Bab 3

Supervised Learning: Konsep dan Implementasi CNN

Supervised learning adalah sebuah teknik dari pembelajaran mesin (ML) yang mana data dilatih untuk mengenali suatu output tertentu pada data yang sudah dilabeli. Setiap data sudah memiliki input (fitur) dan output (label) yang telah ditentukan. Proses pelabelan data ini dilakukan secara mandiri/manual, yang pada kondisi kasus tertentu, harus berdasarkan validasi ahli. Di sini, model dilatih untuk mengenali hubungan antara input dengan output.

Dalam dunia *supervised learning*, banyak teknik yang digunakan untuk memecahkan masalah klasifikasi atau regresi. Untuk data tabular (seperti data angka dan tabel), model seperti Decision Tree, SVM, atau Random Forest sering digunakan. Namun, ketika kita berhadapan dengan data citra atau gambar, model tradisional sering kali kesulitan mengenali pola kompleks di dalam piksel-piksel gambar.

Di sinilah Convolutional Neural Network (CNN) hadir sebagai solusi yang lebih efektif. CNN dirancang khusus untuk memproses dan memahami data gambar dengan memanfaatkan sifat spasial dari citra. Dibandingkan dengan model lain, CNN mampu menangkap pola seperti tepi, sudut, bentuk, dan tekstur dari gambar secara otomatis tanpa perlu melakukan ekstraksi

fitur secara manual. Inilah yang membuat CNN menjadi pilihan utama dalam supervised learning untuk tugas-tugas seperti klasifikasi gambar, deteksi objek, dan segmentasi citra. Dengan kata lain, CNN adalah "mesin" yang sangat kuat dalam supervised learning ketika data yang kita hadapi berupa citra atau data visual lainnya.

CNN berperan penting dalam *supervised learning* terutama dalam dunia visi komputer. Pada awalnya, CNN di desain untuk mengenali fitur dari data citra dan video (Putra, Trisnaningrum, Puspitasari, Wibowo, & Rachmawaty, 2022). Namun, semenjak perkembangan yang pesan di bidang ini, CNN diterapkan juga untuk data teks (Zhou, Li, Chi, Tang, & Zheng, 2022) dan data tiga dimensi berupa voxel (Putra et al., 2023) dan point cloud (Putra et al., 2024).

CNN telah banyak diterapkan di dunia nyata. Yann Lecun (Lecun, Bottou, Bengio, & Haffner, 1998) mengenalkan teknik untuk rekognisi tulisan tangan angka berbasis CNN yang dikenal dengan nama LeNet. Teknik ini cukup efektif dalam mendeteksi angka. Sayangnya, teknik ini rentan terhadap gradien yang hilang atau *vanishing gradient*. *vanishing gradient* adalah kondisi ketika nilai gradien dari fungsi loss sangat kecil ketika backpropagasi. Kondisi ini menyebabkan layer-layer di awal sulit dilatih secara efektif. Akhirnya, kondisi konvergen sulit dicapai oleh model. Kemudian, CNN dikembangkan lebih lanjut lagi oleh Kaiming He dengan teknik yang revolusioner dikenal dengan ResNet (He, Zhang, Ren, & Sun, 2016). ResNet, dengan *skip connection*, mampu mengatasi isu sebelumnya. Kemudian, CNN dikembangkan lebih jauh lagi untuk deteksi objek, lahirlah YOLO (You Only Look Once) (Redmon, Divvala, Girshick, & Farhadi, 2015). Teknik-teknik ini banyak diterapkan di bidang transportasi, kesehatan, dan pertanian. Bab ini, membahas tentang, Konsep CNN, Arsitektur CNN, Implementasi CNN dengan PyTorch, dan Transfer Learning

3.1 Konsep Convolutional Neural Network (CNN)

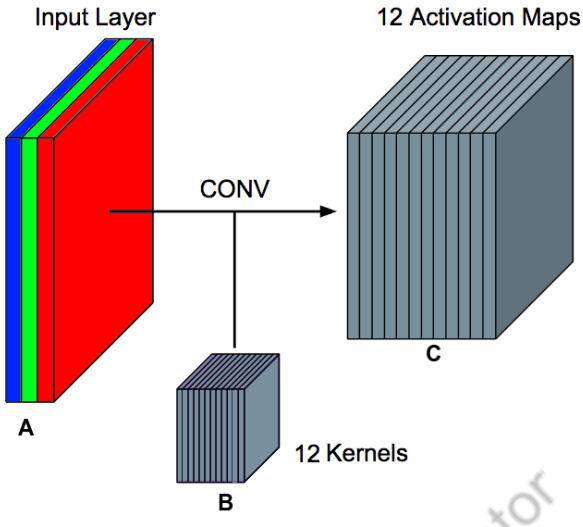
3.1.1 Definisi

CNN adalah metode dalam ekstraksi fitur berbasis konvolusi pada data citra maupun video. Dalam konteks bab ini, CNN yang kita bahas adalah CNN untuk data citra dua dimensi. CNN bekerja dengan cara mengekstrak fitur spasial dari data citra dengan menggunakan pendekatan konvolusi. Berbeda dengan model ML yang lama di mana mengharuskan fitur diolah lebih dahulu di awal.

Metode pembelajaran mesin tradisional kesulitan untuk memproses data mentah secara langsung. Selama bertahun-tahun, membangun sistem pembelajaran mesin membutuhkan desain cermat dan pengetahuan yang mendalam untuk membuat ekstraksi fitur. Ekstraksi fitur ini mengubah data mentah (seperti intensitas piksel pada citra) menjadi format yang lebih bermakna, yang disebut fitur vektor atau fitur map. Nantinya, fitur vektor ini yang digunakan untuk klasifikasi maupun segmentasi objek pada citra.

3.1.2 Cara Kerja CNN

Citra yang berukuran besar terdiri atas ratusan hingga ribuan piksel. Jika hanya bergantung pada ML tradisional dengan menjadikan setiap piksel pada citra sebagai fitur, tentunya layer-layer dari neural network akan sangat besar hingga puluhan ribu neuron. Neuron yang sangat besar ini memerlukan sumber daya baik RAM maupun CPU yang besar juga. CNN, dengan pendekatan konvolusi, mampu mengekstrak informasi dari citra dengan pendekatan spasial, seperti grid atau matriks. Dengan bentuk matriks ini, informasi dari relasi antar piksel yang berdekatan bisa diketahui.



