan lapisan konvolusi awal yang menghasilkan fitur dasar dari gambar masukan. Pada setiap tahap penurunan resolusi, fitur diproses melalui blok residual (*ResNet blocks*), yang dirancang untuk mempertahankan informasi penting tanpa mengorbankan detail visual yang esensial. Pada resolusi tertentu, mekanisme perhatian (*attention*) diterapkan untuk menonjolkan elemen-elemen yang relevan dan signifikan dalam gambar. Setelah melalui tahap penurunan resolusi, fitur dilanjutkan ke blok tengah yang terdiri dari dua blok *Resnet* tambahan serta satu mekanisme perhatian untuk memproses informasi dengan kedalaman yang lebih tinggi. Proses ini ditutup dengan normalisasi menggunakan GroupNorm dan diakhiri dengan lapisan konvolusi final untuk menghasilkan fitur akhir dengan dimensi tertentu yang siap untuk digunakan dalam tahap selanjutnya. Untuk implementasi lengkap dapat dilihat pada tabel ini:

```
class ContentEncoder(nn.Module):
    def __init__(self,
                 in_channels : int,
                 intermediate_channels : int,
                 channel_multipliers : list,
                 resblock_counts : int,
                 attn_resolutions : list,
                 dropout : float = 0.0,
                 resample_with_conv : bool = True,
                 resolution : int = 256,
                 z_channels : int = 256,
```

```
                 double_z : bool = True):
        super(ContentEncoder, self).__init__()
        self.in_channels = in_channels
        self.intermediate_channels = intermediate_channels
        self.time_embedding_channels = 0
        self.num_resolutions = len(channel_multipliers)
        self.resblock_counts = resblock_counts
        self.resolution = resolution

        self.conv_in = nn.Conv2d(in_channels=in_channels,

out_channels=self.intermediate_channels,
                                 kernel_size=3,
                                 stride=1,
                                 padding=1)

        current_resolution = resolution
        in_channels_multiplier       =       (1,       )       +
tuple(channel_multipliers)
        self.in_channels_multipliers = in_channels_multiplier
        self.down = nn.ModuleList()

        for i_level in range(self.num_resolutions):
            block = nn.ModuleList()
            attn = nn.ModuleList()
            block_in       =       intermediate_channels       *
in_channels_multiplier[i_level]
            block_out       =       intermediate_channels       *
channel_multipliers[i_level]

            for i_block in range(self.resblock_counts):
                block.append(
                    ResNetBlock(in_channels=block_in,
                                out_channels=block_out,

time_embedding_channel=self.time_embedding_channels,
                                dropout=dropout)
                )
                block_in = block_out
                if current_resolution in attn_resolutions:
                    attn.append(
```

```
LinearAttentionBlock(in_channels=block_out)
                    )

            downblock = nn.Module()
            downblock.block = block
            downblock.attn = attn

            if i_level != self.num_resolutions - 1:
                downblock.downsample   =   Downsample(block_out,
with_conv=resample_with_conv)
                current_resolution = current_resolution // 2

            self.down.append(downblock)

        self.mid = nn.Module()
        self.mid.block_1   =   ResNetBlock(in_channels=block_in,
out_channels=block_in,
time_embedding_channel=self.time_embedding_channels,
dropout=dropout)
        self.mid.attn                                          =
LinearAttentionBlock(in_channels=block_in)
        self.mid.block_2   =   ResNetBlock(in_channels=block_in,
out_channels=block_in,
time_embedding_channel=self.time_embedding_channels,
dropout=dropout)

        self.norm_out = Normalize(block_in)
        self.conv_out  =  nn.Conv2d(block_in,  2  *  z_channels  if
double_z else z_channels, kernel_size=3, stride=1, padding=1)
```

Tabel 5. 2 Implementasi ContentEncoder

5.1.3. Lpips

LPIPS (*Learned Perceptual Image Patch Similarity*) adalah sebuah metrik untuk mengukur kesamaan visual antara dua gambar berdasarkan fitur yang dipelajari oleh jaringan neural. Metrik ini dikembangkan dengan tujuan untuk lebih mirip dengan

persepsi manusia terhadap perbedaan gambar, berbeda dengan metrik konvensional seperti PSNR atau SSIM yang mengandalkan kesamaan *pixelwise.* LPIPS menggunakan jaringan CNN yang dilatih untuk mengekstraksi fitur pada berbagai level resolusi dan memahami distorsi gambar yang lebih halus, sehingga menghasilkan penilaian yang lebih sesuai dengan persepsi manusia.

