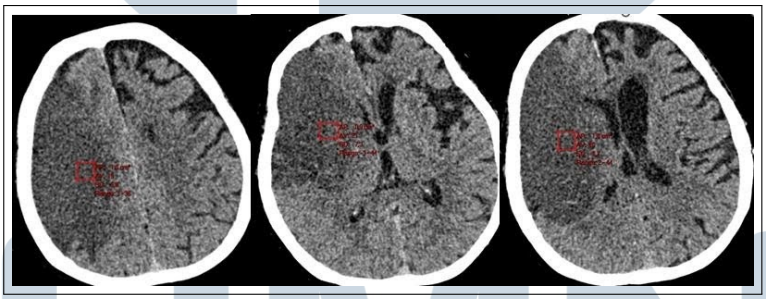
BAB 2
LANDASAN TEORI

2.1 Stroke Iskemik pada CT Scan Non-Kontras

Stroke Iskemik diakibatkan oleh penyumbatan sirkulasi darah ke jaringan otak yang mengakibatkan sel-sel otak mengalami penurunan suplai oksigen secara mendadak. Kekurangan oksigen ini akan memicu kegagalan produksi energi yang memicu kerusakan pada membran sel. Kondisi tersebut akan menyebabkan akumulasi natrium dan kalsium di dalam sel sehingga menyebabkan edema sitotoksik [15]. Pada CT Scan non-kontras, stroke iskemik umumnya menunjukan area hipodense yang tampak lebih gelap dibandingkan jaringan otak normal. Nilai hipodense ini dapat di ukur secara kuantitatif menggunakan nilai *Hounsfield Unit* (HU) [16]. Gambar CT Scan Non-Kontras untuk stroke iskemik dapat dilihat pada Gambar 2.1



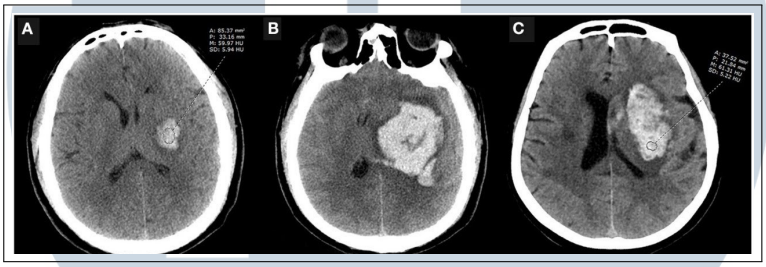
Gambar 2.1. CT Scan Non-Kontras stroke iskemik dengan area hipodens
Sumber: [16]

Berdasarkan penelitian Aslam dan Fauzi (2025) pada stroke iskemik nilai HU menurun sesuai fase, pada fase akut 25.84 ± 1.07 HU, fase subakut 18.11 ± 0.96 HU, dan fase kronis 9.24 ± 0.96 HU . Penurunan nilai HU ini mencerminkan evolusi jaringan otak secara bertahap, mulai dari edema sitotoksik pada fase akut, edema vasogenik disertai nekrosis pada fase subakut, hingga *gliosis* dan *encephalomalacia* pada fase kronis [16].

2.2 Stroke Hemoragik pada CT Scan Non-Kontras

Stroke hemoragik diakibatkan karena terjadinya ruptur pembuluh darah pada otak yang akan menyebabkan keluarnya darah bertekanan tinggi mengalir ke

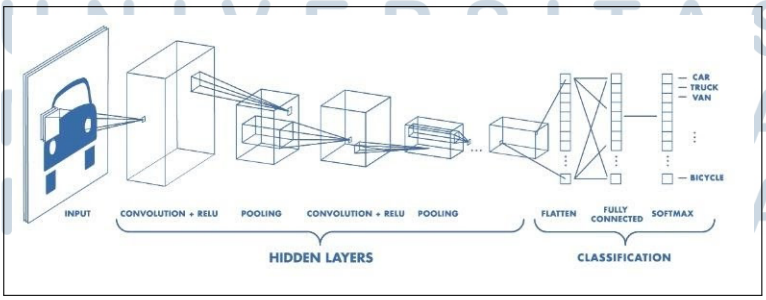
jaringan parenkim yang memicu kerusakan jaringan otak memalui efek massa. Pada pemeriksaan CT scan non-kontras pendarahan pada otak menunjukan karakteristik *hyperdense* dari pendarahan yang dapat diidentifikasi secara visual [17]. Nilai HU pada stroke hemoragik memiliki angka yang lebih tinggi dibandingkan dengan stroke iskemik. Berdasarkan hasil studi yang dilakukan oleh Nur Adlina et al. (2025) rata rata nilai HU pada stroke hemoragik adalah 57.17 ± 20.10 HU [18]. Gambar CT scan non kontras untuk stroke hemoragik ditunjukan dalam Gambar 2.2.



