LAMPIRAN

1 Lampiran A: Detail Arsitektur HTR Recognizer

Lampiran ini menyajikan spesifikasi lengkap arsitektur Hybrid CNN-Transformer yang digunakan sebagai HTR Recognizer dalam framework GAN-HTR. Detail ini melengkapi ringkasan high-level yang disajikan pada Bab III (Metodologi).

1.1 Spesifikasi Lengkap CNN Backbone

Tabel 1: Detail arsitektur CNN Backbone untuk ekstraksi fitur visual

Layer	Type	Spec	Output Shape	Params
Block 1: Initial Feature Extraction				
conv1_1	Conv2D	32 filters, 3×3, stride 1, ReLU	(H, W, 32)	896
conv1_2	Conv2D	32 filters, 3×3, stride 1, ReLU	(H, W, 32)	9,248
bn1	BatchNorm	-	(H, W, 32)	128
pool1	MaxPool2D	2×2	(H/2, W/2, 32)	0
Block 2: Mid-Level Features				
conv2_1	Conv2D	64 filters, 3×3, stride 1, ReLU	(H/2, W/2, 64)	18,496
conv2_2	Conv2D	64 filters, 3×3, stride 1, ReLU	(H/2, W/2, 64)	36,928
bn2	BatchNorm	-	(H/2, W/2, 64)	256
pool2	MaxPool2D	2×2	(H/4, W/4, 64)	0
Block 3: High-Level Features with Residual				
conv3_1	Conv2D	128 filters, 3×3, stride 1, ReLU	(H/4, W/4, 128)	73,856
conv3_2	Conv2D	128 filters, 3×3, stride 1, ReLU	(H/4, W/4, 128)	147,584
residual3	Add	Skip connection from conv3_1	(H/4, W/4, 128)	0
bn3	BatchNorm	-	(H/4, W/4, 128)	512
pool3	MaxPool2D	2×2	(H/8, W/8, 128)	0
Block 4: Deep Features				
conv4_1	Conv2D	256 filters, 3×3, stride 1, ReLU	(H/8, W/8, 256)	295,168
conv4_2	Conv2D	256 filters, 3×3, stride 1, ReLU	(H/8, W/8, 256)	590,080
residual4	Add	Skip connection from conv4_1	(H/8, W/8, 256)	0
bn4	BatchNorm	-	(H/8, W/8, 256)	1,024
pool4	MaxPool2D	2×2	(H/16, W/16, 256)	0
Block 5: Final Feature Extraction				
conv5_1	Conv2D	512 filters, 3×3, stride 1, ReLU	(H/16, W/16, 512)	1,180,160
conv5_2	Conv2D	512 filters, 3×3, stride 1, ReLU	(H/16, W/16, 512)	2,359,808
bn5	BatchNorm	-	(H/16, W/16, 512)	2,048
Projection Layer (proj_ln)				
proj	Conv2D	512 filters, 1×1 (projection)	(H/16, W/16, 512)	262,656
proj_ln	LayerNorm	-	(H/16, W/16, 512)	1,024
reshape	Reshape	Flatten spatial dims	(W/16, 512)	0
			Total CNN Parameters:	**4,979,872**

Catatan Implementasi:

- Input shape: (128, 1024, 1) — grayscale images

- Residual connections: Setiap 2 conv layers untuk mencegah vanishing gradient
- BatchNorm: Setelah setiap block untuk stabilitas training
- Activation: ReLU untuk non-linearity
- Pooling strategy: Max pooling 2×2 untuk spatial downsampling progresif
- Final output: (64, 512) sequence untuk Transformer input

