

Fundamental CNN

1. Input Data (Gambar)

Script: Membuat gambar 32×32 , separuh atas hitam, separuh bawah putih.

Konsep CNN: Input adalah matriks piksel (grayscale atau RGB).

Yang perlu dipelajari:

- o Representasi gambar sebagai array (nilai intensitas piksel).
- o Normalisasi data (misalnya skala 0-1).
- o Perbedaan grayscale vs RGB (channel).
- o Bentuk input CNN: (height, width, channels).

2. Kernel / Filter

Script: Kernel acak 3×3 .

Konsep CNN: Filter kecil yang bergerak di atas gambar untuk mengekstrak fitur (tepi, tekstur, pola).

Yang perlu dipelajari:

- o Operasi konvolusi (matriks kernel digeser di atas gambar).
- o Peran kernel dalam mendekripsi pola (horizontal edge, vertical edge, dll).
- o Padding, stride, dan efek ukuran kernel.
- o Hubungan kernel dengan feature map.

3. Convolution Operation

Script: cv2.filter2D(img, -1, kernel)

Konsep CNN: Menghasilkan **feature map** dari input dengan kernel.

Yang perlu dipelajari:

- o Bagaimana konvolusi bekerja secara matematis.
- o Feature map sebagai representasi fitur yang terdeteksi.
- o Perbedaan convolution vs correlation.
- o Multi-channel convolution (untuk RGB).

4. Prediction & Loss Function

Script: predict() → rata-rata feature map sebagai skor, lalu loss_fn() → MSE.

Konsep CNN: CNN menghasilkan prediksi, lalu dibandingkan dengan target menggunakan fungsi loss.

Yang perlu dipelajari:

- o Fungsi loss umum: MSE, cross-entropy.
- o Bagaimana loss mengukur “seberapa jauh” prediksi dari target.
- o Peran loss dalam mengarahkan training.

5. Backpropagation & Gradient

Script: compute_grad() dengan finite difference.

Konsep CNN: Gradien menunjukkan arah perubahan kernel agar loss menurun.

Yang perlu dipelajari:

- o Konsep turunan (derivative) dalam optimisasi.
 - o Backpropagation di CNN (chain rule).
 - o Numerical gradient vs analytical gradient.
 - o Bagaimana gradien memodifikasi kernel.
-

6. Training Loop

Script: 20 iterasi update kernel dengan `kernel -= lr * grad`.

Konsep CNN: Kernel diperbarui sedikit demi sedikit agar lebih baik mendeteksi pola.

Yang perlu dipelajari:

- o Optimizer (SGD, Adam, RMSProp).
 - o Learning rate dan dampaknya.
 - o Epoch, batch size, iterasi.
 - o Overfitting vs generalisasi.
-

7. Visualisasi Kernel & Loss

Script: Plot kernel awal, tengah, akhir + plot loss.

Konsep CNN: Visualisasi membantu memahami bagaimana kernel belajar dan loss menurun.

Yang perlu dipelajari:

- o Interpretasi kernel (misalnya kernel edge detector).
 - o Loss curve: apakah menurun stabil atau tidak.
 - o Debugging training dengan visualisasi.
-

Tahap	Fokus Belajar Mendalam	Contoh Latihan
Input Data	Representasi gambar, normalisasi, channel	Coba load gambar asli dengan <code>cv2.imread</code>
Kernel	Konvolusi manual dengan NumPy	Buat kernel Sobel untuk edge detection
Convolution	Feature map, padding, stride	Bandingkan hasil dengan <code>stride=1</code> vs <code>stride=2</code>
Loss	MSE, cross-entropy	Latih model kecil untuk klasifikasi 2 kelas
Backprop	Chain rule, gradient descent	Hitung gradien analitik vs numerik
Training	Optimizer, learning rate	Bandingkan SGD vs Adam
Visualisasi	Kernel evolution, loss curve	Plot feature map tiap epoch

Gambar sederhana → input

Kernel acak → filter

Konvolusi → feature map

Loss → ukuran kesalahan

Backprop → update kernel

Training loop → proses belajar

Visualisasi → bukti kernel belajar

1. Input Data (Gambar)

Input CNN adalah gambar yang direpresentasikan sebagai array numerik.

Grayscale → hanya 1 channel (nilai intensitas 0-255).

RGB → 3 channel (Red, Green, Blue).

Normalisasi → biasanya piksel diubah ke skala 0-1 agar training lebih stabil.

Bentuk input CNN → (height, width, channels).

1. Membuat gambar 32×32 separuh atas hitam, separuh bawah putih (grayscale)

Soal: Buat gambar sederhana 32×32 , separuh atas hitam, separuh bawah putih.

Script:

```
import numpy as np
import matplotlib.pyplot as plt

# Buat gambar 32x32 grayscale
img = np.zeros((32, 32), dtype=np.float32) # semua hitam
img[16:] = 255 # separuh bawah putih

plt.imshow(img, cmap='gray')
plt.title("Separuh atas hitam, bawah putih")
plt.show()
```

2. Normalisasi gambar ke skala 0-1

Soal: Ubah gambar grayscale 0-255 menjadi skala 0-1.

Script:

```
img_norm = img / 255.0
print("Nilai piksel min:", img_norm.min())
print("Nilai piksel max:", img_norm.max())

plt.imshow(img_norm, cmap='gray')
plt.title("Gambar ternormalisasi (0-1)")
plt.show()
```

3. Membuat gambar RGB dengan pola warna

Soal: Buat gambar 32×32 dengan separuh atas merah, separuh bawah biru (RGB).

Script:

```
img_rgb = np.zeros((32, 32, 3), dtype=np.uint8)

# separuh atas merah
img_rgb[:16, :, 0] = 255 # channel R

# separuh bawah biru
img_rgb[16:, :, 2] = 255 # channel B

plt.imshow(img_rgb)
plt.title("Separuh atas merah, bawah biru")
```

```
plt.show()
```

4. Mengecek bentuk input CNN

Soal: Tampilkan bentuk array untuk grayscale dan RGB.

Script:

```
print("Shape grayscale:", img.shape)      # (32, 32)
print("Shape RGB:", img_rgb.shape)        # (32, 32, 3)
```

5. Membuat batch input (beberapa gambar sekaligus)

Soal: Buat batch berisi 10 gambar grayscale 32×32 , lalu cek dimensinya.

Script:

```
batch = np.zeros((10, 32, 32, 1), dtype=np.float32)    # 10
gambar, grayscale
print("Shape batch:", batch.shape)    # (10, 32, 32, 1)
```

Grayscale → (H, W) atau $(H, W, 1)$

RGB → $(H, W, 3)$

Batch → (N, H, W, C)

Normalisasi → penting untuk stabilitas training.

≡ 珊瑚 Load gambar asli menjadi grayscale 32×32

```
import cv2
import matplotlib.pyplot as plt

# Load gambar asli (misalnya "gambar.jpg") dalam mode
grayscale
img_gray = cv2.imread("gambar.jpg", cv2.IMREAD_GRAYSCALE)

# Resize ke  $32 \times 32$ 
img_gray_resized = cv2.resize(img_gray, (32, 32))

# Normalisasi ke skala 0-1 (opsional)
img_gray_norm = img_gray_resized / 255.0

print("Shape grayscale:", img_gray_resized.shape)    # (32,
32)

# Visualisasi
plt.imshow(img_gray_resized, cmap='gray')
plt.title("Grayscale  $32 \times 32$ ")
plt.show()
```

Catatan belajar:

`cv2.IMREAD_GRAYSCALE` → langsung baca gambar jadi 1 channel.

`cv2.resize(img, (width, height))` → ubah ukuran.

Shape hasil: $(32, 32)$.

Ⅱ 珊瑚 Load gambar asli menjadi RGB 224×224

```
import cv2
import matplotlib.pyplot as plt
```

```

# Load gambar asli dalam mode BGR (default OpenCV)
img_bgr = cv2.imread("gambar.jpg", cv2.IMREAD_COLOR)

# Konversi BGR → RGB (karena OpenCV default BGR)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

# Resize ke 224x224
img_rgb_resized = cv2.resize(img_rgb, (224, 224))

# Normalisasi ke skala 0-1 (opsional)
img_rgb_norm = img_rgb_resized / 255.0

print("Shape RGB:", img_rgb_resized.shape) # (224, 224, 3)

# Visualisasi
plt.imshow(img_rgb_resized)
plt.title("RGB 224x224")
plt.show()

Catatan belajar:
    OpenCV default baca gambar sebagai BGR, jadi perlu
    cv2.cvtColor(..., cv2.COLOR_BGR2RGB).

    Shape hasil: (224, 224, 3) → sesuai format CNN input.
    Normalisasi ke 0-1 biasanya dilakukan sebelum masuk ke model.