Gambar 2.2. CT Scan Non-Kontras stroke hemoragik dengan area hyperdense
Sumber: [19]

2.3 Convolutional Neural Network

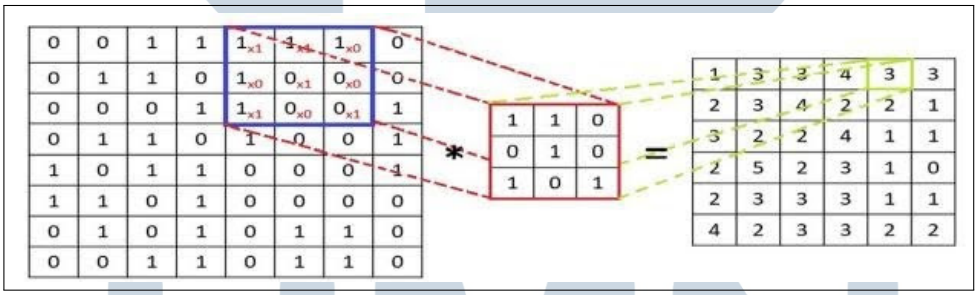
Convolutional Neural Network (CNN) atau yang bisa disebut sebagai ConvNet termasuk ke dalam teknik *Deep learning* yang dikembangkan dari *Artificial Neural Network* (ANN) serta dibangun untuk memperoses data citra. CNN meniru cara kerja sistem visual manusia, di mana setiap *neuron* hanya memproses informasi pada area tertentu yang disebut *receptive field* [20]. Secara umum, CNN terdiri atas beberapa bagian penting, seperti *convolutional layer, pooling layer, activation function,* serta *fully connected layer* [20, 21]. Representasi struktur CNN disajikan di Gambar 2.3.



Gambar 2.3. Arsitektur CNN
Sumber: [21]

2.3.1 Convolutional Layer

Convolutional layer adalah lapisan penting yang memiliki fungsi sebagai ekstraksi fitur dari citra *input* dengan menerapkan operasi konvolusi. Pada tahap ini, citra direpresentasikan sebagai kumpulan nilai numerik dalam bentuk matriks yang mencerminkan intensitas piksel [21]. Operasi konvolusi dilakukan dengan menggunakan sejumlah filter atau *kernel* berukuran lebih kecil dibandingkan citra *input*. Filter tersebut akan bergeser ke setiap bagian citra, kemudian dilakukan operasi perkalian elemen per elemen antara *kernel* dan *receptive field*, yang selanjutnya dijumlahkan untuk menghasilkan nilai pada *feature map* [20, 22]. Proses ini dilakukan berulang kali hingga setiap bagian citra diproses, sehingga memproduksi *feature map* sebagai hasil yang menyajikan ciri-ciri penting pada citra seperti seperti kontur, tekstur, serta pola-pola tertentu. Contoh cara kerja operasi konvolusi pada citra *input* 8×8 menggunakan *kernel* 3×3 di ilustrasikan di Gambar 2.4.



Gambar 2.4. Operasi konvolusi pada citra input

Sumber: [21]

Selain itu, terdapat beberapa parameter penting yang mempengaruhi kinerja *convolutional layer*, yaitu *Kernel size* berfungsi untuk menetapkan dimensi filter yang diterapkan, sedangkan *stride* mengatur jarak perpindahan filter selama proses konvolusi, sementara itu *padding* berfungsi untuk menambahkan nilai tertentu pada bagian tepi citra agar dimensi *output* dapat dikontrol[21]. Secara matematis operasi konvolusi dinyatakan sebagai berikut.

$$f_k(x,y) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} w_{i,j} x_{(i+x),(j+y)} + b \qquad (2.1)$$

Pada Rumus 2.1, $x_{(i+x),(j+y)}$ menyatakan nilai piksel citra masukan pada area yang dilalui oleh *kernel*, $w_{i,j}$ merupakan bobot *kernel* pada posisi (i, j), b

adalah nilai *bias*, sedangkan $f_k(x,y)$ merupakan nilai keluaran berupa *feature map* yang dihasilkan oleh filter [23].

2.3.2 Activation Function

Activation function berperan dalam menentukan *output* dari setiap *neuron* setelah proses konvolusi dilakukan. Fungsi ini diterapkan pada nilai hasil ekstraksi fitur untuk memperkenalkan sifat non-linear ke dalam model, yang memungkinkan jaringan saraf mampu mengidentifikasi hubungan yang rumit antara *input* dan *output*. Tanpa adanya *activation function*, model hanya akan berperilaku seperti fungsi *linear* sederhana yang memiliki keterbatasan dalam menangani permasalahan yang kompleks [24].

Dalam CNN, *activation function* bekerja dengan cara mentransformasikan nilai hasil konvolusi menjadi bentuk tertentu yang dapat digunakan pada proses selanjutnya. Proses ini memungkinkan jaringan untuk menonjolkan fitur-fitur penting sekaligus menekan informasi yang kurang relevan. Dengan demikian, model dapat membangun representasi data yang lebih abstrak dan mendalam pada setiap *layer* [24].

