Program Studi Teknik Informatika
Institut Teknologi Sumatera

Nama	:	**Fathan Andi Kartagama(122140055)**
Mata Kuliah	:	**Pembelajaran Mendalam (IF25-40401)**
Tugas	:	**Eksplorasi Vision Transformer**
Dosen Pengampu	:	1. **Imam Eko Wicaksono, S.Si., M.Si.**
		2. **Martin Clinton Tosima Manullang, S.T., M.T., Ph.D.**
Tanggal	:	November 21, 2025

LAPORAN TUGAS EKSPLORASI

Perbandingan Model Vision Transformer (ViT) Base, Swin Transformer Base, dan DeiT Base

1 Pendahuluan

1.1 Latar Belakang

Dalam 20 tahun terakhir, *Convolutional Neural Networks* (CNN) sudah jadi raja di dunia *computer vision*. CNN berhasil menyelesaikan berbagai tugas seperti klasifikasi gambar, deteksi objek, dan segmentasi dengan performa yang sangat baik [1, 2]. Tapi ternyata, CNN punya kelemahan juga. Arsitekturnya yang bergantung pada konvolusi lokal membuat CNN kurang efektif dalam menangkap hubungan antar bagian gambar yang jaraknya jauh (*long-range dependencies*).

Nah, kesuksesan *Transformer* di bidang *Natural Language Processing* (NLP) [3] membuat para peneliti bertanya-tanya: "Kenapa nggak coba pakai *Transformer* buat gambar juga?". Di tahun 2020, Dosovitskiy dan timnya memperkenalkan *Vision Transformer* (ViT) [4]. Hasilnya cukup mengejutkan bahwa ternyata *transformer* murni tanpa konvolusi bisa mencapai hasil yang setara atau bahkan lebih baik dari CNN terbaik, terutama kalau dilatih dengan dataset besar seperti ImageNet-21k dan JFT-300M.

Cara kerja Vision Transformer sebenarnya cukup sederhana. Gambar dipotong-potong jadi *patches* berukuran kecil, lalu setiap *patch* diperlakukan seperti kata-kata dalam kalimat. Dengan mekanisme *self-attention*, model bisa "melihat" hubungan antar *patches* di seluruh gambar sekaligus. Ini berbeda dengan CNN yang hanya bisa melihat area lokal di sekitarnya. Pendekatan ini memberikan fleksibilitas lebih dalam memahami konteks global dari sebuah gambar.

Kesuksesan ViT memicu banyak inovasi baru. Liu dan timnya mengembangkan *Swin Transformer* [5] yang lebih efisien dengan menggunakan *shifted window attention* dan arsitektur hierarkis untuk menangkap informasi di berbagai skala. Touvron dkk. membuat *Data-efficient Image Transformer* (DeiT) [6] yang bisa dilatih dengan efisien pada dataset yang lebih kecil seperti ImageNet-1k dengan memanfaatkan teknik *knowledge distillation*. Sementara He dkk. mengusulkan *Masked Autoencoder* (MAE) [7] yang belajar dengan cara merekonstruksi bagian gambar yang disembunyikan.

Setiap varian Vision Transformer ini punya kelebihan dan kekurangannya masing-masing. Ada yang lebih akurat tapi berat komputasinya, ada yang lebih cepat tapi butuh data lebih banyak. Makanya penting banget buat saya memahami karakteristik masing-masing model supaya bisa pilih yang paling cocok untuk kebutuhan.

1.2 Motivasi Perbandingan Model

Walaupun berbagai model Vision Transformer sudah menunjukkan hasil yang bagus di paper-paper penelitian, ada beberapa hal yang perlu diperhatikan untuk aplikasi praktis. Pertama, kebanyakan model ini dievaluasi menggunakan dataset yang sangat besar seperti ImageNet-21k atau JFT-300M. Masalahnya, dataset sebesar itu nggak selalu tersedia atau praktis dipakai di dunia nyata. Kedua, jarang ada penelitian yang membandingkan secara lengkap antara akurasi, ukuran model, dan kecepatan inferensi dalam satu eksperimen yang terkontrol.

Di praktiknya, memilih model bukan cuma soal akurasi tertinggi. Ada banyak faktor lain yang perlu dipertimbangkan:

1. **Efisiensi Komputasi**: Berapa banyak parameter dan operasi yang dibutuhkan? Ini penting banget kalau mau deploy di perangkat dengan sumber daya terbatas seperti smartphone atau perangkat IoT.

2. **Kecepatan Inferensi**: Berapa lama waktu yang dibutuhkan untuk memproses satu gambar? Kalau aplikasinya butuh real-time seperti autonomous driving atau video surveillance, kecepatan adalah hal yang sangat krusial.

3. **Efisiensi Data**: Seberapa banyak data yang dibutuhkan untuk training? Mengumpulkan dan memberi label pada data berskala besar itu mahal dan memakan banyak waktu

4. **Kemudahan Implementasi**: Seberapa mudah model di-*fine-tune* untuk domain spesifik? Beberapa arsitektur lebih mudah di-training ulang daripada yang lain.

1.3 Tujuan Eksperimen

Eksperimen ini bertujuan untuk membandingkan secara sistematis tiga arsitektur Vision Transformer yang berbeda. Secara spesifik, tujuannya adalah:

1. **Membandingkan Arsitektur Model**
   - Melihat perbedaan mendasar dalam desain arsitektur antara ViT, Swin Transformer, dan DeiT
   - Memahami bagaimana pilihan desain mempengaruhi karakteristik model
   - Mengidentifikasi kelebihan dan kekurangan masing-masing pendekatan
2. **Mengevaluasi Performa**
   - Mengukur dan membandingkan akurasi pada dataset test
   - Menghitung metrik evaluasi seperti precision, recall, dan F1-score
   - Menganalisis *confusion matrix* untuk melihat di mana model sering salah
   - Membandingkan kurva pembelajaran selama training
3. **Menganalisis Efisiensi**
   - Menghitung jumlah parameter dan ukuran model
   - Membandingkan kompleksitas komputasi dari masing-masing arsitektur
   - Melihat trade-off antara ukuran model dan akurasi yang dicapai
4. **Mengukur Kecepatan**
   - Mengukur waktu inferensi rata-rata per gambar
   - Menghitung berapa gambar yang bisa diproses per detik
   - Melihat apakah model cocok untuk aplikasi real-time atau tidak
5. **Analisis Trade-off**
   - Menganalisis hubungan antara akurasi, jumlah parameter, dan kecepatan
   - Mengidentifikasi model mana yang paling efisien
   - Mengevaluasi trade-off untuk berbagai skenario penggunaan
6. **Memberikan Rekomendasi**
   - Merekomendasikan model terbaik untuk akurasi maksimal
   - Merekomendasikan model terbaik untuk aplikasi dengan resource terbatas
   - Merekomendasikan model terbaik untuk aplikasi real-time
   - Memberikan panduan pemilihan model berdasarkan kebutuhan spesifik

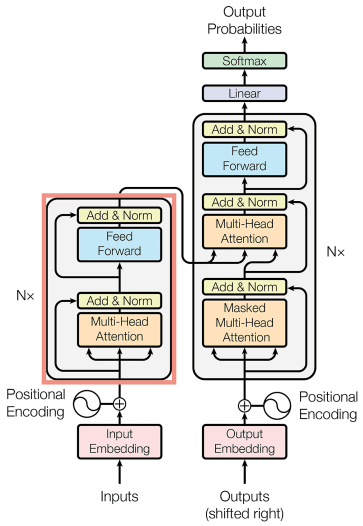
Dengan mencapai tujuan-tujuan di atas, diharapkan eksperimen ini bisa memberikan pemahaman yang lebih baik tentang karakteristik masing-masing arsitektur Vision Transformer, sekaligus memberikan panduan praktis untuk memilih model yang paling sesuai dengan kebutuhan

2 Landasan Teori

2.1 Transformer dan Mekanisme Self-Attention

2.1.1 Arsitektur Transformer

Transformer adalah arsitektur neural network yang diperkenalkan oleh Vaswani et al. [3] pada tahun 2017, yang pertama kali dirancang untuk tugas pemrosesan bahasa alami. Berbeda dengan arsitektur sekuensial seperti Recurrent Neural Networks (RNN) dan Long Short-Term Memory (LSTM), Transformer menghilangkan mekanisme rekurensi dan konvolusi, sepenuhnya bergantung pada mekanisme *attention* untuk menangkap dependensi global antara input dan output.



