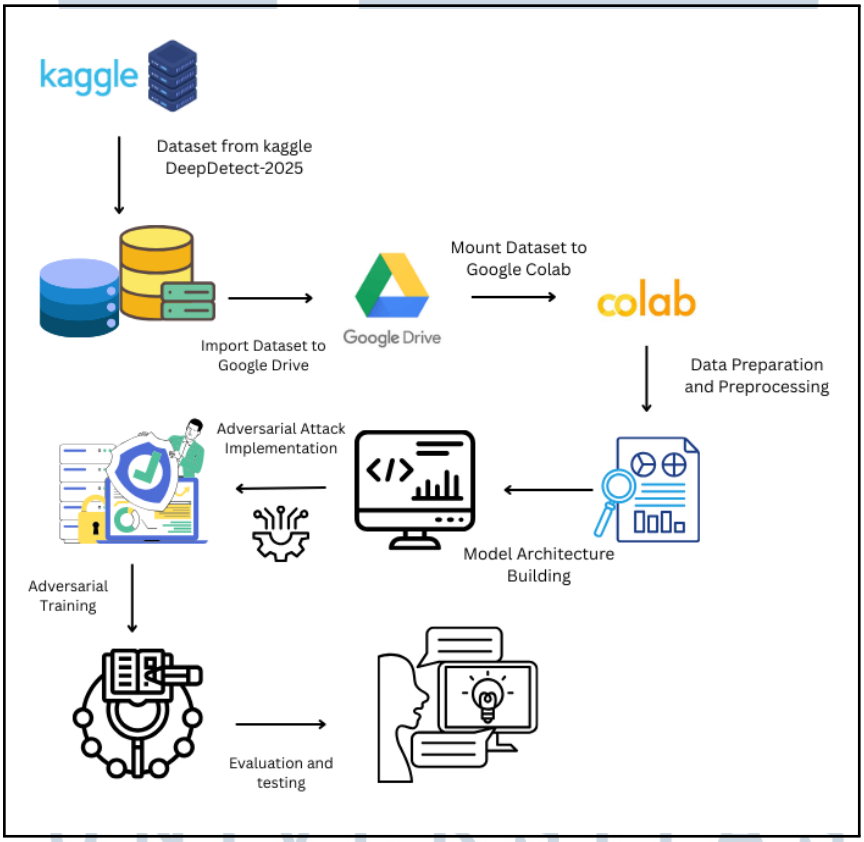
BAB 3
METODOLOGI PENELITIAN

3.1 Diagram Metodologi Penelitian untuk Deteksi Deephoax

Bagian ini menjelaskan Diagram metodologi penelitian yang diterapkan untuk membangun model *EfficientNet-B0* dalam mendeteksi citra *deephoax* dengan mempertimbangkan serangan *adversarial*. Diagram pipeline penelitian disajikan untuk memberikan visualisasi proses secara keseluruhan sebelum masuk ke tahap detail pelatihan dan evaluasi.



Gambar 3.1. Diagram metodologi penelitian untuk deteksi deephoax

Gambar 3.1 menggambarkan alur penelitian mulai dari pengumpulan dataset *DeepDetect-2025* dari platform Kaggle, penyimpanan dan akses data melalui Google Drive dan Google Colab, hingga tahap persiapan data yang meliputi pembersihan, normalisasi citra, dan augmentasi gambar untuk meningkatkan kualitas serta keberagaman data.

Selanjutnya, arsitektur *EfficientNet-B0* disiapkan sebagai model dasar klasifikasi citra *deephoax*. Pipeline ini akan dilanjutkan dengan tahap pelatihan yang menggunakan metode *adversarial training*, di mana model dilatih untuk meningkatkan kemampuan mempertahankan prediksi yang benar terhadap gangguan *adversarial*. Tahap akhir pipeline mencakup evaluasi model menggunakan data uji untuk mengukur performa klasifikasi dan tingkat ketahanan model terhadap serangan *adversarial* [8].

3.1.1 Pengumpulan Data

Tahap awal penelitian melibatkan pengumpulan dan penyiapan dataset yang akan digunakan dalam pengembangan sistem deteksi *deephoax*. Dataset yang digunakan adalah DeepDetect-2025 [58], yaitu dataset publik yang tersedia melalui platform Kaggle dan dirancang khusus untuk tugas klasifikasi citra wajah fakta dan hoax.

Dataset ini terdiri atas citra wajah manusia yang terbagi ke dalam dua kelas utama, yaitu kelas fakta (citra wajah fakta) dan kelas hoax (citra hasil generasi atau manipulasi berbasis kecerdasan buatan). Citra fakta diperoleh dari sumber fotografi autentik dengan variasi latar belakang, pose, dan kondisi pencahayaan. Sementara itu, citra hoax dihasilkan menggunakan berbagai model generatif mutakhir, seperti StyleGAN3, DALL·E 3, Midjourney, dan Stable Diffusion 3.

Seluruh citra dalam dataset telah melalui proses kurasi untuk memastikan kualitas visual yang memadai, resolusi tinggi, serta representasi yang beragam dari segi usia, etnis, dan ekspresi wajah [59]. Untuk menjaga efisiensi komputasi selama proses pelatihan dan evaluasi model, penelitian ini menggunakan subset dataset sebanyak 20.000 citra dengan distribusi kelas yang seimbang. Rincian komposisi dataset yang digunakan disajikan pada Tabel 3.1.

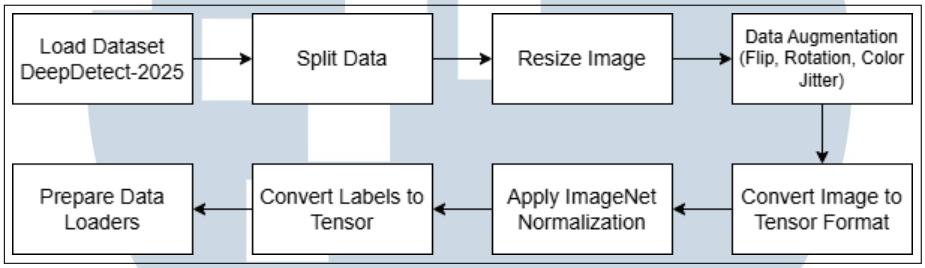
Tabel 3.1. Komposisi dataset DeepDetect-2025 yang digunakan dalam penelitian

Dataset	**Total Sampel**	**Kelas Fakta**	**Kelas Hoax**
DeepDetect-2025 (Subset)	20,000	10,000	10,000

Dataset kemudian diunggah ke Google Drive dan diintegrasikan dengan lingkungan Google Colab menggunakan perintah `drive.mount('/content/gdrive')`. Integrasi ini bertujuan untuk mempermudah akses data serta memanfaatkan akselerasi GPU selama proses pelatihan dan evaluasi model.