LPIPS bekerja dengan cara membandingkan fitur gambar yang dihasilkan oleh jaringan CNN yang dilatih dengan dataset besar. Dalam kasus ini, model VGG19 digunakan untuk ekstraksi fitur. LPIPS menghitung perbedaan antara fitur-fitur ini dengan cara yang lebih sensitif terhadap perbedaan visual yang penting bagi manusia, seperti bentuk, tekstur, dan struktur.

Fungsi __init__ berfungsi untuk menginisialisasi objek dari kelas VGG19. Fungsi ini pertama kali mengatur konfigurasi jaringan berdasarkan parameter yang diberikan, seperti *init_weights, feature_mode, batch_norm,* dan *num_classes*. Parameter *init_weights* digunakan untuk memuat bobot model, baik dengan mendownload bobot yang sudah dilatih sebelumnya dari URL tertentu ataupun memuat bobot dari file lokal. Kemudian, fungsi ini memanggil fungsi *make_layers* untuk membangun lapisan-lapisan konvolusional sesuai dengan konfigurasi yang ditentukan, yang disusun dalam variabel c*fg.* Setelah itu, lapisan *fully connected* ditambahkan untuk tujuan klasifikasi. Jika *init_weights* berisi URL atau *file* bobot, bobot tersebut akan dimuat ke dalam model. Secara keseluruhan, fungsi ini memastikan bahwa arsitektur dan bobot awal

model telah siap untuk pelatihan atau evaluasi lebih lanjut. Untuk implementasi lengkap dapat dilihat pada tabel ini:

```
class VGG19(nn.Module):
    def  __init__(self,  init_weights=None,  feature_mode=False,
batch_norm=False, num_classes=1000, progress=True):
        super(VGG19, self).__init__()
        self.cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256,
256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M']
        self.init_weights = init_weights
        self.feature_mode = feature_mode
        self.batch_norm = batch_norm
        self.num_classes = num_classes
        self.features = self.make_layers(self.cfg, batch_norm)
        self.classifier = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, 4096),
            nn.ReLU(True),
            nn.Dropout(),
            nn.Linear(4096, num_classes),
        )
        if init_weights is not None:
            if init_weights == 'vgg19':
                # progress (bool): If True, displays a progress
bar of the download to stderr
                model_url                                       =
'https://download.pytorch.org/models/vgg19-dcbb9e9d.pth'
                state_dict = load_state_dict_from_url(model_url,
progress=progress)
            else:
                state_dict = torch.load(init_weights)
            self.load_state_dict(state_dict)
```

Tabel 5. 3 Konstruktor Utama LPIPS

Fungsi *make_layers* bertanggung jawab untuk membangun arsitektur lapisan-lapisan konvolusional yang ada dalam model VGG19. Fungsi ini menerima

parameter *cfg* yang berisi konfigurasi jumlah filter untuk setiap lapisan konvolusi dan simbol 'M' untuk lapisan *max pooling*. Dengan menggunakan konfigurasi ini, fungsi membangun lapisan konvolusional secara bertahap, diikuti dengan fungsi aktivasi ReLU setelah setiap lapisan konvolusional. Jika parameter *batch_norm* diaktifkan, *batch normalization* juga diterapkan setelah setiap konvolusi untuk memperbaiki stabilitas pelatihan. Selain itu, fungsi ini menambahkan lapisan *max pooling* yang mengurangi ukuran fitur dengan cara melakukan subsampling. Setelah seluruh lapisan konvolusional dan *pooling* dibangun sesuai konfigurasi, fungsi mengembalikan model yang terdiri dari lapisan-lapisan tersebut dalam bentuk sebuah modul sekuensial. Untuk implementasi *function* dapat dilihat pada tabel ini:

```
def make_layers(self, cfg, batch_norm=False):
    layers = []
    in_channels = 3
    for v in cfg:
        if v == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            conv2d = nn.Conv2d(in_channels, v, kernel_size=3,
padding=1)
            if batch_norm:
                layers += [conv2d, nn.BatchNorm2d(v),
nn.ReLU(inplace=True)]
            else:
                layers += [conv2d, nn.ReLU(inplace=True)]
            in_channels = v
    return nn.Sequential(*layers)
```

