

IF25-40401 · DEEP LEARNING · PERTEMUAN 06 · MATERI PENDAMPING STUDIO 2

HYPERPARAMETER TUNING UNTUK CNN & ViT

Learning rate, optimizer, scheduler, regularisasi, early stopping, strategi fine-tuning, dan pelacakan eksperimen dengan W&B

Tim Dosen Deep Learning · Institut Teknologi Sumatera (ITERA)

Setelah mengikuti sesi ini, mahasiswa diharapkan mampu:

- 1 **Membedakan parameter dan hyperparameter** serta memprioritaskannya
- 2 **Merancang eksperimen yang jujur**: split train/valid/test, satu perubahan per run, hipotesis tertulis
- 3 **Memilih learning rate, optimizer, batch size, dan scheduler** berdasarkan kurva training
- 4 **Mendiagnosis underfitting/overfitting** dan memilih regularisasi serta early stopping yang tepat
- 5 **Membandingkan strategi fine-tuning** (scratch, linear probing, full FT) pada CNN dan ViT
- 6 **Melacak dan membandingkan eksperimen** dengan Weights & Biases

OUTLINE

- 1 Metodologi Eksperimen
- 2 Setup: `train.py` & Weights & Biases
- 3 Learning Rate, Optimizer & Batch Size
- 4 LR Scheduler & Warmup
- 5 Regularisasi & Early Stopping
- 6 Strategi Fine-tuning & Pilihan Model
- 7 Strategi Pencarian & Resep Praktis

Model Sama, Satu Angka Berbeda

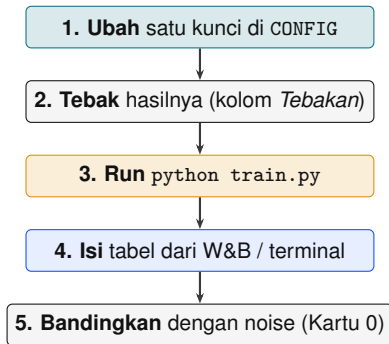
Cara kerja sesi ini

Tebak dulu

ResNet-18 pretrained, data sama, kode sama.
Hanya `lr` yang diganti dari 10^{-3} ke 10^{-1} . Berapa poin akurasi yang hilang? **0? 5? 50?**

Jawabannya kita **ukur sendiri** hari ini: setiap topik punya **kartu eksperimen** yang dijalankan langsung di kelas.

- satu run = `python train.py`, maks. 30 epoch
- RTX 5070: $\pm 2,5$ – $4,5$ menit per run (early stopping sering lebih cepat)
- semua run masuk ke satu proyek W&B



Kembalikan nilai yang diubah ke baseline sebelum pindah kartu.

Parameter vs Hyperparameter

Siapa yang menentukan nilainya?

	Parameter θ	Hyperparameter
Contoh	bobot kernel konvolusi, bias, γ/β BatchNorm, bobot attention	learning rate, optimizer, batch size, epoch, dropout, augmentasi, scheduler
Ditentukan oleh	gradient descent (otomatis)	Anda (sebelum training)
Jumlah	jutaan (ResNet-18: 11,2 M)	belasan (CONFIG di <code>train.py</code>)
Dipilih memakai	data train	data valid

$$\underbrace{\theta^*(\lambda) = \arg \min_{\theta} \mathcal{L}_{\text{train}}(\theta; \lambda)}_{\text{training (loop dalam)}}$$

$$\underbrace{\lambda^* = \arg \max_{\lambda} \text{Skor}_{\text{valid}}(\theta^*(\lambda))}_{\text{hyperparameter tuning (loop luar)}}$$

λ = vektor hyperparameter. Setiap evaluasi loop luar = satu training penuh $\Rightarrow$ mahal, jadi harus dirancang dengan cermat.

BAGIAN 01 · METODOLOGI

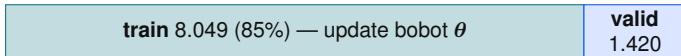
Eksperimen yang Jujur

Split Data, Satu Perubahan per Run, dan Prioritas

Tiga Split, Tiga Peran

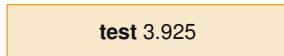
Test set hanya disentuh sekali

Imagenette train (9.469)



pilih hyperparameter,
early stopping, checkpoint (**sekali**, bobot terbaik)

Imagenette val



laporan akhir

Kenapa test tidak boleh dipakai memilih?

Setiap kali skor test memengaruhi keputusan, test ikut “dilatih”. Skor jadi **optimistis** dan tidak lagi mengukur generalisasi.

Analogi di Studio 2

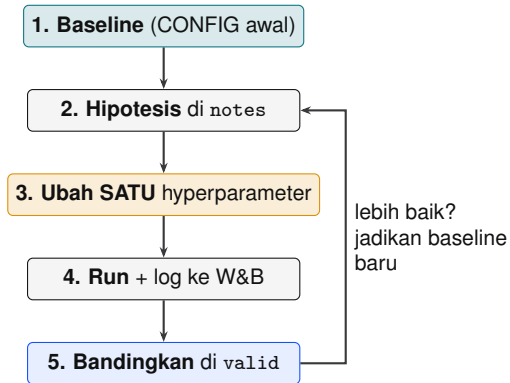
Validation set tersembunyi milik asisten = test set. Split lokal (`my_val/`) = valid set Anda. Tuning hanya pada `my_val/`.