3.1.2 Pra-Pemrosesan Data

Tahap pra-pemrosesan data bertujuan untuk menyiapkan dataset agar sesuai dengan kebutuhan arsitektur model serta meningkatkan kualitas data sebelum digunakan dalam proses pelatihan. Pra-pemrosesan dilakukan mengikuti praktik umum dalam pengolahan citra pada *deep learning*, khususnya untuk tugas klasifikasi berbasis *computer vision* [60]. Diagram pipeline pra-pemrosesan data ditunjukkan pada Gambar 3.2.



Gambar 3.2. Diagram pipeline pra-pemrosesan data

Berdasarkan Gambar 3.2, proses pra-pemrosesan dimulai dengan pemuatan dataset *DeepDetect-2025* yang terdiri dari citra fakta dan hoax. Dataset kemudian dibagi menjadi tiga subset, yaitu data pelatihan, validasi, dan pengujian dengan rasio 70:15:15. Pembagian ini dilakukan untuk memastikan ketersediaan data yang memadai pada tahap pelatihan serta menyediakan data yang representatif untuk proses validasi dan evaluasi model [61].

Tabel 3.2. Pembagian dataset DeepDetect-2025 pada tahap pra-pemrosesan

Data	Rasio	Total Sampel	Kelas Fakta	Kelas Hoax
Training	70%	14,000	7,041	6,959
Validation	15%	3,000	1,469	1,531
Testing	15%	3,000	1,490	1,510
Total	100%	20,000	10,000	10,000

Selanjutnya, setiap citra diubah ukurannya menjadi 224×224 piksel agar sesuai dengan dimensi input yang dibutuhkan oleh arsitektur EfficientNet-B0. Untuk meningkatkan variasi data dan kemampuan generalisasi model, diterapkan teknik augmentasi data berupa *random horizontal flip*, *random rotation*, dan *color jittering*. Teknik augmentasi ini bertujuan untuk mensimulasikan variasi kondisi visual yang mungkin muncul pada data uji [62].

Setelah tahap augmentasi, citra dinormalisasi menggunakan nilai rata-rata dan deviasi standar dataset ImageNet, yaitu [0.485, 0.456, 0.406] dan [0.229, 0.224, 0.225]. Normalisasi dilakukan untuk menstabilkan distribusi nilai piksel dan mempercepat proses konvergensi selama pelatihan, sekaligus mendukung pemanfaatan *pretrained weights* pada model EfficientNet-B0 [43, 63]. Tahap selanjutnya adalah konversi citra ke dalam format tensor serta pengubahan label kelas menjadi representasi numerik. Proses pra-pemrosesan diakhiri dengan pembentukan *data loaders* untuk memfasilitasi pemrosesan data secara efisien dalam bentuk *mini-batch* selama pelatihan dan evaluasi model [64].

3.1.3 Konfigurasi Pelatihan dan Serangan Adversarial

Seluruh eksperimen dalam penelitian ini menggunakan arsitektur *EfficientNet-B0*, yang dipilih karena kemampuannya dalam menyeimbangkan akurasi klasifikasi dan efisiensi komputasi pada tugas pengenalan citra wajah [43]. Seluruh citra masukan distandarkan pada resolusi 224×224 piksel sesuai dengan spesifikasi input arsitektur tersebut, sehingga konsistensi dimensi masukan dapat terjaga pada seluruh proses pelatihan dan evaluasi.

Proses optimisasi model dilakukan dengan *batch size* sebesar 32 untuk menjaga stabilitas estimasi gradien sekaligus menyesuaikan keterbatasan sumber daya komputasi. Optimizer Adam digunakan karena kemampuannya dalam menangani gradien yang tidak stasioner serta mempercepat konvergensi pada model *convolutional neural network*. Nilai *learning rate* awal ditetapkan sebesar 1×10^{-4}, mengikuti praktik yang digunakan pada model CNN untuk deteksi *deephoax* berbasis EfficientNet seperti yang dilaporkan oleh [65, 66], dengan tujuan memastikan pembaruan bobot yang halus selama *fine-tuning* dan menjaga stabilitas optimisasi. *Weight decay* sebesar 1×10^{-4} diterapkan untuk mengendalikan kompleksitas model dan mengurangi risiko *overfitting*, khususnya pada skema pelatihan yang melibatkan contoh adversarial [67, 68].

Untuk mengevaluasi ketahanan model terhadap gangguan terstruktur, penelitian ini menerapkan serangan adversarial berbasis *Projected Gradient Descent* (PGD), yang merupakan pendekatan *white-box attack* dalam ruang ancaman norma L_∞ [8]. Parameter ancaman ditetapkan dengan batas maksimum perturbasi $\varepsilon = 8/255$, ukuran langkah $\alpha = 1/255$, serta jumlah iterasi sebanyak 10 langkah. Pengaturan ini dipilih untuk menghasilkan serangan yang cukup kuat tanpa meningkatkan beban komputasi secara signifikan, serta selaras dengan praktik

umum dalam literatur *adversarial robustness* terkini [69–71].

Tabel 3.3 menyajikan ringkasan seluruh konfigurasi pelatihan dan serangan adversarial yang digunakan dalam penelitian ini. Tabel tersebut berfungsi sebagai acuan parameter global yang diterapkan secara konsisten pada seluruh eksperimen, sehingga memastikan bahwa setiap perbedaan performa model yang diamati dapat diatribusikan secara langsung pada strategi pelatihan yang diuji, bukan pada variasi konfigurasi dasar.

Tabel 3.3. Konfigurasi Pelatihan dan Serangan Adversarial

Parameter	**Nilai**
Model	EfficientNet-B0
Image Size	224×224
Batch Size	32
Optimizer	Adam
Learning Rate	1×10^{-4}
Weight Decay	1×10^{-4}
Serangan Adversarial	PGD (L_∞)
ε	$8/255$
α	$1/255$
PGD Steps	10
Random Start	Ya

Seluruh parameter yang dirangkum pada Tabel 3.3 ditetapkan sebagai konfigurasi tetap (*fixed configuration*) pada seluruh eksperimen. Dengan pendekatan ini, penelitian ini menerapkan prinsip kontrol eksperimen, di mana hanya strategi pelatihan adversarial yang divariasikan pada tahap selanjutnya. Pendekatan tersebut memungkinkan analisis yang lebih objektif terhadap pengaruh pelatihan adversarial terhadap akurasi dan ketahanan model, sekaligus meningkatkan validitas komparatif hasil eksperimen.

