BAB 2
LANDASAN TEORI

Bab ini membahas berbagai teori yang menjadi landasan dalam penelitian, meliputi deskripsi penyakit *powdery mildew* pada tanaman cherry, konsep dasar *Convolutional Neural Network* (CNN), arsitektur SimpleCNN, arsitektur ConvNeXt-Tiny, strategi *feature extraction*, teknik augmentasi data, metrik evaluasi klasifikasi, serta metrik efisiensi komputasi. Pembahasan pada bab ini digunakan sebagai dasar untuk menganalisis kelayakan SimpleCNN sebagai alternatif sederhana terhadap ConvNeXt-Tiny berdasarkan performa klasifikasi dan kebutuhan komputasi model.

2.1 Penyakit *Powdery Mildew* pada Tanaman Cherry

Powdery mildew pada tanaman cherry disebabkan oleh jamur *Podosphaera clandestina*, yaitu jamur *obligate biotrophic* yang hanya dapat bertahan hidup dan berkembang biak pada jaringan tanaman inang yang masih hidup. Penyakit ini masuk ke dalam kategori salah satu penyakit yang paling sering menyerang daun, tunas muda, serta buah cherry di berbagai wilayah budidaya. Jaringan yang muda dan sedang aktif tumbuh cenderung paling rentan terhadap infeksi, sedangkan daun yang lebih tua umumnya mengalami peningkatan ketahanan alami (*ontogenic resistance*) seiring bertambahnya usia jaringan.

Siklus penyakit ini sangat dipengaruhi oleh kondisi lingkungan. Jamur *Podosphaera clandestina* bertahan pada musim dorman dalam bentuk struktur reproduktif kecil berwarna gelap (*chasmothecia*) yang menempel pada daun-daun mati dan sisa tanaman di sekitar tajuk. Pada awal musim tumbuh, hujan atau irigasi memicu pelepasan spora yang kemudian disebarkan oleh angin menuju daun-daun muda. Berbeda dengan banyak jamur lain, *powdery mildew* tidak memerlukan lapisan air bebas untuk berkecambah, sehingga kondisi kelembapan udara yang tinggi disertai suhu hangat sudah cukup untuk memicu perkembangannya. Dalam kondisi yang mendukung, siklus dari spora hingga terbentuknya koloni baru dapat berlangsung sangat cepat, sehingga penyebaran penyakit di lapangan berpotensi terjadi secara masif dalam waktu singkat.

Gejala khas yang dapat diamati secara visual adalah munculnya lapisan berwarna putih menyerupai tepung (*white powdery patches*) pada permukaan

atas maupun bawah daun. Pada tahap awal, gejala dapat berupa bercak melingkar berwarna putih pucat yang kemudian meluas menutupi sebagian besar permukaan daun. Infeksi yang tidak ditangani secara dini dapat menyebabkan daun menggulung, memuntir, dan mengalami distorsi bentuk, serta menghambat pertumbuhan tanaman. Apabila infeksi mencapai buah, lapisan jamur putih pada permukaan buah dapat menurunkan nilai jualnya secara drastis hingga tidak layak dipasarkan. Perbedaan visual antara daun sehat dan daun yang terinfeksi *powdery mildew* ditunjukkan pada Gambar 2.1.

(a) Daun cherry sehat (b) Daun cherry terinfeksi *powdery mildew*

Gambar 2.1. Contoh citra daun cherry pada dataset PlantVillage: (a) daun sehat dan (b) daun terinfeksi *powdery mildew* yang ditandai dengan lapisan putih bertepung pada permukaan daun.

Secara ekonomi, *powdery mildew* tergolong salah satu penyakit terpenting pada tanaman pangan maupun hortikultura. Kerugian yang ditimbulkannya tidak hanya berupa kematian jaringan dan kerontokan daun (*defoliation*), tetapi juga penurunan hasil panen serta kualitas produk akibat rusaknya penampilan visual. Karena penyakit ini juga menyerang beragam komoditas seperti anggur, stroberi, dan tanaman famili Cucurbitaceae, penanganannya menjadi perhatian penting dalam praktik budidaya di banyak wilayah, termasuk yang relevan dengan konteks pertanian di Indonesia.

Deteksi dini penyakit ini secara tradisional mengandalkan inspeksi visual oleh tenaga ahli di lapangan, yang bersifat subjektif, membutuhkan waktu, dan tidak skalabel untuk pertanian berskala besar [11]. Terlebih, gejala pada tahap awal infeksi sering kali samar dan mudah terlewat, sehingga penanganan yang terlambat dapat mempercepat penyebaran penyakit. Pendekatan berbasis citra

digital dengan *deep learning* menawarkan solusi yang lebih objektif, konsisten, dan dapat diotomasi [2], sehingga berpotensi membantu deteksi dini yang lebih cepat dan akurat.