Siklus Eksperimen

Hipotesis → satu perubahan → bukti

Aturan main

- **Satu perubahan per run** — dua perubahan sekaligus = efek tak bisa diatribusikan.
- **Seed tetap** (seed=42), split tetap.
- **Catat semuanya**, termasuk run yang gagal.
- **Ukur noise dulu**: ulangi baseline dengan **seed berbeda**; selisih lebih kecil dari noise = **bukan temuan**.
- Hipotesis ditulis **sebelum** run, bukan setelah melihat hasil.



Kartu 0: Seberapa Besar Noise?

Baseline yang sama, seed berbeda

Ubah di CONFIG

```
seed=42 # baseline
seed=1
seed=2
wandb_group="noise"
```

Hipotesis (tulis di notes)

- Hasil **berbeda** walau konfigurasi sama: seed mengubah init head, urutan batch, augmentasi, dan split.
- Seed sama $\Rightarrow$ hasil identik: run ulang tanpa ganti seed **tidak** mengukur noise.

Seed	Best ep	Valid	Test
42 (baseline)			
1			
2			
Rentang (maks – min)			

Rentang ini = **ambang** untuk semua kartu berikutnya: selisih lebih kecil dari ini **belum tentu nyata**.

Amati di W&B: valid/acc ketiga run bertumpuk di satu panel.

Mana yang Dituning Lebih Dulu?

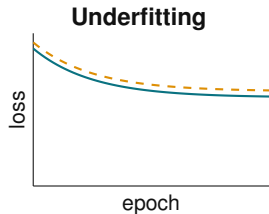
Prioritas untuk fine-tuning pretrained

Prio.	Hyperparameter	Kenapa	Kunci di CONFIG
1	Learning rate	paling sensitif; meleset satu orde = gagal	lr, backbone_lr_mult
1	Pretrained & strategi FT	menentukan titik awal optimasi	pretrained, freeze_backbone
2	Optimizer & weight decay	berinteraksi kuat dengan lr	optimizer, weight_decay
2	Scheduler & epoch	bentuk lr sepanjang training	scheduler, warmup_epochs
2	Batch size	noise gradien & lr optimal	batch_size
3	Regularisasi	penting saat data sedikit / overfit	augment, dropout
3	Early stopping	hemat waktu, checkpoint terbaik	patience, monitor
4	Arsitektur & resolusi	ganti setelah resep training stabil	model, img_size

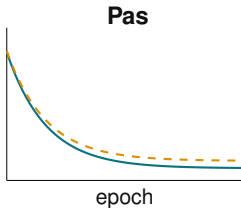
Urutan mengikuti Google *Deep Learning Tuning Playbook* dan Smith (2018): mulai dari efek terbesar, kunci nilainya, lalu lanjut.

Diagnosis Dulu, Baru Tuning

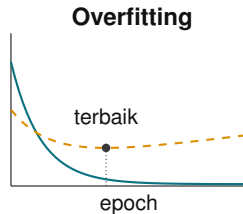
loss train (garis) vs valid (putus-putus)



- keduanya tinggi & datar
- $\uparrow$ lr / epoch, pretrained, **kurangi** regularisasi, model lebih besar



- turun bersama, gap kecil
- pertahankan; coba resolusi / model lain



- train terus turun, valid naik
- augmentasi, dropout, wd, label smoothing, MixUp, **early stopping**

BAGIAN 02 · SETUP

`train.py` & **W&B**

Satu File, Satu Blok CONFIG, Semua Run Tercatat

Dataset & Lima Backbone

Imagenette 160 px · timm pretrained

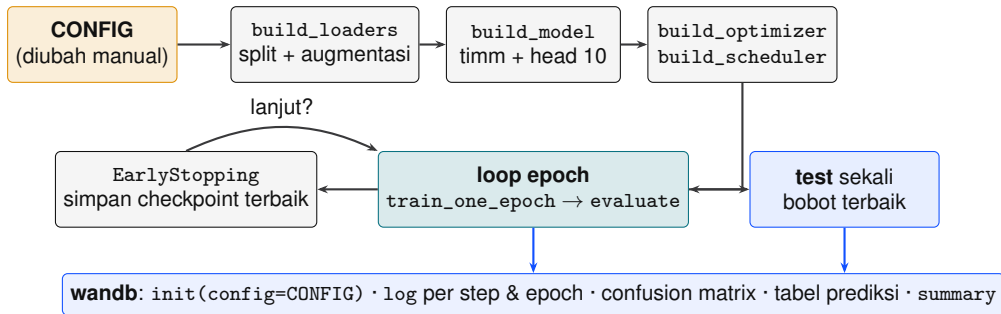
Imagenette (J. Howard, fast.ai)

- subset **10 kelas**
ImageNet yang mudah dibedakan (ikan tench, gereja, parasut, truk sampah, ...)
- 9.469 train · 3.925 val · versi 160 px $\approx$ 94 MB
- unduh otomatis lewat `torchvision`

Kenapa bukan data sampah?
Agar konfigurasi terbaik untuk kompetisi tetap menjadi temuan kelompok Anda sendiri.

model	Keluarga	Param	s/epoch*	30 ep*
resnet18	CNN residual	11,2 M	4,8	2,5 menit
mobilenetv3	depthwise + SE	1,5 M	5,7	3,0 menit
efficientnet_b0	compound scaling	4,0 M	8,6	4,4 menit
convnext_atto	CNN modern	3,4 M	5,4	3,0 menit
vit_tiny	Transformer	5,5 M	6,3	3,3 menit

*terukur di RTX 5070 (batch 64, 160 px, AMP, 8 worker), termasuk validasi. Bottleneck-nya **loading data di CPU**, bukan ukuran model: MobileNetV3 (1,5 M) tidak lebih cepat dari ResNet-18 (11,2 M). Semua: varian **teringan** di keluarganya, head 10 kelas via `timmm.create_model(..., num_classes=10)`.