Tabel 5. 4 Implementasi Fungsi make_layers

Fungsi *forward* mendefinisikan bagaimana data diproses melalui model. Jika *feature_mode* diaktifkan, *input* hanya diproses melalui lapisan konvolusional dan *output* dari lapisan tertentu dapat disimpan dalam *dictionary outputs* berdasarkan parameter *layers_to_save*. Dengan cara ini, pengguna dapat mengakses output spesifik dari lapisan yang diinginkan. Jika *feature_mode* tidak aktif, *input* akan diproses melalui seluruh lapisan konvolusional dan dilanjutkan ke lapisan *fully connected* untuk menghasilkan *output* klasifikasi. Jika ada lapisan yang *output*-nya diminta, fungsi akan mengembalikan *output* dari lapisan-lapisan tersebut bersama hasil akhir model dalam bentuk *tuple*. Untuk implementasi *function* dapat dilihat pada tabel ini:

```
def forward(self, x, layers_to_save=None):
    outputs = {}  # Dictionary to store outputs of specified
layers
    if self.feature_mode:
        module_list = list(self.features.modules())
        for idx, l in enumerate(module_list[1:27]):  # conv4_4
            x = l(x)
            if layers_to_save and idx in layers_to_save:
                outputs[f'layer_{idx}'] = x  # Save the output of
the specified layer
    if not self.feature_mode:
        for idx, l in enumerate(self.features):
            x = l(x)
            if layers_to_save and idx in layers_to_save:
                outputs[f'layer_{idx}'] = x  # Save the output of
the specified layer
        x = x.view(x.size(0), -1)
        for idx, l in enumerate(self.classifier):
            x = l(x)
            classifier_idx = idx + len(self.features)  # Continue
```

```
layer indexing
            if    layers_to_save    and    classifier_idx    in
layers_to_save:
                outputs[f'layer_{classifier_idx}'] = x   # Save
the output of the classifier layer

    if layers_to_save:
        return outputs, x  # Return both the intermediate outputs
and final result
    return x  # Only return the final output if no layers are
specified
```

Tabel 5. 5 Implementasi Fungsi Propagasi Maju LPIPS

5.1.4. VQI2I_AdaIN

VQI2I_AdaIN dirancang untuk memproses gambar dengan teknik transfer gaya dan manipulasi konten menggunakan jaringan syaraf. Pada awalnya, model ini diinisialisasi dengan berbagai parameter yang menentukan struktur jaringan, seperti jumlah saluran *input*, dimensi gaya, dan resolusi gambar. Model ini memiliki dua *encoder* gaya (style_enc_a dan style_enc_b) serta satu encoder konten (content_enc). *Encoder* gaya bertugas untuk mengekstraksi fitur gaya dari gambar, sementara encoder konten bertugas untuk menangkap informasi konten.

Fungsi *encode* bekerja dengan menerima gambar dan label, lalu mengekstraksi representasi konten dari gambar menggunakan *encoder* konten. Fitur konten ini kemudian diproses dengan teknik *quantization*, yaitu dikompresi menggunakan *quant_conv* dan *quantize*. Sementara itu, gaya gambar diambil dari *encoder* gaya yang sesuai dengan label yang diberikan. Proses ini

memungkinkan pemisahan antara konten dan gaya dalam gambar.

Setelah konten dan gaya diekstraksi, fungsi decode_a dan decode_b digunakan untuk mendekode hasil *quantization* dan menghasilkan gambar baru. Fungsi-fungsi ini menerima representasi konten yang telah diproses dan gaya yang diterapkan, kemudian menghasilkan gambar yang merupakan kombinasi dari kedua elemen tersebut.

Fungsi forward bertugas untuk mengatur alur eksekusi model dengan memutuskan apakah menggunakan *decoder* A atau B berdasarkan label *input.* Fungsi ini juga mengembalikan hasil gambar rekonstruksi dan selisih (*difference*) antara *input* dan *output.* Pada kelas VQI2ICrossGAN_AdaIN, terdapat tambahan kemampuan untuk melakukan pertukaran gaya antar *decoder*. Jika parameter *cross* diset ke True, model dapat mengubah gaya yang diterapkan, memungkinkan penerapan gaya dari *decoder* A ke *decoder* B, atau sebaliknya, tergantung pada parameter s_given yang diberikan. Untuk implementasi lengkap dapat dilihat pada tabel ini:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.autograd import Variable

from models.vqi2i.modules.encoders.network import *
from models.vqi2i.modules.vqvae.quantize import VectorQuantizer
#    from    models.vqi2i.modules.discriminators.model    import
NLayerDiscriminator
#    from    models.vqi2i.modules.losses.vqperceptual    import
hinge_d_loss
```