Gambar 3.1: Convolution Layer

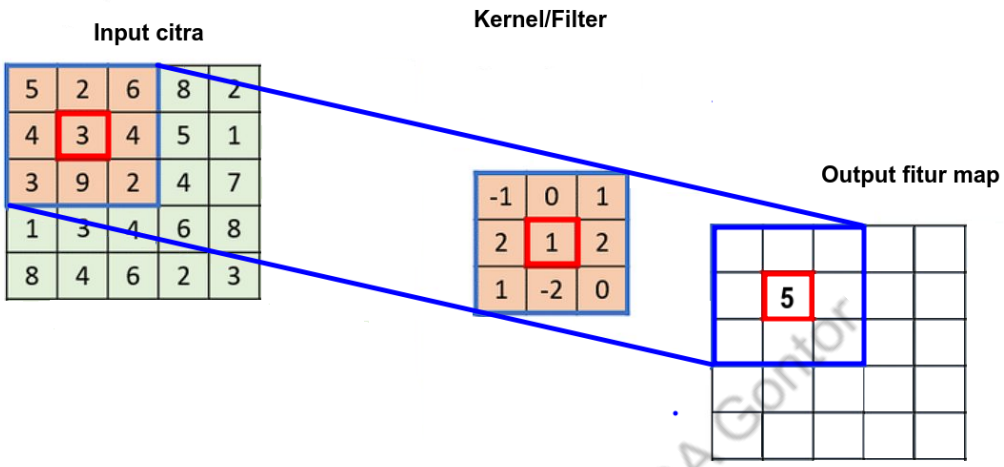
3.2 Komponen CNN

3.2.1 *Convolutional Layer*

Convolution layer (ConvLayer) atau lapisan konvolusi merupakan lapisan yang paling penting dari CNN. Di sini terjadi proses ekstraksi fitur spasial dari citra inputan. Idenya adalah kita memiliki sejumlah filter atau kernel. Filter-filter ini hanyalah *patch* atau tambalan kecil yang mencerminkan semacam fitur visual. Kita menghitung dari setiap fitler dan melakukan perhitungan konvolusi pada citra input untuk menghasilkan satu fitur map atau *activation map*.

Sekarang bayangkan, kita ambil sebagian kecil dari citra pada Gambar 3.1.A dan menjalankan hitungan konvolusi dengan filter atau kernel di atasnya dengan sejumlah K output. Sekarang, geser kernel tersebut ke seluruh citra. Sebagai hasilnya, kita akan mendapatkan citra lain dengan lebar, tinggi, dan kedalaman yang berbeda seperti pada Gambar 3.1.C. Selain R,G,B, kita juga mendapatkan *channel* lain tetapi dengan lebar dan tinggi yang lebih kecil. Jika ukuran tambalan sama dengan ukuran citra, maka akan menjadi jaringan saraf biasa. Dengan adanya tambalan yang kecil ini, ki-

ta memiliki bobot yang lebih sedikit jumlahnya. Adapun proses konvolusi divisualisasikan pada Gambar 3.2.



Gambar 3.2: Proses Konvolusi dengan kernel pada citra input

Sebagai contoh, proses konvolusi pada input akan memberikan fitur map. Untungnya, dalam CNN, kita perlu menulis kode filter, seperti menentukan filter untuk deteksi tepian citra, *sharpening*, *blur*, dan sebagainya. Selama backprop, CNN belajar bahwa mendeteksi fitur seperti garis membantu model CNN bekerja lebih baik sehingga akan mengubah filter seperti halnya dengan memperbarui bobot jaringan saraf tiruan.

3.2.2 *Pooling Layer*

Yang tidak kalah penting dari CNN adalah *pooling layer* (PoolLayer). Layer yang biasanya diletakkan setelah ConvLayer yang berfungsi untuk mengurangi dimensi secara spasial dari fitur map. Pengurangan dimensi ini tentunya berdampak pada penurunan jumlah parameter sehingga mengurangi beban komputasi.

PoolLayer menghitung secara mandiri pada tiap-tiap fitur map dengan tetap menjaga ukuran dimensi dari kedalaman atau *depth* fitur map. Sebagai

ilustrasi singkat, apabila ada sejumlah N fitur map, maka output dari operasi PoolLayer ini juga sebanyak N fitur map.

3.2.3 *Max vs Average Pooling*

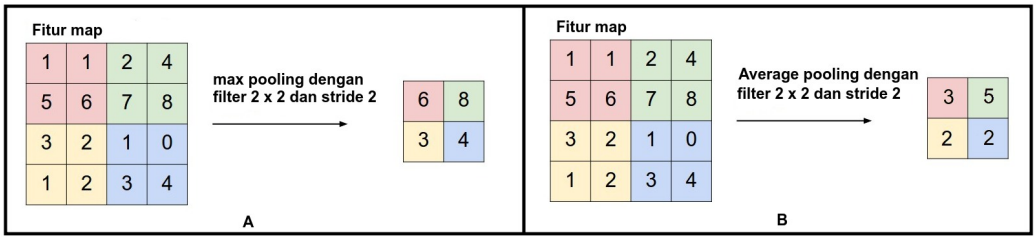
Tidak ada parameter yang perlu dioptimasi pada PoolLayer, berbeda dengan ConvLayer. Adapun yang terjadi di layer ini adalah setiap nilai dari fitur map dihitung dengan suatu fungsi seperti *max pooling* atau *average pooling*.

Pada *max pooling*, layer menggeser jendela atau *window*, misalnya dengan ukuran 2×2, di setiap irisan kedalaman fitur map dan mengambil nilai maksimum di dalam *window* dari setiap langkah. Sebagai contoh, filter 2×2 yang diterapkan dengan 2 *stride* akan mengurangi setiap dimensi spasial (lebar dan tinggi) hingga setengahnya. Tentunya, proses ini, membuang 75% dari aktivasi dalam area yang dikonvolusi dengan *max pooling* tersebut serta mempertahankan fitur yang paling menonjol.