2.2 *Convolutional Neural Network* (CNN)

Convolutional Neural Network (CNN) adalah salah satu jenis jaringan saraf tiruan yang dikembangkan untuk memproses data berbentuk grid, seperti citra digital, dengan memanfaatkan operasi konvolusi untuk melakukan ekstraksi fitur secara bertahap [2]. Melalui proses pelatihan, CNN dapat belajar melalui representasi fitur secara otomatis, mulai dari fitur dasar seperti tekstur dan tepi sampai fitur yang lebih kompleks seperti bentuk dan pola semantik, tanpa memerlukan ekstraksi fitur secara manual.

Pada jaringan saraf tiruan konvensional (*fully connected network*), setiap neuron terhubung ke seluruh neuron pada lapisan sebelumnya. Apabila pendekatan ini diterapkan langsung pada citra, jumlah parameter akan membengkak dan hubungan spasial antarpiksel menjadi hilang. CNN mengatasi permasalahan tersebut melalui tiga prinsip utama, yaitu konektivitas lokal (*local connectivity*), berbagi bobot (*weight sharing*), dan invariansi terhadap pergeseran (*translation invariance*). Dengan konektivitas lokal, setiap neuron hanya memproses sebagian kecil wilayah citra yang disebut *receptive field*; dengan berbagi bobot, filter yang sama digunakan di seluruh bagian citra sehingga jumlah parameter jauh berkurang; dan dengan invariansi terhadap pergeseran, pola yang dikenali tetap dapat terdeteksi meskipun posisinya berubah pada citra.

2.2.1 Operasi Konvolusi

Operasi konvolusi merupakan inti dari CNN. Pada operasi ini, sebuah filter (*kernel*) kecil, contohnya '3×3', digeser secara spasial di seluruh permukaan citra masukan. Pada masing masing posisi, dilakukan perkalian elemen (*element-wise multiplication*) antara nilai filter dan nilai piksel yang bersesuaian, kemudian hasilnya dijumlahkan menjadi satu nilai keluaran. Hasil dari keseluruhan rangkaian ini membentuk *feature map* yang merepresentasikan keberadaan suatu pola tertentu pada citra.

Terdapat beberapa parameter penting yang memengaruhi hasil operasi konvolusi, yaitu:

- ***Kernel size***: Ukuran filter yang menentukan luas wilayah citra yang diproses dalam satu operasi konvolusi. Ukuran kernel yang lebih besar mampu menangkap konteks spasial yang lebih luas.
- ***Stride***: Besar langkah pergeseran filter pada citra. Nilai *stride* yang lebih besar akan menghasilkan *feature map* berdimensi lebih kecil.
- ***Padding***: Penambahan piksel (umumnya bernilai nol) di sekeliling tepi citra untuk mengendalikan dimensi keluaran serta menjaga informasi pada bagian tepi citra.
- **Jumlah filter**: Banyaknya filter pada suatu lapisan konvolusi yang menentukan kedalaman (*depth*) *feature map* keluaran, karena setiap filter mempelajari pola fitur yang berbeda.

2.2.2 Komponen Utama Arsitektur CNN

Pada umumnya, arsitektur CNN tersusun atas beberapa komponen utama sebagai berikut:

- ***Convolutional Layer***: Menerapkan sejumlah filter (kernel) yang bergeser secara spasial pada citra masukan untuk menghasilkan *feature map*. Setiap filter mempelajari pola fitur yang berbeda secara otomatis selama pelatihan.
- ***Activation Function***: Fungsi aktivasi non-linear seperti *Rectified Linear Unit* (ReLU) digunakan setelah konvolusi untuk memperkenalkan non-linearitas sehingga jaringan mampu memodelkan hubungan yang lebih kompleks. ReLU banyak digunakan karena sederhana secara komputasi dan membantu mengurangi permasalahan *vanishing gradient*.
- ***Batch Normalization***: Menormalkan aktivasi pada setiap *mini-batch* untuk menstabilkan dan mempercepat proses pelatihan, sekaligus berfungsi sebagai regularisasi ringan [12].
- ***Pooling Layer***: Mengurangi dimensi spasial dari *feature map* (misalnya dengan *max pooling*) sehingga mengurangi beban komputasi sekaligus meningkatkan ketahanan fitur terhadap pergeseran kecil pada citra.
- ***Global Average Pooling* (GAP)**: Bentuk khusus *pooling* yang meratakan setiap *feature map* menjadi satu nilai rata-rata, sehingga menghasilkan

satu nilai per kanal [13]. Berbeda dengan *flatten* yang mempertahankan seluruh nilai spasial, GAP mereduksi dimensi secara drastis sehingga jumlah parameter pada lapisan *fully connected* jauh berkurang dan risiko *overfitting* dapat ditekan. Pada *PyTorch*, GAP dapat diwujudkan menggunakan `AdaptiveAvgPool2d((1, 1))`.