```
class VQI2I_AdaIN(nn.Module):
    def __init__(self,
                 in_channels : int = 3,
                 intermediate_channels : int = 128,
                 out_channels : int = 3,
                 style_dim : int = 128,
                 z_channels : int = 128,
                 double_z : bool = True,
                 n_embed : int = 512,
                 embed_dim : int = 512,
                 channel_multipliers : list = [1,1,2,4,8],
                 resblock_counts : int = 2,
                 attn_resolutions : list = [16],
                 dropout : float = 0.1,
                 resolution : int = 256,
                 n_adaresblock : int = 4,
                 ckpt_path : str = None,
                 ignore_keys : list = [],
                 image_key : str = "image",
                 colorize_nlabels=None,
                 monitor=None
                ):

        super(VQI2I_AdaIN, self).__init__()
        self.image_key = image_key

        self.style_enc_a = StyleEncoder(in_channels=in_channels,

hidden_dimensions=intermediate_channels,

style_dimensions=style_dim,
                                        n_downsampling=3)

        self.style_enc_b = StyleEncoder(in_channels=in_channels,

hidden_dimensions=intermediate_channels,

style_dimensions=style_dim,
                                        n_downsampling=3)
```

```
        self.content_enc                                        =
ContentEncoder(in_channels=in_channels,

intermediate_channels=intermediate_channels,

channel_multipliers=channel_multipliers,

resblock_counts=resblock_counts,

attn_resolutions=attn_resolutions,
                                            dropout=dropout,
                                            resolution=resolution,
                                            z_channels=z_channels,
                                            double_z=double_z)

        self.quantize       =       VectorQuantizer(n_e=n_embed,
e_dim=embed_dim, beta=0.25)

        self.quant_conv = nn.Conv2d(z_channels * 2 if double_z
else z_channels, embed_dim, kernel_size=1, stride=1)
        self.post_quant_conv = nn.Conv2d(embed_dim, z_channels *
2 if double_z else z_channels, kernel_size=1, stride=1)

        self.decoder_a = Decoder(out_channels=out_channels,

intermediate_channels=intermediate_channels,

channel_multipliers=channel_multipliers,

resblock_counts=resblock_counts,

attn_resolutions=attn_resolutions,
                                    dropout=dropout,
                                    resolution=resolution,
                                    z_channels=z_channels,
                                    n_adaresblock=n_adaresblock,
                                    style_dim=style_dim,
                                    double_z=double_z)

        self.decoder_b = Decoder(out_channels=out_channels,
```

```
intermediate_channels=intermediate_channels,

channel_multipliers=channel_multipliers,

resblock_counts=resblock_counts,

attn_resolutions=attn_resolutions,
                                    dropout=dropout,
                                    resolution=resolution,
                                    z_channels=z_channels,
                                    n_adaresblock=n_adaresblock,
                                    style_dim=style_dim,
                                    double_z=double_z)

    def encode(self, x, label):
        c = self.content_enc(x)
        c = self.quant_conv(c)
        quant_c, emb_loss, info = self.quantize(c)

        # Encode Style
        if label == 1:
            style_vector = self.style_enc_a(x)
        else:
            style_vector = self.style_enc_b(x)

        return quant_c, emb_loss, info, style_vector

    def encode_style(self, x, label):
        if label == 1:
            style_vector = self.style_enc_a(x)
        else:
            style_vector = self.style_enc_b(x)

        return style_vector

    def encode_content(self, x):
        c = self.content_enc(x)
        c = self.quant_conv(c)
        quant_c, emb_loss, info = self.quantize(c)
        return c, quant_c
```

```
    def decode_a(self, quant_c, style_vector):
        c = self.post_quant_conv(quant_c)
        x_hat = self.decoder_a(c, style_vector)
        return x_hat

    def decode_b(self, quant_c, style_vector):
        c = self.post_quant_conv(quant_c)
        x_hat = self.decoder_b(c, style_vector)
        return x_hat

    def forward(self, x, label):
        if label == 1:
            quant_c,  diff,  _,  style_vector  =  self.encode(x,
label)
            dec = self.decode_a(quant_c, style_vector)

        else:
            quant_c,  diff,  _,  style_vector  =  self.encode(x,
label)
            dec = self.decode_b(quant_c, style_vector)

        return dec, diff

    def get_last_layer(self, label):
        if label == 1:
            return self.decoder_a.conv_out.weight
        else:
            return self.decoder_b.conv_out.weight

class VQI2ICrossGAN_AdaIN(VQI2I_AdaIN):
    def __init__(self,
                 n_embed,
                 embed_dim,
                 ckpt_path=None,
                 ignore_keys=[],
                 image_key="image",
                 colorize_nlabels=None,
                 monitor=None):

        super(VQI2ICrossGAN_AdaIN, self).__init__(
            n_embed=n_embed, embed_dim=embed_dim
```