A ReLU

Rectified Linear Unit (ReLU) adalah salah satu fungsi yang ada dalam pengembangan *Deep Learning*. Fungsi aktivasi ini bekerja dengan mempertahankan nilai *input* yang bernilai positif, sedangkan nilai negatif akan diubah menjadi nol. Penggunaan ReLU memungkinkan jaringan saraf menghasilkan proses pembelajaran yang lebih efisien karena mampu mempertahankan informasi penting pada data tanpa mengubah nilai positif yang diterima. Selain itu, karakteristiknya yang sederhana menjadikan proses komputasi lebih ringan dibandingkan beberapa fungsi aktivasi lainnya, sehingga sangat sesuai diterapkan pada model dengan jumlah lapisan yang banyak [25]. Secara matematis fungsi aktivasi ReLU dinyatakan sebagai berikut:

$$f(x) = \max(0,x) \tag{2.2}$$

Berdasarkan Rumus 2.2, nilai keluaran fungsi aktivasi bergantung pada nilai masukan x. Apabila nilai x lebih besar atau sama dengan nol, maka nilai keluaran

sama dengan nilai masukan. Sebaliknya, apabila nilai x kurang dari nol, maka nilai keluaran akan menjadi nol [22].

B Softmax

Softmax merupakan fungsi yang diterapkan di *output layer*, khususnya untuk mengklasifikasikan data multikelas. Fungsi ini berperan dalam mengubah nilai keluaran mentah dari jaringan saraf menjadi sebaran probabilitas bagi masing-masing kategori, yang memungkinkan model dapat mengidentifikasi kategori dengan tingkat probabilitas paling besar sebagai hasil prediksi. Selain itu, *Softmax* juga membantu memperjelas perbedaan antar nilai keluaran sehingga proses pengambilan keputusan oleh model menjadi lebih terstruktur dan interpretatif [26]. Aktivasi *Softmax* dapat dirumuskan sebagai berikut:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.3}$$

Berdasarkan Rumus 2.3, z_i merupakan nilai *logit* yang dihasilkan oleh *neuron* pada kelas ke-i, sedangkan K menyatakan jumlah seluruh kelas. Fungsi *Softmax* mengubah setiap nilai *logit* menjadi nilai probabilitas dengan rentang 0 hingga 1, di mana jumlah probabilitas dari seluruh kelas bernilai 1 [23].

Dalam implementasinya, *Softmax* umumnya ditempatkan setelah *fully connected layer* sebagai tahap akhir pada proses klasifikasi. Penggunaan fungsi ini melakukan pemetaan data secara nonlinier sehingga karakteristik antar kelas dapat dipisahkan dengan lebih baik. melalui proses ini, model tidak hanya mampu menghasilkan prediksi akhir, tetapi juga memberikan representasi probabilitas yang memperlihatkan tingkat keyakinan model kepada setiap kategori tersedia [26].

2.3.3 Pooling Layer

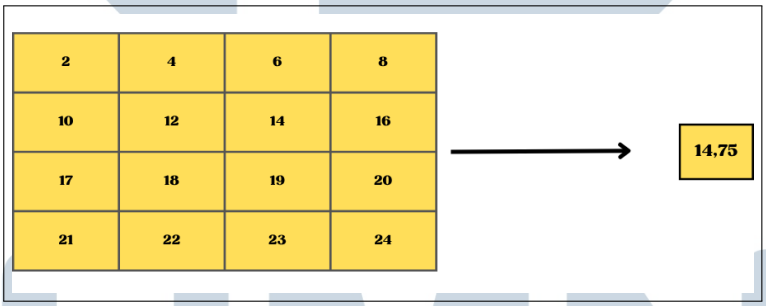
Pooling layer umumnya ditempatkan sesudah *convolutional layer*. Lapisan ini berfungsi untuk melakukan proses *downsampling* terhadap *feature map* dengan cara mereduksi dimensi spasialnya, sehingga jumlah data yang diproses pada lapisan berikutnya menjadi lebih kecil. Proses ini tidak hanya bertujuan untuk menurunkan kebutuhan komputasi, tetapi juga untuk mempertahankan informasi penting yang telah diekstraksi pada tahap konvolusi [27, 28].

Melalui mekanisme agregasi nilai pada area tertentu, *pooling layer* mampu

merangkum karakteristik utama dari fitur yang dihasilkan, sehingga membantu model dalam mengenali pola secara lebih efisien. Selain hal tersebut, penggunaan *pooling layer* juga berkontribusi untuk meningkatkan generalisasi model dengan mengurangi kompleksitas representasi data [28].

A Global Average Pooling

Global Average Pooling (GAP) adalah salah satu teknik yang berfungsi untuk mereduksi informasi pada *feature map* menjadi representasi yang lebih serdehana. Berdasarkan Gambar 2.5 proses kerja GAP dilakukan dengan menjumlahkan nilai rata-rata dari seluruh elemen dalam satu *feature map*, dimana setiap *feature map* direduksi menjadi satu nilai tunggal yang mewakili informasi global dari peta fitur tersebut. Dengan pendekatan ini, dimensi spasial dari *feature map* dapat direduksi tanpa menghilangkan konteks global yang penting dan berkontribusi dalam meningkatkan efisiensi model [29].