- ***Fully Connected Layer***: Menggabungkan semua fitur yang sudah diekstraksi dan memetakannya ke dalam vektor keluaran sesuai jumlah kelas target. Pada tahap akhir, fungsi seperti *softmax* umumnya digunakan untuk menghasilkan distribusi probabilitas antarkelas.

2.2.3 Ekstraksi Fitur Hierarkis

Salah satu kelebihan CNN adalah kemampuannya belajar dari fitur secara hierarkis. Lapisan-lapisan awal cenderung mengambil fitur tingkat rendah (*low-level features*) seperti tepi, sudut, dan tekstur sederhana. Semakin dalam lapisannya, fitur yang dipelajari menjadi semakin abstrak dan kompleks (*high-level features*), seperti bentuk, bagian objek, hingga pola semantik yang lebih spesifik. Struktur hierarkis ini memungkinkan CNN membangun representasi citra yang kaya secara bertahap, yang menjadi alasan utama efektivitasnya pada tugas klasifikasi citra, termasuk deteksi penyakit tanaman [3].

2.2.4 Proses Pelatihan CNN

Proses pelatihan CNN dilakukan melalui dua tahap utama yang berlangsung secara iteratif. Pada tahap *forward pass*, citra masukan diproses melalui seluruh lapisan jaringan hingga menghasilkan prediksi keluaran. Selisih antara prediksi dan label sebenarnya kemudian dihitung menggunakan fungsi *loss*, misalnya *Cross-Entropy Loss* pada tugas klasifikasi. Selanjutnya, pada tahap *backward pass* (*backpropagation*), gradien dari fungsi *loss* terhadap setiap parameter dihitung dan digunakan oleh *optimizer* untuk memperbarui bobot jaringan melalui algoritma penurunan gradien (*gradient descent*).

Proses pembaruan parameter ini dilakukan secara berulang dalam satuan *epoch*, yaitu satu siklus pelatihan penuh terhadap seluruh data latih. Untuk efisiensi komputasi, data latih umumnya dibagi menjadi kelompok-kelompok kecil (*mini-batch*). Beberapa *hyperparameter* penting yang memengaruhi keberhasilan

pelatihan antara lain *learning rate*, ukuran *batch*, jumlah *epoch*, serta pemilihan *optimizer* seperti *Adam* atau *AdamW*.

2.2.5 Fungsi *Loss* (*Cross-Entropy Loss*)

Fungsi *loss* berperan mengukur besarnya selisih antara prediksi model dan label sebenarnya, sehingga menjadi acuan bagi *optimizer* dalam memperbarui bobot. Pada tugas klasifikasi, fungsi *loss* yang umum digunakan adalah *Cross-Entropy Loss*, yang mengukur perbedaan antara distribusi probabilitas prediksi dan distribusi label sebenarnya. Untuk kasus klasifikasi dengan C kelas, nilai *cross-entropy* pada satu sampel dirumuskan sebagai:

$$L = -\sum_{i=1}^{C} y_i \log(\hat{y}_i) \tag{2.1}$$

dengan y_i merupakan label sebenarnya (bernilai 1 untuk kelas yang benar dan 0 untuk kelas lainnya) dan $\hat{y}_i$ merupakan probabilitas prediksi model untuk kelas ke-i. Nilai *loss* akan semakin kecil apabila probabilitas prediksi pada kelas yang benar semakin mendekati 1. Pada penelitian ini digunakan `CrossEntropyLoss` dari *PyTorch*, yang secara internal telah menggabungkan fungsi *softmax* dan *negative log-likelihood*, sehingga keluaran berupa *logit* mentah dapat langsung diproses.

2.2.6 *Optimizer* (Adam dan AdamW)

Optimizer merupakan algoritma yang memperbarui bobot jaringan berdasarkan gradien fungsi *loss* agar nilai *loss* semakin kecil pada setiap iterasi. Salah satu *optimizer* yang paling banyak digunakan adalah *Adam* (*Adaptive Moment Estimation*) [14], yang menggabungkan keunggulan *momentum* dan penyesuaian *learning rate* secara adaptif untuk masing masing parameter. Adam menggunakan estimasi momen pertama (rata-rata gradien) dan momen kedua (rata-rata kuadrat gradien) sehingga proses konvergensi menjadi lebih cepat dan stabil.