1.2 Spesifikasi Lengkap Transformer Encoder

Tabel 2: Detail arsitektur Transformer Encoder untuk sequence modeling

Component	**Specification**	**Parameters**	**Notes**
Positional Encoding			
Encoding Type	Sinusoidal	0 (learned)	Fixed sin/cos encoding
Max Sequence Len	256	-	Supports up to 256 timesteps
Transformer Layer Configuration (6 layers)			
Num Layers	6	-	Stacked encoder layers
Model Dimension (d_{model})	512	-	Feature dimension
Num Attention Heads	8	-	Multi-head attention
Head Dimension (d_k)	64	-	d_{model} / num_heads
FFN Dimension (d_{ff})	2048	-	4× d_{model}
Dropout Rate	0.20	-	Applied to attention & FFN
Per-Layer Components			
Multi-Head Attention	8 heads, 64 dims each	1,048,576	Q, K, V projections + output
LayerNorm (post-attn)	d_{model}=512	1,024	Normalization
FFN Layer 1	512 → 2048, ReLU	1,050,624	Expansion
FFN Layer 2	2048 → 512	1,049,088	Projection back
LayerNorm (post-FFN)	d_{model}=512	1,024	Normalization
Residual Connections	2 per layer	0	Skip connections
		Total per Layer:	3,150,336
		Total 6 Layers:	**18,902,016**

1.3 CTC Output Layer

Tabel 3: Detail CTC output layer dan decoding

Component	**Specification**	**Parameters**
Output Dense Layer	512 $\rightarrow$ 95 (vocab size)	48,735
Activation	Softmax	0
Vocab Size	95 characters	-
	Total Output Layer:	**48,735**

Character Set (95 characters):

- Lowercase: a-z (26 chars)
- Uppercase: A-Z (26 chars)
- Digits: 0-9 (10 chars)
- Punctuation: 30 symbols (.,:;!?'"-/()[]@#$%&*+<>=_~)
- Special: BLANK token, SPACE, newline

1.4 Model Summary

Tabel 4: Ringkasan total parameter HTR Recognizer

Component	**Parameters**
CNN Backbone	4,979,872
Transformer Encoder (6 layers)	18,902,016
CTC Output Layer	48,735
Total Trainable Parameters	**23,930,623**
Model Size (FP32)	**$\sim$96 MB**

2 Lampiran B: Detail Konfigurasi Training Recognizer

2.1 Optimizer Configuration

Tabel 5: Detail konfigurasi AdamW optimizer

Parameter	**Value**	**Justification**
Base Learning Rate	$3{\times}10^{-4}$	Optimal for Transformer (Vaswani et al. 2017)
Beta1 (β_1)	0.9	Standard Adam momentum
Beta2 (β_2)	0.999	Standard Adam RMSProp term
Epsilon (ε)	$1{\times}10^{-8}$	Numerical stability
Weight Decay	$1{\times}10^{-4}$	L2 regularization (decoupled from gradient)
Gradient Clipping	clipnorm=1.0	Prevent exploding gradients

2.2 Learning Rate Schedule

Tabel 6: Cosine annealing learning rate schedule

Parameter	**Value**	**Description**
Schedule Type	Cosine Annealing	Smooth decay without oscillation
Warmup Steps	1000	Linear warmup from 0 to base LR
Max Learning Rate	$3{\times}10^{-4}$	Reached after warmup
Min Learning Rate	$1{\times}10^{-6}$	Final LR at end of training
Total Steps	50,000	Based on dataset size and epochs
Restart	No	Single cosine curve

2.3 Data Augmentation

Tabel 7: Detail data augmentation pipeline

Augmentation	**Parameters**	**Probability**
Photometric Augmentation		
Brightness Adjustment	factor [0.8, 1.2]	0.5
Contrast Adjustment	factor [0.8, 1.2]	0.5
Gamma Correction	gamma [0.8, 1.2]	0.3
Noise Injection		
Gaussian Noise	mean=0, std [0.01, 0.05]	0.4
Salt & Pepper Noise	amount [0.001, 0.01]	0.2
Geometric Augmentation		
Elastic Transform	alpha [50, 150], sigma=5	0.2
Slight Rotation	angle [-2°, +2°]	0.3
Slight Shear	shear [-0.1, +0.1]	0.2