- **Tanpa argparse:** setiap eksperimen = edit CONFIG, simpan, python train.py.
- `main(config)` juga bisa dipanggil dari skrip lain (mis. loop beberapa nilai lr dalam satu skrip).
- Divergen (loss NaN/Inf) $\Rightarrow$ training berhenti rapi dan ditandai `diverged` di W&B.

Blok CONFIG

Semua hyperparameter di satu tempat

```
CONFIG = dict(  
    # --- Data ---  
    img_size=160,          # 128 / 160 / 224  
    train_fraction=1.0,    # 0.2 = 20% data train  
    augment="basic",       # "none"/"basic"/"strong"  
    mixup_alpha=0.0, cutmix_alpha=0.0,  
    # --- Model ---  
    model="resnet18",      # mobilenetv3 / vit_tiny  
    ...  
    pretrained=True,  
    freeze_backbone=False, # True = linear probing  
    dropout=0.0, drop_path=0.0,  
    # --- Optimizer ---  
    optimizer="adamw",     # sgd/adam/adamw/rmsprop  
    lr=1e-3,  
    backbone_lr_mult=0.1,  # lr backbone = lr * 0.1  
    weight_decay=0.05,  
    batch_size=64, epochs=30,  
    label_smoothing=0.0,
```

```
    # --- LR scheduler ---  
    scheduler="cosine",    # none/step/cosine  
                           # onecycle/plateau  
    warmup_epochs=1,  
    # --- Early stopping ---  
    early_stopping=True,  
    patience=5, min_delta=1e-3,  
    monitor="val_loss",   # val_acc / val_f1  
    # --- Weights & Biases ---  
    wandb_project="if25-p6-hyperparameter",  
    wandb_group=None,     # mis. "sweep-lr"  
    wandb_mode="online",  # offline / disabled  
    notes="",             # tulis HIPOTESIS!  
)
```

Nama run dibuat otomatis, mis.

resnet18-adamw-lr0.001-cosine-do0.3,
sehingga mudah difilter di dashboard.

```
run = wandb.init(project="if25-p6-hyperparameter",
                 name=run_name, group="sweep-lr",
                 config=CONFIG,      # semua
                 hyperparameter
                 notes="lr 1e-2 akan divergen?")

# per 10 step
wandb.log({"train/loss": loss, "train/lr": lr,
          "train/grad_norm": gnorm}, step=global_step)

# per epoch
wandb.log({"epoch": ep, "valid/loss": vl,
          "valid/acc": va, "valid/f1": vf})

# akhir run
run.summary.update({"test/acc": acc, "test/f1": f1})
wandb.log({"test/confusion_matrix":
          wandb.plot.confusion_matrix(...)})
wandb.finish()
```

Di dashboard, gunakan:

- **Runs table:** urutkan menurut valid/acc, tampilkan kolom config.
- **Group / filter:** group = "sweep-lr", tag adamw.
- **Parallel coordinates:** hyperparameter mana yang berkorelasi dengan skor.
- **Line plot** train/lr: memastikan scheduler bekerja seperti yang dibayangkan.
- **Report:** rangkuman eksperimen kelompok untuk laporan Studio 2.

Tanpa internet: `wandb_mode="offline"`,
unggah nanti dengan `wandb sync`

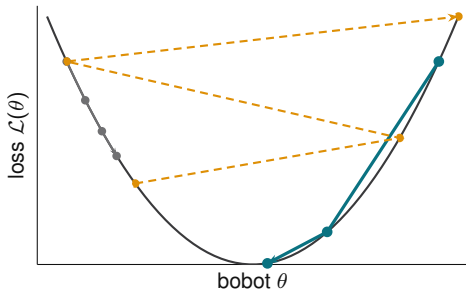
BAGIAN 03 · OPTIMASI

Learning Rate & Optimizer

Hyperparameter Paling Sensitif

Learning Rate: Ukuran Langkah

Hyperparameter paling sensitif



$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta} \mathcal{L}(\theta_t)$$

- η **kecil**: tiap langkah aman tetapi pendek $\Rightarrow$ belum konvergen saat epoch habis (**underfit**).
- η **pas**: turun cepat lalu stabil.
- η **besar**: melompati lembah, loss berosilasi atau **meledak** (NaN). `train.py` mendeteksinya dan menghentikan run (`diverged=True` di W&B).
- Lr optimal bergantung pada optimizer, batch size, dan backbone $\Rightarrow$ ganti salah satunya = cek Lr ulang.

Ilustrasi 1 dimensi; loss sebenarnya berdimensi jutaan.