Gambar 1: Arsitektur Transformer dengan encoder dan decoder. Sumber: Vaswani et al. [3]

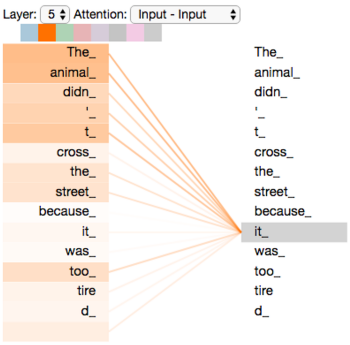
Arsitektur Transformer terdiri dari dua komponen utama: *encoder* dan *decoder*. Encoder memproses input sequence dan menghasilkan representasi abstrak, sementara decoder menggunakan representasi tersebut untuk menghasilkan output sequence. Namun, untuk aplikasi *computer vision* seperti klasifikasi gambar, umumnya hanya komponen encoder yang digunakan.

2.1.2 Mekanisme Self-Attention

Inti dari arsitektur Transformer adalah mekanisme *self-attention*, yang memungkinkan model untuk mempertimbangkan seluruh sequence input secara simultan dan menghitung representasi setiap elemen berdasarkan hubungannya dengan semua elemen lain dalam sequence. Secara matematis, mekanisme attention didefinisikan sebagai:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

di mana Q (Query), K (Key), dan V (Value) adalah matriks yang diproyeksikan dari input, dan d_k adalah dimensi dari key vector. Operasi ini menghitung *attention weights* yang menunjukkan seberapa besar setiap elemen dalam sequence harus memperhatikan elemen lainnya.



Gambar 2: Visualisasi mekanisme self-attention pada Transformer untuk tugas pemrosesan bahasa.

2.1.3 Multi-Head Attention

Untuk meningkatkan kemampuan model dalam menangkap berbagai jenis hubungan, Transformer menggunakan *multi-head attention*, yang menerapkan mekanisme attention secara paralel dengan beberapa set parameter yang berbeda (*heads*):

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \tag{2}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3}$$

di mana W_i^Q, W_i^K, W_i^V, dan W^O adalah parameter yang dapat dipelajari. Setiap *head* dapat mempelajari aspek yang berbeda dari hubungan antar elemen, seperti informasi posisi, semantik, atau sintaksis.

2.1.4 Position Encoding

Karena mekanisme self-attention tidak memiliki informasi tentang urutan atau posisi elemen dalam sequence, Transformer menggunakan *positional encoding* untuk menyediakan informasi posisi. Positional encoding ditambahkan ke input embeddings sebelum diproses oleh encoder. Vaswani et al. [3] mengusulkan penggunaan fungsi sinusoidal:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{4}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{5}$$

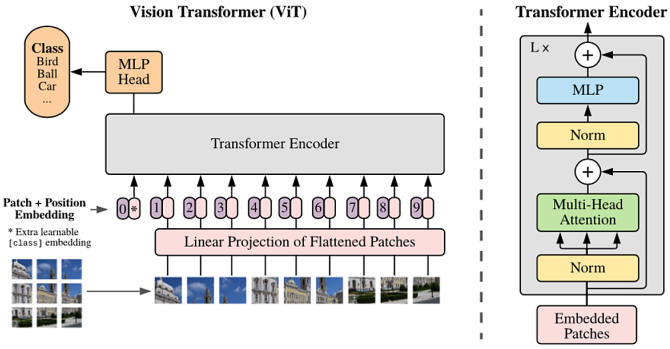
di mana *pos* adalah posisi dan i adalah dimensi. Alternatifnya, beberapa implementasi menggunakan *learned positional embeddings* yang dipelajari selama training.

2.2 Vision Transformer (ViT)

2.2.1 Arsitektur Umum

Vision Transformer (ViT), yang diperkenalkan oleh Dosovitskiy et al. [4], merupakan adaptasi langsung dari arsitektur Transformer untuk domain visi komputer. ViT mendemonstrasikan bahwa arsitektur

transformer murni, tanpa menggunakan konvolusi, dapat mencapai atau bahkan melampaui performa *state-of-the-art* CNN pada tugas klasifikasi gambar, terutama ketika dilatih pada dataset berskala besar.



Gambar 3: Arsitektur Vision Transformer (ViT).

2.2.2 Patch Embedding

Berbeda dengan pemrosesan teks yang secara natural berupa sequence of tokens, gambar perlu ditransformasi menjadi format yang sesuai untuk Transformer. ViT membagi gambar berukuran $H \times W \times C$ menjadi sequence of patches berukuran $P \times P$, menghasilkan $N = HW/P^2$ patches. Setiap patch kemudian di-flatten dan diproyeksikan ke dimensi D menggunakan linear projection:

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1\mathbf{E}; \mathbf{x}_p^2\mathbf{E}; ...; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos} \tag{6}$$

di mana $\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}$ adalah matriks embedding, $\mathbf{x}_p^i$ adalah patch ke-i, dan $\mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$ adalah positional embeddings. Token khusus $\mathbf{x}_{\text{class}}$ ditambahkan di awal sequence, yang representasinya pada output akan digunakan untuk klasifikasi.

2.2.3 Transformer Encoder

Setelah patch embedding, sequence diproses oleh L layer Transformer encoder. Setiap layer terdiri dari:

1. **Multi-Head Self-Attention (MSA)**: Memungkinkan setiap patch untuk berkomunikasi dengan semua patch lainnya

2. **Layer Normalization (LN)**: Normalisasi untuk stabilitas training
3. **Multi-Layer Perceptron (MLP)**: Two-layer feed-forward network dengan GELU activation
4. **Residual Connections**: Skip connections untuk aliran gradien yang lebih baik

Secara matematis, untuk layer ke-ℓ:

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1} \tag{7}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell \tag{8}$$

2.2.4 Classification Head

Output dari class token pada layer terakhir $\mathbf{z}_L^0$ digunakan sebagai representasi gambar dan diproses oleh classification head (typically an MLP) untuk menghasilkan prediksi kelas:

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0)\mathbf{W}_{\text{head}} \tag{9}$$

di mana $\mathbf{W}_{\text{head}} \in \mathbb{R}^{D \times K}$ adalah weight matrix untuk K kelas.

