Pertemuan 13 — *Convolutional Neural Network*

Memberi *neural network* pengetahuan bahwa citra punya struktur

Hendri Karisma

Semester Ganjil 2026/2027

Program Studi Teknik Informatika, STMIK Tazkia

Alur pertemuan hari ini

1. Mengapa MLP boros untuk citra
2. *Local connectivity* dan *weight sharing*
3. Operasi konvolusi, dihitung dengan tangan
4. *Filter*, *feature map*, dan jumlah parameter
5. *Stride*, *padding*, dan rumus dimensi *output*
6. *Pooling* dan *receptive field*
7. Menyusun dan melatih CNN dengan PyTorch
8. Arsitektur klasik: LeNet, AlexNet, VGG, ResNet
9. *Transfer learning* pada citra
10. Visualisasi *filter* dan Grad-CAM
11. Kapan memakai CNN
12. Praktikum: CNN lawan MLP

Sasaran pertemuan ini

Anda dapat menjelaskan operasi konvolusi, *stride*, *padding*, dan *pooling*, menghitung dimensi *output* dan jumlah parameter setiap *layer* tanpa komputer, lalu melatih CNN sederhana dan menafsirkan hasilnya dibandingkan dengan MLP.

Motivasi: citra punya struktur

Menyambung dari Pertemuan 12

Pertemuan lalu menutup tiga gagasan: kedalaman membantu jaringan menyusun representasi bertingkat, *residual connection* menjaga *gradient* tetap mengalir, dan *transfer learning* menghemat data. Hari ini ketiganya bertemu dalam satu keluarga model untuk citra.

Kasus yang akan kita bawa sepanjang pertemuan

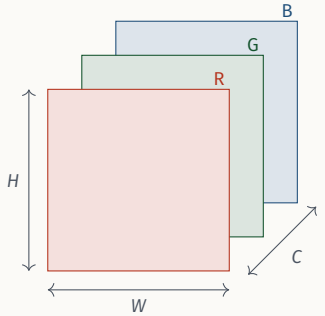
Sebuah koperasi tani di Karawang ingin aplikasi sederhana: petani memotret daun padi, lalu aplikasi menebak **sehat**, **hawar daun bakteri**, atau ***blast***. Foto ponsel berisi ratusan ribu piksel, masing-masing tiga nilai warna, sedangkan foto berlabel yang tersedia hanya beberapa ratus.

Bila seluruh piksel itu dimasukkan ke MLP Pertemuan 10, setiap neuron *layer* pertama memerlukan satu *weight* per piksel. Jumlah *weight* meledak, data tidak sebanding, dan MLP tidak tahu bahwa bercak di pojok kiri atas sama artinya dengan bercak yang sama di tengah daun.

Citra sebagai *tensor*

- Citra skala abu-abu $H \times W$: matriks kecerahan, nilai 0–255 yang diskalakan ke $[0, 1]$.
- Citra berwarna punya tiga ***channel*** (kanal) merah, hijau, biru, sehingga bentuknya $C \times H \times W$ dengan $C = 3$.
- *Mini-batch* berisi N citra berbentuk $N \times C \times H \times W$. Urutan ini disebut **NCHW** dan menjadi bawaan PyTorch.

Dataset	**Citra**	**Jumlah**
MNIST	abu-abu 28×28, angka 0–9	60.000 latih, 10.000 uji
CIFAR-10	RGB 32×32, 10 kelas	50.000 latih, 10.000 uji



Satu citra CIFAR-10 adalah *tensor* $3 \times 32 \times 32$, yaitu 3.072 bilangan.

Mengapa MLP boros untuk citra

Ratakan (*flatten*) citra CIFAR-10 menjadi vektor sepanjang $3 \cdot 32 \cdot 32 = 3.072$. *Layer* tersembunyi dengan 1.000 neuron memerlukan

$$3.072 \times 1.000 + 1.000 = 3.073.000 \text{ parameter, hanya untuk } \textit{layer} \text{ pertama.}$$

Untuk foto ponsel $3 \times 1000 \times 1000$ angkanya menjadi sekitar tiga miliar. Selain boros, ada tiga masalah yang lebih mendasar.

Struktur ruang hilang

Setelah *flatten*, dua piksel bertetangga diperlakukan sama jauhnya dengan dua piksel di pojok berseberangan. MLP harus menemukan sendiri bahwa tetangga itu penting.

Tidak ada pakai ulang

Detektor "tepi tegak" yang dipelajari untuk pojok kiri atas tidak bisa dipakai di pojok kanan bawah. Setiap posisi dipelajari terpisah.

Rentan pergeseran

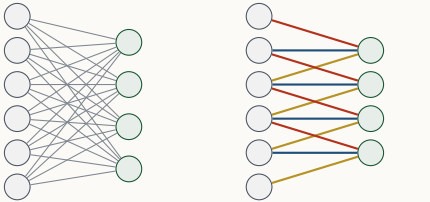
Geser objek beberapa piksel, dan seluruh vektor *input* berubah. MLP dapat memberi jawaban yang berbeda untuk benda yang sama.

Tiga gagasan CNN

- ***Local connectivity***: setiap neuron hanya melihat sepetak kecil *input*, misalnya 3×3 piksel. Petak itu disebut ***receptive field***-nya.
- ***Weight sharing***: neuron di semua posisi memakai *weight* yang sama. Satu set *weight* kecil ini disebut ***filter*** atau ***kernel***, dan ia digeser ke seluruh citra.
- ***Translation equivariance***: akibat dua gagasan pertama. Bila *input* digeser, *output* ikut bergeser dengan cara yang sama. Setelah diringkas dengan *pooling*, kita memperoleh *translation invariance* yang kurang lebih.

MLP: sambungan penuh
$6 \times 4 = 24$ *weight*

CNN: sambungan lokal
3 *weight* dipakai bersama



warna sama = *weight* sama

Versi satu dimensi. Pada CNN setiap neuron *output* hanya melihat tiga *input* bertetangga, dan ketiga *weight*-nya (merah, biru, emas) sama di semua posisi.

Penghematannya, dan apa artinya

Aspek	**MLP, 1.000 neuron**	**Konvolusi, 64 *filter* 3×3**
Jumlah parameter	$3.072 \times 1.000 + 1.000 = 3.073.000$	$64 \times 27 + 64 = 1.792$
Bergantung ukuran citra?	Ya, tumbuh linier dengan jumlah piksel	Tidak
Memanfaatkan tetangga	Harus dipelajari dari data	Ya, lewat *receptive field*
Pola dipakai ulang	Tidak	Ya, *weight sharing*
Respons terhadap pergeseran	Berubah tak teratur	Ikut bergeser (*equivariant*)

Kembali ke Pertemuan 2

CNN adalah MLP yang diberi *inductive bias* yang cocok untuk citra: pola itu lokal dan bisa muncul di mana saja. Kita belajar sejak awal semester bahwa bias induktif yang tepat adalah kunci *generalization*. CNN menuliskan pengetahuan tentang struktur citra langsung ke dalam arsitekturnya, sehingga data yang dibutuhkan jauh lebih sedikit.

