APLIKASI KECERDASAN BUATAN
UNTUK PERTANIAN MODERN

KLASIFIKASI PENYAKIT DAUN TANAMAN

INSTIPER press

Kuncahyo Setyo Nugroho, Mahmud Isnan, Teddy Suparyanto, Bens Pardamean

Aplikasi Kecerdasan Buatan untuk Pertanian Modern

Klasifikasi Penyakit Daun Tanaman

Kuncahyo Setyo Nugroho
Mahmud Isnan
Teddy Suparyanto
Bens Pardamean

INSTIPER press

Aplikasi Kecerdasan Buatan untuk Pertanian Modern

Klasifikasi Penyakit Daun Tanaman

Penulis
Kuncahyo Setyo Nugroho
Mahmud Isnan
Teddy Suparyanto
Bens Pardamean

Editor dan Layout
Kuncahyo Setyo Nugroho

Desain Sampul
Kuncahyo Setyo Nugroho

Jumlah Halaman
v + 62

ISBN
XXX-X-XX-XXXXXX-X
Cetakan Pertama, Maret 2025

Penerbit
INSTIPER PRESS (IKAPI & APPTI)
Jl. Nangka II, Maguwoharjo, Depok, Sleman, DI Yogyakarta

Hak Cipta © 2025 pada Penulis
Hak Cipta dilindungi undang-undang. Dilarang memper banyak atau memindahkan sebagian atau seluruh isi buku ini dalam bentuk apa pun, secara elektronis maupun mekanis, termasuk memfotokopi, merekam, atau dengan teknik perekaman lainnya, tanpa izin tertulis dari penulis.

Kata Pengantar

Pertanian modern menghadapi tantangan besar dalam mendeteksi dan mengelola penyakit tanaman yang dapat menyebabkan penurunan hasil panen. Kemajuan teknologi *artificial intelligence* (AI), khususnya *computer vision* dan *deep learning*, telah membawa solusi inovatif untuk mengatasi masalah ini. Dengan analisis data visual yang cepat dan akurat, sistem berbasis AI dapat mengenali pola penyakit lebih dini, memungkinkan intervensi yang lebih efektif.

Buku ini membahas penerapan *TensorFlow* dalam klasifikasi penyakit daun tanaman menggunakan teknik *deep learning*. Pembahasan dimulai dari dasar-dasar *computer vision*, pengelolaan dataset, hingga implementasi model *Convolutional Neural Network* dan metode *Transfer Learning* menggunakan VGG16. Dengan pendekatan sistematis, buku ini diharapkan menjadi referensi bagi peneliti, akademisi, dan praktisi pertanian dalam menerapkan teknologi AI untuk meningkatkan produktivitas pertanian.

Agar dapat mengikuti pembahasan dalam buku ini, diperlukan perangkat keras dan lunak berikut:

Software/Hardware	***Operating System***
Minimum 10 GB penyimpanan Minimum 8 GB RAM Prosesor Intel i5 atau lebih tinggi GPU NVIDIA 8+ GB atau lebih tinggi Python 3.7 TensorFlow 2.15.0	Windows, Linux, MacOs

Semua kode dalam buku ini dapat dijalankan di Google Colab melalui tautan berikut:

https://s.id/KnrJl

Dengan hadirnya buku ini, diharapkan pembaca dapat memahami konsep dan implementasi *deep learning* dalam pertanian secara praktis. Buku ini ditujukan bagi peneliti, akademisi, serta praktisi yang tertarik menerapkan kecerdasan buatan dalam sistem pertanian cerdas. Semoga buku ini dapat menjadi referensi yang bermanfaat dalam upaya meningkatkan produktivitas dan ketahanan pangan melalui inovasi teknologi.

Jakarta, Maret 2025

Penulis

Daftar Isi

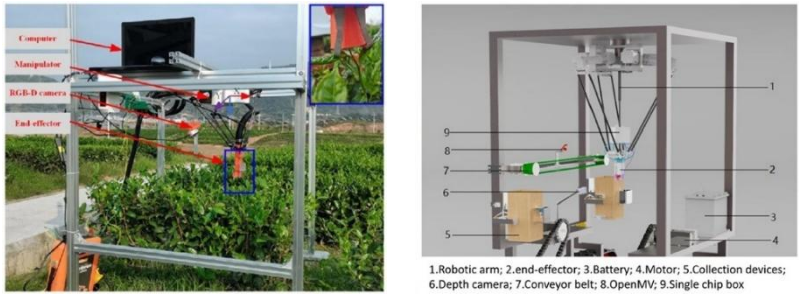
Bab 1: Tantangan Diagnosis Penyakit Tanaman pada Pertanian Modern

Deteksi dini penyakit tanaman sangat penting untuk mengurangi kehilangan hasil panen dan memastikan ketahanan pangan global. Penyakit patogen dapat menyebabkan penurunan produksi hingga 40% setiap tahun pada tanaman bernilai ekonomi tinggi (Venbrux et al., 2023). Deteksi dini memungkinkan intervensi tepat waktu, menghambat penyebaran penyakit, serta mengurangi dampak negatif terhadap ekosistem pertanian (Mahlein, 2016). Teknologi modern berbasis *Artificial intelligence* (AI) menawarkan solusi dengan menganalisis pola visual tanaman secara otomatis untuk diagnosis lebih cepat dan akurat.

Metode konvensional seperti inspeksi visual memiliki keterbatasan karena bergantung pada keahlian manusia, membutuhkan tenaga kerja dalam jumlah besar dan sering kali hanya efektif pada tahap lanjut penyakit. Inovasi dalam sensor kamera dan algoritma *deep learning* telah meningkatkan efisiensi deteksi penyakit tanaman dengan mengidentifikasi pola yang tidak dapat dikenali oleh mata manusia (Jiang et al., 2019). Teknologi ini memungkinkan pemantauan kesehatan tanaman secara *real-time*, memberikan data yang lebih akurat dalam pengambilan keputusan dan mengurangi ketergantungan pada bahan kimia pertanian.

Namun, Implementasi teknologi ini masih menghadapi tantangan, seperti variasi kondisi lingkungan, kualitas gambar yang tidak seragam, serta kebutuhan dataset besar untuk melatih *model deep learning* (Kamilaris & Prenafeta-Boldú, 2018). Faktor-faktor ini dapat memengaruhi akurasi sistem dalam mengidentifikasi penyakit secara konsisten di berbagai kondisi nyata lingkungan pertanian. Oleh karena itu, penelitian lebih lanjut diperlukan untuk meningkatkan skalabilitas dan ketahanan sistem deteksi berbasis *computer vision*, sehingga dapat diterapkan secara luas dalam berbagai skala pertanian dengan efektivitas tinggi.

1.1 Peran AI dalam Mendukung Pertanian Modern



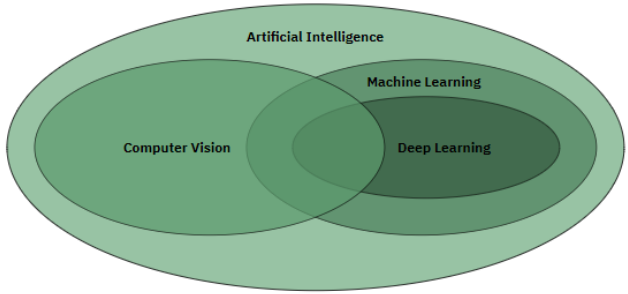
Gambar 1. Penerapan AI dalam pertanian modern. Gambar kiri menunjukkan sistem deteksi penyakit berbasis computer vision, sementara gambar kanan menampilkan sistem pemanenan otomatis dengan lengan robotik dan mekanisme konveyor.

Artificial intelligence (AI) telah membawa transformasi signifikan dalam pertanian modern dengan meningkatkan efisiensi produksi dan optimalisasi sumber daya. Teknologi AI memungkinkan otomatisasi berbagai proses, mulai dari pemantauan pertumbuhan tanaman, analisis kondisi tanah, hingga prediksi hasil panen berdasarkan pola cuaca dan data agronomi (Gupta et al., 2024). Dengan kemampuan menganalisis data dalam jumlah besar secara cepat, AI dapat membantu manusia dalam mengambil keputusan yang lebih tepat, mengurangi pemborosan sumber daya, serta meningkatkan produktivitas pertanian secara keseluruhan.

Gambar sebelumnya menggambarkan penerapan AI dalam pertanian modern, khususnya dalam pemantauan kesehatan tanaman dan pemanenan otomatis (Wang et al., 2023). Gambar sebelah kiri menunjukkan sistem deteksi penyakit berbasis *computer vision* yang menggunakan kamera untuk menganalisis kondisi kesehatan tanaman. Sementara itu, gambar sebelah kanan menampilkan sistem pemanenan otomatis yang dilengkapi dengan lengan robotik, kamera, serta mekanisme konveyor. Teknologi berbasis AI ini meningkatkan produktivitas dengan memastikan identifikasi masalah tanaman secara tepat waktu serta memaksimalkan efisiensi dalam proses panen, menunjukkan bagaimana AI berperan dalam mendukung pertanian modern.

1.2 Mengenal AI, *Machine Learning*, dan *Deep Learning*

Artificial intelligence (AI), *machine learning*, dan *deep learning* adalah tiga konsep yang saling terkait dalam pengembangan teknologi cerdas (Russell & Norvig, 2022). Seperti yang ditunjukkan pada diagram di bawah ini, AI mencakup seluruh upaya untuk menciptakan sistem yang dapat meniru kecerdasan manusia. *Machine learning*, sebagai bagian dari AI, menggunakan algoritma dengan pendekatan berbasis data untuk mengembangkan kemampuan belajar dari pengalaman. Sedangkan *deep Learning* merupakan bagian dari *machine learning* untuk mempelajari pola data yang lebih kompleks.



Gambar 2. Hubungan antara artificial intelligence, machine learning, dan deep learning, serta kaitannya dengan computer vision.

Dalam konteks *computer vision*, *deep learning* telah merevolusi cara komputer memahami dan memproses data visual. Misalnya, tugas seperti klasifikasi dan deteksi objek, serta segmentasi gambar menjadi lebih akurat dan efisien. Penggunaan *artificial neural network* (ANN) memungkinkan pengembangan teknologi mutakhir seperti pengenalan wajah, kendaraan otonom, *augmented reality*, dan sistem keamanan berbasis visual.

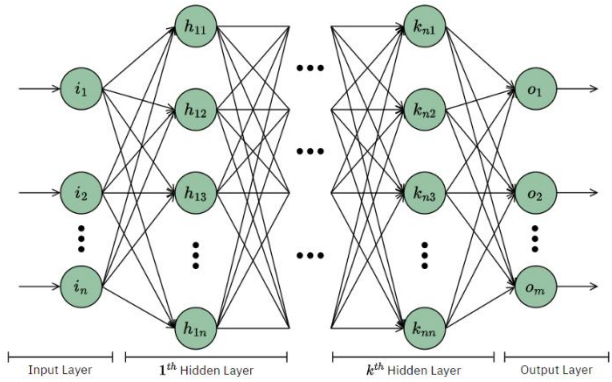
Artificial Neural Network

Artificial Neural Network (ANN) adalah algoritma yang terinspirasi dari cara kerja otak manusia dalam memproses informasi. ANN terdiri dari sekumpulan unit yang disebut *neuron* yang saling terhubung dan bekerja untuk mengenali pola dalam data. ANN menerima data *input*, memprosesnya melalui beberapa *layer*, lalu menghasilkan *output*. Arsitektur ANN dapat disesuaikan dengan berbagai jenis data, baik data terstruktur seperti tabel maupun data tidak terstruktur seperti gambar, teks, atau audio.

Struktur dasar ANN terdiri dari tiga *layer* utama, yaitu:

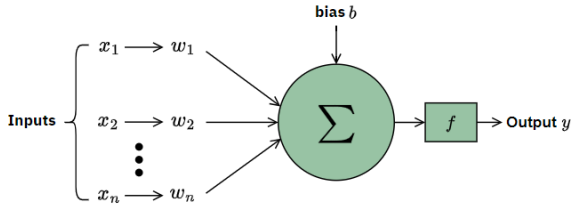
- ***Input layer***. Menerima data *input* sebelum diteruskan ke *layer* berikutnya.
- ***Hidden layer***. Menghubungkan *input layer* dan *output layer*, mengolah data untuk mengenali pola yang lebih kompleks. Jumlah *hidden layer* dapat bervariasi tergantung pada kompleksitas tugas yang dikerjakan.
- ***Output layer***. Menghasilkan prediksi berdasarkan data *input* yang telah diproses *hidden layer*.

Gambar di bawah ini menunjukkan struktur dasar ANN, yang terdiri dari *input layer*, *hidden layers*, dan *output layer*. Jumlah *neuron* pada *output layer* bergantung pada jenis tugas yang diselesaikan. Untuk tugas regresi, hanya ada satu *neuron* pada *output layer*, sedangkan untuk tugas klasifikasi dengan banyak kelas, jumlah *neuron* pada *output layer* akan sesuai dengan jumlah kelas yang diprediksi.



Gambar 3: Struktur dasar arsitektur artificial neural network yang terdiri dari input layer, hidden layer, dan output layer.

Gambar di bawah ini menunjukkan bagaimana sebuah *neuron* mengubah *input* menjadi *output* melalui perhitungan berbasis bobot dan fungsi aktivasi.



Gambar 4. Struktur dasar neuron yang menunjukkan proses penghitungan dari input hingga output menggunakan fungsi aktivasi.

Neuron menghitung output y sebagai berikut:

$$y = f(b + \sum_{i=1}^{n} w_i \cdot x_1)$$

Di mana:

- $x_1, x_2, \ldots, x_n$ adalah variabel *input*.
- b adalah bias.
- $w, w_2, \ldots, w_n$ adalah bobot yang diterapkan pada setiap variabel *input*.
- f adalah fungsi aktivasi yang menambahkan non-linearitas. Fungsi aktivasi memungkinkan neural network menangani hubungan kompleks antara input dan output, sehingga dapat mempelajari pola yang lebih rumit.

Pada tingkat tinggi, ANN terdiri dari banyak *neuron* yang saling terhubung, di mana setiap *neuron* memiliki bobot yang diperbarui selama pelatihan. ANN dapat memiliki arsitektur sederhana atau kompleks tergantung pada jumlah *hidden layer* yang digunakan. Jika jumlah *hidden layer* lebih banyak, maka ANN disebut *deep learning*, yang dirancang untuk menangani tugas kompleks seperti klasifikasi dan deteksi objek.

Bab 2: *Computer Vision* dan TensorFlow untuk Diagnosis Penyakit Tanaman

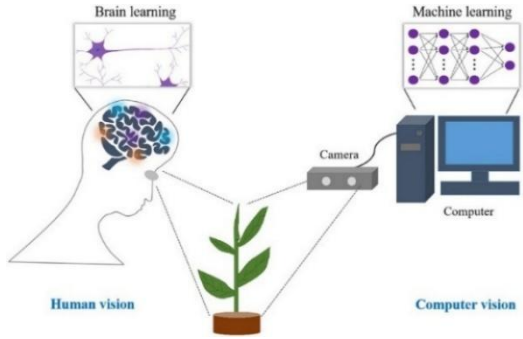
Computer Vision adalah bagian dari AI yang memungkinkan komputer memahami informasi visual dari gambar atau video secara otomatis. Teknologi *computer vision* mengandalkan model *deep learning* untuk mengenali pola, mendeteksi objek, dan menganalisis struktur visual dalam data citra (Voulodimos et al., 2018). Dalam bidang pertanian, *computer vision* memainkan peran penting dalam diagnosis penyakit tanaman dengan mengidentifikasi perubahan warna, bentuk, dan tekstur pada daun, batang, atau buah yang terinfeksi. Sistem berbasis *computer vision* dapat menggantikan proses manual oleh manusia dalam pemantauan kesehatan tanaman, sehingga dapat meningkatkan akurasi, serta mempercepat pengambilan keputusan dalam tindakan pencegahan dan pengobatan.

2.1 Perbedaan Penglihatan Manusia dan *Computer Vison*

Manusia mengenali objek melalui sistem penglihatan yang mengandalkan pengalaman dan intuisi. Cahaya yang dipantulkan dari suatu objek diterima oleh mata, lalu diinterpretasikan oleh otak berdasarkan pola yang telah dipelajari. Proses ini memungkinkan manusia mengenali objek dengan cepat, bahkan dalam kondisi pencahayaan atau bentuk yang tidak sempurna.

Sebaliknya, *computer vision* memungkinkan komputer menganalisis gambar digital menggunakan *machine learning* untuk memproses data visual dengan mendeteksi pola, bentuk, dan warna dalam gambar. Dalam klasifikasi penyakit tanaman, pengamatan manusia bisa bervariasi tergantung pengalaman, sedangkan *computer vision* memberikan hasil yang lebih objektif dan konsisten dengan menganalisis karakteristik visual tanaman secara otomatis.

Gambar di bawah ini menggambarkan perbedaan antara penglihatan manusia dan *computer vision* dalam memahami dan menganalisis objek.



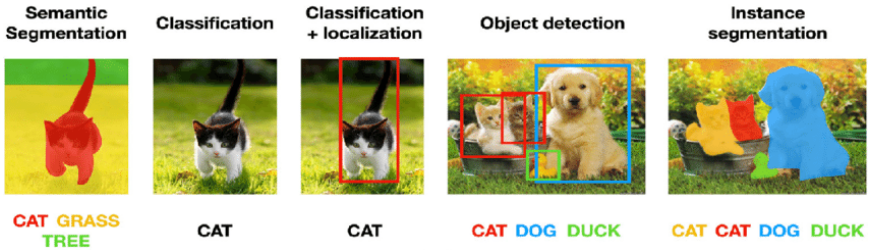
Gambar 5. lustrasi perbedaan antara penglihatan manusia dan computer vision. Manusia mengenali objek berdasarkan pengalaman dan intuisi menggunakan otak, sedangkan computer vision menggunakan model deep learning untuk memproses gambar digital.

Tugas Utama pada *Computer Vision*

Tugas utama *computer vision* dalam pengolahan data visual melibatkan beberapa aspek penting yang mendukung identifikasi dan analisis citra secara otomatis, meliputi:

1. **Klasifikasi objek**. Menentukan kategori suatu objek dalam gambar berdasarkan analisis fitur visual. Model *deep learning* dilatih untuk mengenali apakah suatu gambar termasuk dalam kelas tertentu, seperti mengklasifikasikan daun sebagai sehat atau terinfeksi penyakit.
2. **Deteksi objek**. Mengidentifikasi lokasi spesifik suatu objek dalam gambar. Model tidak hanya mengenali kategori, tetapi juga memberikan *bounding box* pada area yang relevan. Contohnya, menemukan bercak penyakit tertentu pada daun dan menandai lokasinya dalam gambar.
3. **Segmentasi semantik**. Memetakan setiap piksel dalam gambar sesuai dengan kelas tertentu, memungkinkan pemisahan bagian objek dengan lebih rinci. Misalnya, membedakan area daun sehat dan area yang terinfeksi berdasarkan perbedaan warna dan pola tekstur.
4. **Segmentasi *instance***. Mengidentifikasi beberapa objek sejenis dalam satu gambar dan membedakan setiap *instance*. Misalnya, mengenali beberapa jenis penyakit tanaman dalam satu gambar, meskipun bentuknya serupa.

Gambar di bawah ini mengilustrasikan tugas utama dalam *computer vision*:



Gambar 6. Ilustrasi tugas utama dalam computer vision: segmentasi semantik, klasifikasi dan lokalisasi, deteksi objek, serta segmentasi instance.