2.2.5 Varian ViT

Dosovitskiy et al. [4] memperkenalkan tiga varian utama ViT berdasarkan ukuran model:

- **ViT-Base**: 12 layers, hidden size 768, 12 attention heads (∼86M parameters)
- **ViT-Large**: 24 layers, hidden size 1024, 16 attention heads (∼307M parameters)
- **ViT-Huge**: 32 layers, hidden size 1280, 16 attention heads (∼632M parameters)

2.3 Swin Transformer

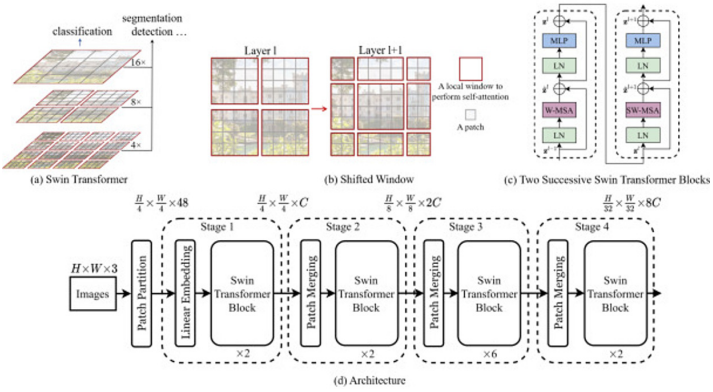
2.3.1 Motivasi dan Konsep Utama

Swin Transformer, yang diperkenalkan oleh Liu et al. [5], mengatasi beberapa keterbatasan ViT, terutama terkait efisiensi komputasi dan kemampuan untuk menangkap informasi multi-skala. Nama "Swin" merupakan singkatan dari *Shifted Windows*, yang merujuk pada mekanisme kunci dari arsitektur ini.

Dua inovasi utama Swin Transformer adalah:

1. **Hierarchical Feature Maps**: Seperti CNN, Swin Transformer membangun representasi hierarkis dengan resolusi yang menurun secara bertahap
2. **Shifted Window Attention**: Menghitung self-attention dalam window lokal yang bergeser antar layer untuk efisiensi dan komunikasi antar-window

2.3.2 Hierarchical Architecture



Gambar 4: Arsitektur hierarkis Swin Transformer. Sumber: Liu et al. [5]

Berbeda dengan ViT yang menggunakan patch size tetap dan menghasilkan feature maps dengan resolusi konstan, Swin Transformer mengadopsi arsitektur hierarkis dengan empat stage. Pada setiap stage, resolusi spasial berkurang setengah melalui *patch merging layer*, sementara jumlah channel meningkat dua kali lipat, mirip dengan arsitektur CNN seperti ResNet [2].

Dimulai dengan patch size 4×4 (lebih kecil dari ViT's 16×16), input gambar berukuran $H \times W \times 3$ diubah menjadi feature maps berukuran $\frac{H}{4} \times \frac{W}{4} \times C$ pada stage pertama. Pada stage berikutnya, patch merging mengurangi resolusi menjadi $\frac{H}{8} \times \frac{W}{8}$, $\frac{H}{16} \times \frac{W}{16}$, dan $\frac{H}{32} \times \frac{W}{32}$, dengan channel yang meningkat menjadi $2C$, $4C$, dan $8C$ respectively.

2.3.3 Window-based Multi-Head Self-Attention (W-MSA)

Untuk mengatasi kompleksitas kuadratik dari global self-attention terhadap ukuran gambar, Swin Transformer membatasi komputasi self-attention dalam window lokal non-overlapping. Dengan membagi feature map menjadi windows berukuran $M \times M$, kompleksitas komputasi untuk gambar dengan $h \times w$ patches menjadi:

$$\Omega(\text{MSA}) = 4hwC^2 + 2(hw)^2C \tag{10}$$

$$\Omega(\text{W-MSA}) = 4hwC^2 + 2M^2hwC \tag{11}$$

di mana C adalah dimensi channel. Untuk $M \ll h, w$ (typically $M = 7$), kompleksitas menjadi linear terhadap jumlah patches, bukan kuadratik.

2.3.4 Shifted Window Multi-Head Self-Attention (SW-MSA)

Meskipun W-MSA efisien, ia membatasi komunikasi antar-window. Untuk mengatasi ini, Swin Transformer menggunakan strategi *shifted window* yang menggeser window partitioning antara consecutive layers:

$$\hat{\mathbf{z}}^{\ell} = \text{W-MSA}(\text{LN}(\mathbf{z}^{\ell-1})) + \mathbf{z}^{\ell-1} \tag{12}$$

$$\mathbf{z}^{\ell} = \text{MLP}(\text{LN}(\hat{\mathbf{z}}^{\ell})) + \hat{\mathbf{z}}^{\ell} \tag{13}$$

$$\hat{\mathbf{z}}^{\ell+1} = \text{SW-MSA}(\text{LN}(\mathbf{z}^{\ell})) + \mathbf{z}^{\ell} \tag{14}$$

$$\mathbf{z}^{\ell+1} = \text{MLP}(\text{LN}(\hat{\mathbf{z}}^{\ell+1})) + \hat{\mathbf{z}}^{\ell+1} \tag{15}$$

Window shifting dilakukan dengan menggeser window sebesar $(\lfloor \frac{M}{2} \rfloor, \lfloor \frac{M}{2} \rfloor)$ pixels, yang memungkinkan koneksi antar-window pada layer sebelumnya.

2.3.5 Relative Position Bias

Berbeda dengan ViT yang menggunakan absolute positional embeddings, Swin Transformer menggunakan *relative position bias* $B \in \mathbb{R}^{M^2 \times M^2}$ yang ditambahkan ke attention computation:

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d}} + B\right)V \tag{16}$$

Relative position bias terbukti lebih efektif untuk transfer learning dan dapat beradaptasi dengan resolusi gambar yang berbeda [5].

2.3.6 Varian Swin Transformer

Liu et al. [5] memperkenalkan empat varian dengan kompleksitas yang berbeda:

- **Swin-T (Tiny)**: $C = 96$, layer numbers $= \{2, 2, 6, 2\}$ (∼29M parameters)
- **Swin-S (Small)**: $C = 96$, layer numbers $= \{2, 2, 18, 2\}$ (∼50M parameters)
- **Swin-B (Base)**: $C = 128$, layer numbers $= \{2, 2, 18, 2\}$ (∼86M parameters)
- **Swin-L (Large)**: $C = 192$, layer numbers $= \{2, 2, 18, 2\}$ (∼197M parameters)

2.4 Data-efficient Image Transformer (DeiT)

2.4.1 Motivasi dan Kontribusi

Data-efficient Image Transformer (DeiT), yang dikembangkan oleh Touvron et al. [6], mengatasi salah satu keterbatasan utama ViT: kebutuhan akan data pre-training yang sangat besar. Sementara ViT original memerlukan pre-training pada dataset berskala ratusan juta gambar (JFT-300M), DeiT menunjukkan bahwa Vision Transformer dapat dilatih secara efektif hanya menggunakan ImageNet-1k (∼1.3 juta gambar) melalui strategi training yang carefully designed.

Kontribusi utama DeiT meliputi:

1. Strategi data augmentation yang agresif
2. Regularization techniques yang efektif
3. Knowledge distillation melalui distillation token
4. Training procedure yang dioptimalkan

2.4.2 Arsitektur DeiT

Dari segi arsitektur, DeiT menggunakan struktur yang sama dengan ViT dengan beberapa modifikasi minor. Namun, DeiT memperkenalkan konsep *distillation token*, sebuah token tambahan selain class token yang khusus didesain untuk knowledge distillation:

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_{\text{dist}}; \mathbf{x}_p^1\mathbf{E}; ...; \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos} \tag{17}$$

di mana $\mathbf{x}_{\text{dist}}$ adalah distillation token yang berinteraksi dengan patch embeddings dan class token melalui self-attention layers.