Gambar 2.5. Global Average Pooling

Untuk memberikan gambaran mengenai cara kerja GAP secara matematis, proses perhitungan nilai rata-rata pada setiap *feature map* dapat dinyatakan sebagai berikut.

$$f_{\text{avg}}(X) = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} X_{ij} \tag{2.4}$$

Berdasarkan Rumus 2.4, dimana N menyatakan lebar (*width*) , M menyatakan tinggi (*height*), sedangkan X_{ij} merupakan nilai aktivasi pada posisi (i, j) dari *feature map* [29].

2.3.4 Fully Connected Layer

Fully connected layer merupakan lapisan yang memiliki koneksi langsung ke seluruh *neuron* pada lapisan sebelumnya, mau itu dari *convolutional layer* maupun *pooling layer* [30]. Seluruh informasi yang telah diekstraksi melalui *convolutinal layer*, disederhanakan oleh *pooling layer*, serta di aktivasi melalui *activation layer* akan diintegrasikan pada lapisan ini untuk menghasilkan klasifikasi akhir [30, 31].

2.3.5 Batch Normalization

Batch Normalization berfungsi untuk mengoptimalkan proses *training* dengan mengatur distribusi nilai *input* pada setiap lapisan. Mekanisme ini berperan dalam mengurangi dampak inisialisai bobot yang kurang optimal, sehingga dapat meminimalkan kemungkinan terjadinya pembelajaran yang tidak efektik akibat bobot awal yang tidak tepat. Dalam penerapannya, *batch normalization* umumnya ditempatkan sebelum lapisan aktivasi [32]. Untuk rumus normalisasi, *scaling* dan *shifting* dapat dinotasikan sebagai berikut.

$$y_{Ni}^{\prime(k)} = \frac{y_i^{(k)} - \mu_B^{(k)}}{\sqrt{\left(\sigma_B^{(k)}\right)^2 + \varepsilon}} \tag{2.5}$$

$$y_{Ni}^{(k)} = \gamma \cdot y_{Ni}^{\prime(k)} + \beta \tag{2.6}$$

Berdasarkan Rumus 2.5, $y_i^{(k)}$ merupakan nilai aktivasi masukan pada lapisan ke-k, sedangkan $\mu_B^{(k)}$ dan $\sigma_B^{(k)}$ masing-masing menyatakan *batch mean* dan *batch standard deviation* dari *mini-batch*. Konstanta ε ditambahkan untuk menghindari pembagian dengan nol [33].

Selanjutnya, hasil normalisasi $y_{Ni}^{\prime(k)}$ diproses menggunakan Rumus 2.6 melalui operasi *scaling* dan *shifting*. Pada persamaan tersebut, γ merupakan parameter penskalaan (*scaling*) dan β merupakan parameter pergeseran (*shifting*) yang dipelajari selama proses pelatihan [33].

2.3.6 Dropout

Dropout merupakan teknik regularisasi untuk meningkatkan kemampuan generalisasi model dengan cara meminimalisir dampak *overfitting*. Mekanisme ini dilakukan dengan cara mematikan sejumlah *neuron* secara *random* selama proses *training*, sehingga pada setiap iterasi hanya sebagian jaringan yang aktif. Akibatnya, model tidak dapat dependen kepada *neuron* tertentu, tetapi ditekan untuk memahami representasi karakteristik yang lebih menyebar dan tidak saling bergantung [34, 35].

Penggunaan *dropout* mampu memperkecil perbedaan kinerja antara data pelatihan dan data validasi serta menghasilkan proses pelatihan yang lebih stabil. Meskipun demikian, efektivitasnya sangat dipengaruhi oleh pemilihan nilai *dropout*, karena nilai yang terlalu besar dapat menghambat proses pembelajaran, sedangkan nilai yang terlalu kecil kurang optimal dalam mengatasi *overfitting* [35]. Rumus *Dropout* diperlihatkan sebagai berikut.

$$h^{(l)} = m^{(l)} \odot f\left(W^{(l)} h^{(l-1)}\right) \tag{2.7}$$

Berdasarkan Rumus 2.7, $h^{(l)}$ menyatakan keluaran pada lapisan ke-l setelah mekanisme *dropout* diterapkan. Parameter $m^{(l)}$ merupakan *dropout mask* yang dibangkitkan secara acak mengikuti distribusi *Bernoulli* untuk menyeleksi *neuron* yang tetap aktif selama pelatihan. Simbol $\odot$ menunjukkan operasi perkalian elemen per elemen (*element-wise multiplication*), sedangkan $W^{(l)}$ menyatakan matriks bobot, $h^{(l-1)}$ merupakan aktivasi dari lapisan sebelumnya, dan $f(\cdot)$ adalah fungsi aktivasi [36].