2.2 Mengenal TensorFlow dan Tensor

TensorFlow[1] adalah *framework open-source* yang dikembangkan oleh Google untuk membangun dan melatih model *deep learning*. TensorFlow menyediakan berbagai fungsi tingkat tinggi untuk mempermudah pengembangan model *deep learning*. Selain itu, TensorFlow mendukung akselerasi melalui perangkat keras seperti

[1] https://www.tensorflow.org

Graphics Processing Unit (GPU) dan *Tensor Processing Unit* (TPU)[2], memungkinkan pelatihan model *deep learning* lebih cepat dan efisien.

Instalasi TensorFlow

Sebelum menggunakan TensorFlow, pastikan Python telah terpasang di sistem dengan versi yang kompatibel, seperti Python 3.8 atau lebih baru. Perintah berikut digunakan untuk memeriksa versi Python yang tepasang di sistem:

```
1 python -V
```

TensorFlow dapat diinstal menggunakan berbagai metode, tergantung pada sistem operasi dan kebutuhan perangkat keras yang digunakan. Untuk instalasi pada CPU, perintah berikut dapat digunakan:

```
1 pip install tensorflow
```

Jika ingin memanfaatkan akselerasi GPU dengan CUDA[3] dan cuDNN[4], maka instalasi dilakukan dengan perintah berikut:

```
1 pip install tensorflow-gpu
```

Alternatif lainnya adalah menggunakan Conda[5], yaitu dengan perintah:

```
1 conda install -c conda-forge tensorflow
```

Sementara itu, untuk instalasi untuk perangkat GPU dengan CUDA 12.1, perintah berikut dapat digunakan:

```
1 conda install -c conda-forge tensorflow tensorflow-gpu
  cudatoolkit=12.1 dapat digunakan.
```

Setelah instalasi selesai, buka terminal Python atau Jupyter Notebook, lalu jalankan kode berikut untuk memverifikasi apakah TensorFlow telah terinstal:

```
1 import tensorflow as tf
2 print(tf.__version__)
```

Jika instalasi berhasil, versi TensorFlow yang terinstal akan ditampilkan.

Tipe Data Tensor

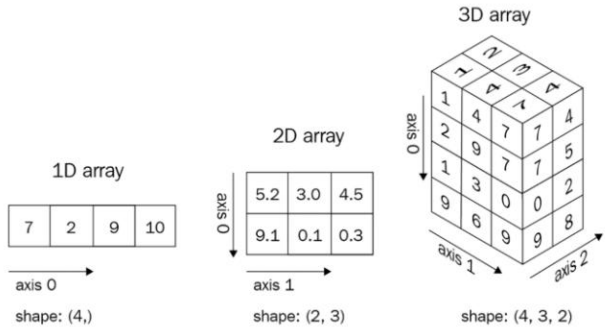
Tensor adalah struktur data dasar pada TensorFlow untuk merepresentasikan data dalam bentuk *array* multidimensi. Tensor mirip dengan *array* pada NumPy, tetapi lebih fleksibel karena dapat berjalan di berbagai perangkat seperti CPU, GPU, maupun TPU. Tensor digunakan untuk menyimpan dan memanipulasi data selama proses komputasi dalam pengembangan model *deep learning*.

[2] https://cloud.google.com/tpu/docs/intro-to-tpu
[3] https://developer.nvidia.com/cuda-toolkit
[4] https://developer.nvidia.com/cudnn
[5] https://anaconda.org

Tensor dapat dikategorikan berdasarkan dimensinya:

1. **Skalar (0-D tensor).** Nilai tunggal tanpa dimensi, seperti angka 5 atau 3.14.
2. **Vektor (1-D tensor).** Kumpulan skalar dalam satu dimensi, misalnya $[1, 2, 3]$.
3. **Matriks (2-D tensor).** *Array* dua dimensi yang sering digunakan dalam perhitungan linear, seperti matriks $[\,[1, 2], [3, 4]\,]$.
4. **Tensor multi-dimensi (n-D tensor):** Struktur data dengan lebih dari dua dimensi, sering digunakan dalam pemrosesan gambar dan data kompleks lainnya.

Secara visual, perbedaan antara *array*, matriks, dan tensor dapat dilihat pada gambar berikut:



Gambar 7. Ilustrasi perbedaan antara array satu dimensi (1D), matriks dua dimensi (2D), dan tensor tiga dimensi (3D).

Misalnya, gambar berwarna dapat direpresentasikan sebagai tensor tiga dimensi dengan struktur $tinggi \times lebar \times 3\ channel$, di mana tiga *channel* mewakili warna merah, hijau, dan biru (RGB). Sementara itu, gambar skala abu-abu (*grayscale*) hanya memiliki satu *channel* warna, sehingga direpresentasikan sebagai tensor dua dimensi dengan struktur $tinggi \times lebar$.

Bab 3: Persiapan Dataset untuk Klasifikasi Penyakit Tanaman

Dalam pengembangan model *deep learning*, dataset memiliki peran yang sangat penting. Model tidak dapat belajar dengan baik tanpa dataset yang representatif. Prinsip *Garbage In, Garbage Out* (GIGO) berlaku: jika dataset memuat data yang tidak relevan atau berkualitas buruk, model yang dihasilkan juga tidak akan akurat, meskipun menggunakan model yang kompleks (Mohammed et al., 2024). Oleh karena itu, pengumpulan dan pengelolaan dataset yang tepat menjadi kunci utama dalam membangun model *deep learning* yang andal.

Dataset dalam *computer vision* untuk klasifikasi, deteksi, maupun segmentasi penyakit tanaman terdiri dari gambar yang menunjukkan berbagai kondisi, baik tanaman sehat maupun yang terinfeksi penyakit tertentu. Agar model dapat mengenali pola dengan akurat, dataset harus mencakup berbagai kondisi dunia nyata, seperti pencahayaan, sudut pengambilan gambar, dan variasi lingkungan lainnya.

Identifikasi Kebutuhan Data

Sebelum mengumpulkan data, penting untuk menentukan kebutuhan dataset agar sesuai dengan tujuan pengembangan model.

- **Tujuan pengumpulan data**. Tentukan tugas yang akan diselesaikan, apakah untuk klasifikasi, deteksi, atau segmentasi. Selanjutnya, tentukan dataset digunakan untuk mendeteksi penyakit tertentu atau meningkatkan ketahanan model terhadap berbagai kondisi lingkungan.
- **Spesifikasi data**. Data harus mencakup variasi penyakit, kondisi cahaya, dan jenis tanaman untuk meningkatkan kemampuan generalisasi model.
- **Frekuensi dan kondisi pengumpulan data**. Dataset harus mencerminkan kondisi nyata agar dapat digunakan dalam berbagai skenario. Pengambilan gambar berkala (misalnya setiap minggu) membantu memantau pertumbuhan tanaman. Variasi waktu (pagi, siang, malam) dan musim (hujan, kemarau) juga penting untuk memastikan model dapat mengenali pola dengan lebih baik.

Kualitas Data

Kualitas dataset menentukan seberapa baik model mengenali pola dalam gambar. Gambar harus memiliki detail yang jelas, pencahayaan yang tepat, dan minim *noise* agar model dapat belajar dengan optimal.

- **Resolusi citra**: Gambar beresolusi tinggi menangkap detail kecil, seperti bintik pada daun atau serangga pada tanaman, yang penting untuk identifikasi penyakit. Resolusi rendah dapat menyebabkan hilangnya detail dan menurunkan akurasi model.

- **Kebersihan data**: Hindari *noise* seperti gambar yang terlalu gelap atau terang. Pastikan pencahayaan terkendali untuk membantu model mengenali fitur yang relevan secara konsisten.

Keragaman Data

Dataset yang bervariasi membantu model mengenali objek dalam berbagai kondisi dan mengurangi risiko bias.

- **Sudut pandang**. Ambil gambar dari berbagai sudut (atas, samping, dekat, jauh) agar model dapat memahami bentuk objek secara lebih luas.
- **Variasi lingkungan**. Ambil gambar dari kondisi berbeda, seperti pencahayaan langsung, bayangan, dalam rumah kaca, dan luar ruangan, untuk memastikan model bekerja dalam berbagai skenario.
- **Distribusi data**. Pastikan dataset mencakup berbagai usia tanaman, ukuran daun, dan kondisi kesehatan agar model tidak bias terhadap satu kelompok data tertentu.

Manajemen Dataset

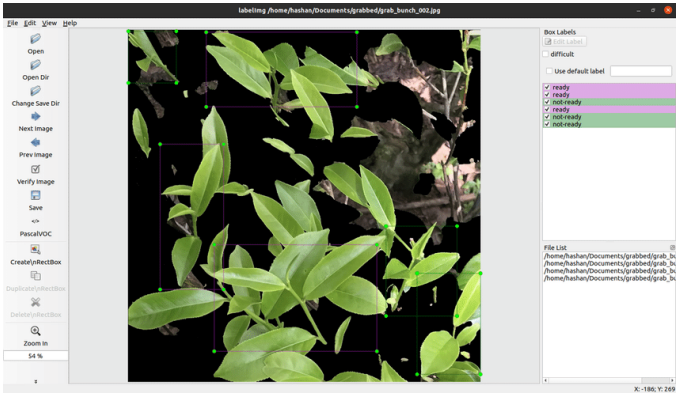
Dataset yang terorganisir dengan baik memastikan model belajar secara optimal dan menghindari bias. Pengelolaan dataset melibatkan pencatatan metadata, pemantauan perubahan data, keseimbangan kelas, dan konsistensi dalam proses pengolahan.

- **Manajemen metadata**. Simpan informasi penting seperti lokasi pengambilan gambar, waktu, kondisi cuaca, dan jenis kamera. Metadata membantu analisis lanjutan dan menjaga relevansi data.
- ***Versioning***. Gunakan *version control* untuk melacak perubahan dataset selama pengembangan model. Dengan *versioning*, setiap modifikasi terdokumentasi dan dapat dibandingkan untuk menjaga konsistensi.
- **Keseimbangan data**. Pastikan jumlah data di setiap kelas seimbang agar model tidak lebih akurat pada satu kategori atau kelas saja. Dataset yang tidak seimbang menyulitkan model dalam mengenali kelas dengan sampel lebih sedikit.
- **Konsistensi**. Data harus dikumpulkan dan diproses secara stabil dan berkelanjutan agar model dapat mengenali pola dengan lebih baik.

Anotasi Data

Anotasi data adalah proses memberi label pada gambar agar model dapat mengenali objek yang relevan. Label yang akurat sangat penting untuk memastikan model belajar dengan benar dan menghasilkan prediksi yang tepat.

- **Anotasi manual**. Dilakukan secara langsung oleh manusia menggunakan *software* seperti LabelImg[6], Label Studio[7] atau VGG Image Annotator[8]. Proses ini memerlukan ketelitian karena objek yang dilabeli harus sesuai dengan kategori atau kelas yang ditentukan. Kesalahan dalam anotasi dapat menyebabkan model belajar dari data yang salah (prinsip GIGO). Oleh karena itu, anotasi sebaiknya dilakukan oleh pakar atau individu yang memahami karakteristik data.
- **Anotasi kolaboratif**. Anotasi sebaiknya dilakukan secara tim untuk mempercepat proses anotasi. Metode ini menerapkan validasi silang guna memastikan label tetap konsisten. Dengan metode kolaboratif, kesalahan anotasi dapat diminimalkan, sehingga dataset yang dihasilkan lebih berkualitas untuk pengembangan model.



Gambar 8. Tampilan software LabelImg untuk melakukan anotasi data.

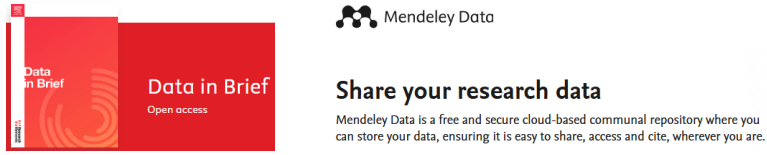
Benchmarking

Evaluasi dataset sangat penting untuk memastikan bahwa data yang digunakan cukup representatif, berkualitas, dan sesuai untuk pelatihan model. Proses *benchmarking* membantu dalam menilai seberapa baik dataset sebelum digunakan dalam model pembelajaran mesin.

- **Evaluasi kualitas dataset.** Tetapkan standar evaluasi untuk memastikan dataset cukup representatif dan berkualitas. Dataset yang baik mencakup berbagai variasi kondisi agar model bekerja dalam berbagai situasi.
- **Menggunakan dataset publik sebagai acuan**. Bandingkan dataset yang telah dikumpulkan dengan referensi dari sumber terpercaya seperti publikasi ilmiah, GitHub, atau Kaggle untuk memastikan kualitas dan relevansi data.

[6] https://github.com/HumanSignal/labelImg
[7] https://github.com/HumanSignal/label-studio
[8] https://www.robots.ox.ac.uk/~vgg/software/via

- **Uji dataset dengan model *pretrained* dari dataset publik**. Gunakan model *pretrained* untuk mengukur kualitas dataset dengan metrik evaluasi standar, seperti akurasi, *mean Average Precision* (mAP) atau *Intersection over Union* (IoU).



Gambar 9. Contoh platform dan sumber data publik yang digunakan dalam benchmarking dataset. Data in Brief (kiri) menyediakan publikasi dataset terbuka, sementara Mendeley Data (kanan) memungkinkan penyimpanan dan berbagi dataset untuk penelitian.

3.1 Dataset Diagnosis Penyakit Tanaman

Seperti yang telah dijelaskan sebelumnya, dataset memiliki peran penting dalam pengembangan model *deep learning*. Namun, mengumpulkan data sendiri memerlukan ketelitian dan waktu yang tidak sedikit. Oleh karena itu, untuk membangun model awal untuk klasifikasi dan deteksi penyakit tanaman, dapat digunakan dataset publik yang telah tersedia.

Dataset publik umumnya berasal dari penelitian terdahulu dan telah diproses agar siap digunakan dalam berbagai tugas *computer vision*. Beberapa dataset mencakup berbagai jenis tanaman dan penyakit dengan jumlah sampel yang cukup besar, memungkinkan model belajar dari pola yang beragam.

Tabel di bawah ini menyajikan contoh dataset publik yang digunakan untuk diagnosis penyakit tanaman.

Tabel 1. Contoh dataset publik untuk diagnosis penyakit tanaman.

Nama	Tugas	Jumlah	Deskripsi
IDBGL: *Black Gram Leaf Disease Dataset*	Klasifikasi	4.038	Koleksi gambar penyakit daun kacang kitam (*Vigna mungo*) yang terdiri dari 5 kelas: *healthy*, *cercospora leaf spot*, *insect*, *leaf crinkle*, and *yellow mosaic*. (Shoib et al., 2025)
BDPapayaLeaf	Klasifikasi, Deteksi	2.159	Koleksi gambar penyakit daun pepaya yang terdiri dari 5 kelas: *healthy*, *anthracnose*, *bacterial spot*, *curl*, and *ring spot*. (Mustofa et al., 2024)
RoCoLe: *A robusta coffee leaf images dataset*	Klasifikasi	1.560	Koleksi gambar penyakit daun kopi robusta yang terdiri dari 6 kelas: *healthy*, *red spider mite*, *rust level* 1, *rust level* 2, *rust level* 3 dan *rust level* 4. (Parraga-Alava et al., 2019)

Nama	Tugas	Jumlah	Deskripsi
Bananas Dataset Tanzania	Klasifikasi, Deteksi, Segmentasi	16.092	Koleksi gambar penyakit daun dan batang pisan yang terdiri dari 3 kelas: *healthy*, *black sigatoka*, dan *fusarium wilt race* 1. (Mduma & Leo, 2023)
IDDMSLD: *Malabar spinach leaf diseases*	Klasifikasi	3.006	Koleksi gambar penyakit daun bayam (Basella alba) yang terdiri dari 5 kelas: *healthy*, *anthracnose*, *bacterial spot*, *downy mildew*, dan *pest damage*. (Sayeem et al., 2025)

3.2 Menggunakan Dataset PlantVillage

Sebagai contoh implementasi klasifikasi penyakit daun tanaman, digunakan dataset publik PlantVillage (Hughes & Salathe, 2016). Dataset ini memiliki 54.305 gambar daun dari 14 jenis tanaman berbeda: apel, blueberry, ceri, jagung, anggur, jeruk, persik, paprika, kentang, raspberry, kedelai, labu, stroberi, dan tomat. Setiap gambar diklasifikasikan sebagai sehat atau terinfeksi penyakit tertentu.

Untuk mempermudah implementasi dan mengurangi beban komputasi, hanya tiga subset kelas dari tanaman tomat yang digunakan, yaitu *Healthy* (daun dalam kondisi sehat), *late blight* (daun yang terinfeksi hawar daun), dan *septoria leaf spot* (daun yang terkena *septoria leaf spot*).

Mengunduh Dataset PlantVillage dari Google Drive

Langkah pertama adalah mengunduh subset dataset PlantVillage dari Google Drive menggunakan *library* `gdown`.

```
#!pip install gdown

!gdown https://drive.google.com/uc?id
=1Gy3hRhjbiVblRYWsmkOq6XkTtXVpRM2T
```

Ekstraksi File Dataset

Setelah dataset berhasil diunduh, langkah berikutnya adalah mengekstrak file yang masih dalam format `.zip`. Proses ini dilakukan dengan perintah `unzip`, yang mengekstrak isi file ke dalam folder sementara. Kemudian, folder dataset utama dibuat dan semua file hasil ekstraksi dipindahkan ke dalamnya. Terakhir, folder sementara dihapus untuk membersihkan memori penyimpanan.

```
!unzip -q tomato-plantvillage-dataset.zip -d temp_extracted
!mkdir -p tomato-plantvillage-dataset
!mv temp_extracted/* tomato-plantvillage-dataset/
!rm -r temp_extracted

print("Done!")
```

3.3 Memuat dan Mengeksplorasi Dataset

Langkah berikutnya adalah memuat dataset dan mengeksplorasi dataset. Proses ini penting untuk memahami struktur dataset, jumlah sampel per kelas, serta karakteristik data sebelum melanjutkan ke tahap pelatihan model.

Mengimpor *Library* yang Dibutuhkan

```
 1  import os
 2  import time
 3  import random
 4  import numpy as np
 5  from collections import Counter
 6  import cv2
 7  from PIL import Image
 8  import matplotlib.pyplot as plt
 9  import seaborn as sns
10  import tensorflow as tf
11  from tensorflow.keras.preprocessing import
    image_dataset_from_directory
12  from tensorflow.keras import layers, models, optimizers,
    metrics, losses
13  from tensorflow.keras.applications import VGG16
14  from sklearn.metrics import classification_report,
    confusion_matrix
15
16  SEED = 69
17  tf.random.set_seed(SEED)
18  np.random.seed(SEED)
19  random.seed(SEED)
20
21  AUTOTUNE = tf.data.AUTOTUNE
```

Kode di atas mengimpor semua *library* yang dibutuhkan untuk memproses dataset, mengeksplorasi data, membangun model *deep learning* hingga evaluasi model. *Library* `os`, `time`, dan `random` digunakan untuk manipulasi file dan pengacakan data, sementara `NumPy` dan `Counter` membantu dalam komputasi numerik dan perhitungan distribusi kelas. Libary OpenCV (`cv2`) dan `PIL` menangani manipulasi gambar, sedangkan `matplotlib` dan `seaborn` digunakan untuk visualisasi data.