2.4.3 Knowledge Distillation Strategy

DeiT menggunakan *distillation* untuk mentransfer pengetahuan dari teacher model (typically a convolutional network seperti RegNetY) ke student model (Vision Transformer). Berbeda dengan traditional distillation yang hanya menggunakan soft labels dari teacher, DeiT menggunakan *hard-label distillation* yang terbukti lebih efektif untuk Vision Transformers.

Fungsi loss untuk training adalah kombinasi dari tiga komponen:

$$\mathcal{L}_{\text{global}} = (1-\lambda)\mathcal{L}_{CE}(\psi(Z_s), y) + \lambda\mathcal{L}_{CE}(\psi(Z_s), y_t) \tag{18}$$

di mana $\mathcal{L}_{CE}$ adalah cross-entropy loss, ψ adalah softmax function, Z_s adalah logits dari student, y adalah ground-truth label, y_t adalah prediction dari teacher, dan λ adalah balancing parameter.

Untuk distillation token, loss tambahan ditambahkan:

$$\mathcal{L}_{\text{distill}} = \mathcal{L}_{CE}(\psi(Z_{\text{dist}}), y_t) \tag{19}$$

Inference dapat menggunakan rata-rata dari kedua classifiers atau hanya salah satu, tergantung pada preferensi accuracy-speed trade-off.

2.4.4 Training Strategy

Keberhasilan DeiT sangat bergantung pada training strategy yang carefully tuned [6]:

1. **Data Augmentation**:
   - Auto-Augment atau RandAugment

- Random Erasing
- CutMix dan Mixup dengan probabilitas tinggi

2. **Regularization**:
   - Stochastic Depth (drop path)
   - Label smoothing
   - Weight decay
3. **Optimization**:
   - AdamW optimizer
   - Cosine learning rate schedule dengan warmup
   - Gradient clipping

2.4.5 Varian DeiT

Touvron et al. [6] memperkenalkan beberapa varian dengan ukuran berbeda:

- **DeiT-Tiny**: 12 layers, hidden size 192, 3 heads (∼5M parameters)
- **DeiT-Small**: 12 layers, hidden size 384, 6 heads (∼22M parameters)
- **DeiT-Base**: 12 layers, hidden size 768, 12 heads (∼86M parameters)

Setiap varian juga memiliki versi distilled (dengan suffix ↑) yang menggunakan distillation token.

2.5 Perbandingan Arsitektur

Tabel 1 merangkum perbedaan kunci antara ketiga arsitektur Vision Transformer yang dibandingkan dalam eksperimen ini.

Tabel 1: Perbandingan Arsitektur Vision Transformer

Aspek	ViT	Swin Transformer	DeiT
Patch Size	16×16	4×4 (initial)	16×16
Attention Scope	Global	Local (windowed)	Global
Architecture Type	Flat (single scale)	Hierarchical (multi-scale)	Flat (single scale)
Complexity	$O(n^2)$	$O(n)$	$O(n^2)$
Position Encoding	Absolute learnable	Relative bias	Absolute learnable
Inductive Bias	Minimal	Moderate (locality)	Minimal
Pre-training Data	JFT-300M / ImageNet-21k	ImageNet-1k/21k	ImageNet-1k
Training Strategy	Standard	Standard	Distillation + augmentation
Special Tokens	[CLS] token	None	[CLS] + [DIST] tokens
Window Size	N/A	7×7 typical	N/A

2.6 Kelebihan dan Kekurangan

Tabel 2 merangkum kelebihan dan kekurangan teoritis dari masing-masing arsitektur berdasarkan desain dan karakteristik mereka.

Tabel 2: Kelebihan dan Kekurangan Masing-masing Arsitektur

Model	Kelebihan	Kekurangan
ViT	• Arsitektur sederhana dan elegan • Global receptive field dari layer pertama • Excellent scalability dengan data • Strong performance pada dataset besar • Mudah diimplementasikan dan dipahami	• Memerlukan data pre-training sangat besar • Kompleksitas kuadratik terhadap jumlah patches • Kurang efisien untuk high-resolution images • Tidak capture hierarchical features • Sulit dilatih dari scratch pada dataset kecil
Swin Transformer	• Kompleksitas linear terhadap image size • Hierarchical features untuk multi-scale tasks • Efisien untuk high-resolution images • Dapat digunakan sebagai backbone umum • Balance antara locality dan global modeling • Lebih baik untuk dense prediction tasks	• Arsitektur lebih kompleks • Window shifting menambah overhead • Implementasi lebih challenging • Relative position bias perlu careful tuning • Sedikit lebih lambat dalam inference per layer
DeiT	• Data-efficient (dapat dilatih pada ImageNet-1k) • Tidak memerlukan pre-training ekstensif • Knowledge distillation meningkatkan performa • Training procedure yang well-documented • Arsitektur sama dengan ViT (familiar) • Cocok untuk practitioners dengan data terbatas	• Memerlukan teacher model yang kuat • Training lebih kompleks (distillation) • Augmentation strategy sangat aggressive • Hyperparameter tuning lebih sensitif • Inference dapat lebih lambat (dua classifiers) • Masih memiliki kompleksitas kuadratik

3 Metodologi

3.1 Dataset

Penelitian ini menggunakan dataset **STL-10**. Dataset ini merupakan dataset standar yang sering digunakan untuk evaluasi algoritma *unsupervised* dan *supervised learning* pada visi komputer. Dataset ini terdiri dari gambar berwarna dengan resolusi asli 96×96 piksel, namun dalam penelitian ini diubah ukurannya (*resize*) menjadi 224×224 piksel untuk menyesuaikan dengan input model Vision Transformer.

Detail pembagian data yang digunakan dalam eksperimen ini adalah sebagai berikut:

- **Jumlah Kelas**: 10 kelas (*airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck*).
- **Original Split**: 5.000 gambar *train* dan 8.000 gambar *test*.
- **Custom Split**: Dalam eksperimen ini, data *train* dibagi kembali menjadi:
  - **Training Set**: 4.000 gambar.
  - **Validation Set**: 1.000 gambar (20% dari data *train* asli).
- **Test Set**: 8.000 gambar (menggunakan data *test* asli).

3.2 Preprocessing dan Augmentasi Data

Untuk meningkatkan kemampuan generalisasi model dan mencegah *overfitting*, diterapkan teknik *preprocessing* dan augmentasi data.

3.2.1 Preprocessing

Semua data (*train*, *val*, *test*) melalui tahapan berikut:

- **Resize**: Mengubah resolusi gambar menjadi 224×224 piksel (interpolasi bikubik).
- **ToTensor**: Mengonversi gambar menjadi format Tensor PyTorch.
- **Normalisasi**: Menormalisasi nilai piksel menggunakan rata-rata (*mean*) $[0.485, 0.456, 0.406]$ dan standar deviasi (*std*) $[0.229, 0.224, 0.225]$, sesuai dengan standar pre-training ImageNet.

3.2.2 Augmentasi

Khusus pada data *training*, diterapkan augmentasi tambahan secara acak setiap kali gambar dimuat:

- **Random Horizontal Flip**: Membalik gambar secara horizontal dengan peluang 50%.
- **Random Rotation**: Memutar gambar secara acak hingga ± 15 derajat.
- **Color Jitter**: Mengubah kecerahan (*brightness*) dan kontras (*contrast*) sebesar 0.2.