AdamW [14] merupakan pengembangan dari Adam yang memisahkan (*decouple*) mekanisme *weight decay* dari perhitungan gradien. Pada Adam standar, *weight decay* tercampur ke dalam pembaruan gradien sehingga kurang efektif sebagai regularisasi, sedangkan pada AdamW *weight decay* diterapkan secara terpisah langsung pada bobot. Hal ini membuat AdamW menghasilkan regularisasi yang lebih baik dan umum digunakan pada pelatihan arsitektur modern. Dalam

penelitian ini, SimpleCNN dilatih menggunakan Adam, sedangkan ConvNeXt-Tiny menggunakan AdamW.

2.2.7 *Learning Rate* dan *Learning Rate Scheduler*

Learning rate adalah *hyperparameter* yang menentukan besarnya langkah pembaruan bobot pada setiap iterasi. Nilai *learning rate* yang terlalu besar dapat mengakibatkan pelatihan tidak stabil atau gagal konvergen, di sisi lain nilai yang terlalu kecil membuat pelatihan berjalan lambat. Oleh karena itu, pemilihan *learning rate* yang tepat sangat memengaruhi keberhasilan pelatihan.

Learning rate scheduler digunakan untuk menyesuaikan nilai *learning rate* secara bertahap selama pelatihan. Salah satu metode yang umum adalah `StepLR`, yang menurunkan *learning rate* dengan faktor tertentu (*gamma*) setiap sejumlah *epoch* tertentu (*step size*). Mekanisme ini memungkinkan model melakukan penyesuaian besar pada awal pelatihan dan penyesuaian yang lebih halus menjelang akhir pelatihan. Dalam penelitian ini, ConvNeXt-Tiny menggunakan `StepLR` dengan `step_size=5` dan `gamma=0.5`.

2.2.8 Overfitting dan Regularisasi

Salah satu tantangan utama dalam pelatihan CNN adalah *overfitting*, yaitu kondisi ketika model terlalu menyesuaikan diri dengan data latih sehingga performanya menurun pada data baru yang belum pernah dilihat. Permasalahan ini cenderung lebih sering terjadi apabila jumlah data latih terbatas atau arsitektur model terlalu kompleks. Untuk mengatasinya, digunakan berbagai teknik regularisasi, di antaranya *dropout* yang menonaktifkan sebagian neuron secara acak selama pelatihan [15], *batch normalization*, augmentasi data untuk memperkaya variasi data latih, serta *weight decay* untuk membatasi besarnya nilai bobot. Teknik-teknik ini bertujuan meningkatkan kemampuan generalisasi model terhadap data uji.

2.2.9 Perkembangan Arsitektur CNN

Arsitektur CNN telah berkembang pesat sejak pertama kali diperkenalkan. Model awal seperti LeNet menunjukkan potensi CNN pada tugas pengenalan citra sederhana, yang kemudian dikembangkan lebih lanjut oleh AlexNet [16] yang membuktikan keunggulan CNN pada klasifikasi citra berskala besar. Perkembangan

berikutnya menghadirkan arsitektur yang lebih dalam seperti VGG dengan filter berukuran kecil yang disusun bertumpuk [17], serta ResNet yang memperkenalkan *residual connection* untuk memungkinkan pelatihan jaringan yang sangat dalam tanpa mengalami degradasi performa [18]. Tren ini terus berlanjut hingga munculnya arsitektur modern seperti ConvNeXt yang mengadopsi beberapa prinsip desain dari *Vision Transformer* namun tetap mempertahankan karakteristik jaringan konvolusi [1].

Meskipun arsitektur modern cenderung menghasilkan akurasi yang semakin tinggi, peningkatan tersebut umumnya diikuti oleh bertambahnya kompleksitas arsitektur, jumlah parameter, dan kebutuhan komputasi. Kondisi ini mendasari perlunya perbandingan antara arsitektur sederhana dan arsitektur modern. Dalam penelitian ini, digunakan dua pendekatan berbasis CNN, yaitu SimpleCNN yang dibangun dari awal (*from scratch*) dan ConvNeXt-Tiny berbasis *transfer learning* dengan strategi *feature extraction*. Kedua model tersebut dibandingkan untuk menganalisis *trade-off* antara performa klasifikasi dan efisiensi komputasi pada tugas klasifikasi *powdery mildew* daun cherry.