Operasi konvolusi

Definisi konvolusi dua dimensi

Input X, *kernel* K berukuran $k \times k$. *Output* pada posisi (i, j) adalah jumlah hasil kali elemen demi elemen antara K dan petak X yang dimulai di (i, j):

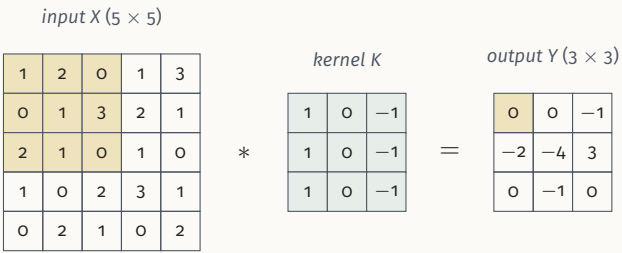
$$Y[i, j] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} K[a, b]\, X[i + a,\, j + b] + b_0$$

dengan b_0 satu bias per *filter*. Bayangkan *kernel* sebagai jendela kecil yang ditempelkan ke citra: hitung skor kecocokannya, geser satu langkah, ulangi.

Konvolusi atau korelasi silang?

Dalam pengolahan sinyal, konvolusi sejati membalik *kernel* lebih dahulu, $\sum K[a, b]\, X[i - a, j - b]$. Rumus di atas tanpa pembalikan sebenarnya ***cross-correlation***, dan PyTorch memakainya sambil tetap menyebutnya konvolusi. Karena *weight kernel* dipelajari, perbedaan ini tidak berpengaruh pada model. Namun hasil `scipy.signal.convolve2d` akan berbeda dari `torch.nn.Conv2d` bila Anda membandingkannya.

Contoh terhitung: *input* 5×5, *kernel* detektor tepi tegak



Kolom kiri *kernel* bernilai 1, kolom tengah 0, kolom kanan -1. Artinya *kernel* ini menghitung **jumlah kolom kiri dikurangi jumlah kolom kanan** pada setiap petak. Petak berwarna menghasilkan $Y[0, 0]$. Tanpa *padding* dan dengan *stride* 1, *kernel* hanya muat di 3×3 posisi, sehingga *output*-nya menyusut dari 5×5 ke 3×3.

Nilai positif besar berarti sisi kiri petak lebih terang daripada sisi kanan; nilai negatif berarti sebaliknya.

Menghitung tiga elemen dengan tangan

$Y[0, 0]$

Petak baris 0–2, kolom 0–2:

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 3 \\ 2 & 1 & 0 \end{bmatrix}$$

Kalikan dengan K lalu jumlahkan:

$$(1+0+0)+(0+0-3)+(2+0+0) = 0$$

Cara cepat: kolom kiri $1+0+2=3$, kolom kanan $0+3+0=3$, selisih 0.

$Y[1, 2]$

Petak baris 1–3, kolom 2–4:

$$\begin{bmatrix} 3 & 2 & 1 \\ 0 & 1 & 0 \\ 2 & 3 & 1 \end{bmatrix}$$

Kolom kiri $3+0+2=5$, kolom kanan $1+0+1=2$, sehingga

$$Y[1, 2] = 5 - 2 = 3$$

Petak ini memotong daerah terang di kiri dan redup di kanan.

$Y[1, 1]$

Petak baris 1–3, kolom 1–3:

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 1 \\ 0 & 2 & 3 \end{bmatrix}$$

Kolom kiri $1+1+0=2$, kolom kanan $2+1+3=6$, sehingga

$$Y[1, 1] = 2 - 6 = -4$$

Respons terkuat, tetapi berlawanan arah: kanan lebih terang.

Ringkasan

Satu elemen *output* adalah k^2 perkalian lalu penjumlahan. Ulangi untuk sembilan posisi dan Anda memperoleh Y di slide sebelumnya. Di dalam *layer* konvolusi tidak ada yang lebih rumit dari itu.

Mengapa disebut detektor tepi

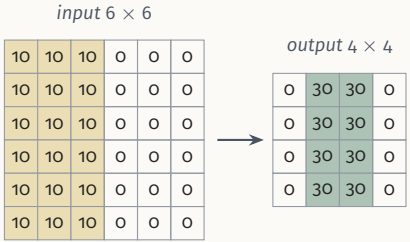
Ambil citra 6×6 yang separuh kirinya terang (nilai 10) dan separuh kanannya gelap (nilai 0). Setiap baris berbentuk $[10, 10, 10, 0, 0, 0]$.

Dengan *kernel* yang sama, setiap baris *output* 4×4 bernilai

$$[\,0,\ 30,\ 30,\ 0\,]$$

- Di dua kolom pinggir petaknya seragam, kolom kiri sama dengan kolom kanan, hasil nol.
- Di dua kolom tengah petak memotong batas: kiri berjumlah 30, kanan 0.

Output menyala tepat di lokasi tepi. Karena itu *feature map* sering disebut **peta respons**: ia menunjukkan *di mana* pola yang dicari ditemukan.



Sebelum *deep learning*, ahli pengolahan citra merancang *kernel* seperti ini dengan tangan. Pada CNN, nilai *kernel* **dipelajari** dari data dengan *backpropagation*.

Kernel klasik yang dulu dirancang manusia

Nama	***Kernel***	**Efek**
Rata-rata (*box blur*)	$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$	Menghaluskan citra, meredam *noise*
Sobel horizontal	$\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$	Menonjolkan tepi tegak (perubahan kecerahan arah horizontal)
Sobel vertikal	$\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$	Menonjolkan tepi mendatar
Sharpen	$\begin{bmatrix}0&-1&0\\-1&5&-1\\0&-1&0\end{bmatrix}$	Mempertajam detail dengan menonjolkan selisih terhadap tetangga

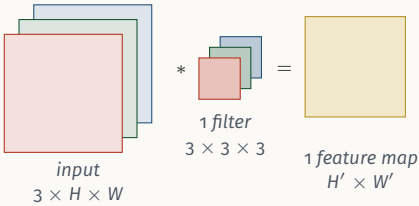
Layer pertama CNN yang terlatih pada ImageNet ternyata menemukan sendiri *filter* yang mirip dengan detektor tepi dan gumpalan warna ini. Kita akan melihatnya di bagian visualisasi.