PoolLayer menerima input dengan ukuran $W_1 \times H_1 \times D_1$ di mana W_1 adalah lebar inputan, H_1 adalah tinggi inputan, dan D_1 adalah kedalaman input atau yang biasa disebut dengan jumlah *channel* dari fitur map. Selain itu, PoolLayer juga membutuhkan dua parameters, yaitu F dan S. F adalah ukuran jendela *pooling*, misal 2×2 atau 3×3. Sedangkan S adalah ukuran *stride* atau pergeseran. *Stride* adalah seberapa banyak piksel yang digerakkan oleh jendela setiap langkah.

Selain *max pooling*, juga ada *average pooling*. Setelah layer konvolusi, dalam arsitekturnya, LeNet menggunakan *pooling* jenis *average pooling*. Perbedaan utama *max pooling* dan *average pooling* terletak dalam perhitungan ketika proses *sliding window*. Pada *max pooling*, nilai intensitas tertinggilah yang diambil oleh filter dalam setiap fitur map. Sedangkan *average pooling*, piksel-piksel dalam *windows*, dihitung rata-ratanya.

Sebagai penjelas, perhatikan Gambar 3.3. PoolLayer menurunkan sampel volume secara spasial, secara independen di setiap irisan kedalaman volume input. Proses *downsampling* yang paling umum adalah *max*, yang menghasilkan *pooling max*, di sini ditunjukkan dengan *stride* 2. Artinya, setiap



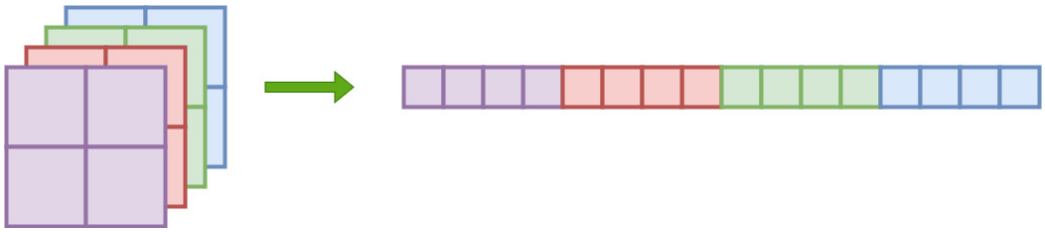
Gambar 3.3: Perbedaan *Max Pooling* (A) dengan *Average Pooling* (B).

max diambil lebih dari 4 angka (kotak kecil 2×2).

3.2.4 *Fully-Connected Layer*

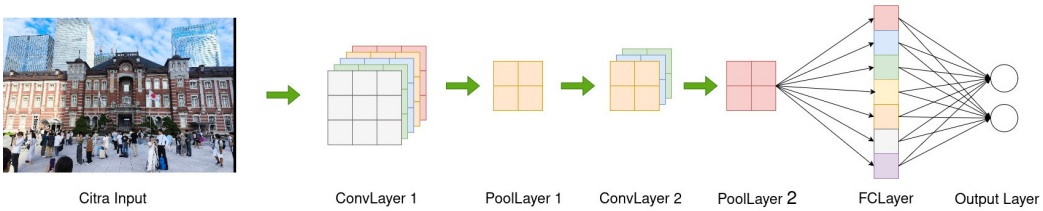
Di setiap akhir dari CNN, terdapat sebuah satu atau lebih *layer* yang setiap neuron terhubung semuanya yang biasa disebut sebagai *fully-connected layer* (FCLayer). Fitur map memiliki bentuk dimensi $m \times n \times d$ di mana $m > 1$, $n > 1$, $d \geq 1$, m adalah lebar fitur map, n adalah tingginya, sedangkan d adalah *depth* atau kedalaman (*channel*). Melihat kondisi ini, bentuk fitur map adalah menyerupai matriks.

Di FCLayer ini, fitur map dari layer sebelumnya ditransformasikan menjadi fitur vektor. Dengan kata lain, mengubah bentuk matriks atau tensor multi-dimensi menjadi vektor satu dimensi. Perhatikan contoh pada Gambar 3.4 untuk perubahan matriks/tensor ke vektor.



Gambar 3.4: Perbedaan Transformasi matriks ke vektor

Gambar 3.5 menunjukkan struktur ujung ke ujung dari jaringan utuh atau model CNN.



Gambar 3.5: CNN yang ditampilkan di sini berisi dua modul konvolusi (konvolusi + pooling) untuk ekstraksi fitur, dan dua FCLayer untuk klasifikasi. Model CNN lain bisa jadi berisi ConvLayer dalam jumlah yang lebih besar atau lebih kecil, dan FCLayer yang lebih banyak atau lebih sedikit.

3.3 Implementasi CNN untuk Klasifikasi Citra

Di sini, kita akan menerapkan CNN untuk beberapa kasus terkait klasifikasi citra. Ada tiga dataset yang biasa digunakan untuk belajar tahap awal CNN, yaitu MNIST, CIFAR-10, dan ImageNet.

3.3.1 Dataset

3.3.1.1 MNIST

MNIST singkatan dari Modified National Institute of Standards and Technology adalah kumpulan dataset citra tulisan tangan (Lecun et al., 1998; L. Deng, 2012). Dataset ini terdiri atas 60.000 citra untuk data latih dan 10.000 citra untuk data uji coba. Dataset ini terdiri atas 10 kelas. Contoh dari data CIFAR-10 bisa dilihat pada Gambar 3.6.