2.2.10 Arsitektur SimpleCNN

SimpleCNN adalah model CNN sederhana yang dirancang secara kustom yang digunakan sebagai model pembanding dalam penelitian ini. Model ini dibangun dari awal (*from scratch*) tanpa menggunakan bobot pra-latih, sehingga seluruh parameter model dipelajari secara langsung dari dataset daun cherry yang digunakan. Pemilihan SimpleCNN bertujuan untuk mengetahui apakah model CNN yang lebih sederhana masih dapat memberikan performa klasifikasi yang memadai dibandingkan dengan arsitektur modern berbasis *transfer learning*.

Penggunaan CNN sederhana sebagai pembanding didukung oleh beberapa penelitian sebelumnya. Imanulloh dkk. [4] menunjukkan bahwa CNN kustom yang tidak kompleks dan ringan dapat digunakan untuk klasifikasi penyakit tanaman berbasis citra daun. Selain itu, Li dkk. [19] meneliti apakah CNN yang sangat dalam selalu dibutuhkan untuk identifikasi penyakit tanaman, dan menunjukkan bahwa pendekatan *shallow CNN* dapat menjadi alternatif yang layak dengan jumlah parameter yang lebih sedikit. Oleh karena itu, SimpleCNN pada penelitian ini tidak dimaksudkan untuk menggantikan ConvNeXt-Tiny secara langsung, melainkan sebagai dasar pembanding untuk menganalisis kelayakan model yang lebih sederhana.

Arsitektur SimpleCNN yang digunakan terdiri atas tiga blok konvolusi pada bagian `features`. Setiap blok terdiri dari lapisan `Conv2d` dengan kernel 3×3 dan `padding=1`, fungsi aktivasi ReLU, *Batch Normalization* [12], dan *Max Pooling* berukuran 2×2. Jumlah filter pada ketiga blok secara berurutan adalah 32, 64, dan 128. Dengan ukuran citra masukan 224×224 piksel, tiga kali operasi *max pooling* mereduksi dimensi spasial *feature map* secara bertahap.

Berbeda dengan arsitektur awal yang menggunakan lapisan `Flatten` berdimensi besar, SimpleCNN pada penelitian ini menggunakan `AdaptiveAvgPool2d((1, 1))` sebelum lapisan *fully connected*. Lapisan ini mereduksi setiap *feature map* menjadi ukuran 1×1, sehingga jumlah fitur yang masuk ke bagian *classifier* menjadi lebih kecil. Setelah itu, fitur diratakan menggunakan `Flatten`, kemudian diproses oleh lapisan *Linear*(128, 64), fungsi aktivasi ReLU, *Dropout* dengan probabilitas 0,5 [15], dan lapisan *Linear*(64, 2) sebagai keluaran akhir untuk dua kelas.

Penggunaan `AdaptiveAvgPool2d` bertujuan untuk mengurangi jumlah parameter pada bagian *classifier* dan menekan risiko *overfitting*, khususnya karena dataset yang digunakan relatif terbatas. Dengan demikian, SimpleCNN dapat merepresentasikan model CNN sederhana yang memiliki ukuran model kecil dan tidak bergantung pada bobot pra-latih.

2.3 Arsitektur ConvNeXt-Tiny

ConvNeXt merupakan arsitektur CNN modern yang diperkenalkan oleh Liu dkk. [1]. Arsitektur ini dikembangkan dengan memodernisasi ResNet secara bertahap menggunakan beberapa prinsip desain dari *Vision Transformer* (ViT), tetapi tetap mempertahankan sifat jaringan konvolusi murni tanpa mekanisme *self-attention*. ConvNeXt dilaporkan mampu bersaing dengan arsitektur Transformer dalam hal akurasi dan skalabilitas, serta mencapai performa tinggi pada tugas klasifikasi citra berskala besar [1].

Pada domain penyakit tanaman, ConvNeXt juga telah digunakan untuk klasifikasi penyakit daun pada beberapa jenis tanaman. Murugesan dkk. [20] menjelaskan bahwa struktur hierarkis, ukuran kernel yang besar, dan desain tahap mendalam pada ConvNeXt memungkinkan model menangkap pola tekstur kompleks dan ketidakteraturan lokal pada citra daun. Hal ini membuat ConvNeXt relevan digunakan sebagai representasi model CNN modern untuk klasifikasi penyakit *powdery mildew* pada daun cherry.

Beberapa inovasi desain utama pada ConvNeXt adalah sebagai berikut:

- ***Depthwise Convolution*** 7×7: Menggunakan konvolusi *depthwise* berukuran besar untuk menangkap konteks spasial yang lebih luas dibandingkan konvolusi standar berukuran kecil.
- ***Inverted Bottleneck***: Memperbesar dimensi fitur pada bagian tersembunyi, mengikuti prinsip desain pada *feed-forward network* dalam Transformer.
- ***Layer Normalization***: Menggunakan *Layer Normalization* untuk menjaga stabilitas pelatihan pada arsitektur modern.
- **Aktivasi GELU**: Menggunakan fungsi aktivasi *Gaussian Error Linear Unit* (GELU), yang umum digunakan pada arsitektur modern berbasis Transformer.
- ***Layer Scale***: Menggunakan parameter skala yang dapat dipelajari untuk membantu stabilitas pelatihan pada jaringan yang lebih dalam.