Input berkanal banyak

Untuk *input* C_{in} kanal, setiap *filter* juga punya C_{in} irisan, sehingga ukurannya $C_{\text{in}} \times k \times k$. Setiap irisan dikonvolusikan dengan kanal pasangannya, lalu **semua hasilnya dijumlahkan**:

$$Y[i,j] = \sum_{c=1}^{C_{\text{in}}} \sum_{a,b} K[c,a,b]\, X[c,\, i+a,\, j+b] + b_{\text{o}}$$

- Satu *filter* selalu menghasilkan **satu** peta, berapa pun kanal *input*-nya.
- C_{out} *filter* menghasilkan *output* berkanal C_{out}, yang menjadi *input layer* berikutnya.



Contoh: *layer* pertama untuk citra RGB dengan 16 *filter* 3×3 memegang 16 *tensor weight* berukuran $3 \times 3 \times 3$, dan *output*-nya 16 *feature map*.

Filter, *feature map*, dan parameter

Satu *layer* konvolusi utuh, dan jumlah parameternya

Satu ***layer* konvolusi** berisi C_{out} *filter*. Setiap *filter* menghasilkan satu ***feature map***. Setelah itu hampir selalu diterapkan ReLU.

Pada contoh tadi, ReLU mengubah Y menjadi $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix}$: hanya respons positif yang diteruskan. Tanpa aktivasi non-linier, tumpukan *layer* konvolusi setara dengan satu konvolusi linier saja.

Rumus jumlah parameter

$$\#\text{param} = C_{\text{out}} \times (C_{\text{in}} \cdot k \cdot k) + C_{\text{out}}$$

Suku terakhir adalah bias. Ukuran citra **tidak** muncul di rumus ini.

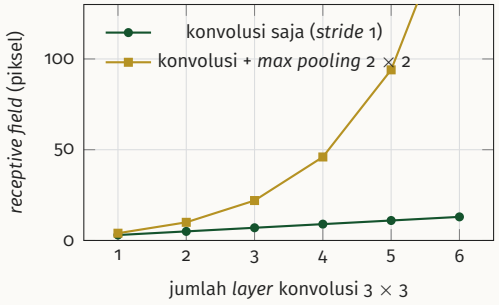
Contoh terhitung

Input $3 \times 32 \times 32$, *layer* `Conv2d(3, 16, kernel_size=3, padding=1)`.

- *Weight*: $16 \times (3 \cdot 3 \cdot 3) = 16 \times 27 = 432$
- Bias: 16, sehingga total 448 parameter
- *Output*: $16 \times 32 \times 32$ (*padding* 1 menjaga ukuran)
- Perkalian: setiap elemen *output* perlu 27, jadi $16 \cdot 32 \cdot 32 \cdot 27 = 442.368$ perkalian

Jadi 448 parameter memerlukan hampir setengah juta perkalian untuk satu citra. *Computational cost* CNN ditentukan ukuran *feature map*, sehingga *layer* awal yang beresolusi tinggi paling mahal.

Hierarki *feature* dan *receptive field* yang membesar



Nilai *layer* ke-6 dengan *pooling*, 190, berada di luar sumbu.

Layer awal menanggapi tepi dan gradasi warna, *layer* tengah tekstur dan bagian objek, *layer* akhir objek utuh. Hierarki ini mungkin karena *receptive field* membesar bersama kedalaman.

Tumpukan L *layer* 3×3 ber-*stride* 1 melihat $2L + 1$ piksel. Rumus umumnya $r_\ell = r_{\ell-1} + (k_\ell - 1)\, j_{\ell-1}$, dengan j hasil kali semua *stride* sebelumnya.

Telusuri: konv 3×3, *pool* 2×2 ($s = 2$), konv 3×3.

- Awal $r_0 = 1$, $j_0 = 1$
- Konv 1: $r_1 = 1 + 2 \cdot 1 = 3$
- *Pool*: $r_2 = 3 + 1 \cdot 1 = 4$, $j_2 = 2$
- Konv 2: $r_3 = 4 + 2 \cdot 2 = 8$

Satu neuron di *layer* ketiga melihat petak 8×8 piksel citra asli.

Stride, *padding*, dan dimensi *output*

Stride dan *padding*

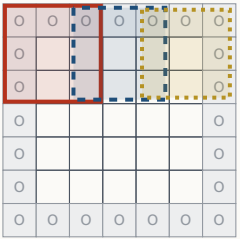
Stride s adalah jarak geser *kernel*. Dengan $s = 2$, *output* kira-kira separuh per sumbu. Ini cara murah untuk *downsampling* sambil tetap belajar.

Tanpa tambahan, konvolusi $k \times k$ memperkecil peta sebesar $k - 1$. Sepuluh *layer* 3×3 membuat 32×32 menyusut ke 12×12, dan piksel pinggir jarang dikunjungi. ***Padding*** menambah bingkai, biasanya nol.

- ***Valid***: $p = 0$, *output* menyusut.
- ***Same***: *output* sama dengan *input* (untuk $s = 1$). Untuk k ganjil $p = (k - 1)/2$: $p = 1$ untuk 3×3, $p = 2$ untuk 5×5.

Itulah alasan *kernel* hampir selalu ganjil: punya titik tengah, dan *padding same*-nya simetris.

5×5 + *padding* $p = 1 \Rightarrow 7 \times 7$



Kernel 3×3 dengan *stride* 2 hanya singgah di tiga posisi per baris.

Rumus dimensi *output*

Dengan ukuran *input* n, *kernel* k, *padding* p, dan *stride* s, ukuran *output* per sumbu adalah

$$n_{\text{out}} = \left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1$$

Alasannya. Setelah *padding*, panjangnya $n + 2p$. Posisi awal *kernel* yang sah adalah $0, s, 2s, \ldots$ selama ujung *kernel* tidak melewati batas, yaitu selama posisi awal $\leq n + 2p - k$. Banyaknya posisi seperti itu $\lfloor (n + 2p - k)/s \rfloor + 1$. Rumus yang sama berlaku untuk *pooling*.

Bentuk *output* satu *layer*

Input $C_{\text{in}} \times H \times W$ menjadi $C_{\text{out}} \times H_{\text{out}} \times W_{\text{out}}$. Jumlah kanal ditentukan jumlah *filter*; ukuran ruang ditentukan k, p, s.

Kebiasaan yang menyelamatkan

Tulis bentuk setiap *layer* di kertas sebelum menulis kode. *Shape mismatch error* paling sering muncul saat merakit CNN.

Latihan rumus dimensi, dikerjakan bersama

n	k	p	s	$\lfloor (n + 2p - k)/s \rfloor + 1$
5	3	0	1	$\lfloor 2/1 \rfloor + 1 = 3$
5	3	1	2	$\lfloor 4/2 \rfloor + 1 = 3$
32	3	1	1	$\lfloor 31/1 \rfloor + 1 = 32$ (*same*)
28	5	0	1	$\lfloor 23/1 \rfloor + 1 = 24$
224	7	3	2	$\lfloor 223/2 \rfloor + 1 = 111 + 1 = 112$

Baris pertama cocok dengan contoh 5×5 tadi, baris kedua dengan gambar *padding* dan *stride*. Baris terakhir adalah *layer* pertama ResNet: citra 224×224 menjadi 112×112.