```

Grayscale → (H, W) atau (H, W, 1)
RGB → (H, W, 3)
 Resize sesuai kebutuhan model (misalnya 32×32 untuk eksperimen,
 224×224 untuk CNN populer seperti ResNet).
 Normalisasi → penting untuk stabilitas training.

Batch input: load 3 gambar sekaligus dari folder /data (misalnya gambar1.jpg, gambar2.jpg, gambar3.jpg) ke dalam array dengan bentuk (N, H, W, C).

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Daftar file gambar
files = ["data/gambar1.jpg", "data/gambar2.jpg",
        "data/gambar3.jpg"]

# Parameter ukuran target (misalnya 64x64 RGB)
target_size = (64, 64)

batch = []

for f in files:

```

```

# Load gambar dalam mode BGR
img_bgr = cv2.imread(f, cv2.IMREAD_COLOR)

# Konversi ke RGB
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

# Resize ke target size
img_resized = cv2.resize(img_rgb, target_size)

# Normalisasi ke skala 0-1
img_norm = img_resized / 255.0

# Tambahkan ke batch
batch.append(img_norm)

# Ubah list menjadi numpy array
batch_array = np.array(batch, dtype=np.float32)

print("Shape batch:", batch_array.shape) # (3, 64, 64, 3)

# Visualisasi ketiga gambar
fig, axes = plt.subplots(1, 3, figsize=(12,4))
for i, ax in enumerate(axes):
    ax.imshow(batch_array[i])
    ax.set_title(f"Gambar {i+1}")
    ax.axis("off")
plt.show()

```

Looping file gambar → memuat semua gambar dari folder /data.

Resize → seragamkan ukuran (misalnya 64×64).

Konversi BGR → **RGB** → karena OpenCV default BGR.

Normalisasi → skala 0-1 agar siap masuk ke CNN.

Batch array → hasil akhir (N, H, W, C) → di sini (3, 64, 64, 3).

CNN butuh input dengan **dimensi konsisten**.

Batch input → (N, H, W, C) di mana:

- o N = jumlah gambar (batch size),
- o H, W = tinggi & lebar,
- o C = channel (1 grayscale, 3 RGB).

Normalisasi penting agar training stabil.

Batch dataset otomatis: load semua file .jpg dalam sebuah folder tanpa harus menuliskan satu per satu. Kita bisa gunakan modul **glob** atau **os** untuk membaca semua file dengan ekstensi .jpg.

```

import cv2
import numpy as np
import matplotlib.pyplot as plt
import glob

```

```

import os

# Folder tempat gambar disimpan
folder_path = "data/"      # misalnya folder bernama 'data'

# Ambil semua file .jpg dalam folder
files = glob.glob(os.path.join(folder_path, "*.jpg"))

print("Ditemukan file:", files)

# Parameter ukuran target (misalnya 64x64 RGB)
target_size = (64, 64)

batch = []

for f in files:
    # Load gambar dalam mode BGR
    img_bgr = cv2.imread(f, cv2.IMREAD_COLOR)

    # Konversi ke RGB
    img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

    # Resize ke target size
    img_resized = cv2.resize(img_rgb, target_size)

    # Normalisasi ke skala 0-1
    img_norm = img_resized / 255.0

    # Tambahkan ke batch
    batch.append(img_norm)

# Ubah list menjadi numpy array
batch_array = np.array(batch, dtype=np.float32)

print("Shape batch:", batch_array.shape)  # (N, 64, 64, 3)

# Visualisasi beberapa gambar
fig, axes = plt.subplots(1, min(len(batch_array), 5),
figsize=(15, 4))
for i, ax in enumerate(axes):
    ax.imshow(batch_array[i])
    ax.set_title(f"Gambar {i+1}")
    ax.axis("off")
plt.show()

```

`glob.glob("*.jpg")` → otomatis mencari semua file .jpg dalam folder.

`Looping file` → load, konversi ke RGB, resize, normalisasi.

`batch_array.shape` → (N, H, W, C) di mana N jumlah gambar.

`Visualisasi` → menampilkan beberapa gambar dari batch.

Dengan cara ini, dataset bisa otomatis terbentuk dari folder gambar.
CNN biasanya butuh input batch (N, H, W, C) dengan ukuran konsisten.

Normalisasi ke 0-1 penting untuk stabilitas training.

2. Kernel / Filter

Kernel adalah jantung CNN: matriks kecil (misalnya 3×3) yang digeser di atas gambar untuk mengekstrak fitur.

Operasi konvolusi: kernel digeser di atas gambar → menghasilkan feature map.

Peran kernel: mendeteksi pola (tepi horizontal, vertikal, tekstur).

Padding & stride: memengaruhi ukuran feature map.

Hubungan kernel-feature map: kernel = "detektor pola", feature map = "hasil deteksi".

1. Kernel acak 3×3 pada gambar sederhana

Soal: Buat kernel acak 3×3 dan terapkan pada gambar separuh hitam–putih.

Script:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Gambar sederhana
img = np.zeros((32, 32), dtype=np.float32)
img[16:] = 255

# Kernel acak 3x3
np.random.seed(0)
kernel = np.random.randn(3, 3).astype(np.float32)

# Konvolusi
feature_map = cv2.filter2D(img, -1, kernel)

plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title("Input")

plt.subplot(1, 2, 2)
plt.imshow(feature_map, cmap='gray')
plt.title("Feature Map (Kernel Acak)")
plt.show()
```

2. Kernel horizontal edge detector

Soal: Buat kernel untuk mendeteksi tepi horizontal.

Script:

```
# Kernel horizontal edge (Sobel-like)
kernel_h = np.array([[-1, -2, -1],
                     [ 0,  0,  0],
```

```

[ 1,  2,  1]], dtype=np.float32)

feature_map_h = cv2.filter2D(img, -1, kernel_h)

plt.imshow(feature_map_h, cmap='gray')
plt.title("Deteksi Tepi Horizontal")
plt.show()

```

3. Kernel vertical edge detector

Soal: Buat kernel untuk mendeteksi tepi vertikal.

Script:

```

# Kernel vertical edge (Sobel-like)
kernel_v = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]], dtype=np.float32)

feature_map_v = cv2.filter2D(img, -1, kernel_v)

plt.imshow(feature_map_v, cmap='gray')
plt.title("Deteksi Tepi Vertikal")
plt.show()

```

4. Efek padding pada konvolusi

Soal: Bandingkan hasil konvolusi dengan dan tanpa padding.

Script:

```

# Kernel blur sederhana
kernel_blur = np.ones((3,3), dtype=np.float32) / 9

# Tanpa padding (default)
fmap_no_pad = cv2.filter2D(img, -1, kernel_blur)

# Dengan padding (gunakan border replicate)
fmap_pad = cv2.filter2D(img, -1, kernel_blur,
borderType=cv2.BORDER_REPLICATE)

plt.subplot(1,2,1)
plt.imshow(fmap_no_pad, cmap='gray')
plt.title("Tanpa Padding")

plt.subplot(1,2,2)
plt.imshow(fmap_pad, cmap='gray')
plt.title("Dengan Padding")
plt.show()

```

5. Efek stride pada konvolusi (manual)

Soal: Terapkan kernel dengan stride=2 (manual implementasi).

Script:

```

def conv_stride(img, kernel, stride=2):
    h, w = img.shape
    kh, kw = kernel.shape
    out_h = (h - kh) // stride + 1

```

```

        out_w = (w - kw) // stride + 1
        out = np.zeros((out_h, out_w))
        for i in range(0, h-kh+1, stride):
            for j in range(0, w-kw+1, stride):
                region = img[i:i+kh, j:j+kw]
                out[i//stride, j//stride] = np.sum(region * kernel)
        return out

kernel_simple = np.array([[1,0,-1],
                         [1,0,-1],
                         [1,0,-1]], dtype=np.float32)

fmap_stride2 = conv_stride(img, kernel_simple, stride=2)

plt.imshow(fmap_stride2, cmap='gray')
plt.title("Feature Map dengan Stride=2")
plt.show()

```

Kernel = detektor pola.