Kartu 1: Sweep Learning Rate

Skala log, bukan linear

Ubah di CONFIG

```
lr=1e-5
lr=1e-4
lr=1e-3 # baseline
lr=1e-2
lr=1e-1
wandb_group="sweep-lr"
```

Hipotesis (tulis di notes)

- Akurasi vs $\log_{10} \eta$ berbentuk **U terbalik**.
- lr terlalu besar: berhenti cepat (divergen / early stopping).

lr	Tebakan	Best ep	Henti ep	Valid	Test
10^{-5}					
10^{-4}					
10^{-3}					
10^{-2}					
10^{-1}					

Setelah orde terbaik ketemu, haluskan dengan faktor $\pm 3 \times$ (mis. $3 \cdot 10^{-4}$, $3 \cdot 10^{-3}$).

Amati di W&B: train/loss per step dan valid/acc; panel *Parallel coordinates* $lr \rightarrow \text{valid/acc}$.

SGD, Momentum, dan Adam

Aturan update yang dipilih optimizer

SGD + momentum ("sgd", momentum=0.9)

$$\mathbf{v}_t = \mu \mathbf{v}_{t-1} + \mathbf{g}_t, \quad \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \mathbf{v}_t$$

- satu lr untuk semua bobot; momentum meredam zig-zag
- butuh lr lebih besar (umumnya 10^{-2}) dan lebih sensitif terhadap lr
- sering sedikit lebih baik dalam generalisasi bila dituning cermat

Adam ("adam"): lr adaptif per bobot

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t, \quad \mathbf{s}_t = \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t + \epsilon}}$$

- bobot dengan gradien kecil mendapat langkah relatif besar
- lr default 10^{-3} biasanya "langsung jalan"
- "rmsprop": hanya $\mathbf{s}_t$, tanpa $\mathbf{m}_t$ dan koreksi bias

Kartu 2: Optimizer

Setiap optimizer punya lr idealnya

Ubah di CONFIG

```
optimizer="sgd", lr=1e-3  
optimizer="sgd", lr=1e-2  
optimizer="adam", weight_decay=0.0  
optimizer="adam" # wd 0.05 = L2  
optimizer="rmsprop"  
optimizer="adamw" # baseline
```

Hipotesis (tulis di notes)

- SGD butuh lr $\sim 10\times$ lebih besar dari Adam.
- Adam + L2 dengan λ ala AdamW **lebih buruk** dari Adam tanpa decay (slide berikut).

Optimizer	lr	wd	Tebakan	Valid	Test
SGD+mom.	10^{-3}	0,05 L2			
SGD+mom.	10^{-2}	0,05 L2			
Adam	10^{-3}	0			
Adam	10^{-3}	0,05 L2			
RMSprop	10^{-3}	0,05 L2			
AdamW	10^{-3}	0,05 dec.			

L2 = decay lewat gradien; dec. = decoupled (AdamW).
momentum=0.9 untuk SGD & RMSprop.

Amati di W&B: kurva valid/acc per optimizer di satu panel (filter group).

Adam + L2 ("adam", weight_decay= λ): decay masuk ke gradien

$$\mathbf{g}_t \leftarrow \nabla \mathcal{L} + \lambda \boldsymbol{\theta} \Rightarrow \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}$$

$\lambda \boldsymbol{\theta}$ ikut **dinormalisasi** oleh $\sqrt{\hat{\mathbf{s}}_t}$: bobot dengan gradien kecil di-decay sangat kuat, yang gradiennya besar hampir tidak.

AdamW ("adamw"): decay **dipisah** dari gradien

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \left(\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} + \lambda \boldsymbol{\theta} \right)$$

Semua bobot menyusut dengan laju yang sama $\eta \lambda$ — perilaku yang memang diinginkan dari weight decay.

Cek di Kartu 2

Bandingkan tiga baris:

- Adam, $\lambda = 0$
- Adam + L2, $\lambda = 0,05$
- AdamW, $\lambda = 0,05$

Jika Adam + L2 paling rendah: λ yang wajar untuk AdamW **terlalu kuat** sebagai L2.

Di PyTorch:

```
torch.optim.Adam(weight_decay=...) =  
L2; torch.optim.AdamW = decoupled.
```

Kartu 3: Batch Size

Goyal et al. (2017): linear scaling rule

$$B \rightarrow kB \Rightarrow \eta \rightarrow k\eta$$

Batch $4\times$ lebih besar = langkah $4\times$ lebih sedikit per epoch; kompensasi dengan lr $4\times$ lebih besar (perlu warmup).

- Batch besar: gradien lebih akurat, GPU lebih efisien, tetapi **lebih sedikit update**.
- Batch kecil: gradien berisik (efek regularisasi), lebih lambat per epoch.
- CUDA out of memory: turunkan `batch_size` dan lr secara proporsional.

Hipotesis: baris $256 + 4 \cdot 10^{-3}$ menyamai baseline; $256 + 10^{-3}$ tertinggal.

batch_size	lr	Step/ep	Tebakan	s/epoch	Test
32	10^{-3}	252			
64 (baseline)	10^{-3}	126			
256	10^{-3}	32			
256	$4 \cdot 10^{-3}$	32			

Step/ep = $\lceil 8.049/B \rceil$ (data train 85%).
`wandb_group="batch"`.