Hasil baris kedua. *Input* 5×5 contoh kita, *padding* 1, *stride* 2, *kernel* yang sama:

$$Y = \begin{bmatrix} -3 & 0 & 3 \\ -2 & -4 & 6 \\ -2 & -1 & 3 \end{bmatrix}$$

Elemen tengah -4 sama dengan $Y[1, 1]$ sebelumnya, karena petaknya kebetulan sama. Elemen pinggir kini memakai nol dari bingkai. Silakan periksa dengan kode NumPy beberapa slide lagi.

Coba sendiri

Input $64 \times 56 \times 56$, *layer* dengan 128 *filter* 3×3, $s = 2$, $p = 1$. Hitung bentuk *output* dan jumlah parameternya. (Jawaban di akhir slide tugas.)

Ketika pembagiannya tidak bulat

Bila $(n + 2p - k)$ tidak habis dibagi s, baris atau kolom terakhir tidak pernah dikunjungi *kernel* dan diam-diam terbuang.

Contoh terkenal adalah *layer* pertama AlexNet. Makalahnya menyebut *input* 224×224, *kernel* 11×11, *stride* 4:

$$\frac{224 - 11}{4} + 1 = 54{,}25$$

tidak bulat. Ukuran 227×227 memberi tepat $(227 - 11)/4 + 1 = 55$, sehingga banyak implementasi ulang memakai 227.

Hiper-parameter	**Bila diperbesar**	**Pilihan umum**
Ukuran *kernel* k	*Receptive field* lebih luas, parameter naik kuadratik	3×3; 7×7 hanya di *layer* pertama
Jumlah *filter* C_{out}	Lebih banyak pola, memori dan komputasi naik	32, 64, 128, berlipat dua setelah tiap *downsampling*
Stride s	*Output* mengecil kira-kira $1/s$ per sumbu	1; 2 untuk *downsampling*
Padding p	*Output* membesar, pinggir lebih terwakili	$(k - 1)/2$ (*same*)

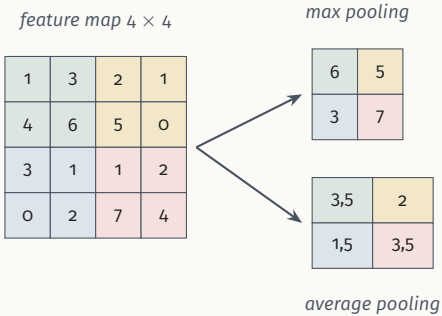
Pooling

Max pooling dan *average pooling*

Pooling meringkas setiap petak kecil *feature map* menjadi satu angka. Tujuannya mengecilkan peta, memperluas *receptive field*, dan membuat representasi sedikit lebih tahan pergeseran. *Pooling* bekerja per kanal dan **tidak punya parameter**.

- ***Max pooling***: nilai terbesar petak. "Apakah pola ini muncul di mana pun dalam petak?"
- ***Average pooling***: rata-rata petak. Lebih halus.

Contoh 2×2**,** $s = 2$**.** Petak kiri atas $\{1, 3, 4, 6\}$: maksimum 6, rata-rata $14/4 = 3{,}5$. Kanan atas $\{2, 1, 5, 0\}$: 5 dan 2. Kiri bawah $\{3, 1, 0, 2\}$: 3 dan 1,5. Kanan bawah $\{1, 2, 7, 4\}$: 7 dan 3,5. Dimensinya $\lfloor (4-2)/2 \rfloor + 1 = 2$.



Warna menunjukkan petak asal setiap nilai *output*.

Global average pooling, *gradient*, dan pilihan desain

Global average pooling (GAP)

Merata-ratakan seluruh *feature map* menjadi satu angka per kanal: $512 \times 7 \times 7$ menjadi vektor sepanjang 512. GAP menggantikan *layer fully connected* besar di ujung jaringan, memangkas parameter, dan membuat jaringan menerima citra berbagai ukuran. ResNet memakainya sebelum *layer classification*.

Gradient melewati *pooling*

Max pooling meneruskan *gradient* hanya ke posisi yang menjadi maksimum; posisi lain menerima nol. *Average pooling* membagi *gradient* rata ke semua posisi di petak.

Pooling atau *stride*?

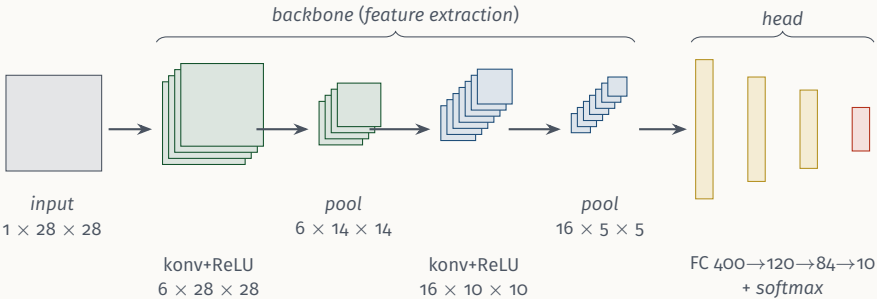
LeNet, AlexNet, dan VGG memakai *pooling* untuk *downsampling*. Banyak arsitektur sesudahnya, termasuk ResNet, lebih banyak memakai konvolusi ber-*stride* 2 yang dapat belajar cara meringkas, ditambah GAP di akhir.

Ringkasan

Untuk proyek pertama Anda, pola "konvolusi, ReLU, *max pooling*" yang diulang beberapa kali tetap pilihan yang aman dan mudah dibaca.

Menyusun dan melatih CNN

Pola arsitektur: *backbone* lalu *head*



1. **Ekstraksi *feature***: beberapa blok konvolusi, aktivasi, *pooling*. Ukuran ruang mengecil ($28 \to 14 \to 10 \to 5$), jumlah kanal membesar ($1 \to 6 \to 16$).
2. ***Flatten*** atau GAP: *tensor* menjadi vektor.
3. ***Classification***: *layer fully connected*, diakhiri *softmax*.

Bagian pertama disebut ***backbone***, bagian akhir ***head***. Kedua istilah ini kembali saat membahas *transfer learning*.