Feature map = hasil deteksi.

Horizontal/vertical edge → contoh klasik.

Padding → menjaga ukuran output.

Stride → mengurangi resolusi feature map.

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

def show_result(img, kernel, title):
    fmap = cv2.filter2D(img, -1, kernel)
    fig, axes = plt.subplots(1,2, figsize=(6,3))
    axes[0].imshow(img, cmap='gray')
    axes[0].set_title("Input")
    axes[0].axis("off")
    axes[1].imshow(fmap, cmap='gray')
    axes[1].set_title(title)
    axes[1].axis("off")
    plt.show()

# 1. Separuh atas hitam, bawah putih + horizontal edge
img1 = np.zeros((32,32), dtype=np.float32); img1[16:] = 255
kernel_h = np.array([[-1,-2,-1],[0,0,0],[1,2,1]], dtype=np.float32)
show_result(img1, kernel_h, "Horizontal Edge")

# 2. Separuh kiri hitam, kanan putih + vertical edge
img2 = np.zeros((32,32), dtype=np.float32); img2[:,16:] = 255

```

```

kernel_v = np.array([[-1,0,1],[-2,0,2],[-1,0,1]], dtype=np.float32)
show_result(img2, kernel_v, "Vertical Edge")

# 3. Kotak putih di tengah + blur
img3 = np.zeros((32,32), dtype=np.float32);
img3[10:22,10:22] = 255
kernel_blur = np.ones((3,3), dtype=np.float32)/9
show_result(img3, kernel_blur, "Blur")

# 4. Garis diagonal putih + sharpen
img4 = np.zeros((32,32), dtype=np.float32)
np.fill_diagonal(img4, 255)
kernel_sharp = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]], dtype=np.float32)
show_result(img4, kernel_sharp, "Sharpen")

# 5. Lingkaran putih di tengah + edge detection
img5 = np.zeros((32,32), dtype=np.uint8)
cv2.circle(img5, (16,16), 8, 255, -1)
kernel_edge = np.array([[-1,-1,-1],[-1,8,-1],[-1,-1,-1]], dtype=np.float32)
show_result(img5, kernel_edge, "Edge Detection")

# 6. Garis horizontal putih + vertical edge
img6 = np.zeros((32,32), dtype=np.float32); img6[15:17,:]
= 255
show_result(img6, kernel_v, "Vertical Edge on Horizontal Line")

# 7. Garis vertikal putih + horizontal edge
img7 = np.zeros((32,32), dtype=np.float32); img7[:,15:17]
= 255
show_result(img7, kernel_h, "Horizontal Edge on Vertical Line")

# 8. Checkerboard hitam-putih + blur
img8 = np.indices((32,32)).sum(axis=0) % 2 * 255
show_result(img8.astype(np.float32), kernel_blur, "Blur Checkerboard")

# 9. Gradien kiri ke kanan + sharpen
img9 = np.tile(np.linspace(0,255,32),
(32,1)).astype(np.float32)
show_result(img9, kernel_sharp, "Sharpen Gradient")

# 10. Titik putih di tengah + Laplacian
img10 = np.zeros((32,32), dtype=np.float32); img10[16,16]
= 255
kernel_lap = np.array([[0,1,0],[1,-4,1],[0,1,0]], dtype=np.float32)

```

```
show_result(img10, kernel_lap, "Laplacian Point")
```

Edge kernels → mendeteksi batas horizontal/vertikal.

Blur kernel → menghaluskan detail.

Sharpen kernel → menonjolkan detail.

Laplacian kernel → mendeteksi tepi/titik.

Feature map = hasil konvolusi, menunjukkan pola yang kernel “lihat” dari gambar.

eksperimen padding & stride.

Tujuannya: melihat bagaimana ukuran **feature map** berubah ketika kita mengubah **padding** dan **stride** pada operasi konvolusi.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def conv_stride(img, kernel, stride=1, padding=0):
    # Tambahkan padding
    if padding > 0:
        img = np.pad(img,
((padding,padding), (padding,padding)), mode='constant')
    h, w = img.shape
    kh, kw = kernel.shape
    out_h = (h - kh)//stride + 1
    out_w = (w - kw)//stride + 1
    out = np.zeros((out_h, out_w))
    for i in range(0, h-kh+1, stride):
        for j in range(0, w-kw+1, stride):
            region = img[i:i+kh, j:j+kw]
            out[i//stride, j//stride] = np.sum(region * kernel)
    return out

# Gambar sederhana: kotak putih di tengah
img = np.zeros((32,32), dtype=np.float32)
img[10:22,10:22] = 255

# Kernel edge sederhana
kernel = np.array([[[-1,-1,-1],
                   [-1, 8,-1],
                   [-1,-1,-1]]], dtype=np.float32)

experiments = [
    ("Padding=0, Stride=1", 0, 1),
    ("Padding=1, Stride=1", 1, 1),
    ("Padding=2, Stride=1", 2, 1),
    ("Padding=0, Stride=2", 0, 2),
    ("Padding=1, Stride=2", 1, 2),
    ("Padding=2, Stride=2", 2, 2),
```

```

        ("Padding=0, Stride=3", 0, 3),
        ("Padding=1, Stride=3", 1, 3),
        ("Padding=2, Stride=3", 2, 3),
        ("Padding=3, Stride=3", 3, 3),
    ]

```

```

# Plot hasil
fig, axes = plt.subplots(5,2, figsize=(10,20))
for ax, (title, pad, stride) in zip(axes.ravel(), experiments):
    fmap = conv_stride(img, kernel, stride=stride,
    padding=pad)
    ax.imshow(fmap, cmap='gray')
    ax.set_title(f"{title}\nOutput shape: {fmap.shape}")
    ax.axis("off")
plt.tight_layout()
plt.show()

```

Padding → menambahkan border di sekitar gambar.

- o **Padding=0** → output lebih kecil.
 - o **Padding besar** → output lebih besar, menjaga tepi.
- Stride** → langkah geser kernel.
- o **Stride=1** → kernel geser satu piksel → output besar.
 - o **Stride=2/3** → kernel geser lebih jauh → output mengecil.
- Kombinasi padding & stride → memengaruhi **ukuran feature map**.

Padding menjaga informasi di tepi gambar.

Stride mengontrol resolusi feature map (semakin besar stride → semakin kecil output).

CNN modern sering pakai **padding="same"** agar ukuran output sama dengan input.

Eksperimen ini menunjukkan secara visual bagaimana kernel “melihat” gambar dengan konfigurasi berbeda.

multi-kernel: satu gambar sederhana dilewatkan oleh **3 kernel berbeda** sehingga menghasilkan **3 feature map sekaligus**. Ini meniru cara kerja CNN layer sebenarnya, di mana banyak filter bekerja paralel pada input.

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

# 1. Buat gambar sederhana: kotak putih di tengah
img = np.zeros((32,32), dtype=np.float32)
img[10:22, 10:22] = 255

# 2. Definisikan 3 kernel berbeda
kernel_h = np.array([[-1,-2,-1],
                     [ 0, 0, 0],

```

```

[ 1, 2, 1]], dtype=np.float32)    #
horizontal edge

kernel_v = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]], dtype=np.float32)    #
vertical edge

kernel_sharp = np.array([[0,-1,0],
                        [-1,5,-1],
                        [0,-1,0]], dtype=np.float32)    #
sharpen

kernels = [kernel_h, kernel_v, kernel_sharp]
titles = ["Horizontal Edge", "Vertical Edge", "Sharpen"]