3.3.1.2 CIFAR-10

Dataset CIFAR-10 terdiri dari 60.000 citra berwarna berukuran 32×32 piksel (Krizhevsky & Hinton, 2009). Dataset ini terdiri atas 10 kelas, dengan 6.000 citra tiap kelasnya. Data ini juga dibagi menjadi 50.000 citra untuk data latih, dan 10.000 citra untuk data ujicoba. CIFAR-10 ini dikembangkan oleh

Gambar 3.6: Sampel dataset MNIST

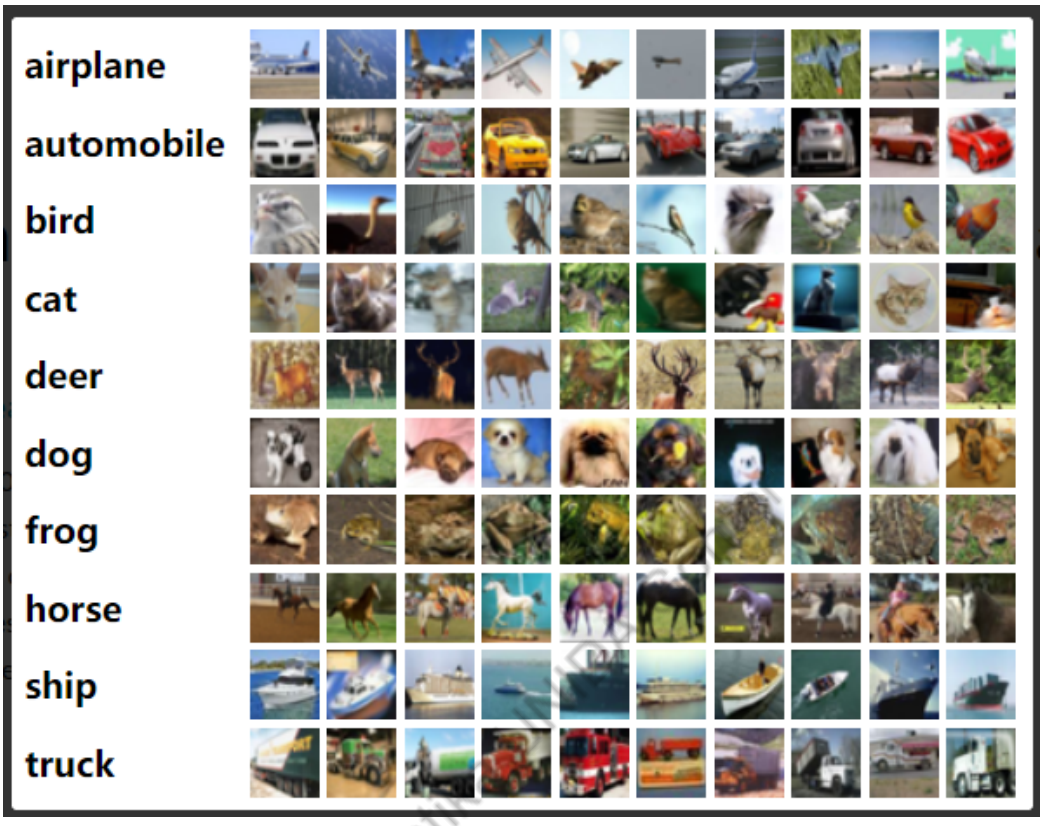
Canadian Institute for Advanced Research. Contoh dari data CIFAR-10 bisa dilihat pada Gambar 3.7.

3.3.2 Preprocessing data

Beragam bentuk data baik itu dari sisi variasi dan rentang menentukan bagaimana model memprediksi data. Untuk meminimalkan sensitivitas model, maka perlu preprocessing data. Preprocessing pada data merupakan langkah yang sangat penting dalam *deep learning* karena berkaitan dengan kinerja model yang akan dilatih. Ada beberapa tahapan yang lazim digunakan, yaitu normalisasi, standardisasi, dan augmentasi.

3.3.2.1 Normalisasi Data

Normalisasi data adalah proses mengubah data menjadi rentang tertentu seperti $[0, 1]$ atau $[-1, 1]$. Ide dasarnya adalah bahwa data mentah cenderung memiliki beragam variasi. Sebagai contoh:



Gambar 3.7: Sampel dataset CIFAR-10

- Untuk data citra (RGB atau Grayscale), intensitas piksel citra terletak antara $[0, 255]$
- Data gaji pegawai, misalnya antara Rp 3,5 juta hingga 6 juta
- Data suhu, misalnya berkisar antara 20 - 35 derajat celsius.

Melihat contoh-contoh tersebut, data yang beragam, akan menyulitkan model selama proses pelatihan di mana model cenderung fokus pada fitur dengan nilai angka yang tinggi. Dengan adanya normalisasi, semua angka numerik dari fitur dikonversi menjadi skala yang sama. Berikut ini contoh persamaan normalisasi:

$$x_{normalisasi} = \frac{x - x_{min}}{x_{maks} - x_{min}} \tag{3.1}$$

Dengan menerapkan Persamaan 3.1, setiap data dikonversi menjadi $[0, 1]$.

Normalisasi itu penting karena membuat proses *gradient descent* bekerja lebih efisien. Dengan data yang dinormalisasi, model dapat mengalami konvergensi yang lebih cepat selama pelatihan. Selain itu, normalization mencegah situasi di mana satu fitur mendominasi fitur lainnya hanya karena memiliki nilai numerik yang lebih besar. Secara keseluruhan, teknik ini menghasilkan pelatihan yang lebih stabil dan membantu meningkatkan kinerja model dalam proses pembelajaran mesin.

3.3.2.2 Standardisasi Data

Standardisasi data adalah salah satu teknik prapemrosesan yang digunakan untuk mengubah nilai angka fitur sehingga memiliki nilai rata-rata nol (*zero means*) dan varians unit *unit variance*. Tidak seperti normalisasi, yang mengubah skala data ke rentang tertentu (misalnya, $[0, 1]$ atau $[-1, 1]$), standardisasi berfokus pada penyesuaian distribusi data. Standardisasi data juga sangat berguna ketika fitur-fitur dalam dataset memiliki skala atau satuan yang berbeda.

Standardisasi memastikan bahwa setiap fitur berkontribusi secara merata pada proses pembelajaran model dengan mengubah data ke dalam skala yang sama. Sebagai contoh, bayangkan sebuah dataset di mana satu fitur, misalnya tinggi badan, dalam sentimeter (berkisar antara 150 hingga 180) dan fitur lainnya mewakili pendapatan dalam rupiah (berkisar antara 30.000 hingga 150.000). Tanpa standarisasi, model mungkin akan "menganggap" pendapatan lebih penting hanya karena nilai numeriknya jauh lebih besar.