`TensorFlow` adalah *library* utama untuk memuat dataset gambar, membangun model, serta mengoptimalkan pelatihan dengan berbagai komponen seperti *optimizer* dan fungsi *loss*. Model `VGG16` disertakan untuk implementasi *transfer learning*. Sedangkan evaluasi model dilakukan dengan menggunakan `classification_report` dan `confusion_matrix` dari `sklearn.metrics`.

Untuk memastikan replikasi hasil, nilai `SEED` ditetapkan pada *library* `TensorFlow`, `NumPy`, dan `random`. Terakhir, `AUTOTUNE` pada `TensorFlow` digunakan untuk mengoptimalkan *pipeline* data agar pemrosesan data berjalan lebih efisien.

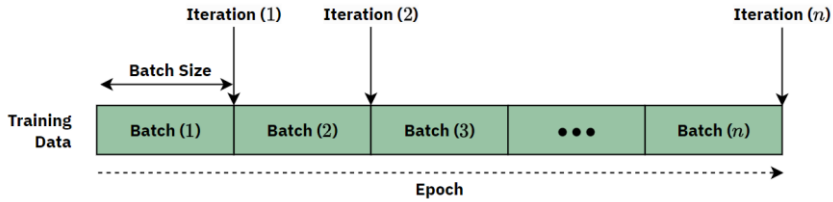
Memuat Dataset

Kode di atas digunakan untuk memuat dataset gambar langsung dari direktori menggunakan fungsi `image_dataset_from_directory()`[9] dari TensorFlow:

```
DATASET_DIR = "/content/tomato-plantvillage-dataset"
IMG_SIZE = (256, 256)
BATCH_SIZE = 32

dataset = tf.keras.utils.image_dataset_from_directory(
    DATASET_DIR,
    image_size=IMG_SIZE,
    batch_size=BATCH_SIZE,
    seed=SEED,
    shuffle=True
)
```

Variabel `DATASET_DIR` menentukan lokasi dataset, sedangkan `IMG_SIZE=(256, 256)` memastikan setiap gambar diubah ukurannya menjadi 256×256 piksel sebelum diproses. `BATCH_SIZE=32` mengatur jumlah gambar yang diproses dalam satu *batch* selama pelatihan. Parameter `shuffle=True` memastikan urutan gambar diacak untuk menghindari model belajar berdasarkan pola tertentu dalam dataset.

Batch size adalah parameter penting dalam pelatihan model karena menentukan jumlah sampel yang diproses sekaligus dalam satu iterasi. Gambar di bawah ini mengilustrasikan bagaimana data dibagi menjadi beberapa *batch* selama proses pelatihan:



Gambar 10. Ilustrasi batch size dalam pelatihan model. Data dibagi menjadi beberapa batch, dengan setiap iterasi memproses satu batch hingga seluruh epoch selesai.

Dalam konteks ini:

- *Batch* (n) adalah subset dari dataset yang digunakan dalam satu iterasi. Sebagai contoh, jika dataset memiliki 1.000 gambar dan *batch size* ditetapkan 50, maka terdapat sekitar 20 *batch* (1.000 / 50), dengan setiap *batch* berisi 50 gambar.
- *Iteration* (n) adalah proses di mana model mempelajari satu *batch* data dan memperbarui bobot berdasarkan informasi dari *batch* tersebut.
- *Epoch* adalah satu siklus penuh di mana seluruh dataset telah diproses oleh model. Jika terdapat n *batch*, maka satu *epoch* terdiri dari n iterasi.

[9] https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory

Dalam setiap iterasi pelatihan, model akan memperbarui bobot berdasarkan data dari satu *batch*, bukan dari seluruh dataset sekaligus. Setelah satu *batch* diproses, model menyesuaikan bobot sebelum melanjutkan ke *batch* berikutnya. Pendekatan ini memungkinkan model belajar lebih cepat karena bisa langsung memperbaiki kesalahan dari setiap batch yang telah diproses, tanpa harus menunggu semua data selesai dipelajari dalam satu epoch.

Batch size biasanya berada dalam rentang 32 hingga 1.024, tergantung pada ukuran model dan kapasitas komputasi. Nilai batch size sering dipilih dalam kelipatan dua (32, 64, 128) untuk efisiensi perhitungan. Jika model yang digunakan sangat besar atau komputer memiliki keterbatasan daya pemrosesan, *batch size* dapat dikurangi hingga 16 atau lebih kecil, agar tetap dapat berjalan tanpa menghabiskan terlalu banyak memori.

Kode di bawah ini digunakan untuk menampilkan jumlah kelas dalam dataset:

```
num_batches = len(dataset)
print(f"Total batches in dataset: {num_batches}")

# Total batches in dataset: 149
```

Kode di atas menghitung jumlah batch dengan `len(dataset)`, yang menunjukkan bahwa dataset memiliki 149 *batch* dengan `BATCH_SIZE=32`.

Sedangkan kode di bawah ini digunakan untuk menyimpan dan menampilkan kelas pada dataset:

```
CLASS_NAMES = [name.replace('Tomato___', '') for name in
dataset.class_names]
print(f"Found {len(CLASS_NAMES)} classes: {CLASS_NAMES}")

# Found 3 classes:
# ['healthy', 'late_blight', 'septoria_leaf_spot']
```

Kode di atas menunjukkan bahwa dataset memiliki tiga kelas, yaitu *healthy*, *late blight*, dan *septoria leaf spot*.

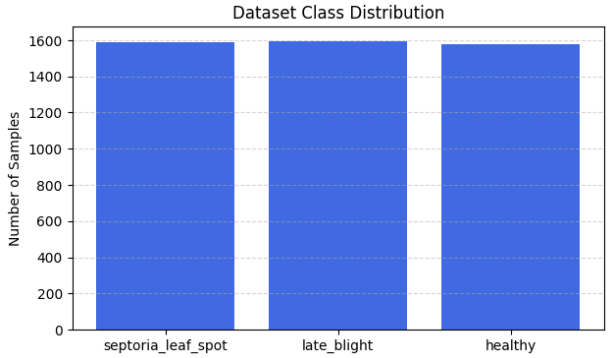
Menampilkan Distribusi Kelas pada Dataset

Distribusi kelas dalam dataset perlu dianalisis untuk memastikan bahwa data yang digunakan seimbang. Jika jumlah sampel di setiap kelas terlalu jauh berbeda, model dapat mengalami bias terhadap kelas dengan jumlah sampel lebih besar. Kode di bawah ini menghitung jumlah sampel dalam setiap kelas dan menampilkannya dalam bentuk diagram batang (*bar chart*):

```
class_counts = Counter()

for _, class_indices in dataset:
    class_counts.update(class_indices.numpy())

plt.figure(figsize=(7, 4))
plt.bar(class_counts.keys(), class_counts.values(),
color="royalblue")
```

```
plt.xticks(ticks=list(class_counts.keys()), labels=CLASS_NAMES)
plt.ylabel("Number of Samples")
plt.title("Dataset Class Distribution")
plt.grid(axis="y", linestyle="--", alpha=0.5)
plt.show()

print("Dataset Class Distribution:")
for class_index, count in sorted(class_counts.items()):
    print(f"{CLASS_NAMES[class_index]}: {count}")
```

Gambar di bawah ini menunjukkan bahwa dataset memiliki tiga kelas, yaitu *healthy*, *late blight*, dan *septoria leaf spot*, dengan jumlah sampel yang cukup seimbang.



Gambar 11. Distribusi kelas pada dataset.

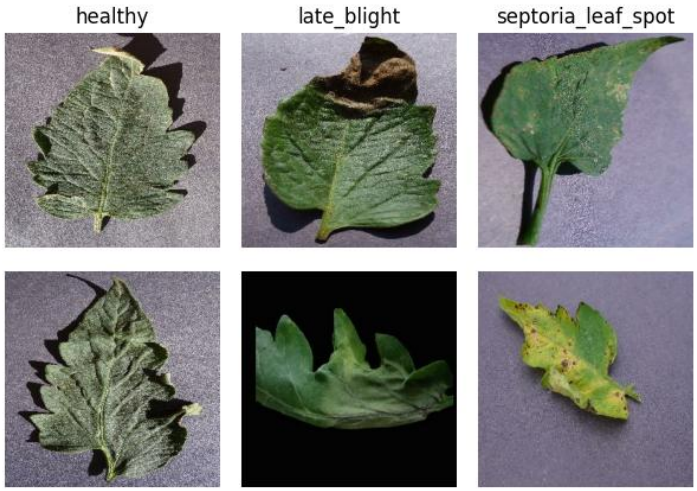
Menampilkan Sampel Gambar pada Dataset

Kode di bawah ini digunakan untuk menampilkan sampel gambar dari setiap kelas dalam dataset:

```
plt.figure(figsize=(len(CLASS_NAMES) * 2.5, 5))

for class_index, class_label in enumerate(CLASS_NAMES):
    images_per_class = [image for image, class_id in dataset.unbatch() if class_id.numpy() == class_index][:2]

    for image_position, image_data in enumerate(images_per_class):
        plt.subplot(2, len(CLASS_NAMES), class_index + 1 + image_position * len(CLASS_NAMES))
        plt.imshow(image_data.numpy().astype("uint8"))
        if image_position == 0:
            plt.title(class_label)
        plt.axis("off")

plt.subplots_adjust(hspace=0.1, wspace=0.1)
plt.show()
```

Gambar di bawah ini menunjukkan bahwa dataset terdiri dari tiga kelas utama: *healthy*, *late blight*, dan *septoria leaf spot*. Setiap kelas direpresentasikan dengan dua contoh gambar daun yang menunjukkan kondisi masing-masing. Daun *healthy*

tampak normal tanpa tanda penyakit, sementara daun dengan *late blight* menunjukkan bercak coklat yang menyebar, dan daun dengan *septoria leaf spot* memiliki bintik-bintik khas yang menandakan infeksi.



Gambar 12. Sampel gambar untuk masing-masing kelas pada dataset.

Memilih Sampel Gambar untuk Pengujian

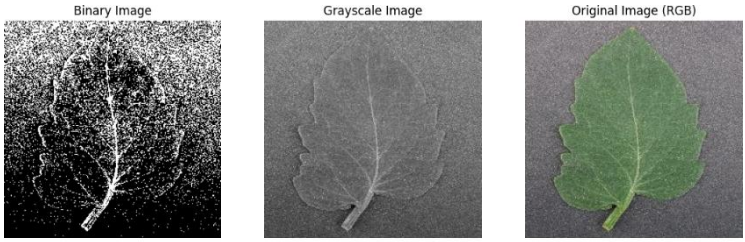
Kode di bawah ini digunakan untuk memilih satu sampel gambar dari setiap kelas dalam dataset sebagai contoh untuk pengujian:

```
TEST_IMAGES, TEST_CLASSES = {}, {}

for image, class_id in dataset.unbatch():
    class_index = class_id.numpy()

    if class_index not in TEST_IMAGES:
        TEST_IMAGES[class_index] = image.numpy()
        TEST_CLASSES[class_index] = CLASS_NAMES[class_index]

    if len(TEST_IMAGES) == len(CLASS_NAMES):
        break
```

Variabel `TEST_IMAGES` digunakan untuk menyimpan gambar untuk pengujian, sementara `TEST_CLASSES` menyimpan label kelas yang sesuai.

3.4 Representasi Gambar Digital dalam *Deep Learning*

Gambar digital (umumnya dengan ekstensi seperti “JPEG” atau “PNG”) merupakan representasi visual berbentuk *array* dua dimensi yang tersusun dari piksel. Setiap piksel adalah unit terkecil yang merepresentasikan warna atau intensitas cahaya dalam gambar.



Gambar 13. Perbandingan tiga jenis representasi gambar digital: gambar biner (kiri), grayscale (tengah), dan berwarna (RGB) (kanan). Gambar biner hanya memiliki dua nilai piksel (hitam dan putih), grayscale merepresentasikan intensitas cahaya dalam skala abu-abu, sedangkan gambar berwarna (RGB) memiliki tiga channel warna.

Nilai piksel dalam gambar tergantung pada tipe gambar yang digunakan:

- **Gambar biner** hanya memiliki dua kemungkinan nilai, yaitu 0 (hitam) atau 1 (putih).
- **Gambar skala abu-abu (*grayscale*)** memiliki nilai piksel antara 0 hingga 255, di mana 0 mewakili hitam, 255 mewakili putih, dan nilai di antaranya membentuk gradasi abu-abu. Representasi ini memungkinkan detail terlihat tanpa informasi warna.
- **Gambar berwarna (RGB)** menggunakan tiga *channel* warna: merah (*red*), hijau (*green*), dan biru (*blue*). Setiap *channel* memiliki nilai dalam rentang 0 hingga 255, dan kombinasi ketiga *channel* menentukan warna piksel tersebut. Misalnya, $(255, 0, 0)$ merepresentasikan merah, sedangkan $(255, 255, 255)$ mewakili putih.

Gambar digital direpresentasikan dalam bentuk matriks tiga dimensi dengan format $h \times w \times c$, di mana:

- h adalah tinggi gambar (jumlah baris piksel).
- w adalah lebar gambar (jumlah kolom piksel).
- c adalah jumlah *channel* warna dalam gambar. Untuk gambar bewarna (RGB), $c = 3$, sedangkan untuk gambar biner dan *grayscale*, $c = 1$.

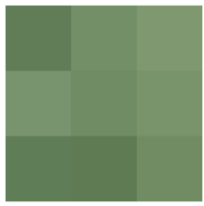
Ekstraksi Nilai Piksel dari Gambar Berwarna

Kode di bawah ini digunakan untuk mengambil bagian kecil (3×3 piksel di tengah gambar) dari gambar berwarna dan menampilkannya dalam bentuk *array*:

```
1  rgb_image =
   TEST_IMAGES[CLASS_NAMES.index("healthy")].astype("uint8")
2
3  rgb_crop = rgb_image[126:129, 126:129, :]
4
5  plt.figure(figsize=(3, 3))
6  plt.imshow(rgb_crop)
7  plt.axis("off")
```

```
plt.show()

print("RGB 3×3 Center Crop Pixel Values:\n", rgb_crop)
```

Setiap nilai dalam *array* menunjukkan intensitas warna merah (*red*), hijau (*green*), dan biru (*blue*) pada masing-masing piksel.



```
RGB 3×3 Center Crop Pixel Values:
 [[[ 97 125  87]
  [115 143 103]
  [127 152 112]]

 [[120 148 110]
  [112 140 100]
  [121 148 107]]

 [[ 95 125  87]
  [ 95 123  83]
  [113 140  99]]]
```

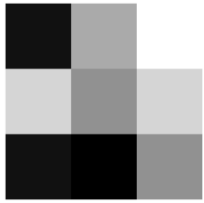
Gambar 14. Ilustrasi ekstraksi nilai piksel dari gambar berwarna (RGB).

Ekstraksi Nilai Piksel dari Gambar *Grayscale*

Kode di bawah ini digunakan untuk mengonversi gambar berwarna (RGB) sebelumnya menjadi *grayscale* dan menampilkan nilai pikselnya:

```
grayscale_image = cv2.cvtColor(rgb_image, cv2.COLOR_RGB2GRAY)

grayscale_crop = grayscale_image[126:129, 126:129]

plt.figure(figsize=(3, 3))
plt.imshow(grayscale_crop, cmap="gray")
plt.axis("off")
plt.show()

print("Grayscale 3×3 Center Crop Pixel Values:\n",
np.round(grayscale_crop))
```

Untuk ekstraksi piksel *grayscale*, fungsi `cvtColor` mengubah gambar dari RGB ke *grayscale* menggunakan fungsi `cv2.COLOR_RGB2GRAY`. Proses ini menghilangkan informasi warna dan hanya mempertahankan tingkat kecerahan setiap piksel.



```
Grayscale 3×3 Center Crop Pixel Values:
 [[112 130 140]
 [135 127 135]
 [112 110 127]]
```

Gambar 15. Ilustrasi ekstraksi nilai piksel dari gambar grayscale.

Gambar *grayscale* hanya memiliki satu *channel*, sehingga setiap piksel direpresentasikan dengan satu angka yang menunjukkan tingkat kecerahan. Nilai yang lebih rendah mendekati 0 menunjukkan piksel lebih gelap (hitam), sedangkan nilai yang lebih tinggi mendekati 255 menunjukkan piksel lebih terang (putih).

Ekstraksi Nilai Piksel dari Gambar Biner

Gambar *grayscale* dapat dikonversi menjadi gambar biner menggunakan fungsi `cv2.threshold()`. Proses ini mengubah nilai setiap piksel berdasarkan ambang batas tertentu (127). Piksel dengan nilai di atas ambang akan diubah menjadi 255 (putih), sedangkan yang lebih rendah menjadi 0 (hitam). Teknik ini digunakan untuk mengekstrak fitur sederhana berdasarkan kontras warna.

```
_, binary_image = cv2.threshold(grayscale_image, 127, 255,
cv2.THRESH_BINARY)

binary_crop = binary_image[126:129, 126:129]

plt.figure(figsize=(3, 3))
plt.imshow(binary_crop, cmap="gray")
plt.axis("off")
plt.show()

print("Binary 3×3 Center Crop Pixel Values:\n",
np.round(binary_crop))
```

Hasil ekstraksi piksel biner dapat dilihat pada gambar di bawah ini:

```
Binary 3×3 Center Crop Pixel Values:
 [[  0 255 255]
 [255   0 255]
 [  0   0   0]]
```

Gambar 16. Ilustrasi ekstraksi nilai piksel dari gambar biner.

3.5 Mempersiapkan Dataset untuk Pelatihan Model

Proses mempersiapkan dataset untuk pelatihan model terdiri dari beberapa langkah yaitu membagi data pelatihan & validasi, normalisasi dataset, dan menerapkan augmentasi data.

Membagi Dataset menjadi *Batch* Pelatihan dan Validasi

Kode di bawah ini digunakan untuk membagi *batch* dataset menjadi dua bagian: 80% untuk pelatihan dan 20% untuk validasi:

```
TRAIN_SPLIT = 0.8

NUM_TRAIN_BATCHES  = int(round(len(dataset) * TRAIN_SPLIT ))
train_dataset = dataset.take(NUM_TRAIN_BATCHES)
val_dataset = dataset.skip(NUM_TRAIN_BATCHES )

print(f"Total batches       : {len(dataset)}")
print(f"Train set batches   : {len(train_dataset)}")
print(f"Validation set batches: {len(val_dataset)}")

```

```
# Total batches       : 149
# Train set batches   : 119
# Validation set batches: 30
```

Variabel `TRAIN_SPLIT=0.8` menentukan proporsi *batch* data yang dialokasikan untuk pelatihan. `NUM_TRAIN_BATCHES` menghitung jumlah *batch* yang digunakan untuk pelatihan dengan mengalikan total *batch* pada dataset dengan `TRAIN_SPLIT`. Dataset kemudian dibagi menjadi:

- `train_dataset = dataset.take(NUM_TRAIN_BATCHES)` mengambil 80% *batch* pertama pada dataset sebagai data pelatihan.
- `val_dataset = dataset.skip(NUM_TRAIN_BATCHES)` melewatkan *batch* yang telah digunakan untuk pelatihan dan menyimpan 20% sisanya sebagai *batch* validasi.

Normalisasi Nilai Piksel pada Dataset

Normalisasi dataset adalah langkah pra-pemrosesan untuk memastikan nilai piksel berada dalam skala yang konsisten. Dalam contoh ini, piksel gambar dinormalisasi ke rentang 0 hingga 1 dengan membagi setiap nilai piksel dengan 255, di mana 0 mewakili hitam dan 1 mewakili putih.