Varian yang digunakan dalam penelitian ini adalah ConvNeXt-Tiny. Varian ini dipilih karena memiliki ukuran yang lebih kecil dibandingkan varian ConvNeXt lainnya, tetapi tetap merepresentasikan arsitektur ConvNeXt modern. Dalam penelitian ini, ConvNeXt-Tiny dimuat dengan bobot pra-latih ImageNet melalui `ConvNeXt_Tiny_Weights.IMAGENET1K_V1`. Bagian `features` digunakan sebagai *backbone* pengekstraksi fitur, sedangkan bagian *classifier* akhir diganti agar sesuai dengan jumlah kelas pada penelitian ini, yaitu dua kelas.

Meskipun ConvNeXt-Tiny memiliki kemampuan ekstraksi fitur yang kuat, model *deep learning* modern umumnya memiliki kebutuhan komputasi dan jumlah parameter yang lebih besar dibandingkan CNN sederhana. Penelitian Pal dkk. [8] menunjukkan bahwa model *state-of-the-art* dalam klasifikasi penyakit tanaman dapat memberikan performa tinggi, tetapi sering kali membutuhkan sumber daya komputasi yang lebih besar sehingga kurang praktis untuk perangkat dengan keterbatasan sumber daya. Oleh karena itu, ConvNeXt-Tiny digunakan dalam penelitian ini sebagai model modern berperforma tinggi yang dibandingkan dengan SimpleCNN sebagai alternatif sederhana.

2.4 *Transfer Learning* dengan Strategi *Feature Extraction*

Transfer learning merupakan metode yang memanfaatkan kembali pengetahuan dari sebuah model yang sudah dilatih terlebih dahulu pada suatu

domain sumber untuk menyelesaikan tugas pada *domain* target yang berbeda [6]. Alih-alih melatih model dari awal dengan bobot acak, pendekatan ini menggunakan bobot pra-latih (*pre-trained weights*) dari model yang telah dilatih pada dataset berskala besar, misalnya ImageNet yang memuat jutaan citra dari seribu kelas. Dengan demikian, model tidak perlu mempelajari kembali representasi fitur dasar dari nol, melainkan cukup menyesuaikan pengetahuan yang telah dimiliki pada tugas yang baru.

Efektivitas *transfer learning* didasarkan pada sifat fitur yang dipelajari oleh CNN. Lapisan-lapisan awal cenderung mempelajari fitur yang bersifat umum (*generic*) seperti tepi, sudut, dan tekstur, yang relevan untuk hampir semua jenis citra. Sebaliknya, lapisan-lapisan akhir mempelajari fitur yang lebih spesifik terhadap tugas asal. Karena fitur tingkat rendah bersifat umum dan dapat dipindahkan (*transferable*) antar-*domain*, model yang telah dilatih pada dataset besar dapat dimanfaatkan kembali secara efektif pada tugas dengan data yang lebih terbatas, seperti klasifikasi penyakit daun cherry pada penelitian ini.

Terdapat dua strategi utama dalam *transfer learning*, yang berbeda pada bagian model mana yang bobotnya diperbarui selama pelatihan:

- ***Feature Extraction***: Seluruh lapisan *backbone* pra-latih **dibekukan** (`requires_grad = False`) sehingga bobotnya tidak diperbarui selama pelatihan. Dalam strategi ini, *backbone* berperan sebagai **pengekstraksi fitur tetap** (*fixed feature extractor*): citra masukan diubah menjadi representasi fitur menggunakan pengetahuan pra-latih, dan hanya lapisan *classifier* baru yang dilatih untuk memetakan fitur tersebut ke kelas target. Strategi ini cocok ketika data target terbatas atau memiliki karakteristik yang mirip dengan data sumber.
- ***Fine-tuning***: Sebagian atau seluruh lapisan *backbone* **diizinkan diperbarui** bersama lapisan *classifier* selama pelatihan pada data target. Strategi ini umumnya memberikan performa lebih tinggi, tetapi membutuhkan lebih banyak data dan kehati-hatian agar tidak terjadi *catastrophic forgetting*, yaitu hilangnya pengetahuan pra-latih akibat pembaruan bobot yang berlebihan.