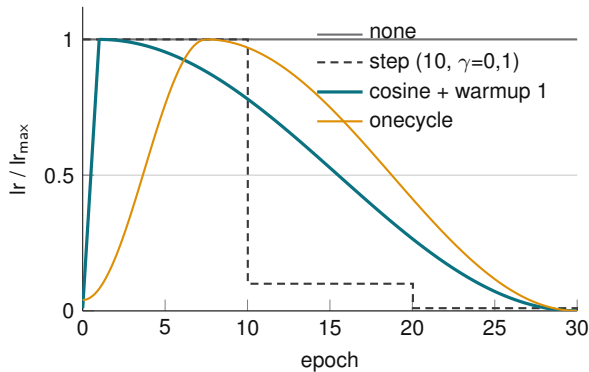
BAGIAN 04 · SCHEDULER

LR Scheduler & Warmup

Learning Rate Tidak Harus Konstan

Bentuk-Bentuk Scheduler

lr relatif terhadap $1r$ di CONFIG, 30 epoch



- **step**: turun $\times \gamma$ tiap `step_size` epoch; sederhana, tapi waktu turunnya harus ditebak.
- **cosine**: turun mulus ke `min_lr`; default modern (ConvNeXt, ViT).
- **onecycle**: naik lalu turun; lr puncak boleh lebih besar.
- **plateau**: turun $\times \gamma$ **hanya jika** `val_loss` macet `plateau_patience` epoch — reaktif, tidak terjadwal.

Di `train.py` semua di-`step()` per batch, kecuali plateau (per epoch).

Kenapa Warmup?

Melindungi bobot pretrained di awal training

- Di awal, **head baru diinisialisasi acak** $\Rightarrow$ loss besar, gradien besar dan “berisik”.
- Gradien itu mengalir ke backbone; dengan lr penuh, fitur pretrained yang bagus bisa **rusak** di beberapa ratus langkah pertama.
- Adam: estimasi $\hat{\mathbf{s}}_t$ belum stabil pada langkah awal.
- **Warmup linear**: lr naik dari 1% ke 100% selama `warmup_epochs`.

$$\eta_t = \eta_{\max} \left(0,01 + 0,99 \frac{t}{T_{\text{warm}}} \right), \quad t < T_{\text{warm}}$$

Kombinasi yang dipakai di `train.py`

LinearLR (warmup) $\rightarrow$ CosineAnnealingLR via SequentialLR, di-step setiap batch.

- ViT umumnya **lebih membutuhkan** warmup daripada CNN dengan BatchNorm.
- `backbone_lr_mult=0.1` adalah perlindungan kedua: backbone bergerak $10\times$ lebih pelan dari head.

Kartu 4: Scheduler

Efeknya bergantung pada lr

Ubah di CONFIG

```
scheduler="none"  
scheduler="step" # step_size=10  
scheduler="cosine", warmup_epochs=0  
scheduler="cosine" # baseline  
scheduler="onecycle"  
scheduler="plateau"  
# dua baris terakhir: lr=1e-2
```

Hipotesis (tulis di notes)

- Pada lr yang sudah pas, scheduler mengubah sedikit.
- Pada lr kebesaran, cosine **menyelamatkan** run.

Scheduler	lr	Tebakan	Henti ep	Test
none	10^{-3}			
step (10, 0,1)	10^{-3}			
cosine, warmup 0	10^{-3}			
cosine, warmup 1	10^{-3}			
onecycle	10^{-3}			
plateau	10^{-3}			
none	10^{-2}			
cosine, warmup 1	10^{-2}			

OneCycle menaikkan lr selama 25% awal ($\approx$ epoch 1–7): early stopping bisa memotong run *sebelum* lr turun. Perhatikan kolom *Henti ep*.

Amati di W&B: train/lr per step (bentuk kurva) dan valid/loss.

BAGIAN 05 · REGULARISASI

Regularisasi & Early Stopping

Menutup Gap antara Train dan Valid

Kartu 5: Memancing Overfitting

10% data (≈ 800 citra), tanpa regularisasi

Ubah di CONFIG

```
train_fraction=0.1
augment="none"
weight_decay=0.0
early_stopping=False
# dropout=0, label_smoothing=0
# (default)
wandb_group="reg"
```

Hipotesis (tulis di notes)

- Akurasi train mendekati 100%, valid tertinggal jauh: **gap** besar.
- Loss train terus turun; loss valid mendatar atau naik.

Epoch	Train acc	Valid	Gap	Valid loss
5				
10				
20				
30				

Run ini cepat ($\pm 1,5$ s/epoch) dan menjadi **baseline Kartu 6**. Studio 2 (± 3.700 citra) juga termasuk “data sedikit”.

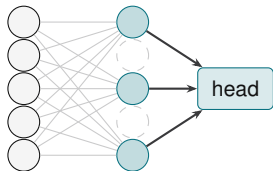
Amati di W&B: epoch/train_acc vs valid/acc dan epoch/gap_acc.

Dropout & Stochastic Depth

dropout dan drop_path

Dropout (Srivastava et al., 2014): matikan neuron acak

$$\tilde{\mathbf{h}} = \frac{\mathbf{m} \odot \mathbf{h}}{1 - p}, \quad m_i \sim \text{Bernoulli}(1 - p)$$



Di timm: diterapkan **sebelum classifier** (`drop_rate`).
Saat `eval()` dropout otomatis mati.