# 3. Terapkan semua kernel ke gambar
feature_maps = [cv2.filter2D(img, -1, k) for k in kernels]

# 4. Plot input + 3 feature map
fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].imshow(img, cmap='gray')
axes[0].set_title("Input")
axes[0].axis("off")

for i, fmap in enumerate(feature_maps):
    axes[i+1].imshow(fmap, cmap='gray')
    axes[i+1].set_title(titles[i])
    axes[i+1].axis("off")

plt.tight_layout()
plt.show()

```

Input: kotak putih sederhana.

Kernel 1 (Horizontal Edge) → menyorot tepi horizontal kotak.

Kernel 2 (Vertical Edge) → menyorot tepi vertikal kotak.

Kernel 3 (Sharpen) → menajamkan kotak, membuat tepi lebih kontras.

Feature maps: hasil konvolusi dari masing-masing kernel.

CNN layer biasanya punya **puluhan hingga ratusan kernel**.

Setiap kernel belajar mendeteksi pola berbeda (edge, tekstur, bentuk).

Output layer CNN = **kumpulan feature maps** dari semua kernel.

Feature maps ini kemudian diproses lebih lanjut (pooling, non-linear activation, dll).

multi-kernel + stacking

Tujuannya: menunjukkan bagaimana CNN menyusun hasil dari banyak filter paralel menjadi sebuah tensor dengan bentuk (H, W, N_filters).

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 1. Buat gambar sederhana: kotak putih di tengah
img = np.zeros((32,32), dtype=np.float32)
img[10:22, 10:22] = 255

# 2. Definisikan beberapa kernel berbeda
kernel_h = np.array([[-1,-2,-1],
                     [ 0, 0, 0],
                     [ 1, 2, 1]], dtype=np.float32)    #
horizontal edge

kernel_v = np.array([[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]], dtype=np.float32)    #
vertical edge

kernel_sharp = np.array([[0,-1,0],
                        [-1,5,-1],
                        [0,-1,0]], dtype=np.float32)    #
sharpen

kernels = [kernel_h, kernel_v, kernel_sharp]
titles = ["Horizontal Edge", "Vertical Edge", "Sharpen"]

# 3. Terapkan semua kernel ke gambar
feature_maps = [cv2.filter2D(img, -1, k) for k in kernels]

# 4. Gabungkan semua feature map menjadi satu tensor (H,
W, N_filters)
feature_tensor = np.stack(feature_maps, axis=-1)

print("Shape feature tensor:", feature_tensor.shape)    #
(32, 32, 3)

# 5. Visualisasi input + semua feature map
fig, axes = plt.subplots(1, 4, figsize=(12,3))

axes[0].imshow(img, cmap='gray')
axes[0].set_title("Input")
axes[0].axis("off")

for i, fmap in enumerate(feature_maps):
    axes[i+1].imshow(fmap, cmap='gray')
    axes[i+1].set_title(titles[i])
```

```
axes[i+1].axis("off")
plt.tight_layout()
plt.show()
```

Input: gambar kotak putih sederhana.

Kernel 1 (Horizontal Edge) → mendeteksi tepi horizontal.

Kernel 2 (Vertical Edge) → mendeteksi tepi vertikal.

Kernel 3 (Sharpen) → menajamkan kotak.

Feature maps: hasil konvolusi masing-masing kernel.

Feature tensor: hasil stacking → $(H, W, N_{filters}) = (32, 32, 3)$.

CNN layer bekerja dengan **banyak kernel paralel**.

Output layer = **tensor feature maps**.

Dimensi $(H, W, N_{filters})$ menunjukkan:

- o H, W = ukuran spasial (tinggi, lebar),
- o $N_{filters}$ = jumlah kernel/filter.

Tensor ini kemudian diproses oleh **activation (ReLU)**, **pooling**, dan layer berikutnya.

hubungan kernel dengan feature map.

Kernel (filter) adalah matriks kecil (misalnya 3×3) yang digeser di atas gambar.

Feature map adalah hasil konvolusi: representasi baru yang menyorot pola tertentu dari gambar.

Hubungan keduanya:

- o Kernel = “detektor pola” (misalnya tepi horizontal, vertikal, blur, sharpen).
- o Feature map = “hasil deteksi” → menunjukkan bagian gambar yang cocok dengan pola kernel.

CNN layer biasanya memiliki banyak kernel → menghasilkan banyak feature map paralel.

1. Kernel Horizontal Edge

```
import cv2, numpy as np, matplotlib.pyplot as plt
```

```
# Gambar sederhana: separuh atas hitam, bawah putih
img = np.zeros((32,32), dtype=np.float32); img[16:] = 255
```

```
# Kernel horizontal edge
kernel_h = np.array([[-1,-2,-1],[0,0,0],[1,2,1]], dtype=np.float32)
```

```
fmap_h = cv2.filter2D(img, -1, kernel_h)
```

```
plt.subplot(1,2,1); plt.imshow(img, cmap='gray');
plt.title("Input")
```

```
plt.subplot(1,2,2); plt.imshow(fmap_h, cmap='gray');
plt.title("Feature Map Horizontal Edge")
plt.show()
```

2. Kernel Vertical Edge

```
img2 = np.zeros((32,32), dtype=np.float32); img2[:,16:] =
255

kernel_v = np.array([[-1,0,1], [-2,0,2], [-1,0,1]], dtype=np.float32)

fmap_v = cv2.filter2D(img2, -1, kernel_v)

plt.subplot(1,2,1); plt.imshow(img2, cmap='gray');
plt.title("Input")
plt.subplot(1,2,2); plt.imshow(fmap_v, cmap='gray');
plt.title("Feature Map Vertical Edge")
plt.show()
```

3. Kernel Blur

```
img3 = np.zeros((32,32), dtype=np.float32);
img3[10:22,10:22] = 255

kernel_blur = np.ones((3,3), dtype=np.float32)/9

fmap_blur = cv2.filter2D(img3, -1, kernel_blur)

plt.subplot(1,2,1); plt.imshow(img3, cmap='gray');
plt.title("Input")
plt.subplot(1,2,2); plt.imshow(fmap_blur, cmap='gray');
plt.title("Feature Map Blur")
plt.show()
```

4. Kernel Sharpen

```
img4 = np.zeros((32,32), dtype=np.float32);
img4[10:22,10:22] = 255

kernel_sharp = np.array([[0,-1,0], [-1,5,-1], [0,-1,0]], dtype=np.float32)

fmap_sharp = cv2.filter2D(img4, -1, kernel_sharp)

plt.subplot(1,2,1); plt.imshow(img4, cmap='gray');
plt.title("Input")
plt.subplot(1,2,2); plt.imshow(fmap_sharp, cmap='gray');
plt.title("Feature Map Sharpen")
plt.show()
```

5. Kernel Laplacian (Edge Detection)

```
img5 = np.zeros((32,32), dtype=np.uint8)
cv2.circle(img5, (16,16), 8, 255, -1)
```

```

kernel_lap = np.array([[0,1,0],[1,-4,1],[0,1,0]],  

dtype=np.float32)

fmap_lap = cv2.filter2D(img5, -1, kernel_lap)

plt.subplot(1,2,1); plt.imshow(img5, cmap='gray');  

plt.title("Input")  

plt.subplot(1,2,2); plt.imshow(fmap_lap, cmap='gray');  

plt.title("Feature Map Laplacian")  

plt.show()

```

Kernel = pola yang ingin dideteksi.

Feature map = hasil konvolusi, menyorot bagian gambar sesuai pola kernel.

CNN layer = kumpulan kernel → kumpulan feature map → representasi fitur yang lebih kaya.

3. Convolution Operation

Convolution operation in CNN with OpenCV and NumPy

You're building intuition the right way: see the math, then feel it with small, visual scripts. Below are concise, reproducible pieces you can drop into your worksheet and extend.

Convolution basics with cv2.filter2D

```

import cv2  

import numpy as np

# Load a simple grayscale image
img = cv2.imread('simple.png', cv2.IMREAD_GRAYSCALE)

# Example kernels
blur_kernel = np.ones((3, 3), dtype=np.float32) / 9.0
sharpen_kernel = np.array([[0, -1, 0],  

                           [-1, 5, -1],  

                           [0, -1, 0]], dtype=np.float32)
edge_kernel = np.array([[-1, -1, -1],  

                       [-1, 8, -1],  

                       [-1, -1, -1]], dtype=np.float32)

# Apply filter2D (correlation by default in OpenCV; see  

# section 3 for convolution vs correlation)
blurred = cv2.filter2D(img, ddepth=-1, kernel=blur_kernel)
sharpened = cv2.filter2D(img, ddepth=-1,  

                        kernel=sharpen_kernel)
edges = cv2.filter2D(img, ddepth=-1, kernel=edge_kernel)

cv2.imwrite('out_blur.png', blurred)

```

```

cv2.imwrite('out_sharpen.png', sharpened)
cv2.imwrite('out_edges.png', edges)

```

How convolution works mathematically

Convolution combines local neighborhoods with learned weights to produce a feature map. For a grayscale image (I) and a kernel (K) of size ($m \times n$), the 2D discrete convolution at position $((i, j))$ is:

$$[(I * K)[i, j] := \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I[i - u, j - v] \cdot K[u, v]]$$

Kernel flip: True convolution flips the kernel horizontally and vertically before the weighted sum.

$$[K^{\text{flip}}[u, v] = K[m-1-u, n-1-v]]$$

Then:

$$[(I * K)[i, j] := \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I[i + u - (m-1), j + v - (n-1)] \cdot K^{\text{flip}}[u, v]]$$

Padding and stride:

- o With zero-padding (p) and stride (s), output size for input ($H \times W$) and kernel ($k_h \times k_w$) is: $[H_{\text{out}} = \left\lfloor \frac{H + 2p - k_h}{s} \right\rfloor + 1, W_{\text{out}} = \left\lfloor \frac{W + 2p - k_w}{s} \right\rfloor + 1]$
- o Padding controls border information retention; stride controls downsampling.