Menelusuri dimensi dan parameter CNN bergaya LeNet-5

Layer	*Configuration*	*Output*	**Parameter**
Input	–	$1 \times 28 \times 28$	0
Konv 1	6 *filter* 5×5, $p = 2$	$6 \times 28 \times 28$	$6 \cdot 25 + 6 = 156$
Max pool	2×2, $s = 2$	$6 \times 14 \times 14$	0
Konv 2	16 *filter* 5×5, $p = 0$	$16 \times 10 \times 10$	$16 \cdot 150 + 16 = 2.416$
Max pool	2×2, $s = 2$	$16 \times 5 \times 5$	0
Flatten	–	400	0
FC 1	$400 \to 120$	120	$48.000 + 120 = 48.120$
FC 2	$120 \to 84$	84	$10.080 + 84 = 10.164$
FC 3	$84 \to 10$	10	$840 + 10 = 850$
Total			**61.706**

Periksa dua baris. Konv 2 menerima 6 kanal, sehingga satu *filter* berisi $6 \cdot 5 \cdot 5 = 150$ *weight*. Ukurannya $\lfloor (14 - 5)/1 \rfloor + 1 = 10$.

Lebih dari tiga perempat parameter berada di *layer fully connected*, padahal konvolusilah yang mengerjakan ekstraksi *feature*-nya.

Pembanding: MLP $784 \to 128 \to 10$ sudah memiliki $100.480 + 1.290 = 101.770$ parameter, tanpa memanfaatkan struktur citra sama sekali.

Konv 1 memakai $p = 2$ agar MNIST 28×28 berperilaku seperti input 32×32 pada LeNet-5 asli.

Konvolusi dari nol dengan NumPy

```python
import numpy as np

def conv2d(x, w, b, stride=1, padding=0):
    """x: (C_in,H,W), w: (C_out,C_in,k,k), b: (C_out,)"""
    c_out, c_in, k, _ = w.shape
    p = padding
    x = np.pad(x, ((0, 0), (p, p), (p, p)))
    h_out = (x.shape[1] - k) // stride + 1
    w_out = (x.shape[2] - k) // stride + 1
    y = np.zeros((c_out, h_out, w_out))
    for o in range(c_out):
        for i in range(h_out):
            for j in range(w_out):
                r, c = i * stride, j * stride
                petak = x[:, r:r+k, c:c+k]
                y[o, i, j] = np.sum(petak * w[o]) + b[o]
    return y

X = np.array([[1,2,0,1,3],[0,1,3,2,1],[2,1,0,1,0],
              [1,0,2,3,1],[0,2,1,0,2]], dtype=float)[None]
K = np.array([[1,0,-1]]*3, dtype=float)[None, None]
print(conv2d(X, K, np.zeros(1)))         # 3x3, sama dengan contoh
print(conv2d(X, K, np.zeros(1), 2, 1))   # stride 2, padding 1
```

- Rumus dimensi *output* muncul apa adanya di baris `h_out` dan `w_out`.
- `np.pad` membuat bingkai nol; kanal tidak diberi *padding*.
- `petak * w[o]` mengalikan seluruh kanal sekaligus, lalu `np.sum` menjumlahkan semuanya menjadi satu angka.

Lambat karena perulangan bersarang, tetapi jujur. *Library* seperti cuDNN memakai teknik ***im2col***: semua petak disalin menjadi kolom matriks besar, sehingga konvolusi menjadi satu perkalian matriks yang cepat di GPU.

CNN bergaya LeNet-5 dengan PyTorch

```python
import torch
from torch import nn

class LeNet(nn.Module):
    def __init__(self, n_kelas=10):
        super().__init__()
        self.backbone = nn.Sequential(
            nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.ReLU(),
            nn.MaxPool2d(2),                    # 6 x 14 x 14
            nn.Conv2d(6, 16, kernel_size=5), nn.ReLU(),
            nn.MaxPool2d(2),                    # 16 x 5 x 5
        )
        self.head = nn.Sequential(
            nn.Flatten(),
            nn.Linear(16 * 5 * 5, 120), nn.ReLU(),
            nn.Linear(120, 84), nn.ReLU(),
            nn.Linear(84, n_kelas),             # logit, tanpa softmax
        )
    def forward(self, x):
        return self.head(self.backbone(x))
model = LeNet()
print(sum(p.numel() for p in model.parameters()))   # 61706
print(model(torch.zeros(8, 1, 28, 28)).shape)        # [8, 10]
```

`nn.Conv2d` menerima kanal *input*, jumlah *filter*, ukuran *kernel*, *stride*, dan *padding*. Komentar bentuk di samping setiap blok adalah kebiasaan baik: Anda menulis hitungan rumus dimensi langsung di kode.

Dua baris terakhir adalah **uji asap** yang wajib: hitung parameter dan jalankan satu *tensor* kosong. Bila $16 \cdot 5 \cdot 5$ salah hitung, *error*-nya langsung muncul di sini sebelum *training* dimulai.

Layer terakhir sengaja tanpa *softmax*: `nn.CrossEntropyLoss` sudah menggabungkan *log-softmax* dan *negative log-likelihood* dengan cara yang stabil secara numerik.

Melatihnya: perulangan yang sama seperti MLP

```
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
tf = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,))])
latih = datasets.MNIST("data", train=True,
                       download=True, transform=tf)
loader = DataLoader(latih, batch_size=64, shuffle=True)

perangkat = "cuda" if torch.cuda.is_available() else "cpu"
model = LeNet().to(perangkat)
opt = torch.optim.Adam(model.parameters(), lr=1e-3)
loss_fn = nn.CrossEntropyLoss()

for epoch in range(5):
    model.train()
    for x, y in loader:
        x, y = x.to(perangkat), y.to(perangkat)
        opt.zero_grad()
        loss = loss_fn(model(x), y)
        loss.backward()     # gradien mengalir juga ke filter
        opt.step()
    print(epoch, loss.item())
```

Tidak ada yang baru dibandingkan Pertemuan 11: *forward*, hitung *loss*, *backward*, langkah *optimizer*.

0,1307 dan 0,3081 adalah rata-rata dan simpangan baku piksel MNIST yang lazim dipakai pada contoh resmi PyTorch. `ToTensor()` sekaligus mengubah urutan HWC menjadi CHW dan membagi nilai dengan 255.

Untuk evaluasi, tambahkan `model.eval()` dan `torch.no_grad()`, lalu hitung *accuracy* pada *test data*. Untuk perbandingan yang adil dengan MLP, samakan *optimizer*, *learning rate*, jumlah *epoch*, dan *batch size*.

Backpropagation melalui *layer* konvolusi

Tidak ada aturan baru: tetap aturan rantai dari Pertemuan 10. Yang khas adalah *weight sharing*. Satu *weight* $K[a, b]$ dipakai di **semua** posisi *output*, sehingga *gradient*-nya adalah jumlah sumbangan dari semua posisi itu:

$$\frac{\partial L}{\partial K[a,b]} = \sum_i \sum_j \frac{\partial L}{\partial Y[i,j]} \, X[i+a, j+b]$$

- Bentuk ini sendiri sebuah *cross-correlation* antara *input* X dan *gradient output*.
- *Gradient* terhadap *input*, $\partial L/\partial X$, adalah konvolusi *gradient output* dengan *kernel* yang dibalik.