```
        )

    def forward(self, x, label, cross=False, s_given=False):
        quant, diff, _, style_vector = self.encode(x, label)

        if label == 1:
            if cross == False:
                output = self.decode_a(quant, style_vector)
            else:
                style_vector = s_given
                output = self.decode_b(quant, style_vector)
        else:
            if cross == False:
                output = self.decode_b(quant, style_vector)
            else:
                style_vector = s_given
                output = self.decode_a(quant, style_vector)

        return output, diff, style_vector
```

Tabel 5. 6 Implementasi VQI2I_adaIN

5.2. Modul Testing NijiGAN

Untuk mengevaluasi kinerja model NijiGAN dalam menghasilkan gambar, dilakukan serangkaian pengujian menggunakan data uji yang tidak dilihat selama proses pelatihan. Dalam pengujian ini, metrik seperti PSNR dan MSE digunakan untuk mengukur kualitas gambar yang dihasilkan serta membandingkannya dengan gambar asli. Proses ini bertujuan untuk memastikan bahwa model dapat menghasilkan gambar yang akurat dan memenuhi standar kualitas yang diharapkan.

Pada bagian *testing* dalam kode ini, model dievaluasi untuk menghasilkan gambar-gambar berdasarkan data yang diberikan dan menghitung beberapa

metrik untuk mengukur kualitas gambar yang dihasilkan. Pertama-tama, model dimuat dari *checkpoint* yang telah disimpan sebelumnya, jika ada, dan dipersiapkan untuk mode evaluasi dengan menonaktifkan operasi yang tidak perlu seperti *dropout*. Setelah itu, gambar diuji menggunakan *test loader*, yang berisi data yang belum pernah dilihat oleh model selama pelatihan.

Setiap batch data diuji dengan menghasilkan gambar dari generator dan menghitung metrik PSNR (*Peak Signal-to-Noise Ratio*) dan MSE (*Mean Squared Error*) antara gambar yang dihasilkan dan gambar asli. Metrik ini digunakan untuk menilai seberapa baik gambar yang dihasilkan mendekati gambar yang sesungguhnya, di mana PSNR mengukur kualitas gambar berdasarkan rasio antara sinyal (gambar asli) dan gangguan (perbedaan antara gambar asli dan yang dihasilkan), sementara MSE mengukur rata-rata kesalahan kuadrat antara keduanya.

Hasil metrik ini disimpan untuk setiap *batch* dan dihitung rata-ratanya di akhir proses *testing*. Hasil dari evaluasi ini membantu untuk memahami sejauh mana model berhasil dalam menghasilkan gambar yang akurat dan berkualitas tinggi. Hasil gambar dan metrik kemudian disimpan dalam folder tertentu untuk analisis lebih lanjut.

Untuk implementasi *function* dapat dilihat pada tabel ini:

```
def test(self, checkpoint_path=None):
        """Run a full testing procedure and compute PSNR and MSE
metrics."""

        # Set up directory to save test results
        test_dir = os.path.join(self.save_dir, 'test_results')
        os.makedirs(test_dir, exist_ok=True)
```

```
        # Load checkpoint if specified
        if checkpoint_path is not None:
            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'
], strict=False)

        # Set generator and feature extractor to evaluation mode
        self.gen.eval()
        self.F.eval()

        print("Starting test evaluation...")

        # Initialize lists to collect MSE and PSNR scores across
batches
        mse_scores = []
        psnr_scores = []

        # Iterate over test loader batches
        for batch_idx, batch in enumerate(self.test_loader):
            x_p, y_p, x, y = 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 test images
            test_images  =  self.generate_validation_images(x_p,
y_p, x, y)

            # Save the generated images to test_dir
            self.save_validation_images(test_images,     'test',
batch_idx, save_dir=test_dir)

            # Calculate metrics between generated and real images
            for         img_type,         generated_img         in
test_images['supervised'].items():
                if img_type == 'rec_xp':  # Example: comparing
real x_p with rec_xp
```

```
                    metrics = self.compute_metrics(x_p,
generated_img)
                    mse_scores.append(metrics['mse'])
                    psnr_scores.append(metrics['psnr'])

            # Print progress and batch metrics
            print(f"Test batch [{batch_idx +
1}/{len(self.test_loader)}] completed and saved. "
                f"MSE: {metrics['mse']:.4f}, PSNR:
{metrics['psnr']:.2f} dB")