3.1.4 Skenario Pelatihan

Berdasarkan konfigurasi pelatihan dan serangan adversarial yang telah dijelaskan pada Subbagian 3.1.3, penelitian ini menerapkan tiga skenario pelatihan utama, yaitu *Baseline*, *Adversarial Training (AT)*, dan *Mixed Adversarial Training (Mixed AT)*. Ketiga skenario ini dirancang untuk mengevaluasi pengaruh strategi pelatihan adversarial terhadap akurasi klasifikasi pada data bersih serta ketahanan model terhadap serangan adversarial.

Perbedaan utama antar skenario terletak pada proporsi penggunaan sampel adversarial selama proses pelatihan, sementara seluruh parameter pelatihan dan konfigurasi serangan adversarial lainnya dipertahankan tetap. Dengan pendekatan ini, perbandingan performa antar skenario dapat dilakukan secara adil dan terkontrol.

Ringkasan pengaturan masing-masing skenario pelatihan disajikan pada Tabel 3.4.

Tabel 3.4. Skenario Pelatihan dan Penggunaan Sampel Adversarial

Parameter	**Baseline**	**Adversarial Training (AT)**	**Mixed AT**
PGD Steps	0	10	10
ε	–	$8/255$	$8/255$
α	–	$1/255$	$1/255$
Rasio Adversarial	0%	100%	50%
Epoch	12	8	8
Fokus Validasi	Clean Loss	Adversarial Loss	Adversarial Loss
Random Start (PGD)	–	Ya	Ya

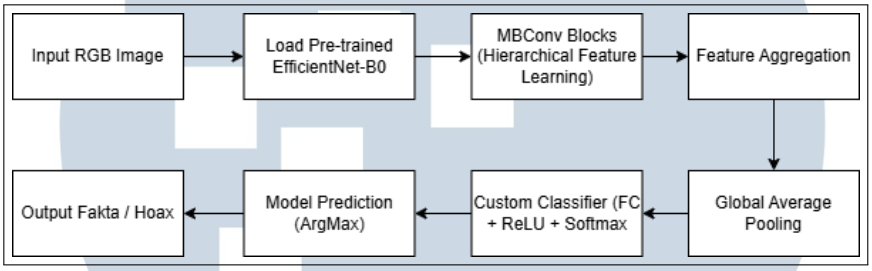
Pada skenario *Baseline*, model dilatih sepenuhnya menggunakan data bersih tanpa melibatkan contoh adversarial. Skenario ini berfungsi sebagai acuan performa dasar untuk mengukur kemampuan model dalam kondisi ideal serta sebagai pembanding terhadap skenario pelatihan yang melibatkan gangguan adversarial.

Skenario *Adversarial Training* (AT) melibatkan pelatihan model menggunakan sampel adversarial yang dibangkitkan melalui serangan *Projected Gradient Descent* (PGD) pada seluruh data latih. Pada setiap iterasi pelatihan, citra masukan diperturbasi secara iteratif untuk memaksimalkan nilai fungsi kerugian model dengan batasan norma L_∞, sehingga proses optimisasi mengikuti formulasi min–max robust optimization sebagaimana dirumuskan oleh Madry et al. [8]. Pendekatan ini bertujuan untuk mengekspos model secara konsisten terhadap gangguan adversarial terburuk (worst-case perturbations), sehingga ketahanan model terhadap serangan adversarial dapat ditingkatkan pada fase inferensi.

Sementara itu, skenario *Mixed Adversarial Training* (Mixed AT) mengombinasikan citra bersih dan citra adversarial dalam setiap *mini-batch* dengan rasio 50:50. Pendekatan ini tetap berada dalam kerangka adversarial training namun dengan proporsi adversarial parsial, yang secara empiris terbukti mampu mencapai tingkat robustness yang kompetitif dibandingkan full adversarial training, sekaligus mempertahankan akurasi pada data bersih dan mengurangi degradasi performa pada data non-adversarial [72, 73].

3.1.5 Pembangunan Arsitektur Model

Arsitektur model dalam penelitian ini dibangun dengan memanfaatkan *EfficientNet-B0* sebagai tulang punggung utama (*backbone*) untuk ekstraksi fitur citra. Model ini dipilih karena memiliki keseimbangan yang baik antara efisiensi komputasi dan kemampuan representasi fitur visual, serta telah terbukti efektif dalam berbagai tugas klasifikasi citra berbasis *deep learning* [43].



Gambar 3.3. Diagram pipeline pembangunan arsitektur model

Gambar 3.3 menunjukkan pipeline arsitektur model yang digunakan dalam penelitian ini. Proses dimulai dari citra masukan RGB dengan ukuran $224 \times 224 \times 3$, yang pertama-tama diproses melalui *Conv Stem* (konvolusi awal) berukuran 3×3, stride 2, dengan 32 output *channels* dan fungsi aktivasi ReLU. Lapisan ini berfungsi untuk menangkap fitur visual dasar seperti tepi, garis, dan tekstur sederhana.

Selanjutnya, citra melewati serangkaian *MBConv Blocks* yang membentuk mekanisme *Hierarchical Feature Learning*. Blok-blok ini bekerja secara bertingkat untuk mengekstraksi fitur dari level lokal hingga global: blok awal menangkap fitur spasial tinggi ($112 \times 112 \times 16$), blok menengah mengekstraksi pola yang lebih kompleks ($56 \times 56 \times 24$ hingga $28 \times 28 \times 40$), dan blok dalam menangkap fitur semantik global ($14 \times 14 \times 80$ hingga $7 \times 7 \times 320$). Pendekatan hierarkis ini memungkinkan model mengenali artefak manipulasi yang tersembunyi pada citra deephoax.

Fitur yang diperoleh dari blok MBConv terakhir kemudian digabungkan melalui tahap *Feature Aggregation* menggunakan *concatenation* atau *weighted sum*, menghasilkan representasi fitur tunggal berukuran $7 \times 7 \times 1280$ yang kompak. Selanjutnya, representasi ini diringkas menjadi vektor fitur global berdimensi 1×1280 melalui *Global Average Pooling* (GAP), sehingga mempermudah pemrosesan oleh lapisan klasifikasi.

Vektor fitur global ini kemudian diproses oleh *Custom Classifier*, yang terdiri dari satu lapisan fully connected (FC) dengan 128 neuron dan fungsi aktivasi

ReLU untuk mempelajari kombinasi fitur non-linear, diikuti oleh lapisan output FC dengan 2 neuron dan fungsi aktivasi softmax untuk menghasilkan probabilitas prediksi kelas biner: *fakta* atau *hoax*. Prediksi akhir diambil dengan memilih kelas dengan probabilitas tertinggi (argmax).