Ringkasan

Backward pass CNN juga tersusun dari konvolusi, sehingga GPU mempercepatnya seperti *forward pass*. Karena *gradient* dijumlahkan dari semua posisi, satu *filter* belajar dari seluruh citra sekaligus.

Kesalahan umum saat merakit CNN

Dimensi dan data

- **Salah urutan dimensi.** PyTorch mengharapkan NCHW, sedangkan citra dari PIL atau OpenCV berbentuk HWC. Pakai `ToTensor()` atau `permute`.
- **Salah hitung ukuran *flatten*.** Hitung dengan rumus, atau jalankan satu *tensor* kosong lalu cetak bentuknya.
- **Lupa *normalization*.** Piksel 0–255 yang tidak diskalakan membuat *training* lambat atau tidak stabil.

Training dan evaluasi

- ***Softmax* sebelum `CrossEntropyLoss`.** Hasilnya *softmax* ganda dan *gradient* yang lemah.
- **Lupa `model.eval()`** saat evaluasi bila memakai *dropout* atau *batch normalization*.
- ***Augmentation* yang merusak label.** Mencerminkan angka “2” pada MNIST menghasilkan simbol yang bukan angka. *Augmentation* hanya untuk *training data*.

Arsitektur klasik: dari LeNet sampai ResNet

LeNet-5 (1998) dan AlexNet (2012)

LeNet-5 — LeCun dkk.

Dirancang untuk angka tulisan tangan, dan versi-versinya dipakai membaca cek bank di Amerika Serikat. *Input* 32×32; konvolusi 5×5, *subsampling*, konvolusi lagi, lalu *fully connected*. Makalahnya menyebut sekitar 60 ribu parameter.

Hampir semua bahan CNN modern sudah ada di sini. Yang belum tersedia saat itu adalah data dan komputasi untuk masalah yang lebih besar. Versi di slide sebelumnya adalah versi modern (ReLU, *max pooling*); LeNet asli memakai aktivasi berbentuk *tanh* dan *subsampling* dengan koefisien yang dilatih.

AlexNet — Krizhevsky, Sutskever, Hinton

Menandai kebangkitan *deep learning*. Pada ILSVRC 2012 (sekitar 1,2 juta citra latih, 1.000 kelas), *top-5 error*-nya sekitar 15,3%, sedangkan peserta terbaik berikutnya sekitar 26,2%. Lima *layer* konvolusi, tiga *layer fully connected*, sekitar 60 juta parameter.

- ReLU menggantikan *tanh* dan sigmoid
- *Training* di GPU (dua GPU sekaligus)
- *Dropout* pada *layer fully connected*
- *Augmentation*: potongan acak dan pencerminan

VGG (2015): cukup 3×3, tetapi dalam

Simonyan dan Zisserman bertanya: bagaimana jika semua *kernel* dibuat 3×3 dan jaringan dibuat 16 sampai 19 *weight layer*? Dua *layer* 3×3 bertumpuk punya *receptive field* 5×5, dan tiga *layer* punya 7×7.

Contoh terhitung (*input* dan *output* berkanal C, bias diabaikan)

- Satu *layer* 7×7: $7^2C^2 = 49C^2$ parameter, satu ReLU
- Tiga *layer* 3×3: $3 \times 3^2C^2 = 27C^2$ parameter, tiga ReLU

Untuk $C = 256$, dengan $C^2 = 65.536$:

$$49 \cdot 65.536 = 3.211.264$$

$$27 \cdot 65.536 = 1.769.472$$

Ringkasan

Penghematan sekitar 45% dengan *receptive field* yang sama, ditambah dua non-linieritas lagi sehingga fungsinya lebih ekspresif.

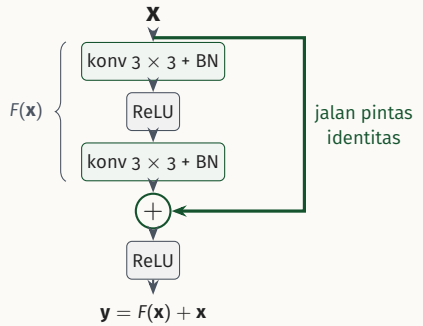
Kelemahan VGG ada pada ukurannya: VGG-16 sekitar 138 juta parameter, sebagian besar di *layer fully connected*. Keteraturannya membuat VGG tetap populer sebagai *feature extractor* dan bahan ajar.

ResNet (2016): *residual connection*

Ketika jaringan biasa diperdalam, jaringan 56 *layer* justru punya *training error* lebih tinggi daripada 20 *layer* (CIFAR-10). Karena *error* pada *training data* pun naik, penyebabnya kesulitan *optimization* (***degradation problem***), dan *overfitting* tidak dapat menjelaskannya.

He dkk. membuat setiap blok mempelajari sisanya, $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$, lalu menambahkan *input* kembali: $\mathbf{y} = F(\mathbf{x}) + \mathbf{x}$.

- *Layer* tambahan yang tidak berguna cukup membuat $F(\mathbf{x}) \approx \mathbf{o}$.
- $\partial \mathbf{y}/\partial \mathbf{x} = \partial F/\partial \mathbf{x} + I$ selalu memuat identitas, sehingga *gradient* punya jalan pintas ke *layer* awal.
- Dengan *batch normalization*, He dkk. melatih 152 *layer* dan memenangi ILSVRC 2015.



Ide yang sama muncul di Pertemuan 12, dan akan muncul lagi di setiap blok *transformer* minggu depan.

Konvolusi 1×1, dan keempat arsitektur berdampingan

Konvolusi 1×1 tidak melihat tetangga; ia mencampur kanal di setiap piksel, seperti *layer fully connected* kecil yang dijalankan di setiap posisi. Gunanya mengubah jumlah kanal secara murah, misalnya $256 \to 64$ sebelum konvolusi 3×3 yang mahal. Mulai ResNet-50, blok memakai bentuk ***bottleneck*** $1 \times 1 \to 3 \times 3 \to 1 \times 1$.