        # Calculate average metrics across all batches
        avg_mse = sum(mse_scores) / len(mse_scores)
        avg_psnr = sum(psnr_scores) / len(psnr_scores)
        print(f"\nAverage MSE: {avg_mse:.4f}, Average PSNR:
{avg_psnr:.2f} dB")


        print("Testing complete. Generated images and metrics
saved to:", test_dir)
```

Tabel 5. 7 Modul Testing NijiGAN

5.3. Deployment Model CartoonGAN

Proses ini dilakukan untuk deployment model CartoonGAN, yang bertujuan untuk membandingkan performanya dengan model yang kami buat, yaitu NijiGAN. Proses dimulai dengan mendefinisikan URL untuk dua dataset yang akan digunakan, yaitu dataset Shinkai dan dataset CycleGAN, yang diunduh menggunakan *library* opendatasets. Setelah dataset berhasil diunduh, proses ini mengunduh model pretrained yang diperlukan dari repositori GitHub dan memindahkannya ke dalam folder proyek. Selanjutnya, penyalinan dataset dilakukan ke dalam direktori yang sesuai di proyek CartoonGAN dan menjalankan skrip test.py dengan parameter yang telah ditentukan, seperti direktori *input* data, gaya gambar (dalam hal ini gaya Shinkai), serta perangkat GPU yang digunakan. Hasil

inferensi dari model CartoonGAN disimpan di direktori yang telah ditentukan. Setelah itu, proses ini melanjutkan dengan melakukan perhitungan *Fréchet Inception Distance* (FID) untuk mengevaluasi kualitas gambar yang dihasilkan dengan membandingkannya dengan gambar asli dari dataset Shinkai. Berikut adalah implementasi lengkapnya:

```
import os
!pip install opendatasets

import os
import opendatasets as od

# Set Kaggle API credentials directly
os.environ['KAGGLE_USERNAME'] = 'kevinputrasantoso'
os.environ['KAGGLE_KEY'] = 'bb537e4a9e83643c8364bd526b572f24'

# URLs for the datasets
shinkai_url                                                    =
'https://www.kaggle.com/datasets/kevinputrasantoso/shinkai-set'
cyclegan_url                                                   =
'https://www.kaggle.com/datasets/kevinputrasantoso/dataset-
cyclegan-uji'

# Create a folder for the datasets
download_path = './DATASET/'

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

print("Datasets downloaded successfully!")

!git   clone   https://github.com/Yijunmaverick/CartoonGAN-Test-
Pytorch-Torch

!cp -r DATASET CartoonGAN-Test-Pytorch-Torch/
```

```
# Commented out IPython magic to ensure Python compatibility.
# %cd CartoonGAN-Test-Pytorch-Torch

!sh pretrained_model/download_pth.sh

!python    test.py    --input_dir    "DATASET/dataset-cyclegan-
uji/Dataset CycleGAN UJI" --style Shinkai --gpu 0

# Commented out IPython magic to ensure Python compatibility.
# %cd ..

import os
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/CartoonGAN-Test-Pytorch-
Torch/test_output"

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

for idx, item in enumerate(real_items):
    real_items[idx] = os.path.join(real_dir, item)

for idx, item in enumerate(shinkai_items):
    shinkai_items[idx] = os.path.join(shinkai_dir, item)

for idx, item in enumerate(trans_items):
    trans_items[idx] = os.path.join(trans_dir, item)