3.3 Konfigurasi Training

Ketiga model (DeiT Base, ViT Base, Swin Base) dilatih menggunakan konfigurasi *hyperparameter* yang sama untuk memastikan perbandingan yang adil. Pelatihan dilakukan menggunakan *fine-tuning* dari bobot *pre-trained* ImageNet.

Tabel 3: Konfigurasi Hyperparameter Training

Parameter	**Nilai**
Optimizer	AdamW
Learning Rate	3×10^{-5} (0.00003)
Batch Size	64
Epochs	20
Loss Function	Cross Entropy Loss
Learning Rate Scheduler	Cosine Annealing LR ($T_{max} = 20$)
Weight Decay	0.05
Mixed Precision	Ya (menggunakan `torch.cuda.amp.GradScaler`)

3.4 Library dan Framework

Implementasi dan eksperimen dilakukan menggunakan bahasa pemrograman **Python** dengan bantuan pustaka (*library*) berikut:

- **PyTorch**: Framework utama untuk pembangunan arsitektur dan pelatihan model *deep learning*.
- **torchvision**: Menyediakan dataset STL-10 dan modul transformasi gambar.
- **timm (PyTorch Image Models)**: Menyediakan implementasi model *state-of-the-art* Vision Transformer (DeiT, ViT, Swin) beserta bobot *pre-trained*-nya.
- **NumPy & Pandas**: Digunakan untuk manipulasi data numerik dan pencatatan *log* hasil eksperimen.
- **Matplotlib & Seaborn**: Digunakan untuk visualisasi grafik *loss*, akurasi, dan *confusion matrix*.
- **Scikit-learn**: Digunakan untuk perhitungan metrik evaluasi seperti *classification report* dan *confusion matrix*.

3.5 Spesifikasi Hardware

Seluruh proses pelatihan dan evaluasi model dilakukan pada lingkungan komputasi berkinerja tinggi dengan spesifikasi sebagai berikut:

- **GPU**: NVIDIA A40 (VRAM 44.42 GB)
- **CPU**: 96 Cores
- **RAM**: 50.51 GB
- **Platform**: Linux (Environment berbasis Cloud/RunPod)

3.6 Metrik Evaluasi

Untuk mengukur kinerja dan efisiensi dari model yang dibandingkan, penelitian ini menggunakan beberapa metrik evaluasi standar dalam klasifikasi gambar.

3.6.1 Akurasi (Accuracy)

Akurasi adalah rasio antara jumlah prediksi yang benar (positif dan negatif) dengan keseluruhan jumlah data. Akurasi dihitung dengan persamaan:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{20}$$

Dimana:

- TP (*True Positive*): Data positif yang diprediksi benar positif.
- TN (*True Negative*): Data negatif yang diprediksi benar negatif.
- FP (*False Positive*): Data negatif yang diprediksi salah sebagai positif.
- FN (*False Negative*): Data positif yang diprediksi salah sebagai negatif.

3.6.2 Precision, Recall, dan F1-Score

Selain akurasi, digunakan juga metrik *Precision*, *Recall*, dan *F1-Score* untuk memberikan gambaran yang lebih komprehensif, terutama jika terdapat ketidakseimbangan kelas.

Precision mengukur seberapa akurat model dalam memprediksi kelas positif:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{21}$$

Recall (atau Sensitivitas) mengukur kemampuan model untuk menemukan semua data positif yang sebenarnya:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{22}$$

F1-Score adalah rata-rata harmonik dari *Precision* dan *Recall*, yang memberikan keseimbangan antara keduanya:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{23}$$

3.6.3 Confusion Matrix

Confusion Matrix digunakan untuk memvisualisasikan performa model dalam bentuk tabel matriks, yang menunjukkan perbandingan antara label prediksi model dengan label sebenarnya (*ground truth*) untuk setiap kelas.

3.6.4 Efisiensi Komputasi

Selain performa akurasi, efisiensi model juga dievaluasi menggunakan parameter berikut:

- **Training Time**: Total waktu yang dibutuhkan untuk melatih model selama jumlah *epoch* yang ditentukan.
- **Inference Time**: Rata-rata waktu yang dibutuhkan model untuk memproses satu gambar saat pengujian.
- **Throughput**: Jumlah gambar yang dapat diproses oleh model dalam satu detik, dihitung dengan rumus:

$$\text{Throughput} = \frac{\text{Jumlah Total Gambar}}{\text{Total Waktu Inferensi (detik)}} \tag{24}$$

- **Ukuran Model**: Kompleksitas model diukur berdasarkan jumlah parameter yang dapat dilatih (*trainable parameters*) dan ukuran memori yang dibutuhkan untuk menyimpan bobot model (dalam MB).

4 Hasil dan Analisis

Bab ini menyajikan hasil evaluasi tiga model Vision Transformer (ViT Base, DeiT Base, Swin Base) pada dataset STL-10 dengan 20 epoch pelatihan, meliputi perbandingan parameter, metrik performa, waktu inferensi, visualisasi, dan analisis mendalam

4.1 Perbandingan Jumlah Parameter

Tabel 4: Perbandingan Jumlah Parameter dan Ukuran Model

Model	Total Parameters	Model Size (MB)
DeiT Base	85,806,346	327.33
ViT Base	85,806,346	327.33
Swin Base	86,753,474	332.44

DeiT Base dan ViT Base memiliki parameter identik (85.8M, 327.33 MB), sementara Swin Base sedikit lebih besar (86.7M, 332.44 MB). Perbedaan hanya 1.1%, sehingga ketiga model setara dalam kompleksitas arsitektur.

4.2 Perbandingan Metrik Performa

Tabel 5: Perbandingan Metrik Performa pada Test Set

Model	Accuracy (%)	Macro F1-Score	Training Time (min)
DeiT Base	98.11	0.9811	5.5
ViT Base	99.08	0.9907	5.5
Swin Base	99.36	0.9936	8.5

Swin Base mencapai akurasi tertinggi (99.36%), diikuti ViT Base (99.08%) dan DeiT Base (98.11%). Perbedaan akurasi kecil (1.25%) menunjukkan semua model efektif untuk STL-10. Swin Base memerlukan waktu training 54% lebih lama (8.5 vs 5.5 menit) karena kompleksitas shifted window attention.