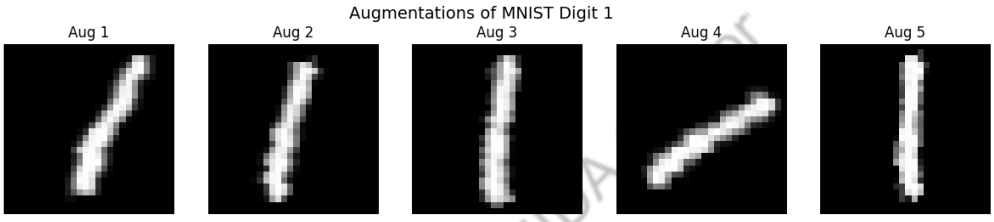
Standardisasi data dihitung dengan:

$$x_{std} = \frac{x - \mu}{\sigma} \tag{3.2}$$

di mana x adalah nilai angka asli, μ adalah nilai rata-rata, σ adalah standar deviasi atau simpangan baku. Persamaan 3.2 mentransformasi data untuk mengikuti distribusi normal standar (distribusi Gaussian dengan $mean = 0$ dan $varians = 1$).

3.3.2.3 Augmentasi Data

Ada suatu kondisi ketika data yang dimiliki jumlahnya terbatas dan belum lagi distribusi atau sebaran data pada setiap kelas berbeda-beda. Kadang juga, ketika dilakukan pelatihan model pada data yang sedemikian, menyebabkan kinerja dari model kurang bagus. Ada juga kondisi dimana dataset yang dimiliki tingkat variasinya terlalu rendah sehingga model mengalami kondisi yang kurang general. Untuk mengatasi ini, bisa menggunakan teknik yang dinamakan sebagai augmentasi data.



Gambar 3.8: Contoh Augmentation Data dengan Random Crop, Rotation $[-30, 30]$

Augmentasi data adalah teknik dalam pembelajaran mesin atau *deep learning* untuk menambah data secara arfisial baik secara kuantitas maupun variasinya dengan cara modifikasi dari dataset yang sudah ada. Teknik ini sering digunakan terutama untuk kasus dataset berupa citra. Beberapa teknik augmentasi yang sering digunakan adalah:

- Rotasi. Teknik rotasi bisa menambahkan dataset citra baru berdasarkan citra asli yang dirotasi dengan derajat tertentu secara acak. Misalnya: citra dirotasi -30°atau 30°.
- *Horizontal/Vertical Flip.* Ini adalah salah satu teknik yang digunakan untuk menambahkan citra baru dengan membalikkan citra secara horizontal atau vertikal.
- *Random Crop* berfungsi untuk menghasilkan citra baru dengan cara memotonng secara acak pada area tertentu dari citra asli.

- *Color Jitter* adalah teknik augmentasi untuk menambah citra baru dengan cara mengubah nilai kecerahan (*brightness*), kontras (*contrast*), saturasi (*saturation*), atau hue.*Color jitter* berfungsi untuk menghasilkan variasi citra baru dengan cara

- *Affine Transformation*. Jika masih ingat materi translasi data, teknik *affine transformation* ini adalah salah satu penerapannya. Di sini, citra baru ditentukan berdasarkan transformasi matriks seperti translasi, skala, atau *shearing*.

Adapun contoh dari Augmentasi Data bisa dilihat pada Gambar 3.8.

3.3.3 Membangun arsitektur CNN

Sebelum membangun model CNN, ada beberapa hal yang perlu diperhatikan yaitu ukuran input citra, ukuran *batch*, jumlah layer (ConvLayer, PoolLayer, FCLayer, dan sebagainya), dan urutannya.

Untuk saat ini, kita buat model sederhana dengan meniru dari LeNet dengan dataset MNIST. LeNet tersusun atas beberapa layer Conv2D, diikuti dengan *Average Pooling* layer, kemudian layer Conv2D lagi, dan *Average Pooling* layer. Layer pertama (Conv2D) memerlukan inputan $1 \times 32 \times 32$ diikuti dengan *Max Pooling* dengan ukuran 2×2. Kemudian, pada Conv2D berikutnya memiliki ukuran $16 \times 10 \times 10$ dan diikuti juga dengan *Max Pooling* dengan ukuran 2×2. Di layer akhir, ada 2 layer FCLayer dengan ukuran/jumlah neuron masing-masing 120 dan 84. Adapun dioutput layer, terdapat 10 kelas. Adapun arsitektur model CNN kita bisa dilihat pada Tabel 3.1.

3.3.4 Fungsi Loss

Fungsi loss adalah fungsi matematis yang digunakan untuk mengukur seberapa bagus kinerja suatu model dalam memprediksi data. Selain itu, fungsi loss juga berfungsi sebagai petunjuk dalam proses pelatihan.

Ada banyak jenis fungsi loss, tergantung pada masalah yang dihadapi:

Tabel 3.1: Bentuk arsitektur model CNN berbasis LeNet dengan *Max Pooling* pada dataset MNIST

Layer	**Ukuran output**
Input	$1 \times 28 \times 28$
Conv1 (5x5, pad=2)	$6 \times 28 \times 28$
MaxPool1 (2x2)	$6 \times 14 \times 14$
Conv2 (5x5)	$16 \times 10 \times 10$
MaxPool2 (2x2)	$16 \times 5 \times 5$
Flatten	400
FC1	120
FC2	84
FC3 (Output)	10

- Untuk data numerik, kita sering menggunakan loss seperti Mean Squared Error (MSE), yang menghitung rata-rata kesalahan kuadrat.

- Untuk data kategori, seperti memprediksi apakah sebuah gambar adalah kucing atau anjing, kita menggunakan loss seperti Cross-Entropy.