Minimal NumPy implementation (with kernel flip)

```
import numpy as np
```

```

def conv2d_valid(img, kernel):
    # Flip kernel for true convolution
    k = np.flipud(np.fliplr(kernel))
    H, W = img.shape
    kh, kw = k.shape
    out_h, out_w = H - kh + 1, W - kw + 1
    out = np.zeros((out_h, out_w), dtype=np.float32)
    for i in range(out_h):
        for j in range(out_w):
            patch = img[i:i+kh, j:j+kw]
            out[i, j] = np.sum(patch * k)
    return out

# Example
img = np.array([[0,0,0,0,0],
                [0,50,50,50,0],
                [0,50,100,50,0],
                [0,50,50,50,0],
                [0,0,0,0,0]], dtype=np.float32)

sobel_x = np.array([[[-1, 0, 1],
                     [-2, 0, 2],
                     [-1, 0, 1]]], dtype=np.float32)

out = conv2d_valid(img, sobel_x)
print(out)

```

Feature map as detected feature representation

In CNNs, each kernel acts like a detector for a specific pattern (e.g., horizontal edges, corners, texture). The output of applying a kernel over the spatial dimensions is a feature map:

Activation meaning: Large positive values indicate strong presence of the kernel's pattern; negative values indicate opposite patterns; near-zero means weak presence.

Stacking features: Multiple kernels applied to the same input produce multiple feature maps, forming a depth dimension for the next layer.

Nonlinearity and normalization: After convolution, CNNs usually apply a bias, a nonlinearity (e.g., ReLU), and sometimes normalization, which sparsify and stabilize feature maps.

Visualizing feature maps quickly

```
import cv2
import numpy as np

img = cv2.imread('simple.png', cv2.IMREAD_GRAYSCALE)

kernels = {
    'blur3': np.ones((3,3), np.float32)/9.0,
    'sharpen': np.array([[0,-1,0],[-1,5,-1],[0,-1,0]], np.float32),
    'sobel_x': np.array([[-1,0,1],[-2,0,2],[-1,0,1]], np.float32),
    'sobel_y': np.array([[-1,-2,-1],[0,0,0],[1,2,1]], np.float32)
}

feature_maps = {}
for name, k in kernels.items():
    fmap = cv2.filter2D(img, -1, k)
    feature_maps[name] = fmap
    cv2.imwrite(f'feature_{name}.png', fmap)

# Optional: ReLU to emphasize positive detections
for name, fmap in feature_maps.items():
    relu = np.maximum(fmap, 0)
    cv2.imwrite(f'feature_{name}_relu.png', relu)
```

Convolution vs correlation

Definition difference:

- o **Correlation** does not flip the kernel: $[(I \star K)[i, j] := \sum_{u=0}^{m-1} \sum_{v=0}^{n-1} I[i+u, j+v] \cdot K[u, v]]$
- o **Convolution** flips the kernel before summation (see earlier equations).

Practical impact:

- o If a kernel is symmetric (e.g., Gaussian blur), convolution and correlation yield the same result.
- o For directional kernels (e.g., Sobel), correlation and convolution can differ: one responds to left-to-right edges, the other to right-to-left edges.

OpenCV note:

- o cv2.filter2D performs correlation. To emulate true convolution, flip the kernel first: `kernel_flipped = cv2.flip(kernel, -1)` then pass `kernel_flipped` to filter2D.

Side-by-side comparison

```
import cv2
import numpy as np

img = cv2.imread('simple.png', cv2.IMREAD_GRAYSCALE)
sobel_x = np.array([[-1,0,1],
                   [-2,0,2],
                   [-1,0,1]], dtype=np.float32)

# Correlation (default)
corr = cv2.filter2D(img, -1, sobel_x)

# Convolution (flip kernel)
sobel_x_flipped = cv2.flip(sobel_x, -1)
conv = cv2.filter2D(img, -1, sobel_x_flipped)

cv2.imwrite('corr_sobel_x.png', corr)
cv2.imwrite('conv_sobel_x.png', conv)
```

Multi-channel convolution for RGB

In CNNs, a single filter spans all input channels. For an RGB image ($I \in \mathbb{R}^{H \times W \times 3}$) and a kernel ($K \in \mathbb{R}^{k_h \times k_w \times 3}$), the output at $((i, j))$ is:

$$[O[i, j] := \sum_{c=0}^2 \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} I[i+u, j+v, c] \cdot K^{\text{flip}}[u, v, c]] + b]$$

Channel coupling: The kernel learns how to combine color channels; edges may pop more in one channel than another.

Multiple filters: If you have (F) filters, you get (F) output feature maps, one per filter.

Implementing multi-channel convolution in NumPy

```
import numpy as np

def conv2d_rgb_valid(img_rgb, kernel_rgb, bias=0.0):
    # img_rgb: H x W x 3
    # kernel_rgb: kh x kw x 3 (one filter spanning all
    # channels)
    kh, kw, C = kernel_rgb.shape
```

```

H, W, _ = img_rgb.shape
out_h, out_w = H - kh + 1, W - kw + 1
out = np.zeros((out_h, out_w), dtype=np.float32)

# Flip spatial dims for true convolution
k_flip = np.flipud(np.fliplr(kernel_rgb))

for i in range(out_h):
    for j in range(out_w):
        patch = img_rgb[i:i+kh, j:j+kw, :]
        # kh x kw x 3
        out[i, j] = np.sum(patch * k_flip) + bias
return out

# Example: RGB edge-like filter biased toward green
channel
kernel_rgb = np.stack([
    np.array([[-1, 0, 1],
              [-2, 0, 2],
              [-1, 0, 1]], dtype=np.float32), # R
    np.array([[-1, 0, 1],
              [-2, 0, 2],
              [-1, 0, 1]], dtype=np.float32) * 1.5, # G
    weighted
    np.array([[-1, 0, 1],
              [-2, 0, 2],
              [-1, 0, 1]], dtype=np.float32) # B
], axis=-1) # kh x kw x 3

img_rgb = cv2.imread('color.png', cv2.IMREAD_COLOR) # H
x W x 3 (BGR in OpenCV)
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB)

feature = conv2d_rgb_valid(img_rgb.astype(np.float32),
                           kernel_rgb)
feature_relu = np.maximum(feature, 0)

# Normalize for visualization
vis = (feature - feature.min()) / (feature.max() -
feature.min() + 1e-6)
vis = (vis * 255).astype(np.uint8)
cv2.imwrite('feature_rgb.png', vis)

Multiple filters producing a stack of feature maps
def conv2d_rgb_bank_valid(img_rgb, kernels_rgb,
                          biases=None):
    # kernels_rgb: list of kh x kw x 3 filters
    F = len(kernels_rgb)
    feats = []
    for idx, K in enumerate(kernels_rgb):
        b = 0.0 if biases is None else biases[idx]
        feats.append(conv2d_rgb_valid(img_rgb, K, b))

```

```

        return np.stack(feats, axis=-1) # H_out x W_out x F

# Create a small bank
kernels = [kernel_rgb,
            kernel_rgb * np.array([[1.0, 0.5, 1.0]]), #
variant
            np.flipud(kernel_rgb)] # different
orientation

feature_stack =
conv2d_rgb_bank_valid(img_rgb.astype(np.float32), kernels)
print('Feature stack shape:', feature_stack.shape) #
(H_out, W_out, F)

# Save each map
for f in range(feature_stack.shape[-1]):
    fmap = feature_stack[..., f]
    vis = (fmap - fmap.min()) / (fmap.max() - fmap.min()
+ 1e-6)
    cv2.imwrite(f'feature_rgb_{f}.png', (vis *
255).astype(np.uint8))

```

Quick experiment ideas you can add to your worksheet

Kernel symmetry check:

- o **Goal:** Show when convolution == correlation.
- o **Action:** Compare outputs for Gaussian kernel vs Sobel kernel using flipped vs non-flipped.