Stochastic depth / drop path (Huang et al., 2016): matikan **seluruh cabang residual**

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \frac{b_l}{1 - p} F_l(\mathbf{x}_l), \quad b_l \sim \text{Bernoulli}(1 - p)$$

- jaringan “acak lebih dangkal” tiap batch $\Rightarrow$ ensemble implisit
- standar di ConvNeXt & ViT (`drop_path` 0,1–0,2)
- didukung kelima model di `train.py` (timm `drop_path_rate`)

Weight Decay & Label Smoothing

weight_decay dan label_smoothing

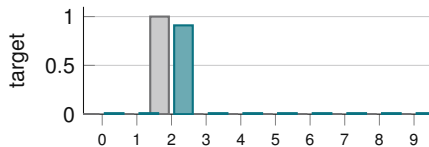
Weight decay: menarik bobot ke nol

$$\mathcal{L}_{\text{total}} = \mathcal{L} + \frac{\lambda}{2} \|\theta\|^2 \Rightarrow \theta \leftarrow (1 - \eta\lambda) \theta - \eta \mathbf{g}$$

- setara untuk SGD; **tidak setara** untuk Adam (lihat bagian optimizer)
- AdamW: λ wajar 0,01–0,1; SGD: 10^{-4} – $5 \cdot 10^{-4}$

Label smoothing (Szegedy et al., 2016): target tidak lagi 100% yakin

$$y_k^{\text{LS}} = (1 - \varepsilon) y_k + \frac{\varepsilon}{K}$$



abu-abu: one-hot • teal: $\varepsilon = 0,1$, $K = 10$. Mencegah logit membesar tanpa batas; model lebih terkalibrasi.

Augmentasi Data, MixUp & CutMix

`augment, mixup_alpha, cutmix_alpha`

augment	Transformasi train
none	resize + center crop (sama dengan valid)
basic	RandomResizedCrop(0,35-1) + flip horizontal
strong	basic + TrivialAugmentWide + RandomErasing(p=0,25)

Valid & test **tidak pernah** diaugmentasi.

Augmentasi harus masuk akal untuk domain: flip vertikal untuk foto sampah? Mungkin. Untuk angka “6” vs “9”? **Tidak**.

MixUp (Zhang et al., 2018): campur dua citra *dan* labelnya

$$\tilde{\mathbf{x}} = \lambda \mathbf{x}_i + (1 - \lambda) \mathbf{x}_j, \quad \tilde{\mathbf{y}} = \lambda \mathbf{y}_i + (1 - \lambda) \mathbf{y}_j$$

dengan $\lambda \sim \text{Beta}(\alpha, \alpha)$, $\alpha = \text{mixup_alpha}$.

CutMix (Yun et al., 2019): tempel potongan citra j ke citra i ; λ = proporsi luas yang tersisa.

- label menjadi **soft** $\Rightarrow$ CrossEntropyLoss menerima probabilitas
- diterapkan per batch lewat `collate_fn` (`v2.MixUp`, `v2.CutMix`)
- butuh **lebih banyak epoch** agar manfaatnya terlihat

Kartu 6: Regularisasi

Di atas konfigurasi Kartu 5

Ubah di CONFIG

```
augment="basic"  
augment="strong"  
dropout=0.5  
weight_decay=0.05  
label_smoothing=0.1  
mixup_alpha=0.2, cutmix_alpha=1.0  
# semua di atas sekaligus
```

Hipotesis (tulis di notes)

- Augmentasi paling ampuh saat data sedikit.
- Gap mengecil $\neq$ test pasti naik.

Perubahan	Tebakan	Gap*	Valid	Test
(Kartu 5)				
aug. basic				
aug. strong				
dropout 0,5				
wd 0,05				
label smooth. 0,1				
MixUp + CutMix				
semua				

*train — valid di epoch terakhir; dengan augmentasi/MixUp akurasi train dihitung pada citra teraugmentasi, jadi gap bisa negatif. Noise di data sedikit lebih besar: ulangi Kartu 5 dengan seed lain.

Amati di W&B: epoch/gap_acc semua run dalam satu panel.

Kartu 7: Early Stopping

patience, min_delta, monitor

Aturan (kelas EarlyStopping di train.py):
berhenti jika

$\underbrace{m_t > m^* - \delta}_{\text{tidak membaik}}$ selama P epoch berturut-turut

- $m = \text{monitor}$ (val_loss turun; val_acc/val_f1 naik), $\delta = \text{min_delta}$, $P = \text{patience}$
- setiap kali membaik: **simpan checkpoint**; test memakai checkpoint ini, *bukan* bobot epoch terakhir
- Studio 2 dinilai dengan macro-F1 $\Rightarrow$ `monitor="val_f1"`

	Penuh	$P = 5$	$P = 2$
early_stopping	False	True	True
Epoch dijalankan			
Epoch terbaik			
Waktu (menit)			
Test			

Konfigurasi baseline (data penuh, 30 epoch); hanya early_stopping dan patience yang diubah.

Hipotesis: test hampir sama (sama-sama memakai checkpoint terbaik), waktu jauh lebih singkat; P terlalu kecil berhenti di plateau sementara.