Ringkasan alur pemrosesan data beserta ukuran fitur pada setiap tahap disajikan dalam Tabel 3.5, yang memberikan gambaran struktural pipeline model secara sistematis.

Tabel 3.5. Ringkasan arsitektur model EfficientNet-B0 untuk deteksi deephoax

Tahap	**Komponen**	**Output Shape**
1	Input Image	$224 \times 224 \times 3$
2	Conv Stem (3×3, stride 2)	$112 \times 112 \times 32$
3	MBConv Block 1	$112 \times 112 \times 16$
4	MBConv Block 2 (stride 2)	$56 \times 56 \times 24$
5	MBConv Block 3 (stride 2)	$28 \times 28 \times 40$
6	MBConv Block 4 (stride 2)	$14 \times 14 \times 80$
7	MBConv Block 5	$14 \times 14 \times 112$
8	MBConv Block 6 (stride 2)	$7 \times 7 \times 192$
9	MBConv Block 7	$7 \times 7 \times 320$
10	Conv 1×1 Head	$7 \times 7 \times 1280$
11	Global Average Pooling	1×1280
12	Fully Connected	1×128
13	Fully Connected Output	1×2

Berdasarkan Tabel 3.5, dapat dilihat bahwa EfficientNet-B0 secara progresif mereduksi dimensi spasial citra sambil meningkatkan kedalaman representasi fitur. Lapisan awal fokus pada karakteristik visual dasar, sedangkan lapisan dalam menangkap pola semantik kompleks, termasuk potensi artefak manipulasi pada wajah. Representasi fitur tingkat tinggi yang dihasilkan kemudian diringkas melalui GAP sebelum diteruskan ke *Custom Classifier* untuk menghasilkan prediksi akhir.

Pemilihan arsitektur ini mempertimbangkan kebutuhan efisiensi komputasi untuk mendukung *adversarial training* berbasis *Projected Gradient Descent* (PGD). EfficientNet-B0 memiliki jumlah parameter yang relatif kecil namun mampu mempertahankan kinerja yang baik terhadap serangan PGD pada sumber daya terbatas [11–13], sehingga model ini dipilih sebagai backbone utama dalam penelitian ini.

3.2 Perhitungan Model Architecture

Bagian ini menyajikan contoh perhitungan manual secara sederhana untuk menjelaskan alur pemrosesan data pada arsitektur model yang digunakan. Perhitungan dilakukan pada beberapa tahap utama, yaitu lapisan konvolusi awal (*conv stem*), mekanisme *Global Average Pooling*, dan lapisan *fully connected*. Contoh ini bertujuan memberikan ilustrasi konseptual mengenai cara kerja arsitektur, bukan merepresentasikan bobot aktual pada model *EfficientNet-B0* pralatih.

3.2.1 Perhitungan Lapisan Konvolusi (Conv Stem dan MBConv)

Operasi konvolusi merupakan komponen utama dalam arsitektur EfficientNet-B0 dan digunakan baik pada lapisan konvolusi awal (*conv stem*) maupun pada setiap blok MBConv. Secara umum, setiap lapisan konvolusi dua dimensi yang menggunakan kernel berukuran $K \times K$, *stride* S, dan *padding* P akan menghasilkan ukuran keluaran spasial yang mengikuti persamaan konvolusi standar [74]:

$$H_{out} = \left\lfloor \frac{H_{in} - K + 2P}{S} \right\rfloor + 1, \quad W_{out} = \left\lfloor \frac{W_{in} - K + 2P}{S} \right\rfloor + 1$$

Sebagai contoh awal, lapisan konvolusi pertama (*conv stem*) menerima citra masukan berukuran $224 \times 224 \times 3$ (RGB). Pada tahap ini digunakan kernel konvolusi berukuran 3×3 dengan nilai *stride* sebesar 2, *padding* sebesar 1, dan jumlah *channel* sebanyak 32. Setiap *channel* diterapkan pada seluruh kanal input untuk mengekstraksi fitur dasar seperti tepi dan pola lokal awal.

Dengan:

1. $H_{in} = W_{in} = 224$,
2. $K = 3$,
3. $S = 2$,
4. $P = 1$.

Maka ukuran spasial keluaran konvolusi dihitung sebagai berikut:

$$H_{out} = W_{out} = \left\lfloor \frac{224 - 3 + 2(1)}{2} \right\rfloor + 1 = 112$$

Dengan demikian, lapisan *conv stem* menghasilkan feature map keluaran berukuran:

$$\boxed{112 \times 112 \times 32}$$

Sebanyak 32 *channel* menghasilkan 32 feature map yang berbeda, sehingga dimensi kanal keluaran menjadi 32. Setelah itu, fungsi aktivasi non-linear seperti *SiLU* diterapkan pada setiap elemen feature map sebelum fitur diteruskan ke tahap ekstraksi fitur berikutnya.

Prinsip perhitungan dimensi konvolusi ini juga berlaku pada seluruh blok MBConv di dalam EfficientNet-B0. Pada blok MBConv, operasi *depthwise convolution* dengan kernel 3×3 atau 5×5 dapat menggunakan nilai *stride* sebesar 1 atau 2. Ketika *stride* bernilai 2, ukuran spasial feature map akan berkurang, misalnya dari 112×112 menjadi 56×56, sesuai dengan persamaan konvolusi yang sama. Sebaliknya, operasi *pointwise convolution* (1×1) yang digunakan pada tahap ekspansi dan proyeksi kanal tidak mengubah ukuran spasial feature map, melainkan hanya memodifikasi jumlah kanal keluaran.

3.2.2 Contoh Proses Konvolusi dan Pembentukan Feature Map

Untuk mempermudah pemahaman mekanisme operasi konvolusi, pada bagian ini digunakan contoh sederhana berupa citra masukan berukuran 5×5 dengan satu kanal. Contoh ini tidak merepresentasikan ukuran data sebenarnya yang digunakan pada model, melainkan bertujuan untuk mengilustrasikan proses perhitungan konvolusi secara konseptual dan terperinci dalam skala yang lebih kecil.