Padding and stride sweep:

- o **Goal:** Visualize how borders and downsampling affect feature maps.
- o **Action:** Implement zero/reflect padding and stride ($s \in \{1, 2\}$) in your NumPy conv. Plot output sizes and maps.

Channel importance maps:

- o **Goal:** Show how different color channels contribute.
- o **Action:** Zero out R/G/B channels and re-run multi-channel convolution; compare feature intensity.

If you want, I can help you wrap these into a single, clean notebook with toggles for kernels, padding, stride, and per-channel visualization.

contoh script sederhana untuk memahami *Convolution Operation* dengan Python + OpenCV/NumPy. Setiap contoh akan menyoroti aspek berbeda dari konvolusi agar lebih intuitif.

1. Blur (Averaging Filter)

```

import cv2
import numpy as np

```

```
img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

# Kernel rata-rata 3x3
kernel = np.ones((3,3), np.float32) / 9.0

blurred = cv2.filter2D(img, -1, kernel)
cv2.imwrite('blurred.png', blurred)
```

2. Sharpen (Menajamkan)

```
import cv2
import numpy as np

img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

# Kernel sharpen
kernel = np.array([[0,-1,0],
                   [-1,5,-1],
                   [0,-1,0]], np.float32)

sharpened = cv2.filter2D(img, -1, kernel)
cv2.imwrite('sharpened.png', sharpened)
```

3. Edge Detection (Sobel X)

```
import cv2
import numpy as np

img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

# Kernel Sobel X
kernel = np.array([[-1,0,1],
                   [-2,0,2],
                   [-1,0,1]], np.float32)

edges_x = cv2.filter2D(img, -1, kernel)
cv2.imwrite('edges_x.png', edges_x)
```

4. Laplacian (Deteksi Tepi Semua Arah)

```
import cv2
import numpy as np

img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

# Kernel Laplacian
kernel = np.array([[0,1,0],
                   [1,-4,1],
                   [0,1,0]], np.float32)

laplacian = cv2.filter2D(img, -1, kernel)
cv2.imwrite('laplacian.png', laplacian)
```

5. Multi-channel Convolution (RGB)

```

import cv2
import numpy as np

img = cv2.imread('color.png') # BGR

# Kernel edge detection sederhana
kernel = np.array([[-1,-1,-1],
                   [-1, 8,-1],
                   [-1,-1,-1]], np.float32)

# Terapkan ke tiap channel
channels = cv2.split(img)
filtered_channels = [cv2.filter2D(ch, -1, kernel) for ch
in channels]

# Gabungkan kembali
filtered_rgb = cv2.merge(filtered_channels)
cv2.imwrite('filtered_rgb.png', filtered_rgb)

```

Ringkasan

Script 1: Blur → rata-rata tetangga.

Script 2: Sharpen → menonjolkan detail.

Script 3: Sobel X → deteksi tepi vertikal.

Script 4: Laplacian → deteksi tepi semua arah.

Script 5: Multi-channel → konvolusi pada gambar berwarna (RGB).

Visualisasi sebelum/ sesudah

Script Visualisasi Sebelum/Sesudah Konvolusi

```

import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load gambar grayscale
img = cv2.imread('image.png', cv2.IMREAD_GRAYSCALE)

# Definisi beberapa kernel
kernels = {
    "Blur (3x3)": np.ones((3,3), np.float32) / 9.0,
    "Sharpen": np.array([[0,-1,0],
                        [-1,5,-1],
                        [0,-1,0]], np.float32),
    "Sobel X": np.array([[ -1,0,1],
                         [ -2,0,2],
                         [ -1,0,1]], np.float32),
    "Laplacian": np.array([[ 0,1,0],
                          [ 1,-4,1],
                          [ 0,1,0]], np.float32)
}

```

```

# Plot hasil
fig, axes = plt.subplots(1, len(kernels)+1,
figsize=(15,5))

# Tampilkan gambar asli
axes[0].imshow(img, cmap='gray')
axes[0].set_title("Original")
axes[0].axis("off")

# Terapkan tiap kernel dan tampilkan hasil
for i, (name, kernel) in enumerate(kernels.items(),
start=1):
    filtered = cv2.filter2D(img, -1, kernel)
    axes[i].imshow(filtered, cmap='gray')
    axes[i].set_title(name)
    axes[i].axis("off")

plt.tight_layout()
plt.show()

```

Penjelasan

- Original** → gambar asli.
 - Blur** → menghaluskan citra.
 - Sharpen** → menajamkan detail.
 - Sobel X** → mendeteksi tepi vertikal.
 - Laplacian** → mendeteksi tepi dari semua arah.
-

Bonus: Visualisasi Multi-channel (RGB)

```

# Load gambar berwarna
img_rgb = cv2.imread('color.png')
img_rgb = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB)

# Kernel edge detection
kernel = np.array([[-1,-1,-1],
                  [-1, 8,-1],
                  [-1,-1,-1]], np.float32)

# Terapkan ke tiap channel
channels = cv2.split(img_rgb)
filtered_channels = [cv2.filter2D(ch, -1, kernel) for ch
in channels]
filtered_rgb = cv2.merge(filtered_channels)

# Plot sebelum/sesudah
fig, axes = plt.subplots(1,2, figsize=(10,5))
axes[0].imshow(img_rgb)
axes[0].set_title("Original RGB")
axes[0].axis("off")

axes[1].imshow(filtered_rgb)
axes[1].set_title("Filtered RGB")

```

```
axes[1].axis("off")
plt.show()
```

Prediction & Loss Function dalam CNN.

Prediction (Prediksi): CNN mengambil input (misalnya gambar), lalu menghasilkan output berupa skor atau probabilitas.

Loss Function (Fungsi Loss): Digunakan untuk mengukur seberapa jauh hasil prediksi dari target sebenarnya.

Training: CNN belajar dengan cara meminimalkan nilai loss. Semakin kecil loss, semakin baik prediksi mendekati target.

```
import numpy as np

# Misalnya feature map hasil CNN
feature_map = np.array([0.2, 0.4, 0.6, 0.8])

# Prediction: rata-rata feature map sebagai skor
def predict(feature_map):
    return np.mean(feature_map)

# Loss Function: Mean Squared Error (MSE)
def loss_fn(pred, target):
    return np.mean((pred - target)**2)

# Contoh penggunaan
pred = predict(feature_map)      # skor prediksi
target = 0.5                      # target sebenarnya
loss = loss_fn(pred, target)

print("Prediksi:", pred)
print("Target:", target)
print("Loss:", loss)
Prediksi = 0.5
Target = 0.5
Loss = 0.0 (artinya prediksi tepat sekali)
```

MSE (Mean Squared Error):

- o Cocok untuk regresi (prediksi angka).
- o Rumus:
[$\text{MSE} = \frac{1}{n} \sum (y_{\text{pred}} - y_{\text{true}})^2$]
- o Contoh: Prediksi tinggi badan 170 cm, target 172 cm → error = $(170-172)^2 = 4$.

Cross-Entropy Loss:

- o Cocok untuk klasifikasi (misalnya mengenali gambar kucing vs anjing).
- o Mengukur seberapa jauh distribusi probabilitas prediksi dari distribusi target.

- o Contoh:
 - Prediksi: [0.9 kucing, 0.1 anjing]
 - Target: [1.0 kucing, 0.0 anjing]
 - Loss kecil → prediksi bagus.
 - Prediksi: [0.4 kucing, 0.6 anjing] → Loss besar.

Loss berfungsi seperti **kompas**: menunjukkan arah perbaikan.
CNN menggunakan algoritma **backpropagation + gradient descent** untuk menyesuaikan bobot.
Jika loss besar → bobot diperbaiki agar prediksi lebih mendekati target.
Jika loss kecil → berarti CNN sudah belajar dengan baik.

Bayangkan kamu menebak nilai ujian temanmu:

Prediksi: kamu bilang nilainya 80.

Target: ternyata nilainya 85.

Loss: selisih 5 → CNN tahu tebakan masih kurang tepat.

CNN akan belajar supaya tebakan berikutnya lebih dekat ke 85.