2.4 Adam Optimizer

Adam (*Adaptive Moment Estimation*) merupakan algoritma optimasi yang dirancang untuk mengatasi kelemahan *momentum-based Stochastic Gradient Descent* (SGD) dalam menghadapi perubahan gradien yang tidak stabil selama proses pelatihan. Mekanisme Adam memanfaatkan estimasi *first moment* dan *second moment* untuk menyesuaikan *learning rate* pada setiap parameter secara adaptif. Dengan mekanisme tersebut, proses pembaruan bobot menjadi lebih efektif [37].

Estimasi momen pertama dihitung menggunakan *exponential moving*

average dari gradien yang dirumuskan pada Rumus 2.8. Pada rumus tersebut, m_t merupakan estimasi momen pertama pada iterasi ke-t, m_{t-1} adalah estimasi momen pertama pada iterasi sebelumnya, g_t merupakan gradien fungsi *loss*, sedangkan α merupakan *decay rate* yang mengatur kontribusi gradien pada iterasi sebelumnya [37].

$$m_t = \alpha m_{t-1} + (1-\alpha) g_t \tag{2.8}$$

Selanjutnya, estimasi momen kedua dihitung berdasarkan *exponential moving average* dari kuadrat gradien sebagaimana diperlihatkan pada Rumus 2.9. Nilai v_t menunjukkan estimasi momen kedua pada iterasi ke-t, sedangkan β merupakan koefisien peluruhan yang mengatur pengaruh kuadrat gradien dari iterasi sebelumnya. Estimasi ini digunakan untuk menyesuaikan besar pembaruan bobot secara adaptif [37].

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{2.9}$$

Untuk mengurangi *bias* pada iterasi awal, estimasi momen pertama dan momen kedua dikoreksi menggunakan Rumus 2.10. Hasil koreksi dinyatakan sebagai $\hat{m}_t$ dan $\hat{v}_t$, yang masing-masing merupakan estimasi momen pertama dan momen kedua setelah dilakukan *bias correction* [37].

$$\hat{m}_t = \frac{m_t}{1-\alpha^t}, \qquad \hat{v}_t = \frac{v_t}{1-\beta^t} \tag{2.10}$$

Bobot model kemudian diperbarui menggunakan estimasi yang telah dikoreksi sebagaimana ditunjukkan pada Rumus 2.11. Pada rumus tersebut, w_t merupakan bobot pada iterasi ke-t, η adalah *learning rate*, sedangkan σ merupakan konstanta bernilai sangat kecil yang digunakan untuk menjaga stabilitas numerik dan menghindari pembagian dengan nol [37].

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \sigma} \tag{2.11}$$

2.5 Transfer Learning

Transfer learning merupakan teknik pembelajaran pada *deep learning* yang memanfaatkan pengetahuan atau ilmu dari *pre-trained model* yang telah dilatih pada *dataset* berskala besar, seperti *ImageNet*. Pendekatan ini memanfaatkan fitur-fitur umum yang telah dipelajari model, seperti kontur, tekstur, pola-pola tertentu dan bentuk, sehingga model tidak perlu mempelajari representasi fitur dari awal [38]. Dalam penerapannya, lapisan ekstraksi fitur umumnya dipertahankan, sedangkan lapisan klasifikasi disesuaikan dengan jumlah kelas pada *dataset* target. Proses *transfer learning* dapat dilakukan melalui dua pendekatan, yaitu *feature extraction* dengan membekukan (*freeze*) bobot model dasar atau *fine-tuning* dengan melatih kembali sebagian lapisan model agar lebih sesuai dengan karakteristik data target [38]. Rumus *Transfer Learning* bisa dilihat pada Rumus 2.12.

$$\psi^* = \arg\min_{\psi} \mathbb{E}_{(x,y)\sim D_T} \left[L\left(y, G(x; \theta^*, \psi)\right) \right] \tag{2.12}$$

Berdasarkan Rumus 2.12, parameter θ^* menunjukkan bobot EfficientNetV2-B0 yang diperoleh dari proses *pre-training* pada *ImageNet* dan dipertahankan dalam keadaan *freeze* selama proses pelatihan. Sementara itu, parameter ψ merepresentasikan bobot pada lapisan klasifikasi yang disesuaikan melalui proses optimasi menggunakan data *CT Scan* otak [39].

2.6 EfficientNetV2-B0

EfficientNetV2 merupakan pengembangan lanjutan dari arsitektur *EfficientNet* yang dirancang dengan fokus utama pada peningkatan efisiensi pelatihan serta efisiensi parameter. Arsitektur ini dikembangkan melalui pendekatan *training-aware neural architecture search* yang tidak hanya mempertimbangkan akurasi, tetapi juga waktu pelatihan dan jumlah parameter secara simultan [11].

EfficientNetV2-B0 merupakan varian dasar dari keluarga *EfficientNetV2* yang memiliki ukuran model relatif kecil. Arsitektur *EfficientNetV2-B0* dibangun dari kombinasi dua jenis blok utama, yaitu *MBConv* dan *Fused-MBConv*. *MBConv* merupakan blok konvolusi efisien yang memanfaatkan *depthwise convolution* untuk mengurangi jumlah parameter, sedangkan *Fused-MBConv* merupakan modifikasi yang menggabungkan operasi konvolusi untuk meningkatkan kecepatan komputasi, khususnya pada lapisan awal jaringan. Penggunaan *Fused-MBConv* pada tahap

awal bertujuan untuk mengatasi *bottleneck* pelatihan yang disebabkan oleh rendahnya efisiensi *depthwise convolution* pada perangkat keras modern [11].