Pada implementasi sebenarnya, sesuai dengan arsitektur model yang digunakan, citra masukan memiliki ukuran $224 \times 224 \times 3$ (RGB). Citra tersebut diproses oleh lapisan konvolusi awal (*conv stem*) pada EfficientNet-B0 yang menggunakan kernel berukuran 3×3, *stride* sebesar 2, dan menghasilkan feature map dengan jumlah kanal sebanyak 32. Meskipun ukuran dan jumlah kanal berbeda, prinsip perhitungan konvolusi yang digunakan tetap sama dengan contoh sederhana yang dijelaskan pada bagian ini.

Sebagai ilustrasi, misalkan citra masukan memiliki ukuran 5×5 dengan satu

kanal sebagai berikut:

$$X = \begin{bmatrix} 1 & 2 & 0 & 1 & 0 \\ 0 & 1 & 2 & 1 & 2 \\ 1 & 0 & 1 & 2 & 1 \\ 2 & 1 & 0 & 1 & 0 \\ 1 & 2 & 1 & 0 & 1 \end{bmatrix}$$

Kernel konvolusi yang digunakan berukuran 3×3:

$$K = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Konvolusi dilakukan dengan *stride* = 1 dan *padding* = 0 (*valid convolution*). Dengan konfigurasi tersebut, ukuran feature map output dapat dihitung menggunakan persamaan berikut:

$$H_{\text{out}} = \frac{5-3}{1} + 1 = 3$$

Dengan demikian, citra masukan berukuran 5×5 akan menghasilkan feature map output berukuran 3×3.

A Langkah-Langkah Operasi Konvolusi

Kernel 3×3 digeser di atas citra masukan dari kiri ke kanan dan dari atas ke bawah. Pada setiap posisi, dilakukan operasi perkalian elemen-sejajar antara kernel dan patch input berukuran 3×3, kemudian dijumlahkan untuk menghasilkan satu nilai output. Nilai hasil konvolusi selanjutnya dilewatkan ke fungsi aktivasi ReLU. Fungsi aktivasi yang digunakan pada tahap ini adalah *Rectified Linear Unit* (ReLU), yang didefinisikan sebagai berikut [75]:

$$\text{ReLU}(x) = \max(0, x)$$

Fungsi ReLU berperan dalam memperkenalkan sifat non-linear pada model dengan mempertahankan nilai hasil konvolusi yang bernilai positif dan mengubah nilai negatif menjadi nol. Dengan demikian, hanya fitur-fitur yang memiliki respon kuat terhadap kernel yang akan diteruskan ke lapisan berikutnya.

1. Posisi (1,1) output — kernel berada di kiri atas input:

$$\begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

$$(1\cdot 1)+(2\cdot 0)+(0\cdot -1)+(0\cdot 1)+(1\cdot 0)+(2\cdot -1)+(1\cdot 1)+(0\cdot 0)+(1\cdot -1)=-1$$

Setelah ReLU:

$$\text{ReLU}(-1)=0$$

Nilai ini ditempatkan pada posisi $(1,1)$ pada feature map output.

2. Posisi (1,2) output — kernel digeser satu kolom ke kanan:

$$\begin{bmatrix} 2 & 0 & 1 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

Hasil konvolusi $=0 \Rightarrow \text{ReLU}=0$, ditempatkan pada posisi $(1,2)$.

3. Posisi (1,3) output:

$$\begin{bmatrix} 0 & 1 & 0 \\ 2 & 1 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Hasil konvolusi $=1 \Rightarrow \text{ReLU}=1$, ditempatkan pada posisi $(1,3)$.

4. Posisi (2,1) output — kernel turun satu baris:

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

Hasil konvolusi $=0 \Rightarrow \text{ReLU}=0$.

5. Posisi (2,2) output:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 2 \\ 1 & 0 & 1 \end{bmatrix}$$

Hasil konvolusi $=-1 \Rightarrow \text{ReLU}=0$.

6. Posisi (2,3) output:

$$\begin{bmatrix} 2 & 1 & 2 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

   Hasil konvolusi $= 0 \Rightarrow \text{ReLU} = 0$.

7. Posisi (3,1) output:

$$\begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

   Hasil konvolusi $= 2 \Rightarrow \text{ReLU} = 2$.

8. Posisi (3,2) output:

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{bmatrix}$$

   Hasil konvolusi $= 0 \Rightarrow \text{ReLU} = 0$.

9. Posisi (3,3) output:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

   Hasil konvolusi $= 1 \Rightarrow \text{ReLU} = 1$.

B Feature Map Output

Seluruh nilai hasil konvolusi yang telah melalui fungsi aktivasi ReLU disusun sesuai dengan urutan pergeseran kernel, sehingga diperoleh feature map output berukuran 3×3 sebagai berikut:

$$\text{Feature Map Output} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Sebagai perbandingan, pada arsitektur EfficientNet-B0 yang digunakan dalam penelitian ini, citra masukan berukuran $224 \times 224 \times 3$ setelah melewati lapisan konvolusi awal akan menghasilkan feature map berukuran $112 \times 112 \times 32$. Perbedaan ukuran tersebut disebabkan oleh penggunaan *stride* yang lebih besar

serta jumlah kernel yang lebih banyak, namun mekanisme perhitungan konvolusi yang mendasarinya tetap identik dengan contoh sederhana yang telah dijelaskan.

3.2.3 Perhitungan Global Average Pooling

Keluaran fitur terakhir memiliki ukuran $H \times W \times C$, dengan H dan W masing-masing menyatakan tinggi dan lebar feature map, serta C menyatakan jumlah *channel*. Pada arsitektur EfficientNet-B0, nilai tersebut adalah $H = 7$, $W = 7$, dan $C = 1280$. *Global Average Pooling* menghitung rata-rata nilai pada setiap *channel* fitur [76] sebagai berikut:

$$f_k = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} x_{i,j,k}$$

Pada persamaan tersebut, $x_{i,j,k}$ menyatakan nilai aktivasi feature map pada posisi baris ke-i dan kolom ke-j dari *channel* ke-k. Nilai ini merupakan hasil ekstraksi fitur dari lapisan konvolusi sebelumnya yang telah melalui fungsi aktivasi.

Sebagai contoh, *channel* fitur terakhir memiliki matriks nilai:

$$x_{i,j,k} = \begin{bmatrix} 2 & 1 & 0 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 2 & 1 & 1 \\ 1 & 1 & 0 & 2 & 2 & 1 & 1 \\ 0 & 2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 0 & 2 & 1 \\ 1 & 1 & 1 & 1 & 2 & 0 & 2 \\ 1 & 0 & 1 & 2 & 1 & 1 & 1 \end{bmatrix}$$

Jumlah seluruh elemen dalam *channel*:

$$\sum_{i=1}^{7} \sum_{j=1}^{7} x_{i,j,k} = 98$$

Sehingga nilai GAP untuk *channel* tersebut:

$$f_k = \frac{98}{49} = 2$$

Proses ini menghasilkan vektor fitur berdimensi 1×1280.