Prediction adalah hasil tebakan CNN, sedangkan **Loss Function** adalah penggaris yang mengukur seberapa jauh tebakan dari kenyataan. Tanpa loss, CNN tidak tahu apakah ia sedang belajar dengan benar.

loss curve dan visualisasi prediksi sebelum dan sesudah training

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# 1. Dataset sintetis (dua kelas bulan sabit)
X, y = make_moons(n_samples=500, noise=0.2,
random_state=42)
X = StandardScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# Konversi ke tensor (reshape agar cocok dengan Conv2D)
X_train = X_train.reshape(-1, 2, 1, 1)    # [batch, height,
width, channel]
X_test = X_test.reshape(-1, 2, 1, 1)

# 2. Definisi CNN sederhana
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(4, kernel_size=(2,1),
activation='relu', input_shape=(2,1,1)),
    tf.keras.layers.Flatten(),
```

```

        tf.keras.layers.Dense(1, activation='sigmoid') #  

    output probabilitas  

])  
  

# 3. Compile dengan loss MSE  

model.compile(optimizer='adam', loss='mse',  

metrics=['accuracy'])  
  

# 4. Training  

history = model.fit(X_train, y_train, epochs=50,  

batch_size=32, validation_data=(X_test, y_test),  

verbose=0)  
  

# 5. Plot Loss Curve  

plt.plot(history.history['loss'], label='Train Loss')  

plt.plot(history.history['val_loss'], label='Val Loss')  

plt.title("Loss Curve selama Training")  

plt.xlabel("Epoch")  

plt.ylabel("Loss (MSE)")  

plt.legend()  

plt.show()  
  

# 6. Visualisasi prediksi sebelum & sesudah training  

# Sebelum training: model dengan bobot acak  

model_untrained = tf.keras.Sequential([  

    tf.keras.layers.Conv2D(4, kernel_size=(2,1),  

activation='relu', input_shape=(2,1,1)),  

    tf.keras.layers.Flatten(),  

    tf.keras.layers.Dense(1, activation='sigmoid')  

])  

model_untrained.compile(optimizer='adam', loss='mse')  
  

# Prediksi sebelum training  

pred_before = model_untrained.predict(X_test)  
  

# Prediksi sesudah training  

pred_after = model.predict(X_test)  
  

plt.figure(figsize=(10,4))  

plt.subplot(1,2,1)  

plt.scatter(X_test[:,0,0,0], X_test[:,1,0,0],  

c=pred_before.ravel()>0.5, cmap='coolwarm', alpha=0.7)  

plt.title("Prediksi Sebelum Training")  
  

plt.subplot(1,2,2)  

plt.scatter(X_test[:,0,0,0], X_test[:,1,0,0],  

c=pred_after.ravel()>0.5, cmap='coolwarm', alpha=0.7)  

plt.title("Prediksi Sesudah Training")  

plt.show()

```

Dataset:

1. make_moons → data dua kelas berbentuk bulan sabit.
2. Normalisasi agar lebih stabil.

CNN:

1. Conv2D → mengekstrak pola dari input.
2. Flatten → meratakan hasil feature map.
3. Dense → menghasilkan skor probabilitas (0-1).

Loss Function (MSE):

1. Mengukur jarak prediksi dengan target.

Training:

1. Bobot diperbarui agar loss mengecil.

Plot Loss Curve:

1. Menunjukkan penurunan loss seiring epoch.

Visualisasi Prediksi:

1. **Sebelum training:** tebakan acak, decision boundary berantakan.
 2. **Sesudah training:** CNN sudah belajar, boundary lebih rapi memisahkan dua kelas.
-

Backpropagation & Gradient.

Gradien dalam CNN

Gradien adalah arah perubahan parameter (kernel/filter) yang membuat **loss** menurun. Intinya: kernel di-update mengikuti arah negatif gradien agar prediksi semakin akurat.

Turunan (Derivative) dalam Optimisasi

Turunan menunjukkan seberapa cepat fungsi berubah terhadap suatu variabel. Dalam CNN, turunan loss terhadap kernel memberi tahu kita bagaimana mengubah kernel agar loss mengecil.

Backpropagation (Chain Rule)

CNN terdiri dari banyak lapisan. Backpropagation menggunakan **chain rule** untuk menghitung turunan loss terhadap parameter di setiap lapisan.

Contoh:

$[\frac{\partial L}{\partial W}] = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial W}$] di mana (L) = loss, (W) = kernel, (y) = output.

Numerical Gradient (Finite Difference)

- o Menghitung gradien dengan pendekatan:
 $[\frac{\partial f}{\partial x}] \approx \frac{f(x+h) - f(x-h)}{2h}$]
- o Mudah dipahami, tapi lambat dan kurang presisi.
- o Cocok untuk mengecek kebenaran implementasi backpropagation.

Analytical Gradient (Backpropagation)

- o Menggunakan rumus turunan langsung (chain rule).
- o Cepat dan akurat.
- o Dipakai dalam training CNN nyata.

compute_grad() dengan Finite Difference

Contoh sederhana dengan NumPy:

```
import numpy as np

def loss_function(x, W):
    # misalnya output sederhana: y = W * x
    y = np.dot(W, x)
    loss = np.sum(y**2)  # contoh loss = sum of squares
    return loss

def compute_grad(x, W, h=1e-5):
    grad = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_pos = W.copy()
            W_neg = W.copy()
            W_pos[i, j] += h
            W_neg[i, j] -= h
            grad[i, j] = (loss_function(x, W_pos) -
                           loss_function(x, W_neg)) / (2*h)
    return grad

# Contoh penggunaan
x = np.array([1.0, 2.0])
W = np.array([[0.5, -0.3],
              [0.8, 0.2]])

grad = compute_grad(x, W)
print("Gradient:\n", grad)
```

Setelah gradien dihitung, kernel diperbarui dengan gradient descent:
[$W_{\text{baru}} = W - \eta \cdot \nabla W$] di mana (η) adalah learning rate.

Artinya: kernel bergeser sedikit ke arah yang menurunkan loss.

Proses ini diulang ribuan kali hingga CNN belajar mengenali pola.

-
1. Gradien = arah perubahan kernel agar loss menurun.
 2. Backpropagation = cara menghitung gradien dengan chain rule.
 3. Numerical gradient (finite difference) = metode cek kebenaran.
 4. Analytical gradient = metode cepat untuk training.
 5. Kernel dimodifikasi dengan gradient descent mengikuti gradien.
-

gradien menggeser nilai kernel.

```
import numpy as np
import matplotlib.pyplot as plt

# Fungsi loss sederhana: output = W * x, lalu loss =
jumlah kuadrat output
def loss_function(x, W):
    y = np.dot(W, x)
    return np.sum(y**2)

# Hitung gradien dengan metode finite difference
def compute_grad(x, W, h=1e-5):
    grad = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_pos = W.copy()
            W_neg = W.copy()
            W_pos[i, j] += h
            W_neg[i, j] -= h
            grad[i, j] = (loss_function(x, W_pos) -
loss_function(x, W_neg)) / (2*h)
    return grad

# Visualisasi before/after update kernel
def show_update_examples():
    x = np.array([1.0, 2.0])      # input sederhana
    eta = 0.1                      # learning rate

    fig, axes = plt.subplots(5, 2, figsize=(6, 12))
    fig.suptitle("Kernel Before/After Update (5 contoh)",
    fontsize=14)

    for k in range(5):
        # Kernel acak (2x2)
        W = np.random.randn(2, 2)
        grad = compute_grad(x, W)
        W_new = W - eta * grad      # update kernel

        # Plot Before
        im0 = axes[k, 0].imshow(W, cmap="coolwarm",
vmin=-2, vmax=2)
        axes[k, 0].set_title(f"Before {k+1}")
        axes[k, 0].axis("off")

        # Plot After
        im1 = axes[k, 1].imshow(W_new, cmap="coolwarm",
vmin=-2, vmax=2)
        axes[k, 1].set_title(f"After {k+1}")
        axes[k, 1].axis("off")
```

```
plt.tight_layout()  
plt.show()
```

```
# Jalankan visualisasi  
show_update_examples()
```

-
1. **Kernel Before:** nilai awal acak.
 2. **Gradien:** dihitung dengan finite difference.
 3. **Kernel After:** kernel diperbarui dengan gradient descent.
 4. **Plot warna:** menunjukkan perubahan nilai kernel. Warna bergeser → kernel belajar.
 5. Ada **5 contoh** sehingga siswa bisa melihat variasi perubahan kernel.
-

gradien menggeser kernel.