4.3 Perbandingan Waktu Inferensi

Tabel 6: Perbandingan Waktu Inferensi dan Throughput

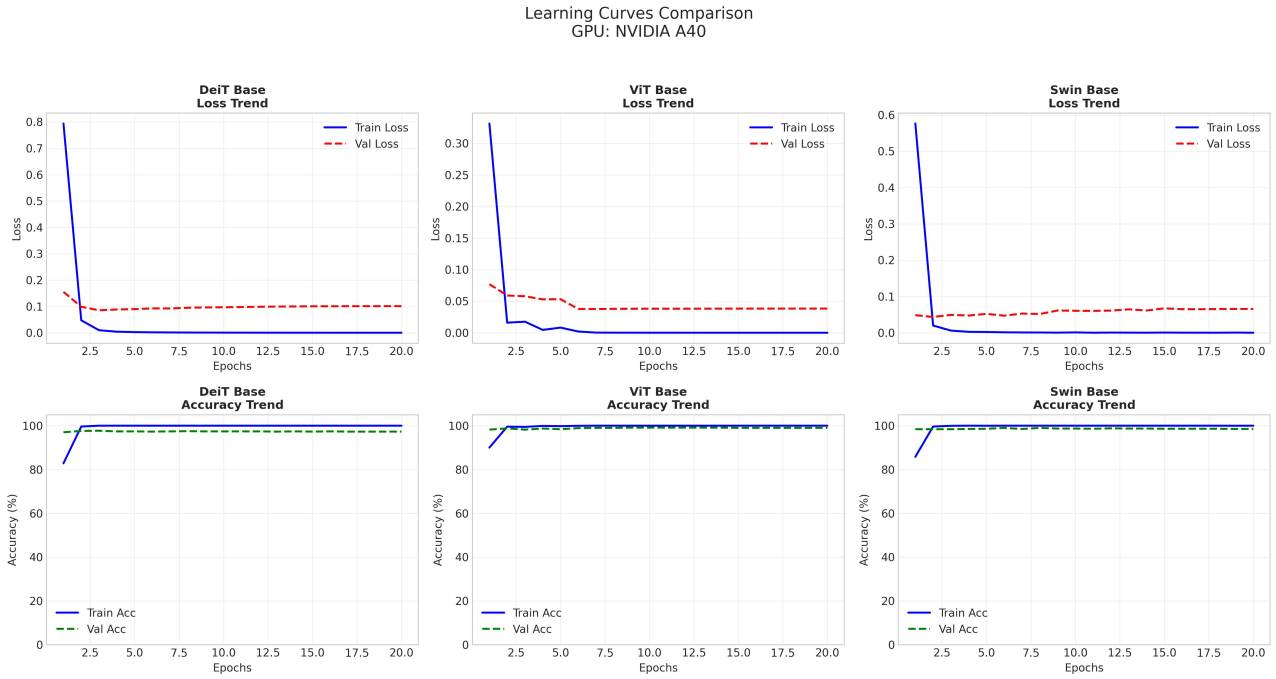
Model	Inference Time (ms/image)	Throughput (images/s)
DeiT Base	2.56	389.9
ViT Base	2.57	388.8
Swin Base	3.42	292.6

DeiT Base dan ViT Base memiliki kecepatan hampir identik (388-389 images/s, 2.56 ms/gambar), sedangkan Swin Base 33% lebih lambat (292.6 images/s, 3.42 ms/gambar) karena overhead shifted window attention.

4.4 Visualisasi

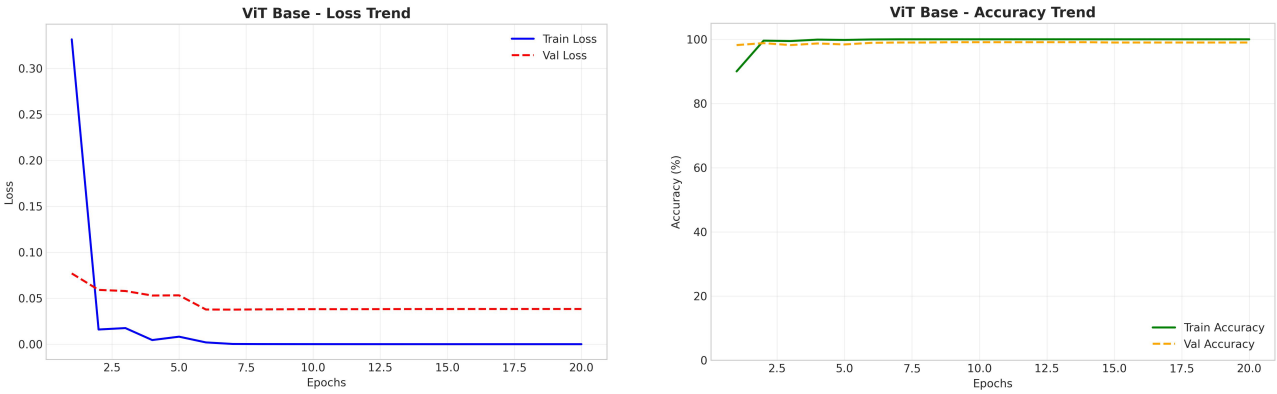
4.4.1 Kurva Learning

Gambar 5 menunjukkan: (1) Semua model konvergen dengan training loss mendekati 0 dan training accuracy 100%, (2) Validation accuracy: Swin Base (99.36%), ViT Base (99.08%), DeiT Base (98.11%),

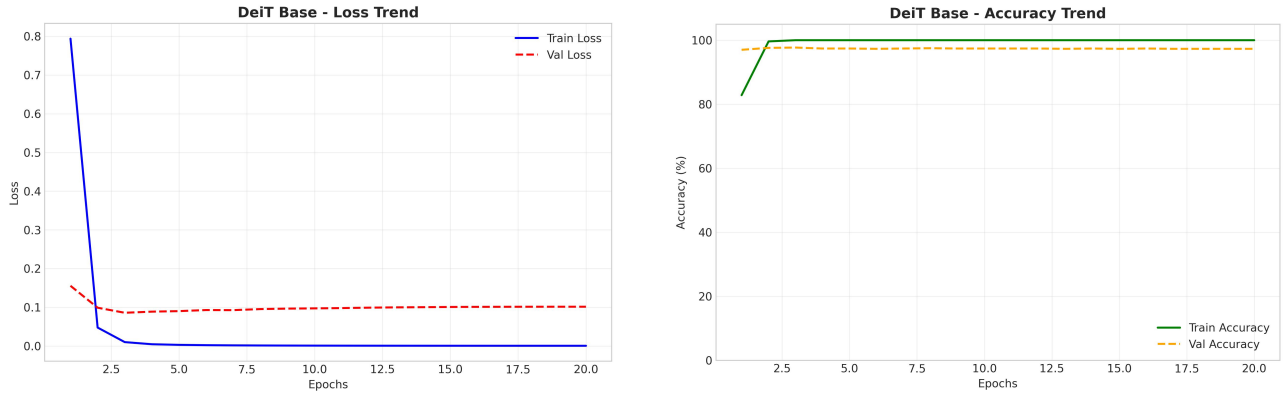


Gambar 5: Kurva Learning: Loss dan Accuracy pada Training dan Validation Set

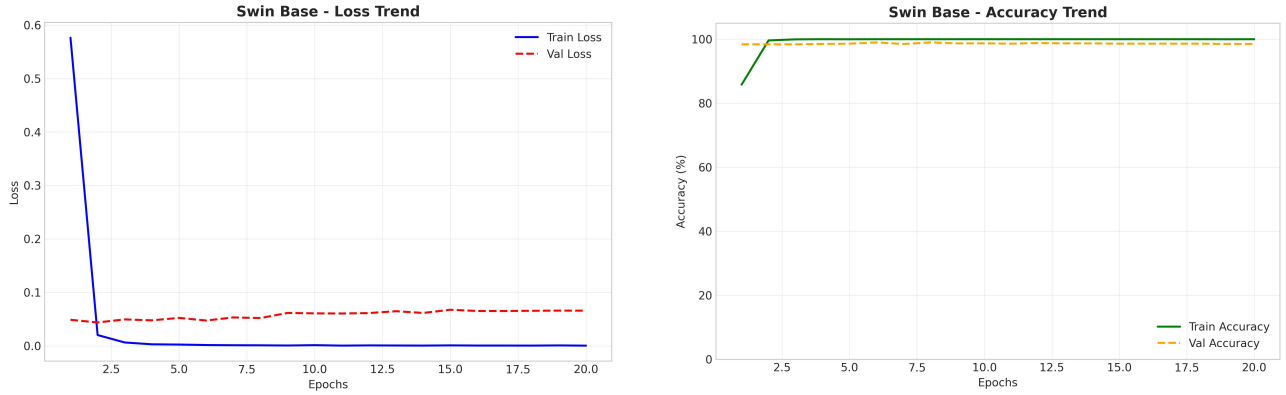
(3) DeiT Base memiliki validation loss lebih tinggi (0.10 vs 0.038-0.065), mengindikasikan gap yang lebih besar antara training dan validation, (4) Overfitting minimal pada semua model.