Selain itu, *EfficientNetV2-B0* juga dirancang dengan strategi *scaling* yang berbeda dibandingkan *EfficientNet* generasi pertama. Pada *EfficientNetV2*, penskalaan tidak dilakukan secara seragam pada seluruh lapisan, melainkan lebih difokuskan pada penambahan kedalaman pada tahap akhir jaringan. Pendekatan ini bertujuan untuk meningkatkan kapasitas model tanpa memberikan beban komputasi yang signifikan. Selain itu, ukuran citra *input* selama pelatihan dibatasi agar tidak terlalu besar, sehingga penggunaan memori lebih efisien dan proses pelatihan dapat berlangsung lebih cepat [11].

Dari sisi proses pelatihan, *EfficientNetV2-B0* memanfaatkan teknik *progressive learning* yang telah ditingkatkan. Metode ini bekerja dengan memulai pelatihan menggunakan ukuran citra yang kecil dan tingkat regularisasi yang rendah, kemudian secara bertahap meningkatkan ukuran citra serta kekuatan regularisasi seperti *dropout* dan augmentasi data. Pendekatan ini memungkinkan model untuk belajar pola sederhana terlebih dahulu sebelum beralih ke representasi yang lebih kompleks, sehingga dapat mempercepat konvergensi sekaligus menjaga akurasi model [11]. Berdasarkan konfigurasi tersebut, rincian arsitektur *EfficientNetV2-B0* dijabarkan di Tabel 2.1.

Tabel 2.1. Arsitektur EfficientNetV2-B0 berdasarkan konfigurasi *v2_base_block*

Stage	Operator	Kernel	Stride	Exp.	Input	Output	Layer	SE
0 (stem)	Conv2D	3×3	2	-	3	32	1	-
1	Fused-MBConv	3×3	1	1	32	16	1	-
2	Fused-MBConv	3×3	2	4	16	32	2	-
3	Fused-MBConv	3×3	2	4	32	48	2	-
4	MBConv	3×3	2	4	48	96	3	0.25
5	MBConv	3×3	1	6	96	112	5	0.25
6	MBConv	3×3	2	6	112	192	8	0.25
7a (head)	Conv2D	1×1	1	-	192	1280	1	-
7b (head)	Global Avg. Pooling	-	-	-	1280	1280	1	-
7c (head)	Fully Connected	-	-	-	1280	*C*	1	-

Sumber: [40]

Arsitektur *EfficientNetV2-B0* yang ditunjukkan pada Tabel 2.1 terdiri dari beberapa tahap utama yang disusun secara bertingkat, dimulai dari lapisan *stem* hingga *head*. Pada tahap awal, digunakan lapisan *Conv2D* dengan *kernel* berdimensi 3×3 dan *stride* 2 untuk menurunkan resolusi citra sekaligus meningkatkan jumlah kanal fitur. lalu, di Stage 1 sampai Stage 3, digunakan blok *Fused-MBConv* yang bertujuan untuk meningkatkan efisiensi komputasi dengan

menggabungkan operasi ekspansi dan konvolusi dalam satu tahap.

Memasuki Stage 4 sampai Stage 6, arsitektur menerapkan blok *MBConv* dengan mekanisme *Squeeze-and-Excitation* (SE) dengan rasio 0,25. SE berfungsi untuk meningkatkan kualitas representasi fitur dengan menekankan kanal yang lebih relevan [41]. Selain itu, pada beberapa tahap digunakan *stride* sebesar 2 diterapkan untuk melakukan proses *downsampling* guna mengurangi ukuran spasial fitur sekaligus meningkatkan jumlah kanal. Pada bagian akhir, fitur diproses melalui lapisan *Conv2D* berukuran 1×1, diikuti dengan *Global Average Pooling*, lalu diakhiri dengan *fully connected layer* sehingga menghasilkan prediksi kelas.

2.6.1 Mekanisme Kerja EfficientNetV2-B0

Mekanisme kerja *EfficientNetV2-B0* diawali dengan citra masukan berukuran $224 \times 224 \times 3$ yang diproses oleh lapisan *Conv2D* menggunakan *kernel* berukuran 3×3 dengan *stride* 2 untuk mengekstraksi fitur-fitur dasar, seperti tepi, tekstur, dan pola lokal. Operasi konvolusi menghasilkan *feature map* baru yang diperoleh melalui proses pergeseran *kernel* pada setiap wilayah citra. Secara matematis, proses konvolusi dapat dilihat pada Rumus 2.1.