```
import numpy as np  
  
x = np.array([1.0, 2.0])  
W = np.array([[0.5, -0.3],  
             [0.8, 0.2]])  
  
grad = (np.dot(W, x) * x).reshape(W.shape) # grad  
analitik sederhana  
W_new = W - 0.1 * grad  
  
print("Before:\n", W)  
print("After:\n", W_new)
```

```
x = np.array([1.0, -1.0, 2.0])  
W = np.random.randn(3,3)
```

```
grad = (np.dot(W, x) * x).reshape(W.shape)  
W_new = W - 0.05 * grad
```

```
print("Before:\n", W)  
print("After:\n", W_new)
```

```
x = np.array([2.0, 1.0])  
W = np.eye(2) # kernel identitas
```

```
grad = (np.dot(W, x) * x).reshape(W.shape)  
W_new = W - 0.1 * grad
```

```
print("Before:\n", W)  
print("After:\n", W_new)
```

```
x = np.array([1.0, 1.0])  
W = np.array([[0.01, 0.02],  
             [0.03, 0.04]])
```

```

grad = (np.dot(W, x) * x).reshape(W.shape)
W_new = W - 0.5 * grad # learning rate lebih besar

print("Before:\n", W)
print("After:\n", W_new)



---


x = np.array([1.0, -2.0])
W = np.array([[0.5, -0.2],
              [-0.1, -0.3]])

grad = (np.dot(W, x) * x).reshape(W.shape)
W_new = W - 0.1 * grad

print("Before:\n", W)
print("After:\n", W_new)

```

Before = kernel awal.

Gradien = arah perubahan berdasarkan input.

After = kernel bergeser mengikuti gradien (gradient descent).

Variasi kernel (acak, identitas, kecil, negatif) membantu siswa melihat bahwa **gradien selalu mengarahkan kernel ke arah yang menurunkan loss**.

5 contoh kernel sederhana tadi dengan plot warna (imshow)

```

import numpy as np
import matplotlib.pyplot as plt

# Fungsi loss sederhana
def loss_function(x, W):
    y = np.dot(W, x)
    return np.sum(y**2)

# Hitung gradien dengan finite difference
def compute_grad(x, W, h=1e-5):
    grad = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_pos = W.copy()
            W_neg = W.copy()
            W_pos[i, j] += h
            W_neg[i, j] -= h
            grad[i, j] = (loss_function(x, W_pos) -
loss_function(x, W_neg)) / (2*h)
    return grad

# Daftar 5 contoh kernel
examples = [
    (np.array([1.0, 2.0]), np.array([[0.5, -0.3], [0.8,
0.2]])), # Contoh 1

```

```

        (np.array([1.0, -1.0, 2.0]), np.random.randn(3,3)),
# Contoh 2
        (np.array([2.0, 1.0]), np.eye(2)),
# Contoh 3
        (np.array([1.0, 1.0]), np.array([[0.01, 0.02], [0.03,
0.04]])),           # Contoh 4
        (np.array([1.0, -2.0]), np.array([[-0.5, -0.2], [-0.1,
-0.3]]))          # Contoh 5
]

# Plot hasil before/after
fig, axes = plt.subplots(len(examples), 2, figsize=(6,
12))
fig.suptitle("Kernel Before/After Update (5 contoh)",
fontsize=14)

for idx, (x, W) in enumerate(examples):
    grad = compute_grad(x, W)
    W_new = W - 0.1 * grad # update kernel

    # Plot Before
    axes[idx, 0].imshow(W, cmap="coolwarm", vmin=-1,
vmax=1)
    axes[idx, 0].set_title(f"Before {idx+1}")
    axes[idx, 0].axis("off")

    # Plot After
    axes[idx, 1].imshow(W_new, cmap="coolwarm", vmin=-1,
vmax=1)
    axes[idx, 1].set_title(f"After {idx+1}")
    axes[idx, 1].axis("off")

plt.tight_layout()
plt.show()

```

Setiap baris = satu contoh kernel.
Kolom kiri (Before) = kernel awal.
Kolom kanan (After) = kernel setelah update dengan gradient descent.

Warna menunjukkan nilai kernel: perubahan warna = kernel bergeser mengikuti gradien.

animasi update kernel berulang kali (20 langkah)

Kita gunakan `matplotlib.animation.FuncAnimation` agar setiap frame menunjukkan kernel setelah update gradien.

```

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

```

```

# Fungsi loss sederhana
def loss_function(x, W):
    y = np.dot(W, x)
    return np.sum(y**2)

# Hitung gradien dengan finite difference
def compute_grad(x, W, h=1e-5):
    grad = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_pos = W.copy()
            W_neg = W.copy()
            W_pos[i, j] += h
            W_neg[i, j] -= h
            grad[i, j] = (loss_function(x, W_pos) -
loss_function(x, W_neg)) / (2*h)
    return grad

# Setup kernel awal
x = np.array([1.0, 2.0])
W = np.random.randn(2, 2)      # kernel acak
eta = 0.1                      # learning rate

# Simpan hasil update untuk 20 langkah
steps = 20
kernels = [W.copy()]
for _ in range(steps):
    grad = compute_grad(x, kernels[-1])
    W_new = kernels[-1] - eta * grad
    kernels.append(W_new)

# Buat animasi
fig, ax = plt.subplots()
im = ax.imshow(kernels[0], cmap="coolwarm", vmin=-2,
vmax=2)
ax.set_title("Kernel Update (Step 0)")
ax.axis("off")

def update(frame):
    im.set_array(kernels[frame])
    ax.set_title(f"Kernel Update (Step {frame})")
    return [im]

ani = animation.FuncAnimation(fig, update,
frames=len(kernels), interval=500, blit=True)

plt.show()

```

Kernel awal ditampilkan pada step 0.
Setiap frame animasi menunjukkan kernel setelah update gradien.

Warna bergeser dari frame ke frame → kernel belajar menurunkan loss.

Dengan 20 langkah, siswa bisa melihat proses belajar secara bertahap, bukan hanya before/after.

grafik loss per langkah di samping animasi. Jadi siswa bisa melihat **kernel berubah** sekaligus **loss menurun** secara paralel.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Fungsi loss sederhana
def loss_function(x, W):
    y = np.dot(W, x)
    return np.sum(y**2)

# Hitung gradien dengan finite difference
def compute_grad(x, W, h=1e-5):
    grad = np.zeros_like(W)
    for i in range(W.shape[0]):
        for j in range(W.shape[1]):
            W_pos = W.copy()
            W_neg = W.copy()
            W_pos[i, j] += h
            W_neg[i, j] -= h
            grad[i, j] = (loss_function(x, W_pos) -
loss_function(x, W_neg)) / (2*h)
    return grad

# Setup kernel awal
x = np.array([1.0, 2.0])
W = np.random.randn(2, 2)      # kernel acak
eta = 0.1                      # learning rate

# Simpan hasil update untuk 20 langkah
steps = 20
kernels = [W.copy()]
losses = [loss_function(x, W)]
for _ in range(steps):
    grad = compute_grad(x, kernels[-1])
    W_new = kernels[-1] - eta * grad
    kernels.append(W_new)
    losses.append(loss_function(x, W_new))

# Buat figure dengan 2 subplot: kernel + grafik loss
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 5))

# Subplot 1: Kernel
im = ax1.imshow(kernels[0], cmap="coolwarm", vmin=-2,
vmax=2)
```

```

ax1.set_title("Kernel Update (Step 0)")
ax1.axis("off")

# Subplot 2: Grafik Loss
line, = ax2.plot([], [], 'b-o')
ax2.set_xlim(0, steps)
ax2.set_ylim(0, max(losses)*1.1)
ax2.set_title("Loss per Step")
ax2.set_xlabel("Step")
ax2.set_ylabel("Loss")

def update(frame):
    # Update kernel plot
    im.set_array(kernels[frame])
    ax1.set_title(f"Kernel Update (Step {frame})")

    # Update grafik loss
    line.set_data(range(frame+1), losses[:frame+1])
    return [im, line]

ani = animation.FuncAnimation(fig, update,
frames=len(kernels), interval=500, blit=True)

plt.tight_layout()
plt.show()

```

Kiri (Kernel): menunjukkan perubahan nilai kernel setiap langkah update.

Kanan (Grafik Loss): menampilkan penurunan loss seiring kernel belajar.

Siswa bisa melihat hubungan langsung: **kernel bergeser → loss menurun.**

Dengan 20 langkah, terlihat proses belajar CNN secara bertahap.