Perlu ditegaskan bahwa istilah *feature extraction* pada konteks *transfer learning* berbeda dengan proses ekstraksi fitur secara umum pada CNN. Pada CNN, ekstraksi fitur merujuk pada proses konvolusi yang menghasilkan *feature map*, sedangkan pada *transfer learning*, *feature extraction* merujuk pada **strategi**

penggunaan *backbone* pra-latih dalam keadaan beku sebagai pengekstraksi fitur, tanpa memperbarui bobotnya.

Dalam penelitian ini, strategi yang digunakan adalah ***feature extraction***. Seluruh parameter pada `model.features` dibekukan (`param.requires_grad = False`), sehingga hanya lapisan *classifier* baru yang ditambahkan yang memiliki gradien aktif dan diperbarui selama pelatihan. Pemilihan strategi ini didasarkan pada beberapa pertimbangan: dataset yang digunakan relatif terbatas, karakteristik citra daun tidak jauh berbeda dengan citra alami pada ImageNet, serta kebutuhan untuk menekan jumlah parameter yang dilatih agar risiko *overfitting* berkurang dan proses pelatihan lebih efisien.

Model dioptimasi menggunakan *optimizer AdamW* dengan *learning rate* '10^{-4}' dan *weight decay* '10^{-4}', serta *learning rate scheduler* `StepLR` dengan `step_size=5` dan `gamma=0.5` untuk menurunkan *learning rate* secara bertahap. Fungsi *loss* yang digunakan adalah *CrossEntropyLoss*.

2.5 Augmentasi Data

Augmentasi data merupakan teknik untuk memperbesar keragaman dataset pelatihan secara artifisial dengan menerapkan serangkaian transformasi pada citra asli [11]. Tujuan utamanya adalah meningkatkan ketahanan (*robustness*) model terhadap variasi kondisi pengambilan gambar sekaligus mengurangi risiko *overfitting*.

Dalam penelitian ini, augmentasi hanya diterapkan pada data latih melalui *pipeline* transformasi berikut secara berurutan:

1. `Resize(224, 224)`: Mengubah ukuran seluruh citra ke dimensi 224×224 piksel sebagai langkah awal penyeragaman.
2. `RandomHorizontalFlip`: Membalik citra secara horizontal dengan probabilitas 0.5 untuk mensimulasikan variasi orientasi daun.
3. `RandomVerticalFlip`: Membalik citra secara vertikal dengan probabilitas 0.5.
4. `RandomRotation(30)`: Memutar citra secara acak hingga ± 30 untuk meningkatkan ketahanan terhadap variasi sudut pandang.

5. `RandomResizedCrop(224, scale=(0.7, 1.0))`: Memotong area citra secara acak dengan skala antara 0.7 hingga 1.0 dari ukuran asli, lalu mengubah ukurannya kembali ke 224×224 piksel.

6. `ColorJitter(brightness=0.3, contrast=0.3, saturation=0.3)`: Mengubah kecerahan, kontras, dan saturasi secara acak dalam rentang ± 0.3 untuk mensimulasikan variasi kondisi pencahayaan.

7. `GaussianBlur(kernel_size=3)`: Menerapkan *blur* Gaussian dengan kernel berukuran 3×3 untuk mensimulasikan variasi ketajaman citra.

8. `ToTensor`: Mengonversi citra dari format PIL menjadi `torch.Tensor` dengan nilai piksel pada rentang [0, 1].

9. `RandomErasing(p=0.5, scale=(0.1, 0.3))`: Menghapus sebagian area citra secara acak dengan probabilitas 0.5 dan skala area penghapusan antara 0.1 hingga 0.3. Transformasi ini bertujuan untuk meningkatkan ketahanan model terhadap bagian citra yang tertutup atau kurang informatif.

10. `Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])`: Menormalkan citra dengan nilai *mean* serta *standard deviation* ImageNet agar distribusi citra masukan sama seperti distribusi data yang digunakan pada model pra-latih.

Sementara itu, data validasi dan data uji hanya melalui `Resize(224, 224)`, `ToTensor`, dan `Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])`, tanpa augmentasi apapun, guna memastikan evaluasi yang objektif dan konsisten.

2.6 Metrik Evaluasi

Evaluasi performa kedua model dilakukan dengan metrik umum pada klasifikasi biner yang berasal dari *confusion matrix*, yaitu *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative* (FN) [11]. Pada penelitian ini, kelas *powdery mildew* atau *disease* diatur sebagai kelas positif, sedangkan kelas *healthy* diatur sebagai kelas negatif. Penetapan tersebut dilakukan agar nilai presisi, *recall*, dan *F1-score* dapat menggambarkan keterampilan model dalam mengidentifikasi daun yang terinfeksi penyakit.