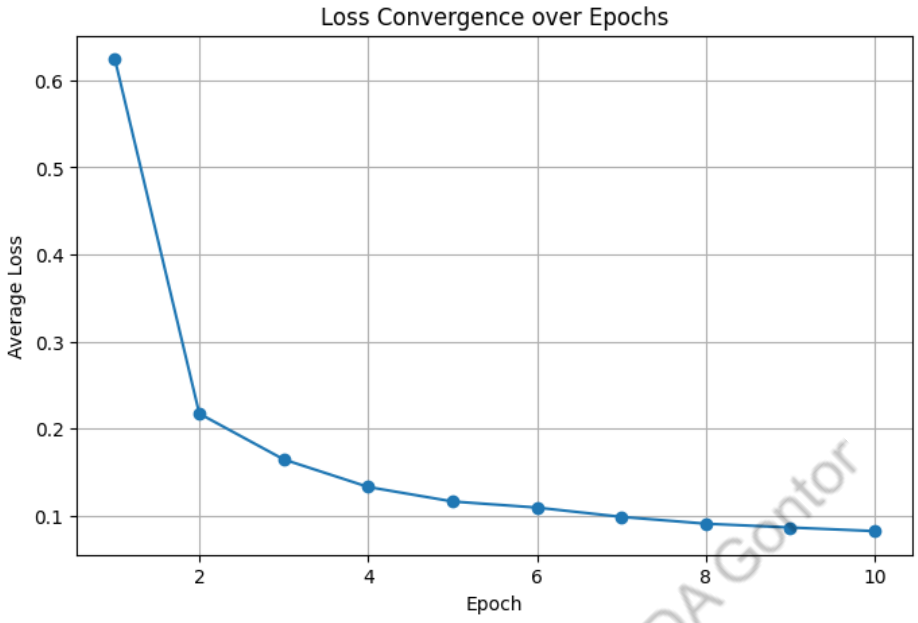
Untuk kasus klasifikasi MNIST, kita menggunakan fungsi loss Cross-Entropy yang diformulasikan sebagai:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i) \tag{3.3}$$

di mana C adalah jumlah kelas, y_i adalah nilai output yang aktual, dan $\hat{y}_i$ adalah nilai yang diprediksi.

Fungsi loss ini sangat penting karena selain menentukan bagaimana bobot diperbarui, juga mempengarui kapan model konvergen [1]. Selain itu, kita juga bisa memantau kinerja model selama proses pelatihan. Conton korvergen bisa dilihat pada Gambar 3.9.

[1]konvergen adalah suatu kondisi dalam pelatihan model *deep learning* bahwa nilai loss sudah tidak mengalami perubahan atau penurunan yang signifikan



Gambar 3.9: Contoh hasil training loss yang telah mencapai kondisi konvergen

3.3.5 Proses pelatihan model

Untuk persiapan pelatihan model CNN, ada beberapa hal yang perlu disiapkan, yaitu:

1. *Learning rate* (LR). Dalam pembelajaran mesin, LR menentukan seberapa besar perubahan parameter dalam setiap epoch/perulangan dalam algoritma *gradient descent*. Jika nilai LR terlalu besar, maka besar kemungkinannya model akan kehilangan *global minima*. Sebaliknya, apabila LR terlalu kecil, model akan lebih lama mengalami konvergen dan rentan jatuh pada *local minima*

2. Ukuran Batch atau *batch size*
   Ukuran batch adalah jumlah data sampel atau data poin yang digunakan sebelum parameter dari model diperbaharui. Ada beberapa cara dalam penerapan ukuran batch

- jika ukuran batch = 1, maka update parameter/bobot akan dilakukan setiap satu data poin.
- jika ukuran batch = 32, maka update parameter/bobot akan dilakukan pada 32 sampel data dan dihitung rata-rata perubahan gradien.
- jika ukuran batch sama dengan ukuran dataset, maka update parameter/bobot dilakukan pada keseluruhan dataset

Ukuran batch berdampak pada penggunakan RAM atau GPU VRAM. Semakin besar ukuran batch, maka semakin besar memory yang dibutuhkan.

3. Jumlah iterasi/epoch. Jumlah iterasi juga membantu menentukan kapan model itu konvergen. Jika terlalu kecil jumlah iterasi, maka juga berisiko kondisi konvergen belum terpenuhi.

4. Fungsi loss. Fungsi loss yang umumnya digunakan adalah Cross-Entropy.

5. Optimizer. Optimizer berperan sebagai fungsi/teknik dalam mengupdate parameter/bobot dari model. Optimizer yang umum dipakai adalah Stochastic Gradient Descent (SGD), ADAM, dan RMSProp.

Adapun algoritma pelatihan data bisa dilihat pada Algoritma 1.

3.3.6 Evaluasi model

Tahapan ini merupakan tahapan yang sangat penting dalam proses pelatihan model. Melalui tahapan ini, kita bisa mengetahui seberapa bagus kinerja dari model yang kita latih terhadap dataset. Umumnya, evaluasi mengukur beberapa kriteria yaitu akurasi, loss,presisi, dan recall. Selain ini, yang tidak kalah penting adalah penggunaan confusion matrix (CM). CM merupakan tabel yang membandingkan hasil kinerja untuk kasus klasifikasi antara hasil prediksi dengan hasil sebenarnya. Dengan CM, kita bisa mengetahui apakah model sering salah memprediksi positif atau sering melewatkan kasus positif.

Algorithm 1 Proses Training CNN dengan Cross-Entropy dan SGD

Input: Dataset $\mathcal{D} = \{(x_i, y_i)\}$, model CNN f_θ, learning rate η
for setiap epoch **do**
    Shuffle dataset $\mathcal{D}$
    for setiap data point (x, y) di $\mathcal{D}$ **do**
        Forward pass: $\hat{y} = f_\theta(x)$
        Hitung loss: $\mathcal{L} = -\sum_{i=1}^{C} y_i \cdot \log(\hat{y}_i)$
        Backward pass: Hitung gradien $\nabla_\theta \mathcal{L}$
        Update parameter: $\theta = \theta - \eta \cdot \nabla_\theta \mathcal{L}$
    end for
end for

CM sangat diperlukan ketika dalam kasus kesehatan dan keamanan. Sebagai ilustrasi CM, perhatikan Tabel 3.2.