Setelah proses ekstraksi fitur awal, *feature map* diproses menggunakan blok *Fused-MBConv* dan *MBConv*. Blok *Fused-MBConv* menggabungkan proses ekspansi kanal dan konvolusi spasial ke dalam satu operasi sehingga lebih efisien pada tahap awal jaringan. Selanjutnya, pada tahap yang lebih dalam digunakan blok *MBConv* yang terdiri atas lapisan ekspansi (*Expansion*), *Depthwise Convolution*, *Squeeze-and-Excitation* (SE), dan *Projection* [42]. Secara umum, proses tersebut dapat dirumuskan sebagai berikut.

$$\text{MBConv}(X) = \text{Proj}\left(\text{SE}\left(\text{DWConv}\left(\text{Expand}(X)\right)\right)\right) + X \tag{2.13}$$

Berdasarkan Rumus 2.13, $Expand(\cdot)$ merupakan konvolusi 1×1 untuk meningkatkan jumlah kanal, $DWConv(\cdot)$ melakukan ekstraksi informasi spasial menggunakan *depthwise convolution*, $\text{SE}(\cdot)$ menyesuaikan bobot setiap kanal berdasarkan tingkat kepentingannya, sedangkan $Proj(\cdot)$ merupakan konvolusi 1×1 untuk mengembalikan jumlah kanal ke dimensi keluaran. Simbol $+X$ menunjukkan *shortcut connection* (*residual connection*) [42].

Di dalam blok *MB-Conv* terdapat mekanisme *Squeeze-and-Excitation* (SE), SE bekerja dalam dua tahap, yaitu *squeeze* dan *excitation*. Tahap *squeeze* meringkas

informasi spasial setiap kanal menjadi satu nilai skalar melalui *global average pooling*, sedangkan tahap *excitation* menghasilkan bobot kepentingan setiap kanal menggunakan dua lapisan *fully-connected*. Secara matematis, mekanisme SE dirumuskan sebagai berikut [42].

$$z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i, j) \tag{2.14}$$

$$s = F'_{ex}(z, W) = \sigma\left(W_2 \cdot \mathrm{ReLU}(W_1 \cdot z)\right) \tag{2.15}$$

$$\tilde{x}_c = s_c \cdot u_c \tag{2.16}$$

Berdasarkan Rumus 2.14, nilai z_c merupakan hasil fungsi *squeeze* $F_{sq}(\cdot)$ yang diperoleh melalui operasi *global average pooling*, yaitu dengan menghitung rata-rata nilai aktivasi u_c pada kanal ke-c dengan tinggi H dan lebar W. Selanjutnya, berdasarkan Rumus 2.15, vektor z diproses oleh fungsi *excitation* $F'_{ex}(\cdot)$ menggunakan bobot W_1 dan W_2 serta fungsi aktivasi ReLU dan *sigmoid* (σ) untuk menghasilkan bobot kepentingan s_c pada setiap kanal. Kemudian, berdasarkan Rumus 2.16, bobot s_c dikalikan dengan *feature map* asli u_c sehingga menghasilkan *feature map* keluaran $\tilde{x}_c$ yang telah disesuaikan berdasarkan tingkat kepentingan masing-masing kanal [43].

Pada tahap akhir, *feature map* yang dihasilkan oleh blok *MBConv* diproses menggunakan lapisan *Conv2D* berukuran 1×1, kemudian direduksi menjadi sebuah vektor fitur melalui *Global Average Pooling*. Untuk rumus *Global Average Pooling* dapat dilihat pada Rumus 2.4.

Vektor hasil GAP selanjutnya diproses oleh lapisan *Dense* dan *Dropout* sebelum dilakukan proses klasifikasi menggunakan fungsi *Softmax*. Untuk rumus *Softmax* dapat dilihat pada Rumus 2.3.

2.6.2 Bottleneck Depthwise Convolution

Setelah memahami mekanisme kerja tiap blok penyusun EfficientNetV2-B0, perlu dikaji pula alasan di balik pemilihan kombinasi blok *Fused-MBConv* digunakan pada tahap awal jaringan alih-alih *depthwise convolution* konvensional. *Depthwise convolution* pada EfficientNet generasi pertama memiliki jumlah parameter dan *FLOPs* yang lebih sedikit dibandingkan konvolusi biasa. Namun,

operasi ini memiliki rasio akses memori (*memory access cost*) yang tinggi terhadap jumlah komputasinya, sehingga kurang mampu memanfaatkan akselerator perangkat keras modern seperti GPU dan TPU secara maksimal, terutama pada lapisan awal yang memiliki ukuran spasial besar [11]. Permasalahan inilah yang mendasari diusulkannya blok *Fused-MBConv*, yang menggantikan *depthwise convolution* 3×3 dan konvolusi ekspansi 1×1 dengan satu operasi konvolusi 3×3, sehingga lebih efisien dijalankan pada akselerator modern [11].

2.6.3 Perbandingan MBConv dan Fused-MBConv

Untuk membuktikan klaim tersebut, Tan & Le (2021) melakukan eksperimen dengan mengganti blok *MBConv* pada *EfficientNet-B4* secara bertahap menggunakan *Fused-MBConv*, sebagaimana ditunjukkan pada Tabel 2.2.