Akurasi mengukur proporsi prediksi yang benar dari keseluruhan sampel uji:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

$$Presisi = \frac{TP}{TP + FP} \tag{2.3}$$

Recall mengukur kemampuan model mendeteksi seluruh sampel positif yang sebenarnya:

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

F1-score adalah rata-rata harmonik antara presisi serta *recall* yang digunakan untuk memberikan gambaran performa model secara lebih seimbang, khususnya pada kondisi distribusi kelas yang tidak merata:

$$F1\text{-}score = \frac{2 \times Presisi \times Recall}{Presisi + Recall} \tag{2.5}$$

2.7 Metrik Efisiensi Komputasi

Tidak hanya metrik performa klasifikasi, tetapi juga penelitian ini menggunakan beberapa indikator efisiensi komputasi untuk menganalisis *trade-off* antara ConvNeXt-Tiny dan SimpleCNN. Pengukuran efisiensi komputasi diperlukan karena model dengan akurasi tinggi belum tentu menjadi pilihan terbaik apabila memiliki ukuran model besar, waktu inferensi tinggi, atau kebutuhan memori yang tidak sesuai dengan keterbatasan perangkat.

Total parameters menunjukkan jumlah keseluruhan parameter atau bobot yang dimiliki oleh model, baik yang dilatih maupun yang tidak dilatih. Semakin banyak parameter yang dimiliki model, semakin tinggi pula kemampuan model dalam mempelajari pola data. Namun, jumlah parameter yang besar juga dapat meningkatkan ukuran model dan kebutuhan penyimpanan.

Trainable parameters menunjukkan jumlah parameter yang diperbarui

selama proses pelatihan. Pada model berbasis *feature extraction*, jumlah *trainable parameters* dapat jauh lebih kecil daripada *total parameters* karena sebagian besar lapisan *backbone* dibekukan. Sebaliknya, pada model yang dilatih dari awal seperti SimpleCNN, seluruh parameter umumnya bersifat *trainable*.

Ukuran model menunjukkan besar file model yang disimpan setelah proses pelatihan. Ukuran model menjadi penting apabila model akan diterapkan pada perangkat dengan kapasitas penyimpanan terbatas atau dikirim melalui jaringan dengan bandwidth terbatas.

Waktu pelatihan menunjukkan total waktu yang dibutuhkan model untuk menyelesaikan proses pelatihan pada jumlah *epoch* tertentu. Waktu pelatihan dipengaruhi oleh kompleksitas model, jumlah parameter yang dilatih, ukuran dataset, spesifikasi perangkat keras, serta kondisi *runtime*.

Waktu inferensi per citra menunjukkan rata-rata waktu yang dibutuhkan model untuk memproses satu citra dan menghasilkan prediksi. Metrik ini penting untuk menilai potensi penggunaan model pada sistem yang membutuhkan prediksi cepat, misalnya sistem deteksi otomatis berbasis citra.

Penggunaan memori menunjukkan jumlah memori maksimum yang digunakan selama proses pelatihan maupun inferensi. Pada tahap pelatihan, penggunaan memori dipengaruhi oleh aktivasi, gradien, optimizer, ukuran batch, dan jumlah parameter yang diperbarui. Pada tahap inferensi, penggunaan memori umumnya lebih rendah karena tidak dilakukan perhitungan gradien.

Estimated GFLOPs menunjukkan estimasi jumlah operasi *floating point* yang dibutuhkan model dalam satu kali proses *forward pass*. GFLOPs merupakan singkatan dari *Giga Floating Point Operations*, yaitu jumlah operasi komputasi dalam satuan miliar operasi. Semakin kecil nilai GFLOPs, semakin sedikit operasi komputasi yang diperlukan model untuk memproses satu citra masukan.

Dalam penelitian ini, nilai GFLOPs digunakan untuk menggambarkan beban operasi komputasi model. Nilai ini diperoleh dari estimasi *multiply-accumulate operations* (MACs), dengan pendekatan bahwa satu MAC terdiri dari satu operasi perkalian dan satu operasi penjumlahan. Oleh karena itu, estimasi GFLOPs dapat dihitung dengan pendekatan:

$$GFLOPs \approx 2 \times GMACs \tag{2.6}$$

Penggunaan metrik efisiensi komputasi membantu memberikan penilaian

yang lebih menyeluruh terhadap kedua model. Dengan demikian, perbandingan antara ConvNeXt-Tiny dan SimpleCNN tidak hanya didasarkan pada akurasi, presisi, *recall*, dan *F1-score*, tetapi juga pada ukuran model, jumlah parameter, waktu pelatihan, waktu inferensi, penggunaan memori, dan estimasi jumlah operasi komputasi.