2.4 Regularization Techniques

Tabel 8: Regularization strategies

Technique	**Configuration**	**Purpose**
Dropout	rate=0.20	Applied after attention and FFN layers
Label Smoothing	ε=0.1	Soft targets for CTC loss
Weight Decay	1×10^{-4}	L2 regularization on model weights
Early Stopping	patience=15 epochs	Prevent overfitting, monitor val CER
Model Checkpoint	Save best val CER	Keep best performing weights

3 Lampiran C: Detail Implementasi Numerik dan Efisiensi

3.1 Stabilisasi Numerik CTC Loss

Tabel 9: Teknik stabilisasi numerik untuk CTC loss computation

Teknik	**Implementation**	**Rationale**
Log-Space Computation	Use log-probabilities	Prevent underflow in probability multiplication
LogSumExp Trick	Numerically stable summation	Avoid overflow/underflow in exponential
Label Smoothing	ε=0.1 on targets	Prevent overconfident predictions
Loss Normalization	Divide by sequence length	Consistent loss across varying lengths
Value Clipping	Max CTC loss = 50.0	Prevent extreme values in logging
Precision	FP32 (pure float32)	Critical for log-space stability
Gradient Clipping	clipnorm=1.0	Prevent gradient explosion

3.2 Justifikasi Presisi Numerik FP32

Tabel 10: Perbandingan FP16/Mixed vs Pure FP32 untuk CTC computation

Aspect	**FP16/Mixed Precision**	**Pure FP32**
CTC Log-Prob	Underflow saat exp(-50)	Stable hingga exp(-100)
Gradient Balance	Loss component dominance	Balanced multi-loss gradients
Convergence	Unstable, oscillating	Smooth, stable convergence
Visual Quality	Suboptimal	Superior quality
Speed	+30-40% faster	Baseline speed
Memory	-40% usage	Baseline memory
Decision	**Pure FP32** chosen: stability > speed trade-off	

3.3 Efisiensi Komputasi

Tabel 11: Profiling waktu eksekusi per komponen (per batch)

Component	**Time (ms)**	**Percentage**	**GPU Util**
Data Loading	15	15%	CPU-bound
Generator Forward	25	25%	85%
Discriminator Forward	10	10%	75%
Recognizer Forward (frozen)	20	20%	80%
Loss Computation	5	5%	60%
Backward Pass	20	20%	90%
Optimizer Step	5	5%	70%
Total per Iteration	**100**	**100%**	**Avg 78%**

Optimization Opportunities:

- Data loading: Implemented prefetch buffer (2 batches ahead)
- Mixed precision: NOT used due to stability concerns
- Batch size: Limited by GPU memory (2 per GPU on RTX 4090)
- Recognizer: Feature extraction cached for GT images

4 Lampiran D: Dokumentasi Desain Software

Lampiran ini menyajikan detail arsitektur software framework GAN-HTR yang mendukung reproduktifitas dan ekstensibilitas penelitian. Konten ini melengkapi metodologi penelitian yang disajikan pada Bab III dengan fokus pada aspek implementasi software engineering.

4.1 Prinsip Desain Modular

Tabel 12: Prinsip desain modular dan implementasinya

Prinsip	Definisi	Implementasi
Loose Coupling	Komponen independen dengan minimal dependencies	Interface-based communication, not direct implementation
High Cohesion	Fungsi terkait grouped dalam modul yang sama	Data processing functions dalam satu Data module
Abstraction Layers	Abstract interfaces untuk implementasi berbeda	Generator interface: U-Net, ResNet, atau custom architectures
Dependency Injection	Runtime configuration of dependencies	Components receive dependencies via constructor/config

4.2 Komponen Inti Framework

Tabel 13: Mapping komponen inti ke modules/classes

Component	Module Path	Key Classes/Functions
Generator	`src/models/generator.py`	`UNetGenerator`, `build_generator()`
Discriminator	`src/models/discriminator.py`	`DualModalDiscriminator`
Recognizer	`src/models/recognizer.py`	`HTRRecognizer`, `freeze_recognizer()`
Loss Functions	`src/losses/`	`PixelLoss`, `RecFeatLoss`, `CTCLoss`
Data Pipeline	`src/data/`	`DataLoader`, `Augmentation`
Training Loop	`src/training/trainer.py`	`GANTrainer`, `train_step()`