Kode di bawah ini digunakan untuk menerapkan normalisasi pada dataset:

```
def normalize(image, class_label):
    image = tf.cast(image, tf.float32) / 255.0
    return image, class_label

train_dataset = train_dataset.map(normalize,
num_parallel_calls=AUTOTUNE)
val_dataset = val_dataset.map(normalize,
num_parallel_calls=AUTOTUNE)
```

Fungsi `normalize()` mengubah tipe data gambar menjadi `float32` dan membaginya dengan 255 agar nilainya berada dalam skala $0-1$. Kemudian, fungsi `map()` digunakan untuk menerapkan normalisasi ke seluruh dataset secara paralel menggunakan `AUTOTUNE`, sehingga proses berjalan lebih efisien.

Normalisasi penting dalam *deep learning* karena mempengaruhi fungsi aktivasi seperti sigmoid. Jika *input* memiliki skala besar, nilai *output* sigmoid dapat menjadi sangat kecil atau besar, menyebabkan gradien menghilang saat pembaruan bobot. Fungsi sigmoid dihitung dengan rumus:

$$sigmoid = \frac{1}{1 + e^{-(input \times weight)}}$$

Tabel di bawah ini mengilustrasikan nilai sigmoid dari kombinasi *input* dan bobot sebelum dan sesudah normalisasi:

Tabel 2. Nilai sigmoid untuk berbagai kombinasi input dan bobot sebelum (kiri) dan setelah (kanan) normalisasi.

Input	Bobot	Sigmoid
255	0.00001	0.501
255	0.0001	0.506
255	0.001	0.563
255	0.1	1.000
255	0.2	1.000
255	0.4	1.000
255	0.8	1.000
255	1	1.000

Input	Bobot	Sigmoid
1	0.00001	0.500
1	0.0001	0.500
1	0.001	0.500
1	0.1	0.525
1	0.2	0.550
1	0.4	0.599
1	0.8	0.690
1	1	0.731

Pada tabel kiri, nilai sigmoid hampir tidak berubah karena *input* memiliki bobot besar. Sebaliknya, tabel kanan menunjukkan bahwa setelah normalisasi, respons sigmoid lebih stabil, membantu model memperbarui bobot dengan lebih baik. Dengan kata lain, normalisasi mengurangi dampak bobot besar, membuat model belajar lebih efektif dan meningkatkan akurasi.

Augmentasi Data

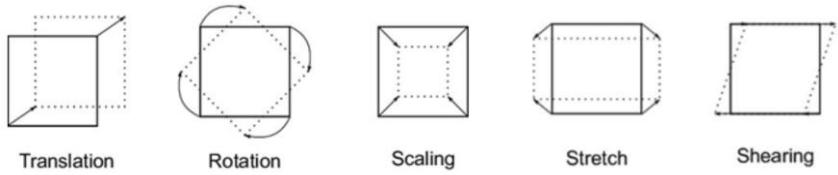
Dalam dunia nyata, gambar sering mengalami berbagai kondisi, seperti:

- Sedikit diputar,
- Diperbesar atau diperkecil (*zoom in*/*out*),
- Mengandung *noise*,
- Memiliki pencahayaan yang terlalu terang atau terlalu redup,
- Dicerminkan (*flipped*),
- Mengalami distorsi (*sheared*).

Jika model hanya dilatih dengan gambar tanpa variasi, performa prediksi bisa menurun, terutama saat menghadapi data dengan distribusi berbeda. Gambar yang tidak mengalami perubahan selama pelatihan dapat menyebabkan model kesulitan mengenali objek dalam kondisi yang berbeda di dunia nyata.

Augmentasi data digunakan untuk mengatasi masalah ini dengan menambahkan variasi pada gambar asli. Teknik ini mencakup rotasi, translasi, perubahan skala, penyesuaian pencahayaan, hingga penambahan noise. Dengan cara ini, model dapat belajar dari berbagai kemungkinan gambar, sehingga lebih *robust* terhadap perubahan pada data.

Salah satu metode yang umum digunakan dalam augmentasi adalah transformasi *affine*, yang mencakup translasi, rotasi, perubahan skala, *stretch*, dan *shear* pada gambar *input*. Transformasi ini memperkaya variasi dataset tanpa mengubah struktur utama gambar, sehingga model dapat mengenali pola dengan lebih baik dan tetap mampu menggeneralisasi dengan baik saat diuji dengan data baru.



Gambar 17. Ilustrasi berbagai jenis transformasi affine seperti translasi, rotasi, scaling, strecth, dan shearing.

Kode di bawah ini digunakan untuk menerapkan augmentasi data:

```
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.2),
    layers.RandomZoom(0.2),
])

def augment(image, class_label):
    return data_augmentation(image), class_label

train_dataset = train_dataset.map(augment,
num_parallel_calls=AUTOTUNE)
```

Kode ini melakukan tiga jenis augmentasi:

- `RandomFlip("horizontal")` digunakan untuk membalik gambar secara horizontal untuk mencerminkan variasi sudut pandang.
- `RandomRotation(0.2)` digunakan untuk memutar gambar secara acak hingga 20% dari sudut penuh, membantu model mengenali objek dalam berbagai orientasi.
- `RandomZoom(0.2)` digunakan untuk memperbesar atau memperkecil gambar hingga 20% untuk mensimulasikan perbedaan jarak kamera saat pengambilan gambar.

Fungsi `augment()` menerapkan augmentasi ke setiap gambar dalam dataset, dan `map()` digunakan untuk mempercepat pemrosesan dengan `AUTOTUNE`. Penting dicatat bahwa augmentasi hanya diterapkan pada data pelatihan, bukan pada data validasi atau pengujian. Hal ini karena augmentasi bertujuan untuk memperluas variasi data yang digunakan dalam peletihan model, sedangkan data validasi dan pengujian harus tetap representatif dengan data dunia nyata tanpa modifikasi tambahan.

Kode di bawah ini digunakan untuk mengoptimalkan *pipeline input* dataset:

```
train_dataset =
train_dataset.cache().prefetch(buffer_size=AUTOTUNE)
val_dataset =
val_dataset.cache().prefetch(buffer_size=AUTOTUNE)
```

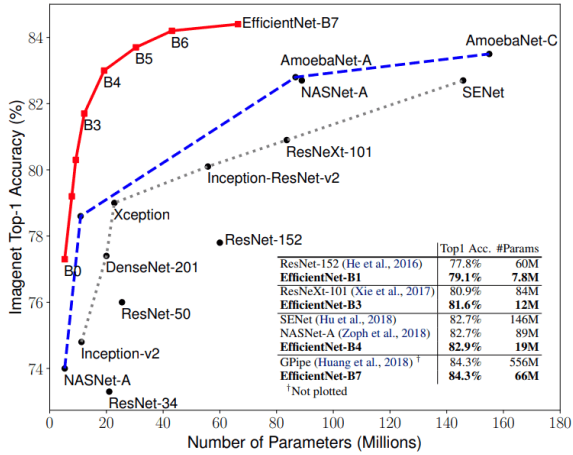
Fungsi `cache()` menyimpan dataset di memori atau penyimpanan lokal untuk menghindari pemuatan ulang dari memori, sehingga mempercepat pelatihan.

Sementara itu, fungsi `prefetch(buffer_size=AUTOTUNE)` memungkinkan pemuatan *batch* berikutnya saat *batch* sebelumnya masih diproses, sehingga mengoptimalkan *pipeline* data. Kombinasi keduanya mengurangi *bottleneck*, meningkatkan efisiensi pelatihan model.

Bab 4: Model *Convolutional Neural Network* untuk Klasifikasi Objek

Pendekatan *deep learning* tradisional, seperti *Multi-Layer Perceptron* (MLP), memiliki keterbatasan dalam mengolah data berbentuk gambar. Model ini memperlakukan setiap piksel sebagai entitas independen tanpa mempertimbangkan hubungan spasial, sehingga kurang efektif dalam mengenali pola visual yang kompleks. Dalam konteks deteksi penyakit tanaman, pendekatan ini tidak cukup tangguh terhadap variasi pencahayaan, rotasi, atau perubahan bentuk daun akibat infeksi. Oleh karena itu, diperlukan metode yang lebih modern, seperti *Convolutional Neural Network* (CNN), yang dapat menangkap pola spasial dalam gambar dengan lebih efisien dan akurat.

Kemajuan CNN telah didorong oleh kompetisi *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC)[10], yang menjadi *benchmark* utama dalam pengembangan arsitektur *deep learning* untuk pengolahan citra. Berbagai pengembangan arsitektur CNN, seperti VGG (Simonyan & Zisserman, 2015), ResNet (He et al., 2015), dan EfficientNet (Tan & Le, 2020), menunjukkan peningkatan signifikan dalam akurasi klasifikasi gambar dengan efisiensi parameter yang lebih baik. Perkembangan ini membuka peluang besar dalam bidang pertanian modern, memungkinkan diagnosis penyakit tanaman dengan akurasi tinggi menggunakan model yang lebih ringan dan dapat diterapkan pada perangkat dengan keterbatasan komputasi.



Gambar 18. Perbandingan arsitektur CNN berdasarkan jumlah parameter dan akurasi pada kompetisi ImageNet menunjukkan peningkatan efisiensi dan akurasi seiring berkembangnya arsitektur deep learning.

[10] https://www.image-net.org/challenges/LSVRC

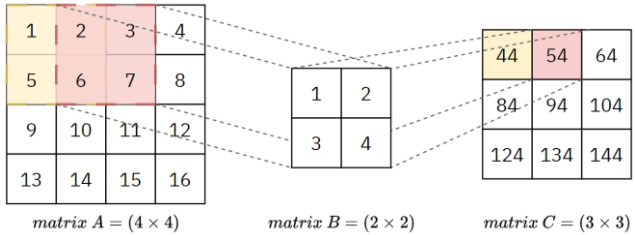
4.1 Memahami Arsitektur dan Komponen CNN

Convolutional Neural Network (CNN) adalah arsitektur *deep learning* yang dirancang untuk mengatasi keterbatasan model tradisional dalam mengenali pola visual pada data gambar. CNN mempelajari representasi hierarkis gambar, di mana *layer* awal menangkap fitur sederhana seperti tepi, sedangkan *layer* lebih dalam mengenali pola kompleks seperti bentuk atau objek tertentu. CNN efektif untuk berbagai tugas *computer vision*, termasuk klasifikasi objek, deteksi objek, dan segmentasi.

Konvolusi (*Convolution*)

Konvolusi adalah operasi utama dalam CNN untuk mengekstraksi fitur dari gambar dengan melakukan perkalian dua matriks. Operasi ini menghitung nilai yang merepresentasikan hubungan antara *input* dengan *filter* yang diterapkan pada gambar. Proses ini membantu mengidentifikasi pola penting dalam gambar dengan menghitung nilai pada *hidden layer* berdasarkan hubungan antara matriks *input* dan bobot *filter*.

Sebagai ilustrasi, misalkan terdapat dua matriks: matriks A berukuran 4×4 yang merepresentasikan gambar dan matriks B berukuran 2×2 sebagai *filter*. Saat proses konvolusi, matriks B bergeser melintasi matriks A, dimulai dari sudut kiri atas. Pada setiap posisi, elemen-elemen dalam matriks B dikalikan dengan elemen-elemen matriks A yang bersesuaian, lalu hasil perkalian dijumlahkan menjadi satu nilai *output*.



Gambar 19. lustrasi operasi convolution: matriks input A berukuran 4×4 dikonvolusi dengan matriks filter B berukuran 2×2 menghasilkan matriks feature map C berukuran 3×3.

Pada contoh operasi konvolusi dari gambar di atas:

1. Perkalian elemen pertama, yaitu $[1, 2, 5, 6]$ dari matriks A dengan $[1, 2, 3, 4]$ dari matriks B menghasilkan:
$$(1 \times 1) + (2 \times 2) + (5 \times 3) + (6 \times 4) = 44$$
2. Perkalian elemen kedua, yaitu $[2, 3, 6, 7]$ dari matriks A dengan $[1, 2, 3, 4]$ dari matriks B menghasilkan:
$$(2 \times 1) + (3 \times 2) + (6 \times 3) + (7 \times 4) = 54$$
3. Proses ini diulang hingga seluruh elemen dalam matriks A telah dikonvolusi.

Hasil dari operasi convolution membentuk matriks baru C berukuran 3×3, yang disebut *feature map* atau *convolved feature*. Ukurannya lebih kecil dari matriks *input* karena setiap kali *filter* bergeser, hanya sebagian elemen yang diproses untuk menghasilkan satu nilai dalam *feature map*.

Filter

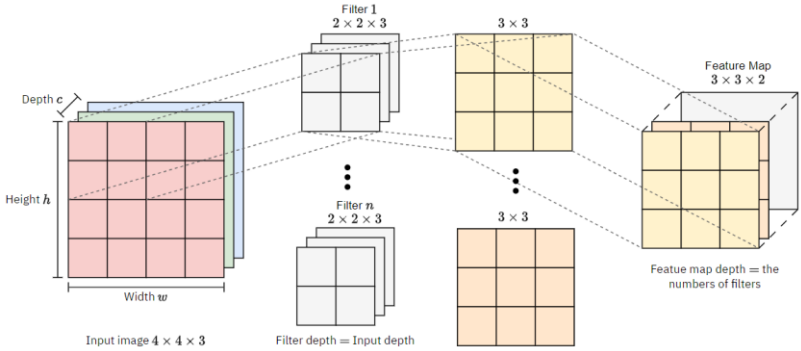
Filter atau *kernel* adalah matriks bobot yang diinisialisasi secara acak saat awal pelatihan. Selama pelatihan, model mempelajari bobot optimal untuk setiap *filter* melalui proses iterasi.

Konsep *filter* dalam CNN dapat dijelaskan dalam dua aspek utama:

1. **Apa yang dipelajari *filter*:** *filter* digunakan untuk mengekstraksi fitur spesifik dari gambar, seperti tepi, pola, tekstur, atau elemen visual lainnya. Semakin banyak *filter* yang digunakan, semakin banyak pola yang dapat dipelajari oleh model.
2. **Bagaimana *filter* direpresentasikan:** *filter* direpresentasikan sebagai matriks kecil yang diterapkan ke seluruh bagian gambar *input* melalui operasi konvolusi.

Pada contoh sebelumnya, sebuah gambar berukuran 4×4 dikonvolusi dengan satu *filter* berukuran 2×2, menghasilkan matriks *output* berukuran 3×3. Jika terdapat 10 *filter*, maka hasilnya adalah 10 set *feature map*, masing-masing berukuran 3×3.

Untuk gambar berwarna yang memiliki tiga *channel* warna (RGB), setiap *filter* juga memiliki tiga *channel* agar dapat dipadankan dengan gambar *input*. Setelah konvolusi, setiap *filter* menghasilkan satu *feature map* skalar. Sebagai contoh, jika satu *filter* diterapkan pada gambar RGB berukuran $64 \times 112 \times 3$, hasilnya adalah *feature map* berukuran $64 \times 112 \times 1$. Jika diterapkan 512 *filter*, maka hasil akhirnya adalah *feature map* dengan ukuran $64 \times 112 \times 512$.



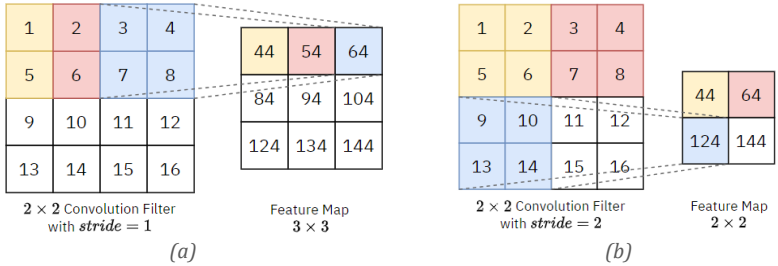
Gambar 20. Ilustrasi penerapan filter pada gambar input berukuran $4 \times 4 \times 3$ dengan filter $2 \times 2 \times 3$, menghasilkan feature maps dengan ukuran $3 \times 3 \times 2$, di mana kedalaman feature map sesuai dengan jumlah filter yang digunakan.

Stride

Stride adalah jarak pergeseran *filter* selama operasi konvolusi pada gambar. Nilai *stride* menentukan seberapa jauh *filter* bergeser dalam setiap langkah, baik secara horizontal maupun vertikal. Secara *default*, *stride* bernilai 1, yang berarti *filter* berpindah satu piksel per langkah sehingga mencakup seluruh area gambar. Jika *stride* lebih besar, *filter* akan melompat beberapa piksel, yang mengurangi jumlah posisi yang diproses tetapi berisiko kehilangan detail penting dalam gambar.

Sebagai ilustrasi, misalkan terdapat matriks A (gambar) berukuran 4×4 dan matriks B (*filter*) berukuran 2×2. Jika *stride* adalah 2, maka *filter* akan melompati dua piksel dalam setiap langkahnya saat bergeser melintasi matriks A.

1. Pada langkah pertama, elemen matriks A yang digunakan adalah $[1, 2, 5, 6]$, yang dikalikan dengan $[1, 2, 3, 4]$ dari matriks B, menghasilkan:
$$(1 \times 1) + (2 \times 2) + (5 \times 3) + (6 \times 4) = 44$$
2. Pada langkah berikutnya, *filter* bergeser dua piksel ke kanan, mengambil elemen $[3, 4, 7, 8]$ dari matriks A, dan dikalikan dengan $[1, 2, 3, 4]$ dari matriks B, menghasilkan:
$$(3 \times 1) + (4 \times 2) + (7 \times 3) + (8 \times 4) = 64$$
3. Proses ini diulang hingga seluruh elemen dalam matriks A telah dikonvolusi.



Gambar 21. Ilustrasi perbandingan hasil konvolusi dengan stride 1 (a) dan stride 2 (b). Stride yang lebih besar menghasilkan feature map dengan dimensi lebih kecil karena pergeseran filter yang lebih jauh.

Pada contoh sebelumnya, jika *stride* adalah 1, *filter* bergerak melintasi setiap piksel matriks A, menghasilkan *output* yang lebih besar, yaitu 3×3. Sebaliknya, jika *stride* adalah 2, *filter* hanya memproses sebagian posisi matriks A, sehingga ukuran *output* menjadi lebih kecil, yaitu 2×2. *Stride* memungkinkan untuk mengontrol ukuran *output* dari operasi konvolusi:

- *Stride* lebih besar menghasilkan *output* dengan dimensi lebih kecil, sehingga mengurangi beban komputasi.

- *Stride* lebih kecil mempertahankan lebih banyak detail dalam gambar, tetapi membutuhkan lebih banyak komputasi karena lebih banyak posisi piksel yang harus diproses.

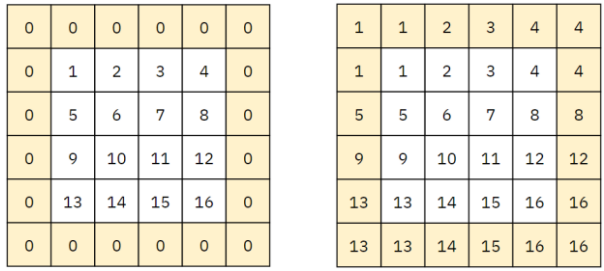
Padding

Padding adalah teknik menambahkan piksel di tepi gambar *input* sebelum operasi konvolusi. Tujuannya untuk menjaga ukuran *output* tetap sama dengan *input* serta mencegah hilangnya informasi di tepi gambar.