3.2.4 Perhitungan Lapisan Fully Connected untuk Dua Kelas

Lapisan *fully connected* menerima vektor fitur hasil *Global Average Pooling* dan menghasilkan skor untuk setiap kelas. Bobot (**w**) dan bias (b) pada lapisan ini merupakan parameter yang dipelajari secara otomatis selama proses pelatihan model melalui mekanisme *backpropagation*. Nilai bobot dan bias pada contoh perhitungan berikut bersifat hipotetik dan hanya digunakan untuk mempermudah ilustrasi proses komputasi. Pada implementasi sebenarnya, seluruh fitur hasil *Global Average Pooling* (misalnya 1280 fitur) digunakan sebagai masukan lapisan ini.

Sebagai ilustrasi, digunakan tiga fitur pertama dari vektor GAP:

$$x = [2, 1, 3]$$

Lapisan *fully connected* memiliki dua neuron keluaran, yaitu kelas fakta dan hoax, dengan bobot dan bias sebagai berikut:

$$w_{\text{fakta}} = [0.5, -0.2, 0.1], \quad b_{\text{fakta}} = 0.1$$

$$w_{\text{hoax}} = [0.3, 0.4, -0.2], \quad b_{\text{hoax}} = 0.2$$

Skor untuk masing-masing kelas dihitung sebagai kombinasi linear antara fitur masukan dengan bobot dan bias.

A Perhitungan skor kelas fakta

$$y_{\text{fakta}} = (2 \cdot 0.5) + (1 \cdot -0.2) + (3 \cdot 0.1) + 0.1 = 1.2$$

B Perhitungan skor kelas hoax

$$y_{\text{hoax}} = (2 \cdot 0.3) + (1 \cdot 0.4) + (3 \cdot -0.2) + 0.2 = 0.6$$

Dengan demikian, vektor skor keluaran lapisan *fully connected* adalah:

$$\mathbf{y} = [y_{\text{fakta}}, y_{\text{hoax}}] = [1.2, 0.6]$$

3.2.5 Perhitungan Softmax dan Prediksi Kelas

Konversi skor ke probabilitas dengan softmax [77]:

$$p_i = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

1. Eksponensial skor:

$$e^{1.2} \approx 3.32, \quad e^{0.6} \approx 1.82$$

2. Jumlah eksponensial:

$$3.32 + 1.82 = 5.14$$

3. Probabilitas:

$$p_{\text{fakta}} = \frac{3.32}{5.14} \approx 0.646$$

$$p_{\text{hoax}} = \frac{1.82}{5.14} \approx 0.354$$

Prediksi akhir:

$$\text{Prediksi} = \arg\max(p_{\text{fakta}}, p_{\text{hoax}}) = \text{fakta}$$

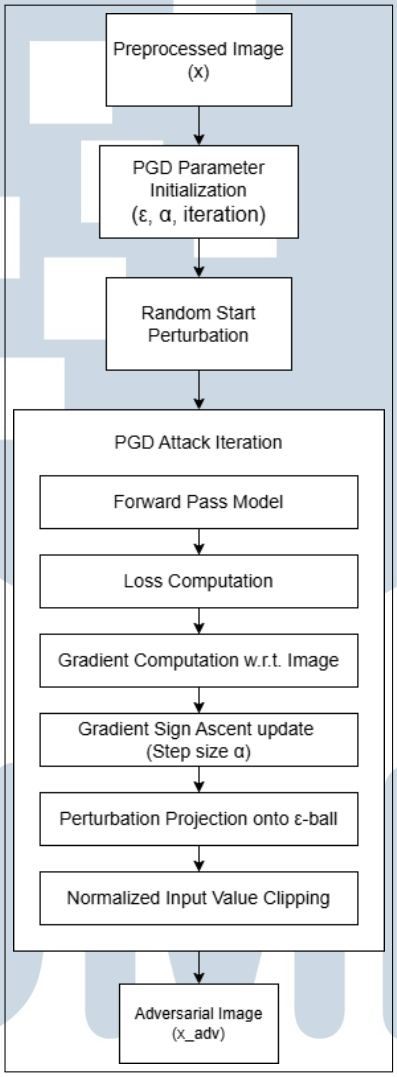
Semua skor neuron dihitung dari bobot dan fitur *Global Average Pooling* (GAP) secara konsisten. Ilustrasi pada bagian ini menggunakan tiga fitur GAP untuk tujuan penyederhanaan, sedangkan pada implementasi sebenarnya seluruh fitur GAP (misalnya 1280 fitur) digunakan pada lapisan *fully connected* untuk klasifikasi dua kelas. Softmax dan ReLU merupakan metode standar yang umum digunakan dalam klasifikasi berbasis CNN [75, 77]. Perhitungan konvolusi serta penentuan ukuran keluaran lapisan mengikuti aturan aritmatika konvolusi standar [74]. Global Average Pooling digunakan untuk mengurangi dimensi fitur secara efektif tanpa melibatkan lapisan *fully connected* berukuran besar [76].

3.2.6 Implementasi Adversarial Attack

Pada penelitian ini, adversarial attack diimplementasikan menggunakan metode *Projected Gradient Descent* (PGD) untuk menghasilkan sampel adversarial yang digunakan dalam proses evaluasi dan pelatihan model. PGD merupakan pengembangan dari metode *Fast Gradient Sign Method* (FGSM) yang bersifat

iteratif, sehingga mampu menghasilkan gangguan (*perturbation*) yang lebih kuat namun tetap berada dalam batasan tertentu [8].

Gambar 3.4 menunjukkan alur implementasi adversarial attack berbasis PGD yang digunakan dalam penelitian ini.



Gambar 3.4. Diagram Pipeline Implementasi Adversarial Attack menggunakan Projected Gradient Descent (PGD)

Proses adversarial attack diawali dengan masukan berupa citra fakta *x* dan label kebenaran *y* yang berasal dari dataset. Citra masukan kemudian dinormalisasi menggunakan nilai *mean* dan *standard deviation* ImageNet agar distribusi data sesuai dengan kondisi pelatihan model *backbone*. Normalisasi ini penting untuk memastikan perhitungan gradien terhadap input berlangsung secara stabil dan konsisten selama proses serangan.