# 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(diff**2) + 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}")
```

Tabel 5. 8 Deployment CartoonGAN

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini menjelaskan tahap uji coba terhadap model NijiGAN. Pengujian dilakukan untuk menganalisis performa model dan kualitas gambar yang dihasilkan oleh model.

6.1. Tujuan Pengujian dan Evaluasi Model

Pengujian dilakukan terhadap Model NijiGAN guna menguji kemampuan arsitektur untuk menghasilkan gambar anime.

6.2. Metrik Evaluasi Model

Untuk menganalisis kinerja model dan kualitas gambar yang dihasilkan, kami menggunakan pendekatan kuantitatif dan kualitatif secara terintegrasi. Pendekatan kuantitatif bertujuan untuk memperoleh nilai eksak yang mencerminkan seberapa baik model menghasilkan gambar sesuai dengan ekspektasi. Dalam hal ini, kami menggunakan *Frechet Inception Distance* (FID) sebagai metrik utama untuk mengevaluasi karakteristik gambar secara numerik berdasarkan referensi gaya anime. Sementara itu, pendekatan kualitatif digunakan untuk menilai kualitas gambar berdasarkan pengamatan manusia. Pendekatan ini memberikan perspektif yang lebih subjektif dan kontekstual dalam penilaian. Untuk menjembatani kedua pendekatan ini, kami menggunakan perhitungan *Mean Opinion Score* (MOS), yang bersifat semi-kuantitatif dan semi-kualitatif. Metrik MOS diperoleh dari penilaian subjektif beberapa evaluator manusia yang membandingkan gambar dengan konten identik yang dihasilkan oleh model NijiGAN dan model perbandingan lainnya. Proses ini memungkinkan kami untuk mendapatkan pemahaman menyeluruh tentang kualitas

gambar yang dihasilkan oleh model, baik dari segi angka objektif maupun penilaian subjektif.

6.3. Evaluasi Kuantitatif

Tabel 6.1 menunjukkan evaluasi kuantitatif terhadap metode yang kami usulkan dibandingkan dengan pendekatan dasar. Metode kami menghasilkan skor *Frechet Inception Distance* (FID) terendah, yang menunjukkan kualitas superior pada gambar yang dihasilkan, sejalan dengan kualitas visual yang lebih baik berdasarkan penilaian kualitatif kami. Sebagai titik referensi, kami juga menghitung FID antara dataset adegan dunia nyata dan dataset adegan anime. Distribusi gaya dari hasil yang dihasilkan oleh model kami secara signifikan lebih dekat dengan domain anime dibandingkan dengan gambar dunia nyata.

Selain menganalisis metrik FID, kami melakukan studi pengguna yang melibatkan 30 partisipan untuk menilai kualitas *rendering* adegan anime berdasarkan tiga aspek: representasi gaya anime yang jelas, konsistensi konten semantik yang akurat, dan performa *rendering* secara keseluruhan. Partisipan diminta memilih hasil terbaik dari enam metode berbeda pada 10 set gambar. Skor preferensi rata-rata, yang dirinci dalam Tabel 6.1, menunjukkan bahwa NijiGAN memiliki performa yang sebanding dengan Scenimefy, yang semakin menegaskan efektivitas pendekatan kami.

FID berfungsi sebagai metrik evaluasi kuantitatif yang mengukur kesamaan distribusi fitur antara gambar hasil terjemahan dan gambar referensi, khususnya adegan

anime. Skor FID yang lebih rendah menunjukkan kualitas gambar hasil terjemahan yang lebih tinggi dalam kaitannya dengan adegan anime. Di sisi lain, kami juga melakukan evaluasi semi-kuantitatif menggunakan MOS, yang didasarkan pada preferensi pengguna. Skor MOS yang lebih tinggi mencerminkan preferensi pengguna yang lebih besar terhadap gambar hasil terjemahan.

Method	CartoonGAN	AnimeGAN	Scenimefy	Ours
FID→	45.79	56.88	60.32	58.71
MOS↑	2.708	2.160	2.232	2.192

Tabel 6. 1 Perbandingan FID dan MOS

Tabel 6.2 menyajikan perbandingan kinerja antara NijiGAN dan Scenimefy. Perbandingan ini mencakup evaluasi efisiensi dan kualitas, memberikan gambaran tentang keunggulan dan kelemahan masing-masing metode berdasarkan konfigurasi yang digunakan.

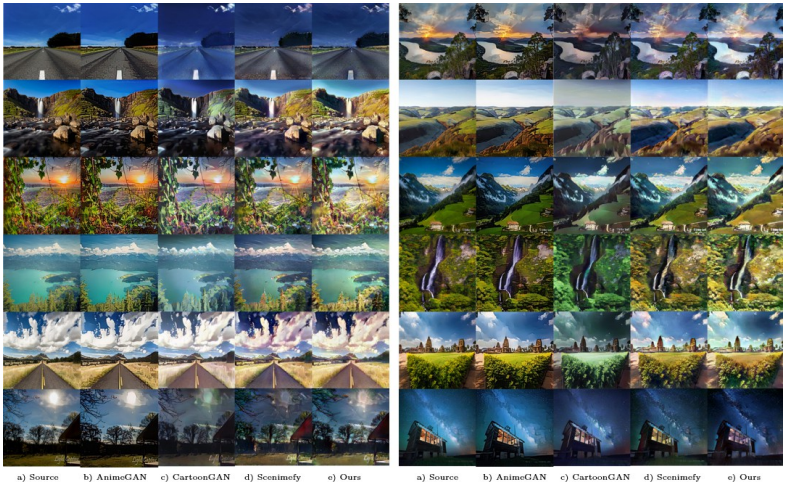
Model	Avg. Five Computation Time (s)	Avg. Fifty Computation Time (s)	Peak Memory (GB)	Model Parameters (M)	TFLOPS
NijiGAN (Ours)	1.533	0.771	9.694	5.477	0.771
Scenimefy	1.253	0.387	9.718	11.378	0.387

Tabel 6. 2 Perbandingan Peforma NijiGAN dan Scenemify