Misalnya, jika terdapat gambar *input* berukuran 4×4 dan *filter* berukuran 2×2 dengan *stride* 2. Jika tanpa *padding*, hanya sebagian matriks yang diproses, menghasilkan *feature map* yang lebih kecil. Namun, jika *padding* diterapkan, ukuran *input* dapat diperbesar menjadi 6×6, sehingga semua piksel bisa tetap diproses.

Jenis *padding* yang umum digunakan adalah:

1. ***Zero Padding***. Menambahkan nilai nol di sekitar tepi matriks *input*. Hal ini berguna untuk menjaga ukuran *output* tetap sesuai dengan *input* sekaligus mencegah hilangnya informasi pada tepi gambar.
2. ***Replication Padding***. Menyalin nilai piksel terluar dari matriks *input* ke tepinya. Dengan cara ini, pola visual gambar tetap terjaga di tepi, membantu model memperoleh informasi lebih baik dari area batas gambar.



Gambar 22. Ilustrasi zero padding (a), yang menambahkan nol di sekitar matriks input, dan replication padding (b), yang mengisi padding dengan nilai dari elemen tepi matriks. Kedua teknik padding digunakan untuk mengatur ukuran input dalam operasi konvolusi.

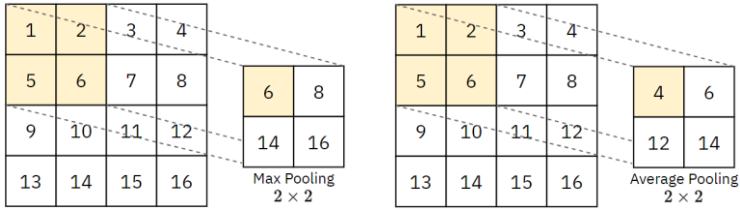
Pooling

Pooling adalah operasi yang digunakan untuk mengurangi dimensi data sambil mempertahankan informasi penting. Teknik ini menggabungkan nilai dalam area kecil (*patch*) dari data *input*, sehingga mengurangi jumlah parameter, meningkatkan efisiensi komputasi, dan mengurangi risiko *overfitting*.

Jenis *pooling* yang umum digunakan adalah:

1. ***Max Pooling***. Memilih nilai maksimum dari setiap *patch*. Metode ini membantu menonjolkan fitur yang paling signifikan dalam gambar.

2. ***Average Pooling***. Mengambil rata-rata nilai dalam setiap *patch*. Teknik ini mempertahankan informasi secara lebih halus dan mengurangi detail yang kurang penting.



Gambar 23. Ilustrasi max pooling (a) dan average pooling (b) pada matriks input 4×4 dengan stride 2. Max pooling memilih nilai maksimum dari setiap patch, sedangkan average pooling menghitung rata-rata nilai dalam patch yang sama.

Fungsi Aktivasi ReLU (*Rectified Linear Unit*)

Fungsi aktivasi ReLU adalah komponen penting dalam CNN untuk menambahkan non-linearitas. Operasi konvolusi pada dasarnya bersifat linear, sementara gambar memiliki pola yang kompleks, seperti tepi, tekstur, dan bentuk yang tidak dapat direpresentasikan hanya dengan operasi linear.

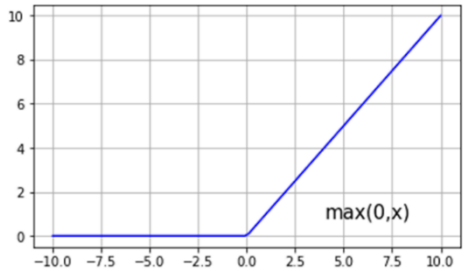
Fungsi ReLU didefinisikan sebagai berikut:

$$f(x) = max(0, x)$$

Di mana:

- Jika x lebih besar dari 0, maka $f(x) = x$.
- Jika x kurang dari atau sama dengan 0, maka $f(x) = 0$.

Karena efektivitasnya dalam meningkatkan efisiensi dan stabilitas pelatihan, ReLU menjadi fungsi aktivasi standar dalam arsitektur CNN modern (Cong & Zhou, 2023). Fungsi aktivasi ReLU diterapkan setelah operasi konvolusi untuk membantu model menangkap fitur-fitur penting dari gambar dengan lebih baik serta meningkatkan performa model.

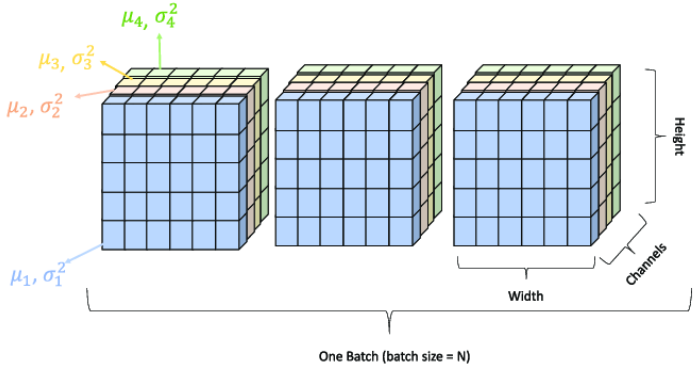


Gambar 24. Grafik fungsi aktivasi ReLU yang menetapkan nilai negatif menjadi nol dan mempertahankan nilai positif secara linear.

Mekanisme *Batch Normalization*

Batch Normalization adalah teknik untuk menormalkan nilai aktivasi pada setiap *layer* CNN berdasarkan statistik *mini-batch* (Ziaee & ÇAno, 2023). Metode ini menghitung rata-rata μ dan varians σ^2 dari setiap *channel* dalam *batch*, lalu menggunakan nilai tersebut untuk menstandarkan data sebelum diteruskan ke *layer* berikutnya. Normalisasi ini membantu model belajar lebih stabil, mempercepat konvergensi, dan mengurangi risiko *vanishing* atau *exploding gradient*.

Gambar di bawah ini mengilustrasikan bagaimana *batch normalization* diterapkan dalam CNN. Setiap *batch* terdiri dari beberapa gambar dengan dimensi *height*, *width*, dan *channels*. Untuk setiap *channel*, nilai μ dan σ^2 dihitung berdasarkan seluruh gambar dalam *batch*, kemudian digunakan untuk menormalkan nilai aktivasi. Setelah itu, parameter γ *(scale)* dan β *(shift)* diterapkan agar model tetap fleksibel dalam menangkap pola kompleks.



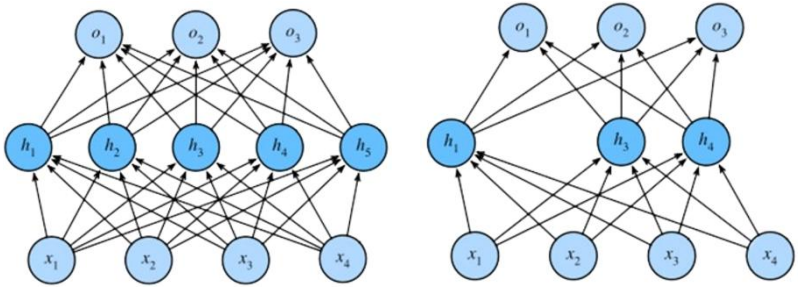
Gambar 25. Ilustrasi penerapan batch normalization dalam CNN. Proses ini menghitung rata-rata μ dan varians σ^2 dari setiap channel dalam satu batch, lalu menggunakan nilai tersebut untuk menormalisasi data sebelum diteruskan ke layer berikutnya.

Mekanisme *Dropout*

Dropout adalah teknik regulasi yang digunakan untuk mencegah *overfitting* selama pelatihan model dengan menonaktifkan sejumlah *neuron* dalam *layer* secara acak di setiap iterasi (Shunk, 2022).

Sebelum *dropout* diterapkan, setiap *neuron* dalam *layer* sepenuhnya terhubung ke *neuron* di *layer* berikutnya. Setelah *dropout* diaktifkan, beberapa *neuron* secara acak dinonaktifkan, sehingga koneksi ke *neuron* terputus. Dengan cara ini, model tidak terlalu bergantung pada *neuron* tertentu dan lebih mampu mengenali pola yang lebih umum. Model menjadi lebih adaptif terhadap variasi data dan memiliki kemampuan generalisasi yang lebih baik saat diuji dengan data baru.

Nilai *dropout* yang umum digunakan berkisar antara 0.2 hingga 0.5, tergantung pada kompleksitas model dan dataset yang digunakan.



(a) *(b)*

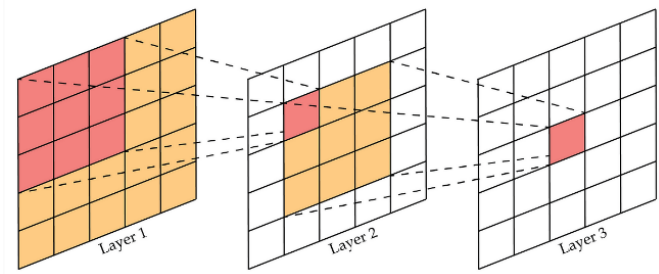
Gambar 26.Ilustrasi mekanisme droput sebelum (a) dan sesudah diterapkan.

Peran Konvolusi dan Pooling dalam Pemrosesan Gambar

Konvolusi dan *pooling* memainkan peran penting dalam pemrosesan gambar di CNN. Saat gambar diproses, konvolusi mengekstraksi fitur, sementara *pooling* mempertahankan informasi utama dengan memilih nilai maksimum atau rata-rata dalam area kecil (*patch*). Proses ini membantu model mengenali pola dalam gambar meskipun terjadi perubahan posisi. Misalnya, dalam *max pooling*, fitur yang paling menonjol, seperti tepi atau pola tekstur, tetap dipertahankan, sehingga model tetap mampu mengenali objek meskipun posisinya sedikit berubah.

Selain itu, konvolusi dan *pooling* memperluas *receptive field*, yaitu area gambar yang mempengaruhi *output neuron* di suatu *layer*. Jika proses ini diterapkan beberapa kali pada gambar berukuran 100×100 piksel, hasil akhirnya bisa menyusut menjadi 25×25 piksel. Setiap piksel dalam hasil akhir tidak hanya merepresentasikan satu titik dalam gambar asli, tetapi juga area yang lebih luas.

Seiring bertambahnya *layer* dalam CNN, *receptive field* semakin luas. Pada *layer* awal, hanya area kecil yang diproses, tetapi di *layer* yang lebih dalam, model mulai memahami pola yang lebih kompleks, seperti bentuk objek secara keseluruhan. Ilustrasi di bawah ini menggambarkan bagaimana receptive field berkembang seiring bertambahnya *layer* dalam CNN.



Gambar 27. Ilustrasi perubahan receptive field dalam jaringan CNN. Pada layer awal, receptive field mencakup area kecil dari input, sedangkan pada layer yang lebih dalam, receptive field membesar, memungkinkan jaringan memahami fitur yang lebih kompleks dan abstrak.

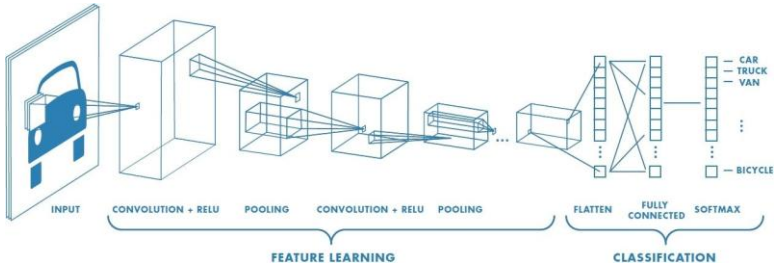
Alur Proses CNN

CNN terdiri dari beberapa tahap, dimulai dari *convolutional layer* hingga *fully connected layer*. Pada tahap awal, *convolutional layer* digunakan untuk mengekstraksi fitur dari gambar menggunakan *filter*, sedangkan *pooling layer* mengurangi dimensi data sambil mempertahankan informasi penting. Setelah melalui beberapa iterasi *convolution* dan *pooling*, data diproses melalui *flatten layer* untuk mengubahnya menjadi vektor satu dimensi sehingga dapat diproses pada *fully connected layer*. Tahap terakhir menghasilkan prediksi dengan fungsi aktivasi seperti *sigmoid* (klasifikasi biner) atau *softmax* (klasifikasi multi-kelas).

Tahapan kerja CNN secara umum mencakup langkah-langkah berikut:

1. ***Convolutional layer***. Gambar *input* diproses menggunakan *filter* untuk mengekstraksi fitur seperti tepi, pola, dan tekstur.
2. ***Pooling layer***. Dimensi data dikurangi untuk menjaga informasi penting dan meningkatkan efisiensi komputasi.
3. **Iterasi *convolution* dan *pooling***. Proses *convolution* dan *pooling* diulang beberapa kali untuk menghasilkan data yang lebih kecil dan lebih terstruktur. Misalnya, jika gambar input berukuran 300×300 piksel, proses *flatten* akan menghasilkan 90.000 nilai *input*. Jika setiap nilai ini dihubungkan ke 100.000 *neuron* di *hidden layer*, jumlah parameter pelatihannya bisa melebihi 9 miliar, sehingga sangat mahal dalam komputasi. Dengan *convolution* dan *pooling*, jumlah parameter dapat dikurangi secara signifikan sebelum diproses di *fully connected layer*.
4. ***Flatten layer***. Data yang telah direduksi diubah ke dalam bentuk linear satu dimensi.
5. ***Fully connected layer***. Data linear diproses untuk menghasilkan prediksi menggunakan fungsi aktivasi.

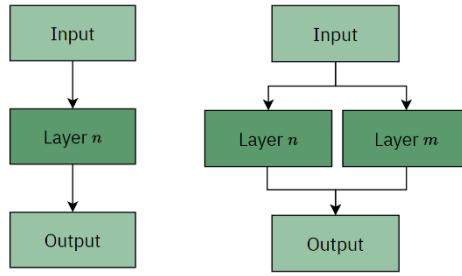
Melalui tahapan-tahapan di atas, CNN mampu mengenali pola secara efisien, mengurangi jumlah parameter yang perlu dipelajari, serta meningkatkan akurasi dan efisiensi dalam pengolahan gambar.



Gambar 28. Diagram alur proses CNN. Proses diawali dengan ekstraksi fitur melalui layer konvolusi dan pooling, kemudian dilanjutkan dengan fully connected layer untuk klasifikasi.

4.2 Membangun Arsitektur CNN

Dalam *deep learning*, rsitektur model menentukan bagaimana data diproses dari *input* hingga *output* untuk menghasilkan prediksi. TensorFlow, melalui Keras[11], menyediakan dua cara utama untuk membangun arsitektur model *deep learning*: Sequential API[12] dan Functional API[13].



(a) *(b)*

Gambar 29. Ilustrasi dua pendekatan dalam membangun model deep learning menggunakan TensorFlow Keras: Sequential API (a) dan Functional API (b).

Sequential API adalah metode paling sederhana dalam mendefinisikan model. Model dibangun secara berurutan, di mana setiap *layer* ditambahkan satu per satu sesuai urutan yang telah ditentukan. Pendekatan ini cocok untuk arsitektur *feedforward*, seperti *Convolutional Neural Network* (CNN), yang memproses data secara hierarkis dari satu *layer* ke *layer* berikutnya.

Dengan Sequential API, arsitektur CNN dapat didefinisikan dengan menambahkan setiap *layer* secara bertahap, dari *input* hingga *output*, seperti kode di bawah ini:

```
cnn_model = models.Sequential([
    # Input layer
    layers.Input(shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),

    # Convolutional layer 1
    layers.Conv2D(filters=32, kernel_size=(3, 3),
activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Convolutional layer 2
    layers.Conv2D(filters=64, kernel_size=(3, 3),
activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Flatten mengubah data ke 1D
```

[11] https://keras.io
[12] https://www.tensorflow.org/guide/keras/sequential_model
[13] https://www.tensorflow.org/guide/keras/functional_api

```
    layers.Flatten(),
    layers.Dropout(rate=0.3),

    # Fully connected layer
    layers.Dense(units=128, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.01)),
    layers.BatchNormalization(),
    layers.Dropout(rate=0.3),

    # Output layer
    layers.Dense(units=len(CLASS_NAMES), activation='softmax')
], name="CNN")
```

Kode di atas mendefinisikan arsitektur model CNN (`cnn_model`) menggunakan TensorFlow dengan fungsi `Sequential()`:

1. ***Input layer*** menerima gambar dengan ukuran yang telah ditentukan dalam variabel `IMG_SIZE`. Gambar memiliki tiga *channel* warna (RGB), sehingga dimensi pada `Input()`[14] adalah $256 \times 256 \times 3$.
2. ***Convolutional layer* ke-1** menggunakan `Conv2D()`[15] dengan 32 *filter* berukuran 3×3 untuk mengekstraksi fitur awal dari gambar. Aktivasi ReLU menambahkan non-linearitas. `BatchNormalization()`[16] diterapkan agar nilai piksel tetap stabil, selanjutnya `MaxPooling2D()`[17] berukuran 2×2 digunakan untuk mengurangi dimensi tanpa kehilangan fitur penting.
3. ***Convolutional layer* ke-2** serupa dengan *layer* konvolusi pertama tetapi menggunakan 64 *filter* agar dapat menangkap fitur lebih kompleks. `BatchNormalization()` diterapkan kembali untuk menjaga stabilitas pelatihan, diikuti oleh `MaxPooling2D()` yang memperkecil dimensi data.
4. ***Flatten layer*** mengubah data 2D hasil konvolusi dan *pooling* menjadi vektor 1D dengan `Flatten()`[18].
5. ***Fully connected layer*** menggunakan `Dense()`[19] dengan 128 *neuron* dan aktivasi ReLU berfungsi sebagai penghubung antara fitur yang diekstraksi dari *layer* konvolusi dan *output layer*. Regularisasi `l2()`[20] diterapkan untuk mencegah *overfitting*, sementara `Dropout()`[21] digunakan untuk meningkatkan generalisasi model.

[14] https://www.tensorflow.org/api_docs/python/tf/keras/Input
[15] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
[16] https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization
[17] https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D
[18] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Flatten
[19] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
[20] https://www.tensorflow.org/api_docs/python/tf/keras/Regularizer
[21] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

6. ***Output layer*** menggunakan `Dense()` dengan jumlah *neuron* sesuai jumlah kelas pada dataset `len(CLASS_NAMES)`. Aktivasi *softmax* digunakan agar model menghasilkan probabilitas untuk setiap kelas.

Setelah model didefinisikan, arsitektur lengkapnya dapat ditampilkan menggunakan fungsi `summary()`.

```
1 cnn_model.summary()
```

Fungsi ini memberikan informasi berikut:

- **Jenis *layer*** yang digunakan dalam model.
- **Ukuran *output*** pada setiap *layer* setelah proses konvolusi, *pooling*, dan *flatten*.
- **Jumlah parameter** yang dapat dilatih (*trainable parameters*) dan tidak dapat dilatih (*non-trainable parameters*).

Dari hasil `cnn_model.summary()` pada gambar di bawah ini, dapat dilihat bahwa model memiliki total 31 juta parameter, dengan mayoritas berasal dari *fully connected layer*. Hal ini menunjukkan bahwa meskipun CNN mengurangi dimensi gambar melalui *convolution* dan *pooling*, jumlah parameter tetap besar karena *dense layer* menangani representasi fitur secara lebih kompleks.