Gambar 6: Kurva Loss dan Accuracy untuk ViT Base

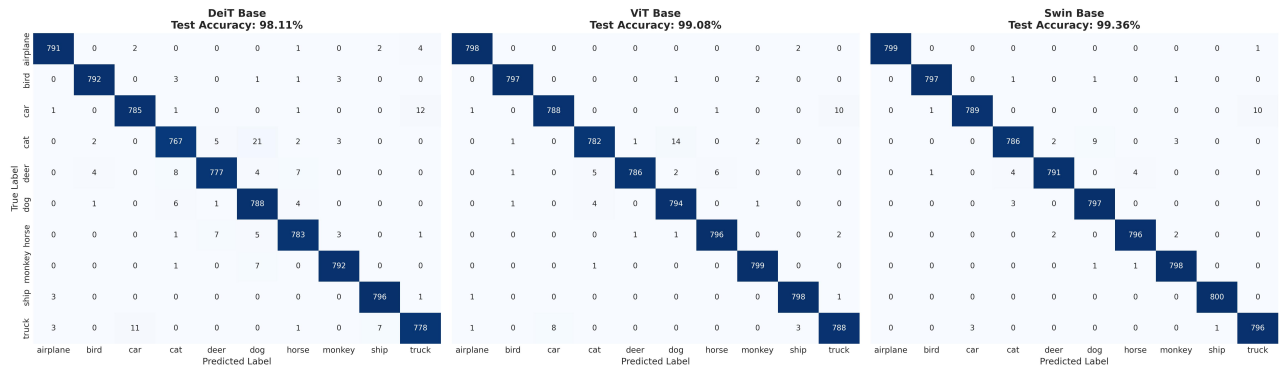


Gambar 7: Kurva Loss dan Accuracy untuk DeiT Base



Gambar 8: Kurva Loss dan Accuracy untuk Swin Base

4.4.2 Confusion Matrix



Gambar 9: Confusion Matrix untuk DeiT Base, ViT Base, dan Swin Base

Confusion matrix (Gambar 9) menunjukkan Swin Base memiliki kesalahan minimal dengan prediksi terkonsentrasi pada diagonal utama. Kesalahan klasifikasi utama pada semua model terjadi antara kelas *cat* dan *dog* yang memiliki kemiripan visual tinggi, dengan Swin Base menunjukkan performa terbaik dalam membedakannya.

4.5 Analisis Mendalam

4.5.1 Analisis Performa Model

Swin Base mencapai akurasi tertinggi (99.36%) karena arsitektur hierarki dengan shifted window attention yang efektif menangkap fitur multi-scale pada resolusi menengah STL-10 (96×96). ViT Base (99.08%) menawarkan keseimbangan optimal dengan arsitektur sederhana dan global receptive field. DeiT Base (98.11%) menunjukkan performa sedikit lebih rendah, kemungkinan karena distillation strategy yang kurang optimal untuk fine-tuning pada dataset kecil.

4.5.2 Trade-off Akurasi, Parameter, dan Kecepatan

Terdapat trade-off yang jelas: Swin Base memberikan akurasi maksimal (+1.25% vs DeiT) dengan biaya kecepatan (-25% throughput). ViT Base menawarkan sweet spot: akurasi tinggi (99.08%, hanya -0.28% vs Swin) dengan kecepatan cepat (33% lebih cepat dari Swin). DeiT Base optimal untuk aplikasi real-time dengan throughput tertinggi (389.9 images/s).

Rekomendasi deployment: (1) Cloud/server dengan prioritas akurasi: Swin Base, (2) Aplikasi umum dengan balance akurasi-kecepatan: ViT Base, (3) Real-time/edge devices: DeiT Base.

4.5.3 Kesesuaian Model dengan Dataset

STL-10 memiliki karakteristik: resolusi menengah (96×96), 10 kelas, 5,000 training images (terbatas), dan variasi intra-class tinggi.

Swin Base paling sesuai karena hierarchical processing optimal untuk resolusi menengah dan kemampuan menangkap multi-scale features untuk membedakan kelas mirip. **ViT Base** efektif dengan global receptive field dan transfer learning yang baik dari ImageNet-21k. **DeiT Base** memiliki ruang optimasi melalui penyesuaian distillation strategy dan data augmentation untuk dataset kecil.

Semua model menunjukkan arsitektur Vision Transformer sangat efektif untuk STL-10, dengan transfer learning dari ImageNet-21k berperan krusial mengingat keterbatasan training data.

5 Kesimpulan dan Saran

5.1 Kesimpulan

Berdasarkan eksperimen perbandingan tiga arsitektur Vision Transformer (ViT Base, DeiT Base, dan Swin Base) pada dataset STL-10, dapat disimpulkan bahwa:

1. Ketiga model menunjukkan performa excellent dengan akurasi >98%. Swin Base mencapai akurasi tertinggi (99.36%, F1: 0.9936), diikuti ViT Base (99.08%, F1: 0.9907), dan DeiT Base (98.11%, F1: 0.9811)

2. Kompleksitas model hampir setara (85-86 juta parameter, ukuran 327-332 MB), namun terdapat perbedaan signifikan pada efisiensi inferensi: DeiT/ViT Base 33% lebih cepat (388-389 images/s) dibanding Swin Base (292.6 images/s)

3. Swin Base memerlukan waktu pelatihan 54% lebih lama (8.5 vs 5.5 menit) karena kompleksitas shifted window attention, namun memberikan akurasi terbaik untuk dataset STL-10 dengan resolusi menengah (96×96)

4. Terdapat trade-off yang jelas: Swin Base optimal untuk akurasi maksimal, ViT Base menawarkan keseimbangan terbaik, dan DeiT Base unggul dalam kecepatan inferensi

Tabel 7: Rekomendasi Model untuk Berbagai Aplikasi

Prioritas	Model	Use Case & Alasan
Akurasi Maksimal	Swin Base	*Medical imaging, quality control, autonomous driving* Akurasi 99.36%, confusion matrix terbaik, cocok untuk cloud deployment
Efisiensi Optimal	ViT Base	*E-commerce, content moderation, general CV tasks* Keseimbangan terbaik: akurasi 99.08% dengan kecepatan tinggi (388.8 img/s)
Real-time	DeiT Base	*Video surveillance, mobile apps, robotics, AR/VR* Throughput tertinggi (389.9 img/s), dapat di-deploy pada edge devices