Hasil menunjukkan bahwa NijiGAN unggul dalam mengoptimalkan alokasi memori dan parameter model, seperti yang terlihat dari penggunaan memori puncak yang lebih rendah dan nilai parameter model yang lebih kecil dibandingkan dengan Scenimefy. Namun, NijiGAN masih kurang efisien dalam hal waktu komputasi rata-rata dan TFLOPS, yang terlihat dari waktu komputasi rata-rata dan skor TFLOPS yang lebih tinggi. Kurang efisiennya proses ini disebabkan oleh penggunaan metode dinamika kontinu

selama pelatihan, yang mengarah pada peningkatan kompleksitas model.

6.4. Evaluasi Kualitatif



Gambar 6.1 Perbandingan kualitas gambar antara NijiGAN dan beberapa model dasar.

Gambar 6.1 menyajikan perbandingan kualitatif antara semua model dasar. NijiGAN menunjukkan hasil yang kompetitif dibandingkan dengan model-model dasar tersebut. Dari sampel yang kami berikan, NijiGAN mencapai representasi yang seimbang antara konten asli dan struktur gaya anime yang diadopsi. NijiGAN berhasil menghasilkan gambar dengan struktur yang lebih halus yang mengingatkan pada lukisan anime. Sebaliknya, CartoonGAN dan AnimeGAN menggunakan *loss* yang dibuat secara manual untuk anime, seperti *edge-smoothing loss*, untuk mencoba menghasilkan tepi yang tajam, yang pada akhirnya membatasi sejauh mana

transfer gaya dapat dilakukan. Selain itu, AnimeGAN menghasilkan gambar yang lebih mirip dengan abstraksi tekstural dengan stylisasi yang lebih lemah, hampir tidak menunjukkan karakteristik gaya anime. Meskipun Scenimefy berhasil melakukan transfer gaya pada beberapa sampel, ia menunjukkan artefak yang signifikan, termasuk konversi dari siang ke malam yang tidak tepat dan pengenalan elemen tekstural yang menyebabkan gambar menjadi kabur di beberapa area. Di sisi lain, NijiGAN secara konsisten memberikan hasil yang stabil dan halus.

6.5. Studi Ablasi

Gambar 6.2 Studi Ablasi Terhadap Pengunaan Komponen NijiGAN

Kami juga menyajikan analisis perbandingan kontribusi komponen-komponen dalam model NijiGAN pada Gambar 6.2. Skenario yang disajikan meliputi kondisi model pada: 1.) Implementasi dengan *VGG Loss* pada 0,15, 2.) Tanpa VGG Loss, 3.) Penggantian NeuralODE dengan ResNet, dan 4.) Arsitektur model

lengkap (VGG Loss pada 0,1). Hasilnya menunjukkan bahwa *VGG loss* memainkan peran penting dalam generasi gambar. Model yang dilatih dengan *VGG loss* menunjukkan konsistensi struktur yang lebih baik, meskipun cenderung menghilangkan detail halus dan banyak karakteristik anime dari gambar sumber. Selain itu, implementasi NeuralODE terbukti efektif dalam menghasilkan gambar dengan karakteristik kelancaran yang lebih baik.

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.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] Jiang, Y., Jiang, L., Yang, S., Loy, C.C.: Scenimefy: learning to craft anime scene via semi-supervised image-to-image translation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 7357–7367 (2023)

[2] Khrulkov, V., Mirvakhabova, L., Oseledets, I., Babenko, A.: Latent transfor- mations via neuralodes for gan-based image editing. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 14428– 14437 (2021)

[3] Lee, H., Seol, J., Lee, S.-g., Park, J., Shim, J.: Contrastive learning for unsuper- vised image-to-image translation. Applied Soft Computing 151, 111170 (2024) https://doi.org/10.1016/j.asoc.2023.111170

[4] Park, T., Efros, A.A., Zhang, R., Zhu, J.-Y.: Contrastive learning for unpaired image-to-image translation. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) Computer Vision – ECCV 2020, pp. 319–345. Springer, Cham (2020)

[5] Han, J., Shoeiby, M., Petersson, L., Armin, M.A.: Dual contrastive learning for unsupervised image-to-image translation. In: Proceedings of the IEEE/CVF Con- ference on Computer Vision and Pattern Recognition (CVPR) Workshops, pp. 746–755 (2021)

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS I

Nama	: Adam Haidar Azizi
Tempat, Tanggal Lahir	: Pureworejo, 30 Oktober 2002
Jenis Kelamin	: Laki-Laki
Telepon	: +6282115890010
Email	: adamhadaizi2002@gmail.com

AKADEMIS

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