Model: "CNN"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	128
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
flatten (Flatten)	(None, 246016)	0
dropout (Dropout)	(None, 246016)	0
dense (Dense)	(None, 128)	31,490,176
batch_normalization_2 (BatchNormalization)	(None, 128)	512
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 3)	387

Total params: 31,510,851 (120.20 MB)
Trainable params: 31,510,403 (120.20 MB)
Non-trainable params: 448 (1.75 KB)

Gambar 30. Ringkasan arsitektur model CNN, menunjukkan jenis layer, output shape, dan jumlah parameter yang digunakan.

Struktur arsitektur model juga dapat divisualisasikan dengan lebih intuitif menggunakan fungsi `plot_model()`. Fungsi ini menampilkan diagram yang menunjukkan susunan *layer* dalam model, termasuk bentuk *input* dan *output* pada setiap *layer*. Dengan visualisasi ini, hubungan antar *layer* dapat lebih mudah dipahami, terutama dalam arsitektur jaringan yang kompleks seperti CNN.

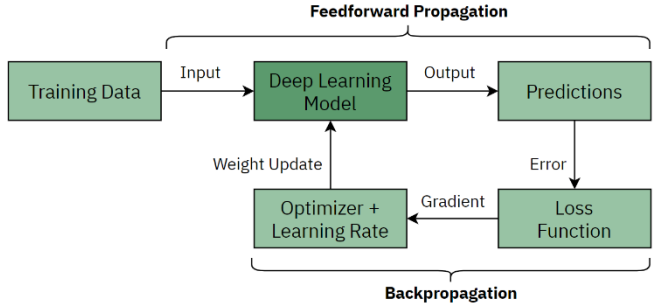
```
1  tf.keras.utils.plot_model(cnn_model, show_shapes=True,
   show_layer_names=True, expand_nested=True, dpi=50)
```

4.3 Melatih dan Validasi Model CNN

Pelatihan model *deep learning* terdiri dari dua proses utama: yaitu:

- ***Feedforward propagation***. Data pelatihan dimasukkan ke dalam model, yang terdiri dari *input layer*, *hidden layers*, dan *output layer*. Model kemudian membuat prediksi berdasarkan bobot awal yang diinisialisasi secara acak.
- ***Backpropagation***. Model membandingkan prediksi dengan label kelas sebenarnya menggunakan fungsi *loss*. Jika terdapat kesalahan (*error*), algoritma *backpropagation* menghitung gradien kesalahan dan memperbarui bobot menggunakan algoritma optimasi berbasis gradien seperti *Adaptive Moment Estimation* (Adam) atau *Stochastic Gradient Descent* (SGD). Proses ini diulang dalam beberapa iterasi hingga model mencapai performa optimal.

Diagram di bawah ini mengilustrasikan bagaimana data mengalir melalui *feedforward propagation*, bagaimana *error* dihitung, dan bagaimana bobot diperbarui menggunakan backpropagation:



Gambar 31. Ilustrasi feedforward propagation dan backpropagation dalam pelatihan model deep learning. Feedforward menghasilkan prediksi, sementara backpropagation memperbarui bobot berdasarkan kesalahan.

Mendefinisikan Hyperparameter dan Kompilasi Model CNN

Sebelum melatih model, perlu mendefinisikan *hyperparameter* untuk proses optimasi dan evaluasi performa model. *Hyperparameter* meliputi fungsi *loss*, algoritma optimasi, dan metrik evaluasi. Kode di bawah ini mendefinisikan *hyperparameter* yang digunakan dalam proses pelatihan:

```
LOSS_FUNCTION = losses.SparseCategoricalCrossentropy()

LEARNING_RATE = 0.0001
OPTIMIZER = optimizers.Adam(learning_rate=LEARNING_RATE)

METRICS = [
    metrics.SparseCategoricalAccuracy(name='accuracy')
]
```

1. `SparseCategoricalCrossentropy()`[22] digunakan sebagai fungsi *loss* untuk klasifikasi multi-kelas dengan label kelas dalam bentuk indeks, bukan *one-hot encoding*.
2. `Adam()`[23] adalah algoritma optimasi yang menyesuaikan *learning rate* secara adaptif untuk setiap parameter. Variabel `LEARNING_RATE=0.0001` menentukan besarnya pembaruan bobot dalam setiap iterasi. Nilai yang terlalu besar dapat membuat model tidak stabil, sedangkan nilai terlalu kecil dapat memperlambat konvergensi.
3. `SparseCategoricalAccuracy()`[24] adalah metrik evaluasi yang menghitung akurasi dengan membandingkan prediksi model dengan label kelas sebenarnya. Akurasi dihitung sebagai rasio prediksi yang benar terhadap total sampel dalam dataset.

Setelah *hyperparameter* didefinisikan, langkah selanjutnya adalah mengompilasi model menggunakan fungsi `compile()`. Proses ini menghubungkan arsitektur model dengan metode pembelajaran yang akan digunakan. Berikut kode untuk mengompilasi model:

```
cnn_model.compile(optimizer=OPTIMIZER, loss=LOSS_FUNCTION,
metrics=METRICS)
```

Proses Pelatihan Model CNN

Setelah model dikompilasi, langkah selanjutnya adalah melatih model menggunakan dataset yang telah disiapkan. Proses pelatihan dilakukan dalam beberapa *epoch*, yaitu satu siklus penuh di mana model mempelajari pola dari seluruh dataset. Pada setiap *epoch*, model menerima *input*, melakukan *feedforward propagation*, menghitung *loss*, lalu memperbarui bobot melalui *backpropagation*.

Pelatihan model dilakukan menggunakan fungsi `fit()` dengan parameter utama:

1. `epochs` adalah jumlah *epoch* model dalam mempelajari dataset.
2. `train_dataset` adalah dataset yang digunakan untuk melatih model.
3. `val_dataset` adalah dataset validasi untuk mengevaluasi performa model di setiap *epoch*.

[22] https://www.tensorflow.org/api_docs/python/tf/keras/losses/sparse_categorical_crossentropy
[23] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
[24] https://www.tensorflow.org/api_docs/python/tf/keras/metrics/SparseCategoricalAccuracy

4. `steps_per_epoch` adalah jumlah *batch* dalam satu *epoch* dataset pelatihan.
5. `validation_steps` adalah jumlah *batch* dalam satu *epoch* dataset validasi.

Kode di bawah ini menunjukkan bagaimana model dilatih selama 20 *epoch*, sekaligus menghitung waktu total pelatihan:

```
EPOCHS = 20

start_time = time.time()

cnn_history = cnn_model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    steps_per_epoch=len(train_dataset),
    validation_steps=len(val_dataset)
)

end_time = time.time()

training_time = end_time - start_time
print(f"Total Training Time: {training_time:.2f} seconds")
```

Waktu komputasi yang dibutuhkan untuk melatih model CNN dengan 20 epoch adalah 245.47 detik.

Menampilkan Grafik Pelatihan Model CNN

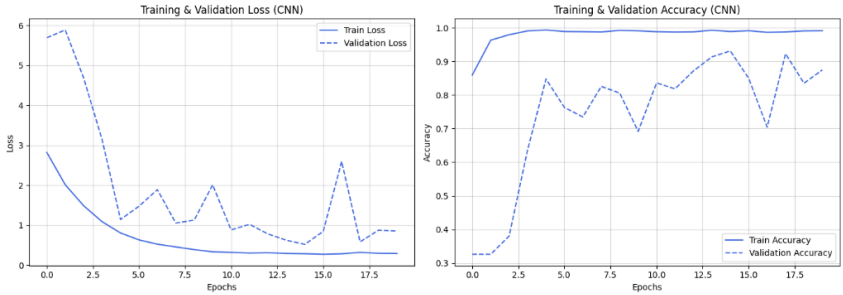
Setelah proses pelatihan selesai, langkah berikutnya adalah memvisualisasikan nilai *loss* dan *akurasi* selama proses pelatihan dan validasi. Visualisasi ini membantu dalam memahami apakah model mengalami *overfitting* atau *underfitting* dengan membandingkan performa pada data pelatihan dan validasi di setiap *epoch*.

Kode berikut digunakan untuk menampilkan grafik nilai *loss* dan akurasi selama proses pelatihan:

```
history_dict = cnn_history.history

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plt.plot(history_dict['loss'], linestyle='-', label='Train
Loss', color='royalblue')
plt.plot(history_dict['val_loss'],  linestyle='--',
label='Validation Loss', color='royalblue')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Training & Validation Loss (CNN)')
plt.grid(alpha=0.5)

plt.subplot(1, 2, 2)
plt.plot(history_dict['accuracy'], linestyle='-', label='Train
Accuracy', color='royalblue')

```

```
    plt.plot(history_dict['val_accuracy'],  linestyle='--',
17  label='Validation Accuracy', color='royalblue')
18  plt.xlabel('Epochs')
19  plt.ylabel('Accuracy')
20  plt.legend()
21  plt.title('Training & Validation Accuracy (CNN)')
22  plt.grid(alpha=0.5)
23
24  plt.tight_layout()
    plt.show()
```

Kode di atas menampilkan dua grafik dalam satu figur: grafik pertama (kiri) memvisualisasikan nilai *loss* pada data pelatihan dan validasi, sedangkan grafik kedua (kanan) menunjukkan akurasi pada data pelatihan dan validasi.



Gambar 32. Grafik performa model CNN selama proses pelatihan. Grafik kiri menunjukkan nilai loss pada data pelatihan dan validasi, sedangkan grafik kanan menampilkan akurasi pada data pelatihan dan validasi seiring bertambahnya jumlah epoch.

Grafik pelatihan menunjukkan bahwa nilai *loss* pada data pelatihan terus menurun, menandakan bahwa model mampu mempelajari pola dalam data. Namun, nilai *loss* pada data validasi berfluktuasi setelah beberapa *epoch*. Pada grafik akurasi, akurasi pada data pelatihan meningkat secara stabil, sedangkan akurasi pada data validasi mengalami fluktuasi setelah mencapai akurasi tinggi. Kedua grafik ini mengindikasikan bahwa model berhasil belajar dengan baik pada awal pelatihan, tetapi semakin sulit mempertahankan performanya terhadap data validasi.

4.4 Evaluasi Model CNN

Evaluasi dilakukan dengan menghitung nilai *loss* dan akurasi pada data validasi yang mencerminkan performa model terhadap data baru menggunakan fungsi `evaluate()`:

```
1  val_loss, val_accuracy = cnn_model.evaluate(val_dataset)
2
3  y_pred = np.argmax(cnn_model.predict(val_dataset), axis=1)
4  y_true = np.concatenate([y.numpy() for _, y in val_dataset],
   axis=0)
5
6  print(f"Validation Loss: {val_loss:.4f}")
7  print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Untuk menganalisis performa klasifikasi secara lebih rinci setelah evaluasi model, diperlukan label kelas asli (`y_true`) dan label kelas prediksi model (`y_pred`):

- `y_true` menyimpan label kelas yang benar untuk setiap sampel dalam dataset.
- `y_pred` menyimpan hasil prediksi model pada data validasi. Model menghasilkan probabilitas untuk setiap kelas, sehingga fungsi `np.argmax()` digunakan untuk mengubah menjadi label kelas yang diprediksi.

Dari hasil evaluasi, model CNN mencapai akurasi pelatihan sebesar 98.71% dengan *loss* 0.3006, menunjukkan model mampu mengenali pola dengan baik. Namun, pada data validasi, akurasi hanya mencapai 87.46% dengan *loss* sebesar 0.8555, yang lebih tinggi dibanding nilai *loss* pada pelatihan. Perbedaan ini mengindikasikan model mengalami *overfitting*, di mana model terlalu menyesuaikan diri dengan data pelatihan sehingga performanya menurun saat diuji dengan data baru.

Menampilkan Confusion Matrix

Confusion matrix adalah matriks yang merepresentasikan performa model klasifikasi dengan membandingkan hasil prediksi model terhadap label kelas sebenarnya. *Confusion matrix* membantu mengidentifikasi kesalahan yang dibuat oleh model, seperti ketika model salah mengklasifikasikan suatu kelas menjadi kelas lain (Heydarian et al., 2022).

Terdapat empat nilai pada *confusion matrix* yang menggambarkan bagaimana model membuat prediksi:

- ***True Positive* (TP)**. Model memprediksi kelas positif dan label sebenarnya juga positif.
- ***True Negative* (TN)**. Model memprediksi kelas negatif dan label sebenarnya juga negatif.
- ***False Positive* (FP)**. Model memprediksi kelas positif, tetapi label sebenarnya negatif (*Type* I *Error*).
- ***False Negative* (FN)**. Model memprediksi kelas negatif, tetapi label sebenarnya positif (*Type* II *Error*).

Tabel berikut mengilustrasikan bagaimana TP, TN, FP, dan FN dikelompokkan dalam *confusion matrix*:

Tabel 3. Ilustrasi confusion matrix.

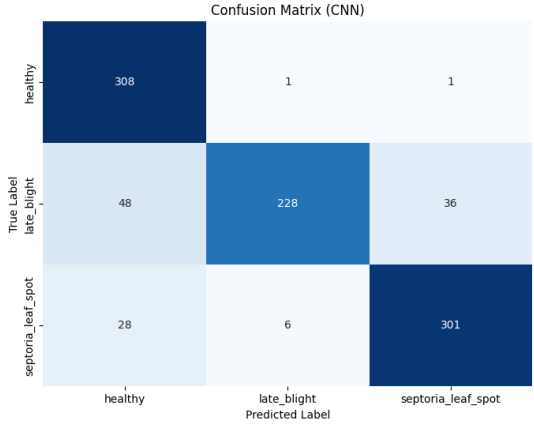
Actual / Predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

Nilai `y_true` dan `y_pred` digunakan untuk membentuk *confusion matrix*, yang divisualisasikan dalam bentuk diagram agar lebih mudah diinterpretasikan.

Kode di bawah ini menghasilkan *confusion matrix* untuk menganalisis performa klasifikasi model:

```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (CNN)")
plt.show()
```

Hasil eksekusi kode di atas menghasilkan *confusion matrix* sebagai berikut:



Gambar 33. Confusion matrix model CNN.

Dari hasil *confusion matrix* di atas, kelas *healthy* diklasifikasikan dengan benar sebanyak 308 gambar, dengan hanya 2 gambar yang salah prediksi. Kelas *late blight* memiliki 228 gambar yang diklasifikasikan dengan benar, tetapi terdapat 48 gambar yang salah diklasifikasikan sebagai *healthy* dan 36 gambar sebagai *septoria leaf spot*. Kelas *septoria leaf spot* diklasifikasikan dengan benar sebanyak 301 gambar, dengan 28 gambar salah diklasifikasikan sebagai *healthy* dan 6 gambar sebagai *late blight*. Kesalahan klasifikasi terutama terjadi pada kelas *late blight*, yang menunjukkan bahwa model mengalami kesulitan dalam membedakan gambar kelas *late blight* dari dua kelas lainnya.

Menampilkan *Classification Report*

Selain akurasi, evaluasi model juga dapat dilakukan menggunakan metrik *precision*, *recall*, dan F1-*score*. Metrik tersebut dihitung berdasarkan nilai TP, TN, FP, dan FN untuk setiap kelas. Evaluasi ini membantu memahami bagaimana model mengklasifikasikan sampel datadengan lebih rinci.

- ***Accuracy*** (akurasi) mengukur seberapa banyak data yang diklasifikasikan dengan benar dari seluruh data yang diuji. Namun, jika distribusi kelas tidak seimbang, akurasi bisa memberi kesan model bekerja dengan baik, padahal bisa saja model kesulitan mengenali kelas yang jumlah sampelnya lebih sedikit.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- ***Precision*** mengukur seberapa banyak prediksi positif yang benar dibandingkan dengan total prediksi positif model. *Precision* yang tinggi berarti model jarang mengklasifikasikan sampel negatif sebagai positif.

$$Precision = \frac{TP}{TP + FP}$$

- ***Recall*** (sensitivitas) mengukur seberapa baik model mendeteksi semua sampel positif yang sebenarnya. *Recall* yang tinggi berarti model jarang melewatkan sampel positif.

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

- **F1-*score*** adalah rata-rata harmonik dari *precision* dan *recall*. Metrik ini digunakan untuk menjaga keseimbangan antara keduanya, terutama saat distribusi kelas pada data tidak seimbang.

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Pasa klasifikasi multi-kelas, metrik *precision*, *recall*, dan F1-*score* dihitung untuk setiap kelas secara individual, lalu dirata-rata menggunakan:

- ***Macro average***: Rata-rata dari semua kelas, tanpa mempertimbangkan jumlah sampel di setiap kelas.
- ***Weighted average***: Rata-rata yang mempertimbangkan jumlah sampel di setiap kelas, sehingga kelas dengan lebih banyak sampel memiliki pengaruh lebih besar terhadap skor akhir.

Metrik evaluasi ini dapat dihitung menggunakan fungsi `classification_report` sebagai berikut:

```
1  report = classification_report(y_true, y_pred,
   target_names=CLASS_NAMES)
2  print(report)
```

Kode ini menghasilkan laporan yang menampilkan *precision*, *recall*, F1-*score*, dan *accuracy* untuk setiap kelas, serta *macro average* dan *weighted average* untuk keseluruhan model.

Tabel 4. Classification report model CNN.

	precision	recall	f1-score	support
healthy	0.80	0.99	0.89	310
late_blight	0.97	0.73	0.83	312
septoria_leaf_spot	0.89	0.90	0.89	335
accuracy			0.87	957
macro avg	0.89	0.87	0.87	957
weighted avg	0.89	0.87	0.87	957

Hasil *classification report* menunjukkan bahwa model memiliki *accuracy* sebesar 87%. Namun, terdapat perbedaan dalam performa tiap kelas. Kelas *healthy* memiliki *recall* tinggi (99%), menunjukkan hampir semua sampel *healthy* diklasifikasikan dengan benar, tetapi *precision*-nya hanya 80%, yang berarti beberapa sampel kelas lain salah diklasifikasikan sebagai *healthy*. Sebaliknya, kelas *late blight* memiliki *precision* tinggi (97%), tetapi *recall*-nya rendah (73%), menunjukkan model sering gagal mengenali sampel dari kelas ini. Kelas *septoria leaf spot* memiliki keseimbangan antara *precision* (89%) dan *recall* (90%), menunjukkan model cukup stabil dalam mengidentifikasi kelas ini. Dari *confusion matrix*, kesalahan klasifikasi lebih banyak terjadi pada kelas *late blight*, yang sering diklasifikasikan sebagai kategori lain.

4.5 Menguji Model pada Gambar Baru

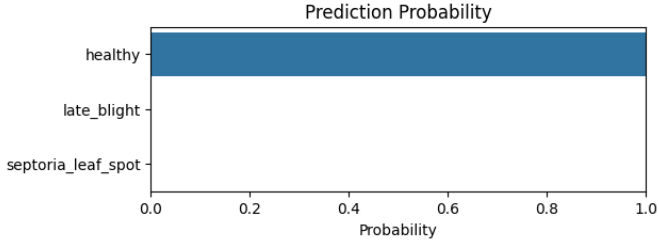
Pengujian model dilakukan dengan memberikan gambar uji sebagai *input*, lalu model yang telah dilatih akan memprediksi kelas yang paling sesuai. Hasil prediksi divisualisasikan dalam bentuk gambar serta grafik probabilitas untuk memahami tingkat keyakinan model terhadap klasifikasi yang diberikan.