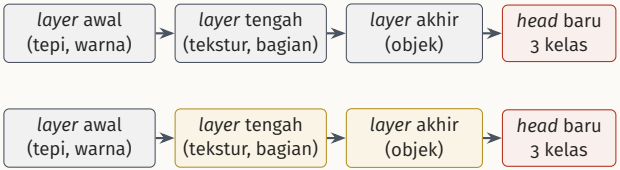
Arsitektur	Tahun	Kedalaman dan ukuran	Gagasan kunci
LeNet-5	1998	7 *layer*; sekitar 60 ribu parameter	Konvolusi + *subsampling* dilatih ujung ke ujung
AlexNet	2012	8 *weight layer*; sekitar 60 juta	ReLU, GPU, *dropout*, *augmentation*; skala ImageNet
VGG-16/19	2015	16/19 *layer*; VGG-16 sekitar 138 juta	Hanya 3×3, dalam dan seragam
ResNet	2016	18 sampai 152 *layer*	*Residual connection*, BN, *bottleneck*, GAP

Sejak sekitar 2021, *vision transformer* (ViT) yang membagi citra menjadi petak lalu memprosesnya dengan *attention* menjadi pesaing CNN, terutama bila data pralatih sangat besar. CNN tetap unggul efisiensi pada data kecil sampai sedang dan pada perangkat terbatas. ResNet dan *derivative*-nya masih *baseline* yang wajib dicoba.

Transfer learning pada citra

Meminjam *backbone* yang sudah terlatih

Melatih ResNet dari nol perlu jutaan citra berlabel dan waktu GPU berhari-hari, sedangkan kasus daun padi hanya punya beberapa ratus foto. *Layer* awal CNN mempelajari tepi, warna, dan tekstur yang berguna untuk hampir semua citra; Yosinski dkk. menunjukkan *feature layer* awal bersifat umum dan *feature layer* akhir makin spesifik pada tugas aslinya.



Feature extraction: bekukan seluruh *backbone*, ganti *head*, latih *head* saja. Cepat, hemat data, sulit *overfitting*. ***Fine-tuning***: setelah *head* terlatih, buka sebagian *layer* dan latih ulang dengan *learning rate* kecil. Lebih akurat bila data cukup atau domainnya berbeda.

Memilih strategi, dan peran *augmentation*

	Domain mirip ImageNet foto benda sehari-hari	**Domain berbeda** citra medis, satelit, mikroskop
Data sedikit	*Feature extraction*; latih *head* saja	*Feature extraction* dari *layer* yang lebih awal, *augmentation* kuat; hasil bisa terbatas
Data banyak	*Fine-tuning* seluruh jaringan dengan *learning rate* kecil	*Fine-tuning* penuh, atau pralatih ulang pada data domain bila sangat banyak

Data augmentation

Untuk citra kita tahu *transformation* mana yang tidak mengubah label: pencerminan horizontal, potongan acak, rotasi kecil, perubahan kecerahan dan kontras. Transformasinya harus masuk akal untuk domainnya. Mencerminkan daun padi tetap menghasilkan daun padi; mencerminkan angka “2” tidak. *Augmentation* hanya diterapkan pada *training data*, tidak pada *validation data* dan uji.

Transfer learning dengan ResNet-18 terlatih

```python
import torch
from torch import nn
from torchvision import models

bobot = models.ResNet18_Weights.DEFAULT
model = models.resnet18(weights=bobot)
praproses = bobot.transforms()   # normalisasi ImageNet

# 1) Feature extraction: bekukan seluruh backbone
for p in model.parameters():
    p.requires_grad = False
model.fc = nn.Linear(model.fc.in_features, 3)  # head baru
opt = torch.optim.Adam(model.fc.parameters(), lr=1e-3)
# ... latih beberapa epoch seperti perulangan tadi ...

# 2) Fine-tuning: buka blok terakhir, learning rate kecil
for p in model.layer4.parameters():
    p.requires_grad = True
opt = torch.optim.Adam([
    {"params": model.layer4.parameters(), "lr": 1e-4},
    {"params": model.fc.parameters(),     "lr": 1e-3},
])
```

`model.fc` adalah *head* ResNet-18: `Linear(512, 1000)` untuk seribu kelas ImageNet. Kita menggantinya dengan `Linear(512, 3)` untuk tiga kelas daun padi.

Layer baru dibuat setelah pembekuan, sehingga parameternya tetap `requires_grad=True` dan hanya ia yang dilatih pada tahap pertama.

Pada tahap kedua, `layer4` (blok residual terakhir) dibuka dengan *learning rate* sepuluh kali lebih kecil daripada *head*. *Layer* awal tetap beku karena *feature*-nya umum.

Jebakan *transfer learning*

- **Praproses harus sama** dengan saat pralatih: ukuran citra serta rata-rata dan simpangan baku ImageNet. Itulah gunanya `bobot.transforms()`.
- ***Learning rate* terlalu besar** saat *fine-tuning* merusak *feature* yang sudah bagus (*catastrophic forgetting*).
- ***Batch normalization*** pada data sangat sedikit bisa tidak stabil; sering lebih aman membekukan statistik BN.
- **Kebocoran data**: foto daun dari tanaman yang sama bisa masuk *training data* dan *test data* dengan sudut berbeda. Bagi data per tanaman atau per lahan.
- **Lisensi**: periksa lisensi *weight* terlatih dan *dataset* sebelum dipakai komersial.

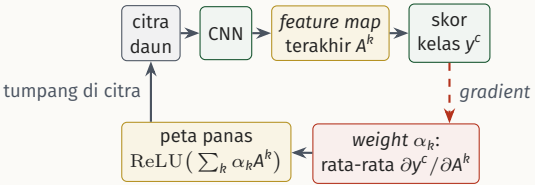
Kembali ke Pertemuan 9

Pembagian data yang salah membuat angka uji terlalu indah. Pada citra, bagi data per objek atau per lokasi, sehingga semua foto dari satu tanaman berada di sisi yang sama.

Visualisasi *filter* dan *feature map*

Melihat isi kepala CNN

1. ***Weight filter layer* pertama** berukuran $C_{\text{out}} \times 3 \times k \times k$ dapat ditampilkan langsung sebagai citra kecil. Pada CNN terlatih, isinya detektor tepi berbagai arah dan gumpalan warna, mirip *filter* Gabor.
2. ***Feature map***: masukkan satu citra, ambil *output layer* tertentu dengan *forward hook*, tampilkan sebagai peta panas. *Layer* awal tampak seperti citra dengan tepi ditonjolkan; *layer* dalam kasar tetapi menyala di bagian objek.
3. ***Saliency map* dan Grad-CAM**: bagian citra mana yang mendorong keputusan kelas ini?



Grad-CAM: *gradient* skor kelas terhadap *feature map* terakhir menentukan *weight* tiap peta; *weighted sum*-nya menjadi peta panas di atas citra.