Selanjutnya, parameter utama PGD diinisialisasi, yang meliputi batas maksimum gangguan ε, ukuran langkah pembaruan α, jumlah iterasi, serta penggunaan skema *random start*. Parameter ε menentukan batas perubahan maksimum yang diperbolehkan pada setiap piksel citra, sedangkan α mengontrol besar perubahan pada setiap iterasi serangan. Untuk meningkatkan kekuatan serangan dan menghindari solusi lokal, sampel adversarial awal x_{adv}^{0} dibentuk dengan menambahkan gangguan acak kecil pada citra fakta, yang dibatasi oleh nilai ε.

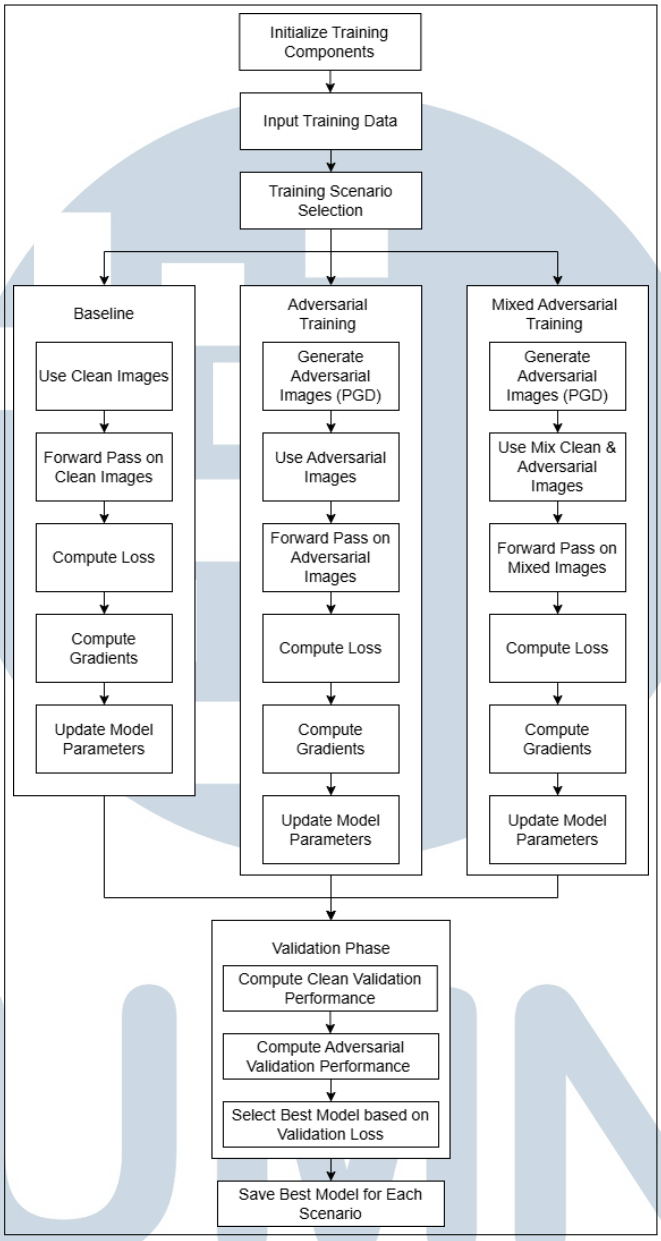
Proses utama PGD dilakukan secara iteratif. Pada setiap iterasi, sampel adversarial sementara dimasukkan ke dalam model melalui proses *forward pass* untuk memperoleh prediksi keluaran, kemudian nilai fungsi kerugian dihitung menggunakan *cross-entropy loss* terhadap label sebenarnya. Gradien fungsi kerugian selanjutnya dihitung terhadap citra masukan untuk menentukan arah perubahan input yang paling meningkatkan nilai kerugian. Sampel adversarial kemudian diperbarui dengan menambahkan gangguan searah tanda gradien (*gradient sign ascent*) yang diskalakan oleh ukuran langkah α.

Agar gangguan yang dihasilkan tetap berada dalam batas yang diperbolehkan, hasil pembaruan tersebut diproyeksikan kembali ke dalam wilayah ε-ball terhadap citra fakta. Selain itu, nilai piksel citra adversarial yang telah ternormalisasi diklip agar tetap berada dalam rentang valid, sehingga tidak menghasilkan nilai input yang tidak realistis bagi model. Langkah-langkah ini diulang hingga jumlah iterasi yang telah ditentukan tercapai.

Hasil akhir dari proses ini adalah sampel adversarial x_{adv} yang secara visual masih menyerupai citra fakta, namun telah dimodifikasi secara terarah untuk memaksimalkan kesalahan prediksi model. Sampel adversarial tersebut selanjutnya digunakan dalam proses adversarial training maupun evaluasi untuk mengukur dan meningkatkan ketahanan model terhadap serangan adversarial berbasis norma ℓ_∞.

3.2.7 *Pipeline* Proses *Training*

Gambar 3.5 menunjukkan diagram *pipeline* proses *training* yang diterapkan dalam penelitian ini. Diagram tersebut menggambarkan alur pelatihan model yang dibagi ke dalam tiga skenario, yaitu *Baseline*, *Adversarial Training*, dan *Mixed Adversarial Training*. Seluruh skenario diawali dengan tahapan inisialisasi komponen pelatihan, pemuatan data latih, serta pemilihan skenario pelatihan sesuai konfigurasi eksperimen.



Gambar 3.5. Diagram pipeline proses *Training*

Pada tahap inisialisasi, sistem memuat arsitektur EfficientNet-B0 sebagai model dasar, mengonfigurasi optimizer AdamW, serta menetapkan fungsi *loss* berupa *Cross-Entropy*. Selain itu, parameter ancaman adversarial ditentukan secara tetap, yaitu batas perturbasi maksimum $\varepsilon = 8/255$, ukuran langkah $\alpha = 1/255$, dan jumlah iterasi serangan PGD sebanyak 10. Konfigurasi ini digunakan secara konsisten pada seluruh skenario untuk memastikan kesetaraan model ancaman selama proses pelatihan dan evaluasi.

Pada skenario *Baseline*, model dilatih menggunakan citra bersih tanpa

penambahan perturbasi adversarial. Setiap batch data diproses melalui *forward pass* untuk menghasilkan prediksi kelas, kemudian dihitung nilai *loss* klasifikasi. Gradien dihitung melalui proses *backpropagation* dan digunakan untuk memperbarui parameter model. Skenario ini merepresentasikan pendekatan *Empirical Risk Minimization* dan digunakan sebagai acuan performa maksimum model pada data bersih.