	Prediksi Positif	**Prediksi Negatif**
Aktual Positif	*True Positive* (TP)	*False Negative* (FN)
Aktual Negatif	*False Positive* (FP)	*True Negative* (TN)

Tabel 3.2: Confusion Matrix untuk Binary Classification

Pada Tabel 3.2 terdapat beberapa istilah:

- *True Positive* (TP)
  Model memprediksi positif, dan label aslinya memang positif. Contoh: Model memprediksi "COVID" dan faktanya pasien memang "COVID"

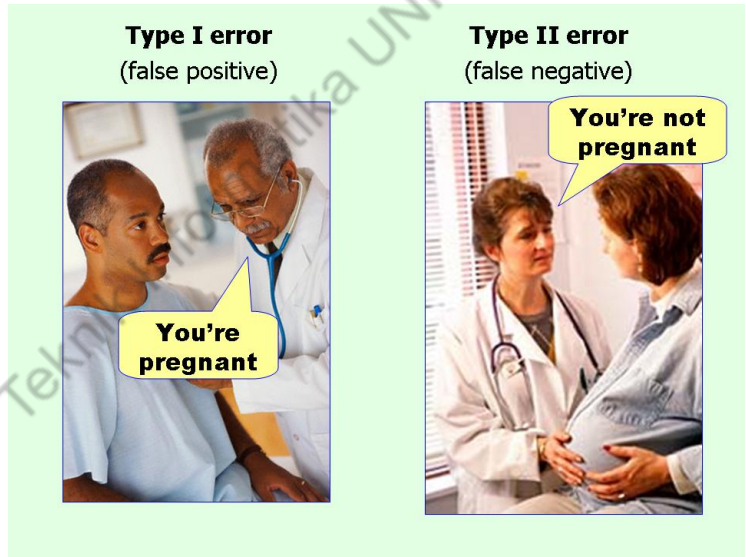
- *True Negative* (TN)
  Model memprediksi negatif, dan label aslinya juga negatif. Contoh: Model memprediksi "non-COVID" dan faktanya memang "non-COVID"

- *False Positive* (FP)
  Model memprediksi positif, tetapi sebenarnya data tersebut negatif (error tipe I). Contoh: Model memprediksi "COVID", padahal pasien "non-COVID"

- *False Negative* (FN)
  Model memprediksi negatif, tetapi sebenarnya data tersebut positif (error tipe II). Contoh: Model memprediksi "non-COVID", padahal pasien "COVID"

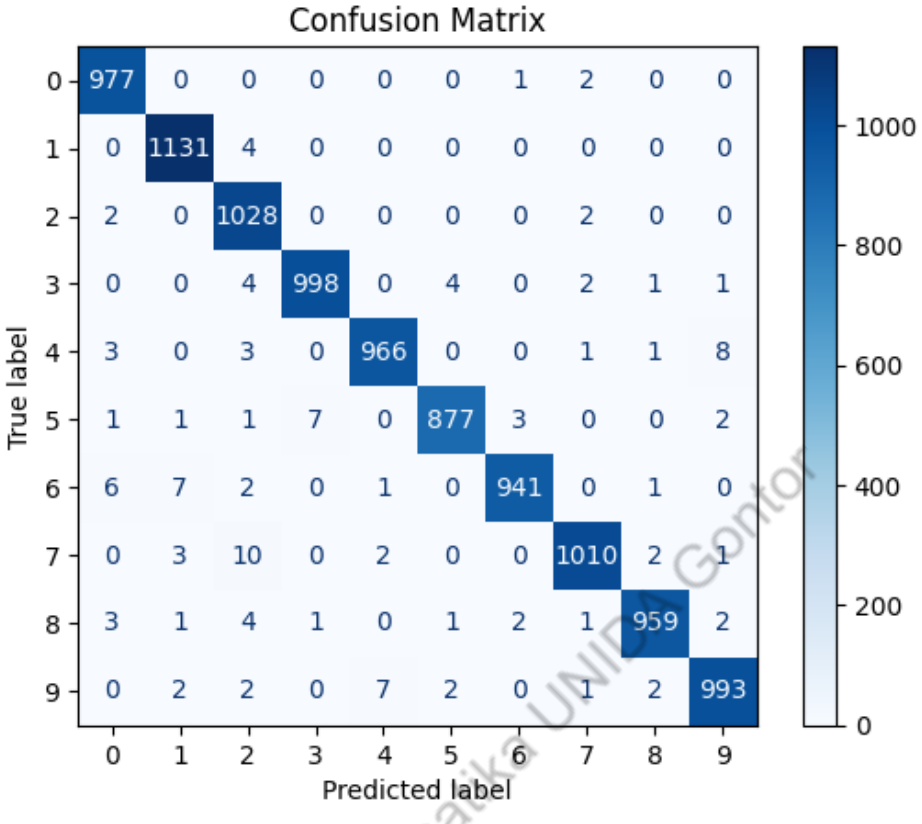
Satu-satunya cara yang bagus untuk mengilustrasikan Error Tipe 1 dan Tipe 2 adalah seperti pada Gambar 3.10[2]

.



Gambar 3.10: Ilustrasi Error Tipe I dan II.

[2]Gambar berasal dari: https://effectsizefaq.com/2010/05/31/i-always-get-confused-about-type-i-and-ii-errors-can-you-show-me-something-to-help-me-remember-the-difference



Gambar 3.11: Confusion Matrix pada data hasil tes

3.3.7 Metrik Kinerja

Dalam melakukan evaluasi kinerja suatu model, seperti kasus klasifikasi biner, CM sangat berperan penting sebagai metrik evaluasi. Metrik ini menyediakan banyak sekali informasi yang mendetil dari perbandingan hasil aktual dengan hasil prediksi model. Sebagai hasil CM, bisa dilihat pada Gambar 3.11

Umumnya, performa metrik terbagi menjadi akurasi, presisi, recall, spesifitas, dan F1-score.

1. Akurasi

   Akurasi didefinisikan sebagai proporsi atau rasio dari prediksi (baik

positif maupun negatif) yang benar terhadap semua prediksi.

$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.4}$$

2. Presisi

   Presisi bisa dikatakan sebagai diantara semua prediksi yang positif, berapa yang hasilnya benar-benar positif.

$$Presisi = \frac{TP}{TP + FP} \tag{3.5}$$

3. Recall

   Recall atau Sensitivitas atau *True Positive Rate* bisa dikatakan dengan seberapa banyak data yang aktual positif, berapa banyak data yang benar diprediksi oleh model.

$$Recall = \frac{TP}{TP + FN} \tag{3.6}$$

4. Spesifitas

   Spesifitas atau *True Negative Rate* bisa dikatakan dengan seberapa banyak data yang aktual negatif, berapa banyak data yang benar diprediksi negatif oleh model.

$$Spesifitas = \frac{TN}{TN + FP} \tag{3.7}$$

5. F1-Score

   F1-score adalah metrik yang digunakan untuk menghitung 'harmoni' dari presisi dan recall.

$$F1 - score = 2 \cdot \frac{Presisi \cdot Recall}{Presisi + Recall} \tag{3.8}$$

3.3.8 Visualisasi hasil prediksi

Seringkali, CNN dianggap sebagai *black box* yang sangat tidak ditekankan terutama dalam penerapannya di dunia kesehatan. Dengan adanya visualisasi, membantu orang dalam memahami kapan dan mengapa model menghasilkan suatu prediksi. Selain itu, kita juga bisa mengetahui fitur mana yang relevan terhadap output.