```
test_class_index = CLASS_NAMES.index("septoria_leaf_spot")
test_label = TEST_CLASSES[test_class_index]
test_image = (TEST_IMAGES[test_class_index] *
255).astype("uint8")
test_image_uint8 = np.clip(test_image * 255, 0,
255).astype("uint8")

predictions = cnn_model.predict(tf.expand_dims(test_image,
axis=0))
predicted_class_index = np.argmax(predictions, axis=1)[0]
predicted_class_name = CLASS_NAMES[predicted_class_index]

plt.figure(figsize=(4, 4))
plt.imshow(test_image_uint8)
plt.title(f"Predicted: {predicted_class_name}\nTrue:
{test_label}")
plt.axis("off")

plt.figure(figsize=(6, 2))
sns.barplot(x=predictions[0], y=CLASS_NAMES, orient='h')
plt.xlabel("Probability")
plt.xlim([0, 1])
```

```
plt.title("Prediction Probability")
plt.show()
```

Kode di atas menghasilkan kelas prediksi model, label kelas sebenarnya dari gambar uji, serta grafik probabilitas untuk setiap kelas.

Gambar 34. Hasil prediksi model CNN pada gambar daun uji. Daun yang sebenarnya terinfeksi septoria leaf spot salah diklasifikasikan sebagai daun sehat (healthy)



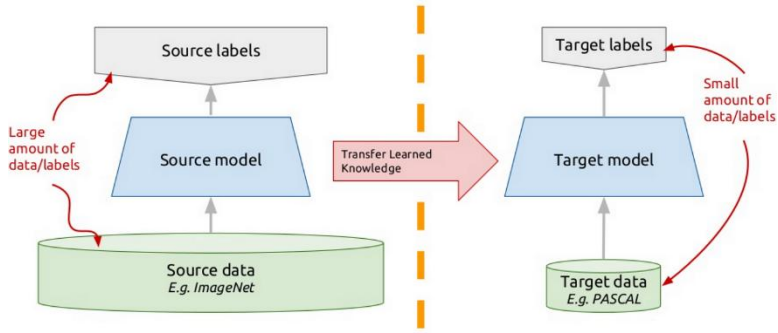
Gambar 35. Grafik probabilitas prediksi model CNN untuk gambar daun uji. Model memiliki tingkat kepercayaan tinggi terhadap kelas healthy, meskipun daun sebenarnya terinfeksi septoria leaf spot, menunjukkan kesalahan klasifikasi

Pada contoh di atas, gambar yang diuji adalah daun yang terinfeksi *septoria leaf spot*, tetapi model salah mengklasifikasikannya sebagai daun sehat. Grafik probabilitas menunjukkan bahwa model memiliki tingkat keyakinan tinggi terhadap prediksi yang salah tersebut. Kesalahan ini menunjukkan bahwa model kesulitan membedakan daun terinfeksi *septoria leaf spot* dari daun sehat.

Bab 5: *Transfer Learning* untuk Meningkatan Performa Model

Transfer learning adalah metode yang memungkinkan model yang telah dilatih sebelumnya digunakan untuk menyelesaikan tugas baru yang serupa. Metode ini sangat bermanfaat ketika jumlah data pelatihan untuk tugas baru terbatas, karena model dapat memanfaatkan fitur-fitur dasar yang telah dipelajari dari dataset sebelumnya.

Sebagai contoh, model yang telah dilatih pada jutaan gambar dari dataset ImageNet[25] telah memahami fitur-fitur dasar gambar seperti bentuk, warna, dan tekstur. Fitur dasar tersebut dapat digunakan kembali untuk menyelesaikan tugas baru, misalnya klasifikasi gambar penyakit tanaman dengan dataset yang lebih kecil. Dengan memanfaatkan model *pretrained*, proses pelatihan tidak perlu dilakukan dari awal, sehingga lebih cepat dan menghemat sumber daya komputasi. Proses ini sering disebut *fine-tuning*.



Gambar 36. Konsep transfer learning.

Gambar di atas mengilustrasikan konsep transfer learning:

- ***Source model* dan *source data***. *Source model* dilatih menggunakan dataset besar seperti ImageNet, sehingga mampu mengenali pola-pola dasar dalam gambar, seperti tekstur dan bentuk.
- ***Transfer knowledge***. Fitur dan bobot yang telah dipelajari oleh *source model* (sering disebut sebagai model *pretrained*) dapat ditransfer ke model target untuk tugas baru yang spesifik.
- **Target model dan target data**. Model target dilatih dengan dataset lebih kecil untuk menyelesaikan tugas tertentu. Dalam tahap ini, *fine-tuning* dilakukan pada beberapa *layer* agar model mengenali karakteristik khusus dalam data baru.

[25] https://www.image-net.org

Proses *transfer learning* untuk klasifikasi gambar terdiri dari beberapa tahap utama, sebagai berikut:

1. **Normalisasi gambar *input***. Gambar *input* dinormalisasi agar sesuai dengan distribusi data yang digunakan saat melatih model *pretrained*.
2. **Menyiapkan arsitektur model *pretrained***. Mengunduh model *pretrained* beserta bobotnya.
3. **Memodifikasi *layer* akhir**. Beberapa *layer* terakhir dihapus dan diganti dengan *layer* baru yang disesuaikan dengan dataset dan tugas yang akan diselesaikan.
4. **Menghubungkan *layer* baru**. *Layer* baru ditambahkan ke model *pretrained*, dengan jumlah *neuron* pada *output layer* disesuaikan dengan jumlah kelas pada dataset target.
5. **Membekukan bobot model *pretrained***. Bobot model *pretrained* tidak dilatih kembali (freeze) karena bobot tersebut sudah optimal untuk mengenali fitur dasar. Hanya *layer* baru yang bobotnya diperbarui.
6. **Melatih model (*fine-tuning*)**. Model dilatih kembali untuk menyesuaikan *layer* baru dengan dataset target selama beberapa *epoch* hingga model mencapai performa optimal.

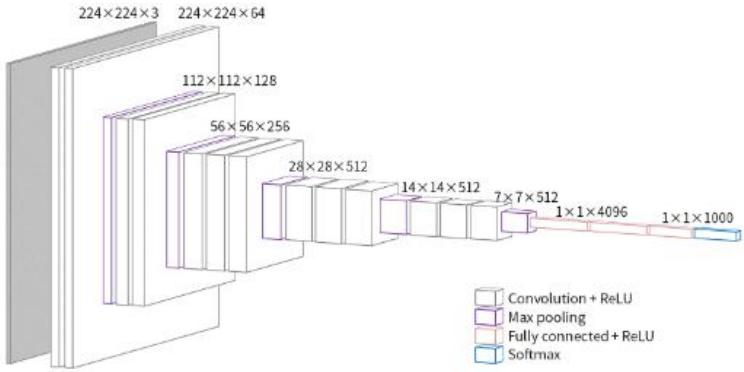
Transfer learning memungkinkan pengembangan model *deep learning* yang lebih efisien, terutama dalam aplikasi dunia nyata yang membutuhkan akurasi tinggi dengan sumber daya komputasi terbatas.

5.1 Memahami Arsitektur VGG

VGG adalah singkatan dari *Visual Geometry Group*[26] yang merupakan kelompok riset dari Universitas Oxford yang mengembangkan *deep neural network* untuk tugas pengenalan objek. Arsitektur VGG merupakan *Convolutional Neural Network* (CNN) yang menggunakan banyak *layer* konvolusi. Nama VGG-16 dan VGG-19 mengacu pada jumlah *layer* konvolusi yang digunakan, yaitu 16 dan 19 *layer*.

Arsitektur VGG menjadi fondasi bagi banyak model pengenalan objek modern, dirancang untuk menangani tugas pengenalan objek berskala besar seperti pada dataset ImageNet. Keunggulannya terletak pada struktur yang sederhana namun efektif, menjadikannya salah satu arsitektur CNN yang paling populer dan banyak digunakan dalam berbagai aplikasi klasifikasi gambar.

[26] https://www.robots.ox.ac.uk/~vgg



Gambar 37. Struktur arsitektur VGG yang menggunakan filter konvolusi kecil (3×3) dan pooling untuk ekstraksi fitur, serta fully connected layer untuk klasifikasi.

Model VGG dibangun menggunakan filter konvolusi kecil untuk mengekstraksi fitur dari gambar secara lebih efisien. Berikut adalah komponen utama dalam arsitektur VGG:

- ***Input layer***. Model menerima input gambar dengan ukuran 224×224 piksel. Pada kompetisi ImageNet, gambar awal dipotong bagian tengahnya agar semua gambar memiliki ukuran *input* yang sama.
- ***Convolutional layers.*** Arsitektur *VGG* menggunakan filter 3×3 untuk menangkap fitur kecil dalam gambar, seperti tepi dan tekstur. Setiap *layer* konvolusi diikuti oleh fungsi aktivasi ReLU, yang membantu mengurangi waktu pelatihan dan meningkatkan stabilitas model.
- ***Hidden layers***: Semua *hidden layer* dalam arsitektur VGG menggunakan fungsi aktivasi ReLU. Tidak seperti model lain, VGG tidak menggunakan *Local Response Normalization* (LRN) karena metode ini meningkatkan konsumsi memori tanpa peningkatan akurasi yang signifikan.
- ***Fully connected layers***. Model VGG memiliki tiga *fully connected layers*, di mana dua *layer* pertama terdiri dari 4096 unit, sementara *layer* terakhir memiliki 1000 unit yang mewakili jumlah kelas dalam dataset ImageNet.

5.2 Mendefinisikan Model VGG16 untuk *Transfer Learning*

Langkah ini terdiri dari dua tahapan utama: mendefinisikan model *pretrained* VGG16 dan mendefinisikan *layer* akhir. Model *pretrained* VGG16 dimanfaatkan untuk mengekstraksi fitur. Sedangkan, *layer* tambahan digunakan untuk menyesuaikan model dengan jumlah kelas pada dataset dan menghasilkan output klasifikasi akhir.

Menggunakan Model *Pretrained* VGG16 sebagai *Feature Extractor*

Kode di bawah ini digunakan untuk mendefinsikan model *pretrained* VGG16 yang telah dilatih dengan dataset ImageNet (`weights='imagenet'`). Model *pretrained* VGG16 digunakan sebagai *feature extractor* untuk mengekstrak fitur dari gambar *input*.

```
pretrained_model = VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=(IMG_SIZE[0], IMG_SIZE[1], 3)
    )

pretrained_model.trainable = False

pretrained_model.summary()
```

Pada kode di atas, `include_top=False` digunakan untuk menghapus *layer* klasifikasi terakhir agar model hanya berfungsi sebagai *feature extractor* tanpa *fully connected layer*. Hal ini memungkinkan penyesuaian arsitektur model dengan menambahkan *layer* klasifikasi baru yang sesuai dengan tugas klasifikasi penyakit tanaman.

Sementara itu, `trainable=False` digunakan untuk membekukan bobot pada model *pretrained*, sehingga bobot yang telah dipelajari dari dataset ImageNet tetap digunakan tanpa perubahan. Dengan demikian, model tidak akan mengalami pembaruan bobot selama pelatihan ulang, yang membantu mempercepat proses pelatihan dan mengurangi risiko *overfitting*.

Gambar di bawah ini menampilkan arsitektur VGG16 menggunakan fungsi `pretrained_model.summary()`.

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 256, 256, 3)	0
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1,792
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36,928
block1_pool (MaxPooling2D)	(None, 128, 128, 64)	0
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73,856
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147,584
block2_pool (MaxPooling2D)	(None, 64, 64, 128)	0
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295,168
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590,080
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590,080
block3_pool (MaxPooling2D)	(None, 32, 32, 256)	0
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 16, 16, 512)	0
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 8, 8, 512)	0

Total params: 14,714,688 (56.13 MB)
Trainable params: 0 (0.00 B)
Non-trainable params: 14,714,688 (56.13 MB)

Gambar 38. Ringkasan arsitektur model pretrained VGG16, menunjukkan jenis layer, output shape, dan jumlah parameter yang digunakan.

Arsitektur VGG16 terdiri dari 16 layer utama, yang sebagian besar merupakan kombinasi `Conv2D` dan `MaxPooling2D`. Layer awal menangkap fitur dasar seperti tepi dan tekstur, sementara layer lebih dalam mengenali pola yang lebih kompleks.

Menambahkan *Layer* Akhir untuk Klasifikasi

Langkah selanjutnya adalah mendefenisikan *layer* terkahir untuk menyesuaikan model dengan tugas yang sesuai dengan dataset yang digunakan. Layer tambahan ini menerima fitur yang telah diekstrak oleh model *pretrained* dan mengubahnya menjadi prediksi akhir.

```
vgg_model = models.Sequential([
    layers.Input(shape=(IMG_SIZE[0], IMG_SIZE[1], 3)),
    pretrained_model,
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),
    layers.Dense(len(CLASS_NAMES), activation='softmax')
])

vgg_model.summary()
```

Pada kode di atas, fungsi `Flatten()` digunakan untuk mengubah *output* dari *layer* konvolusi terakhir menjadi vektor satu dimensi agar dapat diproses oleh *fully connected layer*. Selanjutnya, *layer* `Dense()` dengan 128 *neuron* dan aktivasi ReLU ditambahkan untuk membantu model mengenali hubungan fitur yang lebih kompleks. Untuk mengurangi risiko *overfitting*, `Dropout(0.3)` digunakan dengan menonaktifkan 30% *neuron* secara acak selama pelatihan. *Layer* terakhir, `Dense(len(CLASS_NAMES), activation='softmax')`, berfungsi sebagai *output layer* yang mengonversi *output* menjadi probabilitas untuk setiap kelas.

Gambar di bawah ini menampilkan arsitektur model akhir setelah ditambahkan *layer* tambahan menggunakan fungsi `vgg_model.summary()`.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 8, 8, 512)	14,714,688
flatten_1 (Flatten)	(None, 32768)	0
dense_2 (Dense)	(None, 128)	4,194,432
dropout_2 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 3)	387

Total params: 18,909,507 (72.13 MB)
Trainable params: 4,194,819 (16.00 MB)
Non-trainable params: 14,714,688 (56.13 MB)

Gambar 39. Ringkasan arsitektur model pretrained VGG16, menunjukkan jenis layer, output shape, dan jumlah parameter yang digunakan.

Model akhir memiliki total 18 juta parameter, dengan 14 juta di antaranya merupakan *non-trainable parameters* yang berasal dari VGG16 dan tidak diperbarui selama pelatihan. Sementara itu, 4 juta parameter *trainable* terdapat pada *layer*

tambahan yang akan dilatih untuk menyesuaikan model dengan tugas klasifikasi baru.

5.3 Melakukan *Fine-Tuning*

Fine-tuning dilakukan dengan menyesuaikan kembali bobot pada layer tambahan yang ditambahkan ke model pretrained VGG16. Proses fine-tuning dimulai dengan mendefinisikan fungsi loss, algoritma optimasi, dan metrik evaluasi yang akan digunakan dalam pelatihan.

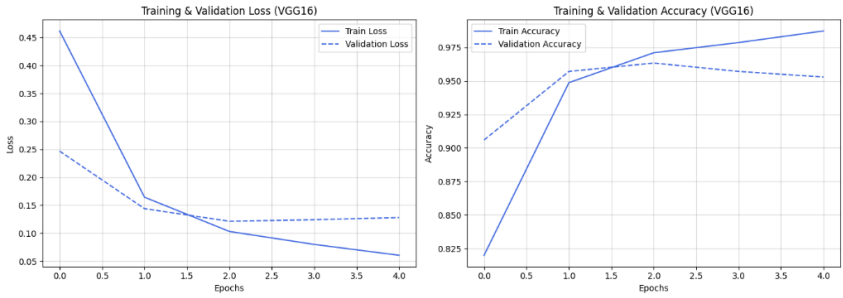
```
LOSS_FUNCTION = losses.SparseCategoricalCrossentropy()

LEARNING_RATE = 0.0001
OPTIMIZER = optimizers.Adam(learning_rate=LEARNING_RATE)

METRICS = [
    metrics.SparseCategoricalAccuracy(name='accuracy')
]

vgg_model.compile(optimizer=OPTIMIZER, loss=LOSS_FUNCTION,
metrics=METRICS)
```

Kode di atas mirip dengan yang digunakan saat melatih model CNN sebelumnya. Setelah kompilasi (`vgg_model.compile()`), model dilatih menggunakan dataset yang telah diproses.

```
EPOCHS = 5

start_time = time.time()

vgg_history = vgg_model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=EPOCHS,
    steps_per_epoch=len(train_dataset),
    validation_steps=len(val_dataset)
)

end_time = time.time()

training_time = end_time - start_time
print(f"Total Training Time: {training_time:.2f} seconds")
```

Proses pelatihan ini serupa dengan model CNN sebelumnya, namun jumlah *epoch* dikurangi menjadi lima karena model sudah memiliki bobot awal dari model *pretrained* VGG16. Penggunaan jumlah *epoch* yang lebih sedikit bertujuan untuk menyesuaikan bobot *layer* tambahan tanpa mengubah fitur yang telah dipelajari sebelumnya.

Waktu komputasi untuk melatih model VGG16 dengan 5 *epoch* adalah 224.42detik. Pengurangan jumlah *epoch* dalam fine-tuning mempercepat pelatihan model dan mengurangi risiko *overfitting*. Gambar di bawah ini menunjukkan grafik *loss* dan akurasi selama pelatihan fine-tuning.



Gambar 40. Grafik performa model VGG16 selama proses pelatihan. Grafik kiri menunjukkan nilai loss pada data pelatihan dan validasi, sedangkan grafik kanan menampilkan akurasi pada data pelatihan dan validasi seiring bertambahnya jumlah epoch.

Grafik pelatihan di atas menunjukkan bahwa nilai *loss* pada data pelatihan terus menurun dengan stabil, menandakan bahwa model mampu menyesuaikan bobotnya terhadap pola dalam data. Sementara itu, *loss* pada data validasi juga mengalami penurunan tanpa fluktuasi yang signifikan, yang menunjukkan bahwa model tidak mengalami *overfitting* selama *fine-tuning*.

Pada grafik akurasi, akurasi pada data pelatihan meningkat dengan cepat dalam beberapa *epoch* pertama dan kemudian stabil. Akurasi validasi juga mengalami peningkatan yang sejalan dengan akurasi pelatihan, menunjukkan bahwa model dapat menggeneralisasi dengan baik. Tidak ada tanda-tanda akurasi validasi yang menurun drastis atau berfluktuasi secara tidak menentu, yang berarti *fine-tuning* berhasil meningkatkan performa model tanpa kehilangan kemampuan generalisasi.

Dibandingkan dengan model CNN sebelumnya, hasil *fine-tuning* pada model VGG16 menunjukkan peningkatan stabilitas dan efisiensi pelatihan. Dengan jumlah *epoch* yang lebih sedikit, model tetap dapat belajar secara efektif dari data tanpa risiko *overfitting* yang signifikan.

5.4 Evaluasi Model VGG16

Setelah model dilatih, langkah berikutnya adalah mengevaluasi performanya pada data validasi. Evaluasi dilakukan dengan menghitung nilai *loss* dan akurasi menggunakan fungsi `evaluate()`, serta menghasilkan prediksi yang disimpan pada variabel `y_pred`.