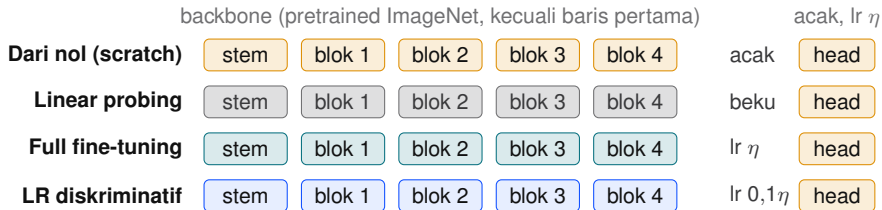
BAGIAN 06 · FINE-TUNING

Fine-tuning & CNN vs ViT

Bagian Mana yang Dilatih, dan Seberapa Cepat

Empat Strategi Melatih Backbone

Bagian mana yang belajar?



Strategi	CONFIG	Cocok ketika	Risiko
Scratch	pretrained=False	domain jauh + data banyak	lambat, haus data
Linear probing	freeze_backbone=True	data sangat sedikit	fitur tak beradaptasi
Full FT	backbone_lr_mult=1.0	data cukup	fitur rusak bila lr besar
LR diskriminatif	backbone_lr_mult=0.1	default yang aman	+1 hyperparameter

Kartu 8: Strategi Fine-tuning

ResNet-18, data penuh

Ubah di CONFIG

```
pretrained=False, backbone_lr_mult=1.0
freeze_backbone=True, lr=1e-3
freeze_backbone=True, lr=1e-2
backbone_lr_mult=1.0
backbone_lr_mult=0.1 # baseline
wandb_group="finetune"
```

Hipotesis (tulis di notes)

- Scratch tertinggal **jauh**: bobot pretrained adalah hyperparameter terpenting.
- Linear probe $\approx$ full FT di Imagenette, karena kelasnya *adalah* kelas ImageNet.

Strategi	Dilatih	Tebakan	Valid	Test
Scratch	11,2 M			
Linear probe, 10^{-3}	5 K			
Linear probe, 10^{-2}	5 K			
Full FT $\times 1$	11,2 M			
Full FT $\times 0,1$	11,2 M			

$\times 1 / \times 0,1 = \text{backbone_lr_mult}$. Data sampah berbeda domain dari ImageNet $\Rightarrow$ ulangi perbandingan ini di data Anda.

Amati di W&B: valid/acc epoch 1: pretrained langsung tinggi, scratch mulai dari $\approx 10\%$.

Kartu 9: CNN vs ViT

Resep sama, backbone berbeda

Ubah di CONFIG

```
model="resnet18" # baseline
model="mobilenetv3"
model="efficientnet_b0"
model="convnext_atto"
model="vit_tiny"
# ulangi dgn train_fraction=0.1
```

Hipotesis (tulis di notes)

- “ViT haus data” berlaku untuk training **dari nol**. Dengan pretraining ImageNet-21k, apakah ViT tetap kalah di 10% data?
- Ganti backbone = **cek lr ulang**: lr 10^{-3} belum tentu cocok untuk ViT.

Model	Param	Tebakan	Test 100%	Test 10%
ResNet-18	11,2 M			
MobileNetV3-S	1,5 M			
EfficientNet-B0	4,0 M			
ConvNeXt-Atto	3,4 M			
ViT-Tiny/16	5,5 M			

CNN: lokalitas & translation equivariance “dibangun” di konvolusi. ViT: attention global antar-patch 16×16 ; di 160 px = 10×10 patch (img_size=160). Run 10% data < 1 menit.

Amati di W&B: best_epoch; jika terbaik di epoch 1–2, lr kemungkinan kebesaran.

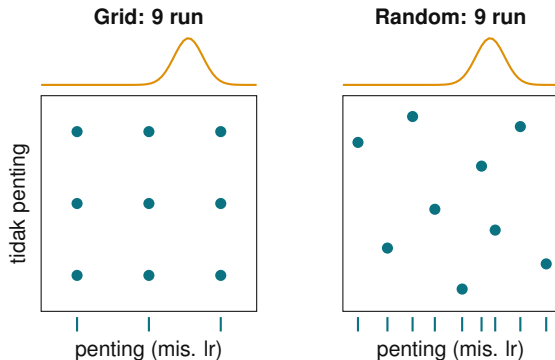
BAGIAN 07 · PENCARIAN

Strategi Pencarian & Resep

Dari Tebakan Manual ke Pencarian Sistematis

Grid Search vs Random Search

Bergstra & Bengio (2012)



- Grid: 9 run, tetapi hanya **3 nilai berbeda** untuk hyperparameter yang penting.
- Random: 9 run $\Rightarrow$ **9 nilai berbeda** untuk setiap dimensi.
- Kurva oranye = skor vs hyperparameter penting; puncaknya sempit dan mudah terlewat oleh grid.
- Sampling lr dalam **skala log**: $\eta = 10^u$, $u \sim \mathcal{U}(-5, -1)$.

Coarse-to-Fine & W&B Sweeps

Langkah berikutnya setelah tuning manual

Coarse-to-fine

- 1 Kasar:** $lr \in \{10^{-5}, \dots, 10^{-1}\}$, epoch pendek, `train_fraction=0.3`.
- 2 Sempit:** sekitar 2–3 nilai terbaik, data penuh.
- 3 Halus:** hyperparameter prioritas 2–3 pada `lr` terbaik.
- 4 Konfirmasi:** ulangi konfigurasi terbaik dengan 2–3 seed.

Tuning manual lewat `CONFIG` di kelas = langkah 1–2. W&B Sweeps mengotomatiskan langkah 2–3.