4.3 Sistem Konfigurasi Hierarkis

Tabel 14: Hierarki konfigurasi dan lokasi file

Level	File	Content
Base Config	`configs/base.yaml`	Default architecture, model dimensions
Experiment Config	`configs/experiment/*.yaml`	HPO results, loss weights, specific settings
Environment Config	`configs/env/*.yaml`	Data paths, GPU settings, storage locations
Runtime Override	Command-line args	Quick parameter adjustments via argparse

Configuration Inheritance Example:

```
# configs/experiment/hpo_best.yaml
base: configs/base.yaml  # Inherit from base

# Override specific parameters
loss_weights:
  pixel: 120.0
  rec_feat: 80.0
  adv: 2.5
  ctc: 10.0  # monitoring only

training:
  batch_size: 2
  epochs: 100
```

4.4 Testing Framework

Tabel 15: Testing strategy dan coverage

Test Type	Location	Coverage
Unit Tests	`tests/unit/`	Individual components (models, losses, data)
Integration Tests	`tests/integration/`	Full pipeline, end-to-end workflow
Smoke Tests	`tests/smoke/`	Quick validation (5 epochs, 100 samples)
Performance Tests	`tests/performance/`	Timing, memory profiling

4.5 Dependency Management

Framework menggunakan Poetry untuk dependency management:

```
# Core dependencies (pyproject.toml)
[tool.poetry.dependencies]
python = "^3.9"
tensorflow = "^2.15.0"
numpy = "^1.24.0"
optuna = "^3.5.0"
mlflow = "^2.10.0"

[tool.poetry.group.dev.dependencies]
```

```
pytest = "^7.4.0"
black = "^23.0.0"
flake8 = "^6.1.0"
```

4.6 MLOps Integration

Tabel 16: MLOps tools dan integrasi

Tool	**Purpose**	**Integration**
MLflow	Experiment tracking	Auto-logging params, metrics, artifacts
Optuna	Hyperparameter optimization	TPE sampler, SQLite database
TensorBoard	Real-time monitoring	Loss curves, image samples
Git	Version control	Auto-commit hash logging in MLflow
Poetry	Dependency management	Reproducible environments

5 Lampiran E: Detail Teknik Stabilitas Training

5.1 Gradient Stabilization

Tabel 17: Comprehensive gradient stabilization techniques

Technique	**Configuration**	**Effect**
Gradient Clipping	clipnorm=1.0 (global norm)	Prevents exploding gradients
Loss Scaling	Not used (FP32 only)	N/A for pure FP32 training
Gradient Accumulation	Not used (batch size=2 fits)	N/A for current setup
Gradient Checkpointing	Not used (memory sufficient)	N/A

5.2 Loss Balancing Strategies

Tabel 18: Multi-loss balancing implementation

Strategy	**Implementation**	**Rationale**
Static Weights	From HPO: pixel=120, rec_feat=80, adv=2.5	Optimal balance found via Ba Optimization
CTC Annealing	Warmup 2 epochs: weight 0→10	Gradual introduction to preven collapse
Loss Normalization	Divide by batch size & sequence length	Consistent scale across batches
Component Monitoring	Log individual losses separately	Detect imbalance during training

5.3 Mode Collapse Prevention

Tabel 19: Strategies untuk mencegah mode collapse

Strategy	**Description**
Adversarial Weight Control	Keep adv_loss_weight 5.0 (proven via HPO)
Label Smoothing	Factor 0.9 for discriminator targets (real=0.9, fake=0.1)
Discriminator Regularization	Instance noise injection (std=0.05 at epoch start, decay)
Diversity Regularization	Not explicitly used (pixel + rec_feat losses provide diversity)
Early Warning System	Monitor PSNR drop below 5 dB $\rightarrow$ trigger alert