Menampilkan 64 *filter layer* pertama ResNet-18

```python
import matplotlib.pyplot as plt
from torchvision import models

bobot = models.ResNet18_Weights.DEFAULT
model = models.resnet18(weights=bobot)
w = model.conv1.weight.detach()          # bentuk (64, 3, 7, 7)
w = (w - w.min()) / (w.max() - w.min())  # skala ke [0, 1]

fig, axes = plt.subplots(8, 8, figsize=(6, 6))
for i, ax in enumerate(axes.flat):
    ax.imshow(w[i].permute(1, 2, 0))     # CHW -> HWC
    ax.axis("off")
plt.savefig("filter_resnet18.png", dpi=150)
```

Perhatikan `permute`: PyTorch menyimpan CHW, matplotlib mengharapkan HWC. Ini kesalahan urutan dimensi yang sama seperti di slide kesalahan umum, hanya ke arah sebaliknya.

Peta panas perlu dikonfirmasi

Peta panas yang tampak masuk akal tidak membuktikan model benar. Model bisa beroleh *accuracy* tinggi karena melihat tanda air, latar belakang, atau penggaris yang kebetulan hanya ada di foto kelas tertentu.

Visualisasi paling berguna untuk **menemukan** jalan pintas semacam itu. Pakai sebagai alat diagnosis, lalu konfirmasi dengan *test data* yang dirancang khusus.

Kapan memakai CNN

CNN cocok untuk data berstruktur grid

Jenis data	Pilihan yang dianjurkan
Tabel, *feature* heterogen	Model klasik dan *gradient boosting* (Pertemuan 4) biasanya lebih baik
Citra, data kecil–sedang	CNN terlatih + *transfer learning*
Citra, data dan komputasi sangat besar	CNN modern atau *vision transformer*
Sinyal 1D (audio, sensor, EKG)	Konvolusi 1D, `nn.Conv1d`
Volume 3D (CT, MRI), video	Konvolusi 3D, atau CNN 2D + model urutan
Teks	Dahulu CNN 1D dan RNN; kini umumnya *transformer* (Pertemuan 14)

Kasus Indonesia

Deteksi penyakit tanaman dari foto daun, pemilahan sampah, pembacaan plat nomor, verifikasi foto KTP saat *onboarding* layanan keuangan digital, dan *classification* tutupan lahan dari citra satelit. Hampir semuanya punya data berlabel terbatas, sehingga *transfer learning* dan *augmentation* menjadi strategi pertama.

Data wajah dan identitas

Pastikan ada dasar hukum pemrosesan sesuai Undang-Undang Pelindungan Data Pribadi (UU 27/2022), batasi akses, dan simpan sesingkat yang diperlukan.

Praktikum, rekap, dan tugas

Praktikum: CNN lawan MLP pada citra kecil

Tujuan. Melatih CNN pada MNIST atau CIFAR-10, lalu membandingkannya dengan MLP secara adil.

1. Hitung dengan tangan bentuk *output* dan jumlah parameter setiap *layer* model Anda. Cocokkan dengan `sum(p.numel() ...)` dan satu *tensor* kosong.
2. Latih LeNet dan MLP $784 \to 128 \to 10$ pada MNIST dengan *optimizer*, *learning rate*, *batch size*, dan jumlah *epoch* yang sama. Catat *test accuracy*, jumlah parameter, dan *training time* per *epoch*.
3. Ulangi pada CIFAR-10: ubah kanal *input* menjadi 3 dan ukuran *flatten* sesuai rumus. MLP menerima 3.072 *input*.
4. Gambarkan kurva *loss* latih dan validasi kedua model dalam satu grafik.
5. Bonus: *transfer learning* ResNet-18 pada subset kecil CIFAR-10, lalu tampilkan *filter layer* pertama.

Yang dinilai

- Kebenaran arsitektur: bentuk dan parameter tiap *layer* sesuai hitungan tangan
- Perbandingan yang adil: pengaturan sama, *seed* dicatat
- Tabel hasil: *test accuracy*, parameter, waktu
- Analisis: mengapa selisih CNN dan MLP lebih besar di CIFAR-10 daripada di MNIST

Kumpulkan *notebook* yang dapat dijalankan ulang dari awal sampai akhir, beserta satu halaman analisis.

Rekap pertemuan ini

- CNN memakai *local connectivity* dan *weight sharing*: parameter sedikit dan respons *translation equivariant*.
- Konvolusi menjumlahkan hasil kali *kernel* dengan setiap petak; semua kanal *input* dijumlahkan menjadi satu peta per *filter*.
- Parameter *layer* $= C_{\text{out}}(C_{\text{in}}k^2) + C_{\text{out}}$, tidak bergantung ukuran citra.
- Ukuran *output* $\lfloor (n + 2p - k)/s \rfloor + 1$; *same* untuk k ganjil: $p = (k-1)/2$.
- *Pooling* meringkas tanpa parameter; GAP merata-ratakan seluruh peta.
- *Receptive field* membesar bersama kedalaman: tepi, tekstur, bagian, objek.
- LeNet memberi resep, AlexNet membuktikan skala, VGG memakai tumpukan 3×3, ResNet memungkinkan jaringan sangat dalam.
- *Transfer learning* adalah strategi utama saat data sedikit; visualisasi membantu diagnosis.

Ringkasan

Asumsi bahwa pola citra bersifat lokal dan berulang di banyak posisi membuat LeNet cukup dengan 61.706 parameter (MLP $784 \to 128 \to 10$: 101.770).

Tugas untuk pertemuan berikutnya

1. Baca Prince, *Understanding Deep Learning*, Bab 10, dan Goodfellow dkk., Bab 9.
2. Latihan tangan: konvolusikan $X = \begin{bmatrix} 2 & 0 & 1 & 3 \\ 1 & 4 & 2 & 0 \\ 0 & 1 & 3 & 1 \\ 2 & 2 & 0 & 1 \end{bmatrix}$ dengan $K = \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix}$ ($s = 1$, $p = 0$), lalu terapkan ReLU dan *max pooling* 2×2 ber-*stride* 1.
3. Tunjukkan bahwa tiga *layer* 3×3 ber-*stride* 1 punya *receptive field* 7×7. Berapa untuk lima *layer*?
4. Selesaikan praktikum CNN lawan MLP dan kumpulkan *notebook* beserta analisisnya.
5. Lanjutkan tugas besar UAS: pastikan data sudah terbagi dengan benar dan *baseline* pertama sudah berjalan.

Jawaban "coba sendiri": output $128 \times 28 \times 28$*, karena* $\lfloor (56 + 2 - 3)/2 \rfloor + 1 = 28$*; parameter* $128 \times (64 \cdot 9) + 128 = 73.856$.

Pertemuan depan

Dari citra ke urutan. RNN berbagi *weight* sepanjang waktu, tetapi kesulitan menghubungkan kata yang berjauhan dalam kalimat panjang. Jalan keluarnya adalah *attention*: setiap posisi menengok langsung ke posisi lain. Dari sana kita membangun *transformer*.

Terima kasih

Pertanyaan?