Ada beberapa bentuk visualisasi yang sangat membantu, yaitu *confusion matrix*, sampel prediksi model, fitur map, t-SNE, dan Principal Component Analysis (PCA).

3.3.9 Kode PyTorch untuk CNN

Kode implementasi bab ini tersedia di sini https://s.id/week3DL.

3.4 Keterbatasan CNN

Walaupun CNN sangat kuat dalam memproses data citra dan banyak digunakan di berbagai aplikasi dunia nyata, CNN juga memiliki beberapa keterbatasan yang perlu kita ketahui:

1. Memerlukan Data yang Besar
   CNN biasanya bekerja dengan baik jika memiliki banyak data pelatihan (Alzubaidi et al., 2021). Jika data yang digunakan terlalu sedikit, model ini cenderung mengalami *overfitting*, yaitu model terlalu hafal data latih namun tidak bisa generalisasi ke data baru.

2. Boros Sumber Daya Komputasi
   CNN memerlukan GPU atau hardware yang kuat untuk melatih model, apalagi jika arsitektur yang digunakan sangat dalam (deep) dan dataset berukuran besar. Tanpa dukungan komputasi yang baik, pelatihan bisa memakan waktu yang lama.

3. Kurang Efektif untuk Data Non-Visual
   CNN dirancang khusus untuk data spasial seperti gambar atau video. Untuk data yang berbentuk teks berurutan atau data tabular, CNN kurang cocok dan biasanya digantikan oleh model seperti RNN, LSTM, atau Transformer (Otter, Medina, & Kalita, 2021; Yin, Kann, Yu, & Schütze, 2017).

4. CNN khususnya *supervised learning* membutuhkan data anotasi atau pelabelan yang lama terhadap output
   CNN untuk *supervised learning* pada kasus klasifikasi memerlukan pelabelan dataset. Apabila datanya besar, maka memerlukan waktu yang lebih lama untuk anotasi ini (Afham et al., 2022; Putra, Ogata, Yuniarno, & Purnomo, 2025).

Pada bab ini, kita telah mempelajari bagaimana CNN menjadi salah satu model yang efektif dalam *supervised learning* untuk kasus citra. CNN memiliki keunggulan dalam mengekstraksi fitur spasial dan mampu mengenali pola pada citra. Kita juga telah mempelajari komponen utama dari CNN, proses pelatihan, evaluasi, serta pentingnya visualisasi prediksi model untuk memahami kinerja CNN.

Meskipun CNN memiliki beberapa keunggulan, model ini juga memiliki beberapa keterbatasan. Pada bab berikutnya, kita akan membahas tentang optimasi model, teknik regularisasi untuk meningkatkan performa CNN, *Transfer Learning* yang dapat membantu ketika data yang tersedia terbatas, dan *Self-Supervised Learning* untuk menanggulangi masalah data anotasi manual.

3.5 Asesmen

3.5.1 Sub-CPMK yang dicapai

- Memahami konsep supervised learning dalam deep learning
- Membangun dan melatih model supervised learning seperti CNN dan RNN

3.5.2 Judul Tugas

Analisis dan Implementasi CNN untuk Klasifikasi Citra Sederhana

3.5.3 Deskripsi Tugas

Anda diminta untuk merancang, membangun, dan melakukan evaluasi model CNN sederhana untuk tugas klasifikasi citra. Dataset yang digunakan dapat berupa dataset standar seperti MNIST atau CIFAR-10, atau dataset lain yang setara.

3.5.4 Lingkup Pekerjaan

1. Desain dan Implementasi Model
   Bangun arsitektur CNN Anda sendiri (minimal memiliki ConvLayer, PoolingLayer, dan FCLayer). Jelaskan mengapa Anda memilih jumlah layer, kernel size, pooling type, dan activation function tertentu (gunakan argumen berdasarkan kebutuhan dataset).

2. Preprocessing dan Augmentasi
   Terapkan minimal dua teknik augmentasi data yang menurut Anda paling cocok untuk dataset ini. Berikan penjelasan kritis mengapa augmentasi tersebut Anda pilih dan bagaimana pengaruhnya terhadap model (diukur dari hasil training dan validasi).

3. Evaluasi dan Interpretasi
   Lakukan proses training, validasi, dan uji model. Visualisasikan hasil prediksi dengan confusion matrix. Hitung dan analisis metrik seperti akurasi, presisi, recall, dan F1-score. Analisis kritis: Jika model Anda menunjukkan nilai Recall tinggi tetapi Presisi rendah, atau sebaliknya, berikan evaluasi kritis terkait apa yang mungkin terjadi pada model dan dataset Anda.

4. Visualisasi Black Box CNN
   Ambil salah satu citra uji (test image) dan tunjukkan fitur map dari ConvLayer pertama dan terakhir (gunakan heatmap atau teknik visualisasi lainnya). Jelaskan apa yang CNN pelajari dari fitur map tersebut (misalnya: apakah CNN fokus pada tepi, bentuk, atau pola tertentu?).

5. Refleksi

   Tuliskan refleksi pribadi Anda: "Apa tantangan terbesar yang Anda alami dalam memahami dan membangun CNN dari awal?" "Jika Anda ingin meningkatkan model ini lebih jauh, strategi apa yang akan Anda lakukan?"

3.5.5 Output yang dikumpulkan

1. Source code CNN (format notebook atau .py)
2. Laporan proyek (maks. 8 halaman) berisi:

   (a) Desain dan justifikasi arsitektur

   (b) Hasil augmentasi dan alasan penggunaannya

   (c) Evaluasi metrik + confusion matrix

   (d) Visualisasi fitur map dan analisis

   (e) Refleksi dan rencana perbaikan