```
val_loss, val_accuracy = vgg_model.evaluate(val_dataset)

y_pred = np.argmax(vgg_model.predict(val_dataset), axis=1)
y_true = np.concatenate([y.numpy() for _, y in val_dataset],
axis=0)

print(f"Validation Loss: {val_loss:.4f}")
print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
```

Dari hasil evaluasi, model VGG16 mencapai akurasi pelatihan sebesar 99.04% dengan *loss* 0.0572, menunjukkan bahwa model dapat mengenali pola dalam data pelatihan dengan sangat baik. Akurasi pada data validasi mencapai 95.30% dengan *loss* 0.1281, yang jauh lebih rendah dibandingkan model CNN sebelumnya. Performa ini mengindikasikan bahwa penggunaan model *pretrained* dengan *fine-tuning* berhasil meningkatkan generalisasi model pada data baru.

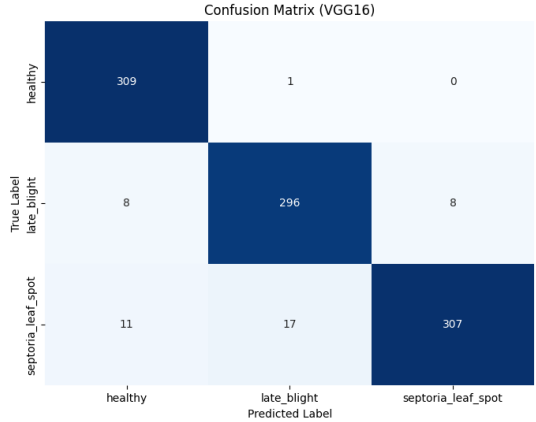
Namun, selisih akurasi pelatihan dan validasi masih menunjukkan sedikit kecenderungan *overfitting*. Nilai *loss* yang lebih tinggi pada validasi dibandingkan pelatihan menandakan bahwa model lebih optimal mengenali pola dalam data pelatihan dibandingkan data yang belum pernah dilihat sebelumnya.

Menampilkan *Confusion Matrix* dan *Classification Report*

Untuk memahami lebih lanjut performa model dalam mengklasifikasikan setiap kelas, *confusion matrix* dan *classification report* ditampilkan.

```
cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False,
xticklabels=CLASS_NAMES, yticklabels=CLASS_NAMES)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix (VGG16)")
plt.show()

report = classification_report(y_true, y_pred,
target_names=CLASS_NAMES)
print(report)
```

Seperti yang telah dijelaskan sebelumnya, *confusion matrix* menampilkan jumlah prediksi benar dan salah untuk setiap kelas, sedangkan *classification report* memberikan metrik evaluasi lebih rinci seperti *precision*, *recall*, F1-*score*.



Gambar 41. Confusion matrix model VGG16.

Tabel 5. Classification report model VGG16.

	precision	recall	f1-score	support
healthy	0.94	1.00	0.97	310
late_blight	0.94	0.95	0.95	312
septoria_leaf_spot	0.97	0.92	0.94	335
accuracy			0.95	957
macro avg	0.95	0.95	0.95	957
weighted avg	0.95	0.95	0.95	957

Hasil *confusion matrix* menunjukkan bahwa model VGG16 mampu mengklasifikasikan sebagian besar gambar pada data validasi dengan benar. Kelas *healthy* diklasifikasikan dengan benar sebanyak 309 gambar, dengan hanya 1 gambar salah prediksi sebagai *late blight*. Kelas *late blight* memiliki 296 prediksi benar, tetapi masih terdapat 8 gambar yang salah diklasifikasikan sebagai kelas *healthy* dan 8 sampel lain diklasifikasikan sebagai kelas *septoria leaf spot*. Sementara itu, kelas *septoria leaf spot* memiliki 307 prediksi benar, dengan 11 gambar salah diklasifikasikan sebagai *healthy* dan 17 gambar diklasifikasikan sebagai *late blight*.

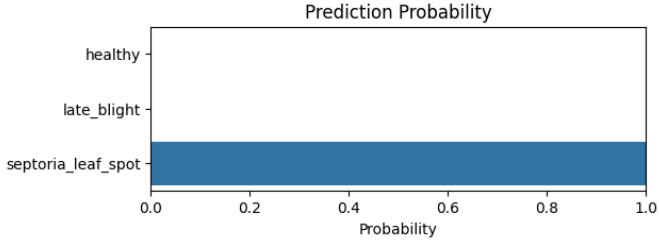
Classification report menunjukkan bahwa model VGG16 memiliki *precision*, *recall*, dan F1-*score* yang cukup tinggi pada setiap kelas, dengan nilai rata-rata keseluruhan sebesar 0.95. Nilai *precision* yang tinggi mengindikasikan bahwa model jarang membuat prediksi yang salah untuk setiap kelas. *Recall* yang tinggi menunjukkan bahwa model mampu menangkap hampir semua sampel dalam masing-masing kategori dengan baik. F1-*score* yang mendekati *precision* dan *recall* menunjukkan keseimbangan performa antara kedua metrik tersebut.

Meskipun hasil evaluasi menunjukkan performa yang baik, kesalahan klasifikasi pada kelas *late blight* dan *septoria leaf spot* masih terjadi. Hal ini dapat disebabkan oleh karakteristik visual yang mirip antara kedua kategori tersebut, sehingga model mengalami kesulitan dalam membedakannya. Untuk meningkatkan performa, metode seperti augmentasi data atau penyesuaian hyperparameter dapat dipertimbangkan guna mengurangi kesalahan klasifikasi lebih lanjut.

Menguji Model VGG16 pada Gambar Baru

Setelah model dilatih dan dievaluasi, langkah berikutnya adalah menguji performanya terhadap gambar baru. Proses ini dilakukan dengan memberikan gambar uji yang sama ke model VGG16 dan melihat prediksi yang dihasilkan. Model akan memproses gambar tersebut dan mengklasifikasikannya ke dalam salah satu kategori yang telah dipelajari.

Gambar 42. Hasil prediksi model VGG16 pada gambar daun uji. Model berhasil mengklasifikasikan daun yang terinfeksi septoria leaf spot dengan benar.



Gambar 43. Grafik probabilitas prediksi model VGG16 untuk gambar daun uji. Model memiliki tingkat kepercayaan tinggi terhadap kelas septoria leaf spot, menunjukkan bahwa model mampu mengidentifikasi penyakit ini dengan baik.

Hasil pengujian model VGG16 pada data baru menunjukkan peningkatan akurasi dibandingkan model CNN sebelumnya. Jika sebelumnya model CNN salah mengklasifikasikan gambar, model VGG16 berhasil memprediksi dengan benar sebagai *septoria leaf spot*. Grafik probabilitas juga menunjukkan bahwa model sangat yakin dengan prediksinya, dengan probabilitas mencapai 100%.

Keberhasilan ini mencerminkan keunggulan *transfer learning* dalam meningkatkan performa model, terutama dalam membedakan kelas yang sebelumnya sulit diklasifikasikan. Model VGG16 mampu mengenali pola yang lebih kompleks dengan lebih baik dibandingkan model CNN yang dilatih dari awal, menunjukkan efektivitas pemanfaatan model *pre-trained* untuk tugas klasifikasi gambar.

Bab 6: Penutup

Penerapan *deep learning* dalam deteksi penyakit tanaman melalui *computer vision* telah menunjukkan efektivitas tinggi dalam meningkatkan akurasi identifikasi serta efisiensi pemantauan kesehatan tanaman. Meskipun demikian, masih terdapat ruang untuk pengembangan lebih lanjut agar teknologi ini dapat lebih optimal dan luas diterapkan dalam skala pertanian yang lebih besar.

6.1 Potensi Pengembangan Selanjutnya

Pengembangan *computer vision* dalam diagnosis penyakit tanaman masih memiliki banyak peluang untuk ditingkatkan. Salah satunya adalah dengan mengadopsi model berbasis transformer dan *self-supervised learning*, yang mampu meningkatkan efisiensi dan akurasi klasifikasi gambar. Selain itu, integrasi dengan sensor hiperspektral dan *Internet of Things* (IoT) memungkinkan sistem untuk tidak hanya mengenali gejala penyakit dari citra visual, tetapi juga menganalisis faktor lingkungan yang dapat mempengaruhi kesehatan tanaman. Dengan memanfaatkan teknologi ini, deteksi penyakit dapat dilakukan secara lebih akurat dan adaptif terhadap kondisi pertanian yang dinamis.

Selain peningkatan akurasi, pengembangan sistem *edge computing* menjadi solusi untuk mengatasi keterbatasan infrastruktur komputasi di lapangan. Dengan teknologi ini, analisis data dapat dilakukan langsung pada perangkat yang lebih ringan, seperti *drone* atau kamera pintar, tanpa harus mengandalkan pemrosesan berbasis *cloud*. Hal ini tidak hanya mengurangi ketergantungan pada konektivitas internet, tetapi juga memungkinkan petani untuk mendapatkan hasil deteksi penyakit secara *real-time*. Implementasi teknologi ini dapat memberikan manfaat besar terutama bagi petani kecil dan menengah yang memiliki keterbatasan dalam akses terhadap infrastruktur komputasi canggih.

6.2 Tantangan

Meskipun kemajuan signifikan telah dicapai, tantangan utama dalam implementasi teknologi ini meliputi kebutuhan dataset besar dan beragam untuk melatih model secara optimal. Variasi kondisi lingkungan, pencahayaan, serta latar belakang gambar dapat memengaruhi akurasi deteksi. Selain itu, keterbatasan infrastruktur komputasi di daerah pertanian terpencil dan biaya pengadopsian teknologi menjadi kendala dalam penerapan skala besar. Oleh karena itu, diperlukan strategi yang berkelanjutan, seperti kolaborasi antara peneliti (perguruan tinggi), industri, dan pemerintah untuk memastikan teknologi ini dapat diakses dan dimanfaatkan secara luas dalam mendukung pertanian modern.

Daftar Pustaka

Cong, S., & Zhou, Y. (2023). A review of convolutional neural network architectures and their optimizations. *Artificial Intelligence Review*, *56*(3), 1905–1969. https://doi.org/10.1007/s10462-022-10213-5

Gupta, D. K., Pagani, A., Zamboni, P., & Singh, A. K. (2024). AI-powered revolution in plant sciences: Advancements, applications, and challenges for sustainable agriculture and food security. *Explor Foods Foodomics*. https://doi.org/10.37349/eff.2024.00045

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition* (No. arXiv:1512.03385). arXiv. https://doi.org/10.48550/arXiv.1512.03385

Heydarian, M., Doyle, T. E., & Samavi, R. (2022). MLCM: Multi-Label Confusion Matrix. *IEEE Access*, *10*, 19083–19095. IEEE Access. https://doi.org/10.1109/ACCESS.2022.3151048

Hughes, D. P., & Salathe, M. (2016). *An open access repository of images on plant health to enable the development of mobile disease diagnostics* (No. arXiv:1511.08060). arXiv. https://doi.org/10.48550/arXiv.1511.08060

Jiang, P., Chen, Y., Liu, B., He, D., & Liang, C. (2019). Real-Time Detection of Apple Leaf Diseases Using Deep Learning Approach Based on Improved Convolutional Neural Networks. *IEEE Access*, *7*, 59069–59080. IEEE Access. https://doi.org/10.1109/ACCESS.2019.2914929

Kamilaris, A., & Prenafeta-Boldú, F. X. (2018). Deep learning in agriculture: A survey. *Computers and Electronics in Agriculture*, *147*, 70–90. https://doi.org/10.1016/j.compag.2018.02.016

Mahlein, A.-K. (2016). Plant Disease Detection by Imaging Sensors – Parallels and Specific Demands for Precision Agriculture and Plant Phenotyping. *Plant Disease*, *100*(2), 241–251. https://doi.org/10.1094/PDIS-03-15-0340-FE

Mduma, N., & Leo, J. (2023). Dataset of banana leaves and stem images for object detection, classification and segmentation: A case of Tanzania. *Data in Brief*, *49*, 109322. https://doi.org/10.1016/j.dib.2023.109322

Mohammed, S., Budach, L., Feuerpfeil, M., Ihde, N., Nathansen, A., Noack, N., Patzlaff, H., Naumann, F., & Harmouch, H. (2024). *The Effects of Data Quality on Machine Learning Performance* (No. arXiv:2207.14529). arXiv. https://doi.org/10.48550/arXiv.2207.14529

Mustofa, S., Ahad, M. T., Emon, Y. R., & Sarker, A. (2024). BDPapayaLeaf: A dataset of papaya leaf for disease detection, classification, and analysis. *Data in Brief*, *57*, 110910. https://doi.org/10.1016/j.dib.2024.110910

Parraga-Alava, J., Cusme, K., Loor, A., & Santander, E. (2019). RoCoLe: A *robusta* coffee leaf images dataset for evaluation of machine learning based

methods in plant diseases recognition. *Data in Brief*, *25*, 104414. https://doi.org/10.1016/j.dib.2019.104414

Russell, S., & Norvig, P. (2022). *Artificial Intelligence: A Modern Approach* (4th ed.). https://aima.cs.berkeley.edu

Sayeem, A. R., Omi, J. F., Hasan, M., Mojumdar, M. U., & Chakraborty, N. R. (2025). IDDMSLD: An image dataset for detecting Malabar spinach leaf diseases. *Data in Brief*, *58*, 111293. https://doi.org/10.1016/j.dib.2025.111293

Shoib, Md. M. H., Saeem, S., Tonima, A. B. A., & Mojumdar, M. U. (2025). IDBGL: A unique image dataset of black gram (Vigna mungo) leaves for disease detection and classification. *Data in Brief*, *59*, 111347. https://doi.org/10.1016/j.dib.2025.111347

Shunk, J. (2022). *Neuron-Specific Dropout: A Deterministic Regularization Technique to Prevent Neural Networks from Overfitting & Reduce Dependence on Large Training Samples* (No. arXiv:2201.06938). arXiv. https://doi.org/10.48550/arXiv.2201.06938

Simonyan, K., & Zisserman, A. (2015). *Very Deep Convolutional Networks for Large-Scale Image Recognition* (No. arXiv:1409.1556). arXiv. https://doi.org/10.48550/arXiv.1409.1556

Tan, M., & Le, Q. V. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* (No. arXiv:1905.11946). arXiv. https://doi.org/10.48550/arXiv.1905.11946

Venbrux, M., Crauwels, S., & Rediers, H. (2023). Current and emerging trends in techniques for plant pathogen detection. *Frontiers in Plant Science*, *14*, 1120968. https://doi.org/10.3389/fpls.2023.1120968

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, *2018*(1), 7068349. https://doi.org/10.1155/2018/7068349

Wang, H., Gu, J., & Wang, M. (2023). A review on the application of computer vision and machine learning in the tea industry. *Frontiers in Sustainable Food Systems*, *7*. https://doi.org/10.3389/fsufs.2023.1172543

Ziaee, A., & ÇAno, E. (2023). Batch Layer Normalization A new normalization layer for CNNs and RNNs. *Proceedings of the 6th International Conference on Advances in Artificial Intelligence*, 40–49. https://doi.org/10.1145/3571560.3571566

Tentang Penulis

Ir. Kuncahyo Setyo Nugroho, S.Kom., M.Kom.

Peneliti di *Bioinformatics & Data Science Research Center* (BDSRC) dan NVIDIA BINUS *Artificial Intelligence Research & Development Center* (AIRDC). Menyelesaikan program sarjana komputer dengan konsentrasi *Business Intelligence* di Fakultas Teknik, Universitas Widya Gama pada tahun 2019. Memperoleh gelar magister komputer dengan konsentrasi *Intelligent Systems* di Fakultas Ilmu Komputer, Universitas Brawijaya pada tahun 2022. Menyelesaikan program profesi insinyur di bidang *Software Engineering* di Sekolah Interdisiplin Manajemen dan Teknologi, Institut Teknologi Sepuluh Nopember pada tahun 2023. Saat ini, menjadi NVIDIA *Deep Learning Institute* (DLI) *Certified Instructor* dan *University Ambassador* di Indonesia. Memiliki minat pada bidang penelitian *affective computing-enginerring* dan *artificial intelligence*, dengan fokus pada pengembangan dan penerapan teknologi AI dalam berbagai domain.

Mahmud Isnan, S.Kom., M.Eng.

Meraih Pendidikan S1 dibidang Sistem Informasi dari STMIK Indonesia Padang dan S2 dibidang *Artificial Intelligence* and *Internet of Things* dari Thammasat University, *joint degree* dengan *National Science and Technology Development Agency* (NSTDA), Thailand dan Tokyo *Institute of Technology*, Jepang. Saat ini dia sedang berprofesi sebagai dosen *Data Science* di Bina Nusantara University dan sebagai peneliti di Artificial Intelligence Research and Development Center (AIRDC). Mahmud juga menjadi *certified instructor* NVIDIA Deep Learning Institute (DLI).

Teddy Suparyanto, S.Pd., M.T.I.

Menjadi *Research Associate* di *Bioinformatics & Data Science Research Center* (BDSRC) serta AI *Research & Development Center* (AIRDC), sekaligus dosen Teknik Informatika Pertanian di Institut Pertanian STIPER, Yogyakarta, Indonesia. Memiliki pemahaman mendalam dalam pengembangan dan manajemen sistem IT di sektor pendidikan dan agroindustri. Lulusan Fakultas Pendidikan Guru Matematika, dengan latar belakang pendidikan dalam bidang matematika, keterampilan mengajar, psikologi pendidikan, filsafat pedagogi, strategi pengajaran, serta media pembelajaran. Keahliannya dalam bidang statistik sangat membantu dalam proses analisis data. Menyelesaikan program magister di bidang Teknologi Informasi di Universitas Bina Nusantara.

Prof. Bens Pardamean, B.Sc., M.Sc., Ph.D.

Mendapatkan gelar sarjana dari program studi Ilmu Komputer di California State University, gelar magister dari program studi Pendidikan Komputer California State University, dan gelar doktor dari program studi Teknik Informatika University of Southern Cafifornia. Ia merupakan Kepala dari lembaga riset *Bioinformatics & Data Science Research Center* (BDSRC) | BINUS-NVIDIA AI *Research & Development Center* (AIRDC) yang didirikannya pada 2014 | 2017. Selain mengemban amanah sebagai Kepala BDSRC | AIRDC, ia juga merupakan Guru Besar di Universitas Bina Nusantara (BINUS). Dalam bidang bioinformatika, penelitian utamanya berfokus pada *genome wide association study* (GWAS). Selain itu, ia juga memiliki banyak ketertarikan di bidang AI. Dia telah menerbitkan lebih dari 500 kontribusi ilmiah, termasuk hampir 400 makalah penelitian yang ditinjau oleh rekan sejawat, 20 buku, dan 40 HKI. Ia merupakan penerima Sarwono Award 2024, sebuah penghargaan bergengsi yang diberikan kepada para peneliti Indonesia yang berprestasi seumur hidup di bidang ilmu pengetahuan dan teknologi yang memberikan dampak nyata bagi masyarakat.

Aplikasi Kecerdasan Buatan untuk Pertanian Modern

Klasifikasi Penyakit Daun Tanaman

Buku ini mengupas penerapan *computer vision* untuk klasifikasi penyakit daun tanaman menggunakan TensorFlow. Dengan pendekatan sistematis, pembahasan mencakup dasar-dasar pemrosesan citra, pengolahan dataset, hingga implementasi *Convolutional Neural Network* dan metode *Transfer Learning* untuk meningkatkan performa klasifikasi. Buku ini ditujukan bagi akademisi, peneliti, dan praktisi pertanian, buku ini menawarkan solusi kecerdasan buatan guna meningkatkan efisiensi pemantauan kesehatan tanaman dan ketahanan pangan untuk pertanian modern.