Tabel 2.2. Perbandingan Penggantian *MBConv* dengan *Fused-MBConv*

Konfigurasi	Params (M)	FLOPs (B)	Top-1 Acc.	TPU (img/s)	V100 (img/s)
No fused	19.3	4.5	82.8%	262	155
Fused stage1-3	20.0	7.5	83.1%	362	216
Fused stage1-5	43.4	21.3	83.1%	327	223
Fused stage1-7	132.0	34.4	81.7%	254	206

Sumber: [11]

Berdasarkan Tabel 2.2, penerapan *Fused-MBConv* pada *stage* awal (*stage1-3*) meningkatkan kecepatan komputasi secara signifikan dengan penambahan parameter dan *FLOPs* yang relatif kecil. Namun, jika *Fused-MBConv* diterapkan pada seluruh *stage* (*stage1-7*), jumlah parameter dan *FLOPs* meningkat drastis sementara kecepatan komputasi justru menurun dan akurasi berkurang. Hal ini menunjukkan bahwa kombinasi *MBConv* pada *stage* dalam dan *Fused-MBConv* pada *stage* awal, sebagaimana diterapkan pada EfficientNetV2-B0, merupakan pendekatan yang lebih optimal dibandingkan penggunaan salah satu blok secara menyeluruh.

2.6.4 Perbandingan EfficientNetV2-B0 terhadap EfficientNet-B1

Untuk menunjukkan efisiensi arsitektur yang digunakan pada penelitian ini, Tabel 2.3 membandingkan EfficientNetV2-B0 dengan *EfficientNet-B1* yang memiliki tingkat akurasi setara.

Tabel 2.3. Perbandingan EfficientNetV2-B0 dan EfficientNet-B1

Model	Top-1 Acc.	Params (M)	FLOPs (B)	Throughput (img/s)
EfficientNet-B1	79.0%	7.8	0.7	2.675
EfficientNetV2-B0	78.7%	7.4	0.7	5.739

Sumber: [11]

Berdasarkan Tabel 2.3, EfficientNetV2-B0 memiliki jumlah parameter yang lebih sedikit dan *throughput* inferensi sekitar dua kali lipat lebih tinggi dibandingkan EfficientNet-B1, dengan akurasi yang relatif setara. Hal ini menunjukkan bahwa EfficientNetV2-B0 lebih efisien secara komputasi.

2.7 Evaluasi Model

Evaluasi model adalah cara mengukur seberapa baik model dapat melakukan prediksi secara akurat [44]. Beberapa metrik yang digunakan berupa *confusion matrix*, *accuraccy*, *precision*, *recall*, dan *F1-Score*.

2.7.1 Confusion Matrix

Confusion matrix merupakan bentuk tabel yang digunakan untuk menilai performa model klasifikasi melalui perbandingan antara prediksi model dan label aktual. Matriks tersebut memiliki empat komponen utama, yakni *true positive* (TP) yang menunjukkan jumlah sampel positif yang berhasil dikenali sebagai kelas positif, *true negative* (TN) yang menunjukkan jumlah sampel negatif yang berhasil dikenali sebagai kelas negatif, *false positive* (FP) yang menunjukkan jumlah sampel negatif yang salah dikenali sebagai kelas positif, serta *false negative* (FN) yang menunjukkan jumlah sampel positif yang salah dikenali sebagai kelas negatif. Keempat komponen tersebut menggambarkan seluruh kemungkinan hasil prediksi model, sehingga menjadi dasar dalam perhitungan berbagai metrik evaluasi [44].

2.7.2 Accuracy

Accuracy merupakan metrik yang digunakan untuk mengukur proporsi jumlah prediksi yang benar terhadap keseluruhan data yang diuji. Metrik ini memberikan gambaran umum mengenai seberapa baik model dalam melakukan klasifikasi [44]. Rumus untuk akurasi dapat dilihat pada Rumus 2.17:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.17}$$

2.7.3 Recall

Recall atau yang dikenal juga sebagai *True Positive Rate* mengukur kemampuan model dalam mendeteksi seluruh data positif yang sebenarnya bersifat positif [44]. Untuk *Recall* rumusnya dapat dilihat pada Rumus 2.18:

$$Recall = \frac{TP}{TP + FN} \tag{2.18}$$

2.7.4 Precision

Precision mengukur jumlah prediksi positif yang benar-benar merupakan kasus positif yang sesungguhnya [44]. Rumus *Precision* dapat dilihat pada Rumus 2.19 sebagai berikut:

$$Precision = \frac{TP}{TP + FP} \tag{2.19}$$

2.7.5 F1-Score

F1-score merupakan metrik gabungan yang menggabungkan *precision* dan *recall* dalam satu nilai tunggal melalui rata-rata harmonis. Metrik ini digunakan untuk menyeimbangkan *trade-off* antara *precision* dan *recall*, terutama pada kondisi dataset yang tidak seimbang [44]. Lalu untuk rumus *F1-score* dapat dilihat pada Rumus 2.20:

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{2.20}$$