5.2 Rekomendasi Model Berdasarkan Use Case

5.3 Saran untuk Pengembangan Lebih Lanjut

1. **Optimasi Deployment**: Implementasi quantization (INT8/FP16), knowledge distillation, dan pruning untuk meningkatkan efisiensi tanpa degradasi akurasi signifikan
2. **Dataset Diversity**: Evaluasi pada dataset dengan karakteristik berbeda (resolusi lebih tinggi, lebih banyak kelas, class imbalance, fine-grained categories)
3. **Arsitektur Terbaru**: Eksplorasi varian modern seperti BEiT, MAE, DINOv2 (self-supervised), dan lightweight models (MobileViT, EfficientFormer)
4. **Ensemble & Robustness**: Implementasi model ensemble, adversarial training, dan evaluasi out-of-distribution detection
5. **Interpretability**: Attention visualization, Grad-CAM, dan analisis feature representations untuk memahami keputusan model
6. **Hyperparameter Tuning**: Extensive search untuk learning rate schedule, augmentation strategy, dan regularization parameters, terutama optimasi distillation strategy untuk DeiT pada STL-10

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 25, 2012.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[4] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[5] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 10 012–10 022. [Online]. Available: https://arxiv.org/abs/2103.14030

[6] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International Conference on Machine Learning (ICML)*. PMLR, 2021, pp. 10 347–10 357. [Online]. Available: https://arxiv.org/abs/2012.12877

[7] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, "Masked autoencoders are scalable vision learners," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 16 000–16 009. [Online]. Available: https://arxiv.org/abs/2111.06377

Lampiran

A. Source Code

A.1 Import dan Setup Environment

```python
import os
import time
import gc
import psutil
import random
from datetime import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split
import torchvision.transforms as transforms
from torchvision.datasets import STL10
import timm
from timm import create_model
from sklearn.metrics import (accuracy_score, precision_recall_fscore_support,
                             confusion_matrix, classification_report)
import warnings

# Setup Tampilan & Warning
warnings.filterwarnings('ignore')
plt.style.use('seaborn-v0_8-whitegrid')
pd.set_option('display.max_columns', None)

# Random Seed untuk Reproducibility
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True

set_seed(42)
```

Kode 1: Import Libraries dan Setup Environment

A.2 Konfigurasi Model

```python
# Hyperparameters (Disesuaikan untuk GPU A40)
BATCH_SIZE = 64      # Aman untuk model Base di GPU 40GB+
NUM_WORKERS = 8      # Optimal untuk 9 vCPU
EPOCHS = 20          # Jumlah epoch training
LR = 3e-5            # Learning rate kecil untuk fine-tuning
IMG_SIZE = 224

# Daftar Model (Varian BASE)
MODELS_CONFIG = {
    'DeiT Base': 'deit_base_patch16_224',       # ~86M Params
    'ViT Base':  'vit_base_patch16_224',        # ~86M Params
    'Swin Base': 'swin_base_patch4_window7_224' # ~88M Params
}
```

Kode 2: Hyperparameters dan Model Configuration

A.3 Data Loading dan Augmentation

```python
imagenet_mean = [0.485, 0.456, 0.406]
imagenet_std = [0.229, 0.224, 0.225]

# Augmentasi Data
train_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(15),
    transforms.ColorJitter(brightness=0.2, contrast=0.2),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_mean, imagenet_std)
])

val_test_transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(imagenet_mean, imagenet_std)
])

```

```python
# Load STL-10 Dataset
train_full = STL10(root='./data', split='train',
                   download=True, transform=train_transform)
test_ds = STL10(root='./data', split='test',
                download=True, transform=val_test_transform)

# Split Train/Val (80% Train, 20% Val)
train_size = int(0.8 * len(train_full))
val_size = len(train_full) - train_size
train_ds, val_ds = random_split(train_full, [train_size, val_size])

# Dataloaders
train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE,
                          shuffle=True, num_workers=NUM_WORKERS,
                          pin_memory=True, prefetch_factor=2)
val_loader = DataLoader(val_ds, batch_size=BATCH_SIZE,
                        shuffle=False, num_workers=NUM_WORKERS,
                        pin_memory=True)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE,
                         shuffle=False, num_workers=NUM_WORKERS,
                         pin_memory=True)
```

Kode 3: Dataset Loading dan Data Augmentation

A.4 Training Loop

```python
def train_one_epoch(model, loader, criterion, optimizer, scaler):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        with torch.cuda.amp.autocast():
            outputs = model(images)
            loss = criterion(outputs, labels)

        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()

        running_loss += loss.item() * images.size(0)
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    return running_loss / total, 100. * correct / total
```

Kode 4: Main Training Loop

B. Output Training Log

B.1 DeiT Base Training Log

Tabel 8: Training Log DeiT Base (20 Epochs)

Epoch	Train Loss	Train Acc (%)	Val Loss	Val Acc (%)
1	0.7940	82.83	0.1556	97.00
2	0.0477	99.63	0.0990	97.60
3	0.0101	100.00	0.0859	97.70
4	0.0046	100.00	0.0888	97.40
5	0.0030	100.00	0.0902	97.40
6	0.0022	100.00	0.0929	97.30
7	0.0017	100.00	0.0927	97.40
8	0.0014	100.00	0.0954	97.50
9	0.0012	100.00	0.0966	97.40
10	0.0010	100.00	0.0973	97.40
20	0.0006	100.00	0.1015	97.30
Test Accuracy: 98.11% | Throughput: 389.9 img/s				

B.2 ViT Base Training Log

Tabel 9: Training Log ViT Base (20 Epochs)

Epoch	Train Loss	Train Acc (%)	Val Loss	Val Acc (%)
1	0.3315	90.00	0.0770	98.20
2	0.0160	99.58	0.0591	98.80
3	0.0176	99.45	0.0579	98.20
4	0.0046	99.90	0.0530	98.70
5	0.0082	99.80	0.0532	98.40
6	0.0020	99.95	0.0378	98.90
7	0.0003	100.00	0.0376	99.00
8	0.0002	100.00	0.0379	99.00
9	0.0001	100.00	0.0380	99.10
10	0.0001	100.00	0.0382	99.10
20	0.0001	100.00	0.0383	99.00
Test Accuracy: 99.08% | Throughput: 388.8 img/s				

B.3 Swin Base Training Log

Tabel 10: Training Log Swin Base (20 Epochs)

Epoch	Train Loss	Train Acc (%)	Val Loss	Val Acc (%)
1	0.5762	85.80	0.0488	98.40
2	0.0203	99.63	0.0438	98.40
3	0.0064	99.95	0.0495	98.40
4	0.0030	100.00	0.0478	98.50
5	0.0025	99.98	0.0524	98.60
6	0.0016	100.00	0.0474	99.00
7	0.0012	100.00	0.0532	98.50
8	0.0011	100.00	0.0520	99.00
9	0.0007	100.00	0.0616	98.70
10	0.0014	100.00	0.0609	98.70
20	0.0005	100.00	0.0659	98.50
Test Accuracy: 99.36% | Throughput: 292.6 img/s				

Google Drive Folder isi Weights

Link Google Drive Berisi Weights

https://drive.google.com/drive/folders/1ZFYb8suViXVsruRy3Uq4W_j-8iY0hJKi?usp=sharing

Link Github Repository

https://github.com/pataanggs/VisionTransformer-Comparison

Percakapan dengan LLM

Link Percakapan

Untuk transparansi penuh, beberapa percakapan penting dapat diakses di:

- Gemini: https://gemini.google.com/share/f1c3bf2b475b
- Claude: https://claude.ai/share/8a89ddbb-bc92-4593-82e2-f6c23d5cd7a3
- Claude: https://claude.ai/share/1f8a9633-8b2d-4f14-8f1b-92ac054b5024
- Claude: https://claude.ai/share/dad30974-f724-4fce-b270-ebecbd659f9f