W&B Sweeps (Bayesian / random / grid)

```
sweep = {  
    "method": "random",          # "bayes" / "grid"  
    "metric": {"name": "valid/acc",  
               "goal": "maximize"},  
    "parameters": {  
        "lr": {"distribution": "log_uniform_values",  
               "min": 1e-5, "max": 1e-2},  
        "dropout": {"values": [0.0, 0.2, 0.5]},  
        "optimizer": {"values": ["sgd", "adamw"]},  
    },  
}  
sweep_id = wandb.sweep(sweep, project="if25-p6")  
wandb.agent(sweep_id, function=..., count=20)
```

Agar `train.py` bisa di-sweep: baca nilai dari `wandb.config` setelah `wandb.init()`, bukan dari `CONFIG`.

Resep Praktis untuk Studio 2

Urutan kerja yang disarankan

- 1 **Baseline** starter kit + split lokal tetap; ukur noise dengan 2–3 seed (Kartu 0).
- 2 **Pretrained**; bandingkan **linear probe** vs **full FT** di data sendiri (Kartu 8).
- 3 **Sweep lr** dalam skala log; ganti optimizer/batch/backbone $\Rightarrow$ cek lr ulang.
- 4 **AdamW** (bukan Adam + L2), cosine + warmup 1 epoch, `backbone_lr_mult 0,1`.
- 5 Lihat kurva: gap besar $\Rightarrow$ augmentasi dulu, lalu label smoothing / MixUp / dropout.
- 6 **Early stopping** pada `val_f1` (metrik kompetisi), patience 3–5 (lebih besar untuk OneCycle).
- 7 Baru setelah itu: **backbone lain** & resolusi lebih besar.

Jangan

- mengubah 3 hal sekaligus
- memilih model memakai skor preview kompetisi
- mengklaim selisih $<$ noise
- lupa mencatat run yang gagal

Bawa pulang

Bandingkan tabel kartu Anda dengan tabel prioritas di awal: **mana yang benar-benar berefek** melebihi noise?

- 1 Buat proyek W&B kelompok; isi `wandb_entity`.
- 2 Jalankan **baseline** `train.py` apa adanya (catat `valid/acc`).
- 3 Setiap anggota memilih **satu** hyperparameter dari tabel prioritas, menulis hipotesis di `notes`, lalu menjalankan 2–3 nilai.
- 4 Bandingkan di dashboard: siapa yang mengalahkan baseline melebihi noise?
- 5 Terapkan temuan terbaik ke **data kompetisi sampah** (`train_baseline.py` di starter kit).

Dikumpulkan (Studio 2)

- tautan **W&B Report**: tabel run, kurva valid, hipotesis vs hasil
- 3 kalimat: hyperparameter apa yang paling berpengaruh pada data sampah, dan buktinya

Tips laptop (CPU/MPS):

`model="mobilenetv3",`
`train_fraction=0.3, epochs=5.` GPU (RTX 5070): 30 epoch $\pm 2,5$ – $4,5$ menit.

-  L. N. Smith. A Disciplined Approach to Neural Network Hyper-Parameters: Part 1 — Learning Rate, Batch Size, Momentum, and Weight Decay. *arXiv:1803.09820*, 2018.
-  D. P. Kingma, J. Ba. Adam: A Method for Stochastic Optimization. *ICLR*, 2015.
-  I. Loshchilov, F. Hutter. Decoupled Weight Decay Regularization. *ICLR*, 2019.
-  I. Loshchilov, F. Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts. *ICLR*, 2017.
-  P. Goyal et al. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv:1706.02677*, 2017.
-  L. N. Smith, N. Topin. Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates. *arXiv:1708.07120*, 2017.
-  J. Bergstra, Y. Bengio. Random Search for Hyper-Parameter Optimization. *JMLR* 13, 2012.
-  V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, Z. Nado. Deep Learning Tuning Playbook. https://github.com/google-research/tuning_playbook, 2023.

-  N. Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15, 2014.
-  G. Huang, Y. Sun, Z. Liu, D. Sedra, K. Weinberger. Deep Networks with Stochastic Depth. *ECCV*, 2016.
-  C. Szegedy et al. Rethinking the Inception Architecture for Computer Vision. *CVPR*, 2016.
-  H. Zhang, M. Cisse, Y. Dauphin, D. Lopez-Paz. mixup: Beyond Empirical Risk Minimization. *ICLR*, 2018.
-  S. Yun et al. CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. *ICCV*, 2019.
-  S. G. Müller, F. Hutter. TrivialAugment: Tuning-free Yet State-of-the-Art Data Augmentation. *ICCV*, 2021.
-  S. Kornblith, J. Shlens, Q. V. Le. Do Better ImageNet Models Transfer Better? *CVPR*, 2019.
-  A. Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR*, 2021.
-  A. Steiner et al. How to Train Your ViT? Data, Augmentation, and Regularization in Vision Transformers. *TMLR*, 2022.
-  R. Wightman. PyTorch Image Models (timm) · J. Howard. Imagenette · Weights & Biases Docs.
<https://docs.wandb.ai>

IF25-40401 · DEEP LEARNING

Terima Kasih

Kode: `materials/hyperparameter-tuning/train.py` · Kompetisi: Studio Proyek 2
— Klasifikasi Sampah

Tim Dosen Deep Learning · Institut Teknologi Sumatera (ITERA)