Pada skenario *Adversarial Training*, setiap batch data latih terlebih dahulu digunakan untuk menghasilkan contoh adversarial dengan serangan *Projected Gradient Descent* (PGD). Contoh adversarial tersebut selanjutnya dijadikan masukan utama pada proses *forward pass*. Nilai *loss* dihitung berdasarkan prediksi terhadap data adversarial, sehingga gradien yang diperoleh mengarahkan proses optimasi untuk meminimalkan kerugian terburuk akibat perturbasi yang dibatasi oleh ε. Prosedur ini sejalan dengan formulasi optimasi robust berbasis *minimax* yang umum digunakan dalam *adversarial training* [8].

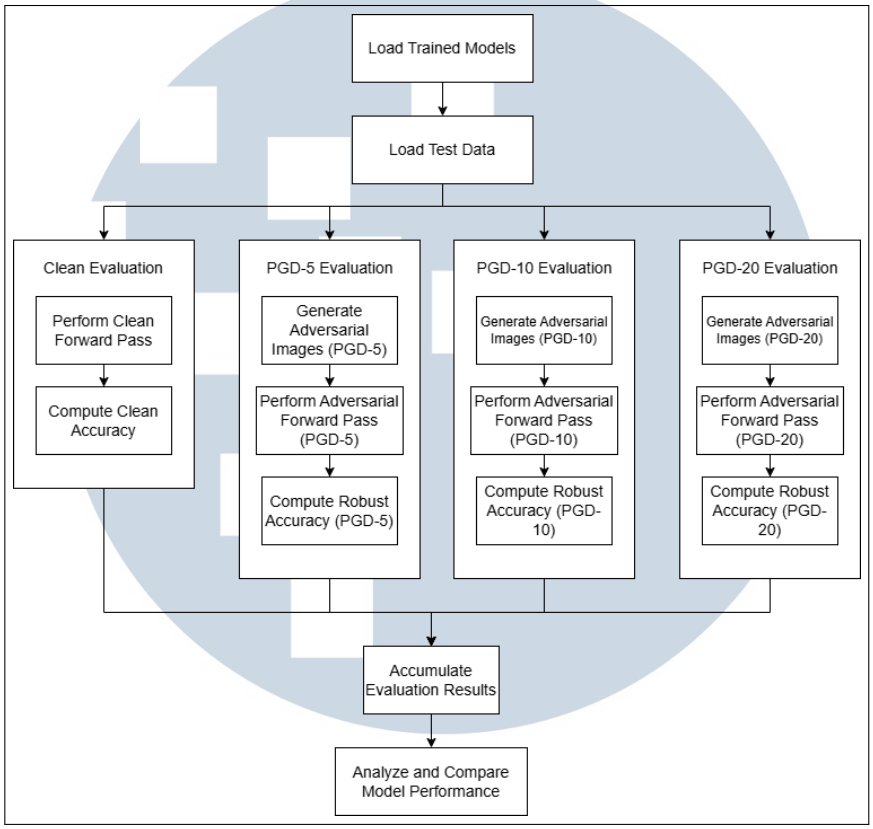
Pada skenario *Mixed Adversarial Training (Mixed AT)*, setiap *mini-batch* berisi kombinasi data bersih dan data adversarial yang dihasilkan menggunakan PGD sesuai rasio yang ditentukan (misalnya 50%). Model melakukan *forward pass* terhadap batch campuran ini dan menghitung satu fungsi *loss* untuk memperbarui parameter melalui *backpropagation*. Pendekatan ini bertujuan menyeimbangkan akurasi pada data bersih dan robustnes terhadap serangan adversarial, sehingga mengurangi degradasi performa yang biasa terjadi pada pelatihan adversarial penuh [72, 73].

Setelah setiap epoch pelatihan, model memasuki tahap validasi. Evaluasi dilakukan menggunakan data validasi bersih dan data validasi adversarial untuk memperoleh gambaran performa yang komprehensif. Pemilihan model terbaik pada setiap skenario didasarkan pada fokus validasi yang telah ditetapkan, yaitu loss bersih untuk skenario *Baseline* dan loss adversarial untuk skenario berbasis *adversarial training*. Model dengan performa terbaik selanjutnya disimpan sebagai model akhir untuk masing-masing skenario pelatihan.

3.2.8 Evaluasi Model

Evaluasi performa model dilakukan menggunakan data pengujian (*test set*) yang tidak digunakan pada tahap pelatihan maupun validasi. Tujuan utama evaluasi ini adalah untuk mengukur performa deteksi *deephoax* pada kondisi normal serta ketahanan model terhadap serangan *adversarial* dengan tingkat kekuatan

yang berbeda. Proses evaluasi mengikuti alur *pipeline* sistematis sebagaimana ditunjukkan pada Gambar 3.6.



Gambar 3.6. Diagram pipeline proses evaluasi model

Tahapan evaluasi diawali dengan pemuatan model terlatih dari setiap skenario pelatihan, yaitu *Baseline*, *Adversarial Training*, dan *Mixed Adversarial Training*. Selanjutnya, data pengujian dimuat secara terpisah untuk memastikan bahwa seluruh hasil evaluasi mencerminkan kemampuan generalisasi model pada data yang belum pernah dilihat sebelumnya.

Evaluasi dilakukan melalui empat skenario pengujian. Pada skenario pertama, yaitu *Clean Evaluation*, model diuji menggunakan citra uji tanpa penambahan perturbasi. Pada tahap ini, dilakukan *forward pass* standar untuk memperoleh prediksi kelas, yang kemudian digunakan untuk menghitung *clean accuracy* sebagai representasi performa dasar model pada kondisi normal.

Selanjutnya, evaluasi ketahanan model dilakukan menggunakan serangan *Projected Gradient Descent* (PGD). Pada skenario *PGD-5 Evaluation*, citra uji terlebih dahulu diubah menjadi contoh adversarial menggunakan lima iterasi PGD. Contoh adversarial tersebut kemudian digunakan sebagai masukan model

untuk menghitung *PGD accuracy* pada tingkat serangan tersebut. Prosedur yang sama diterapkan pada skenario *PGD-10 Evaluation* dan *PGD-20 Evaluation*, dengan jumlah iterasi PGD masing-masing sepuluh dan dua puluh, sehingga merepresentasikan serangan dengan tingkat kekuatan yang semakin meningkat.

Pada tahap evaluasi adversarial, seluruh eksperimen menggunakan batas perturbasi maksimum yang sama, yaitu $\varepsilon = 8/255$, untuk menjaga konsistensi *threat model*. Ukuran langkah serangan PGD ditetapkan tetap, yaitu $\alpha = 1/255$, pada seluruh skenario evaluasi. Penetapan nilai α yang konstan bertujuan untuk menjaga stabilitas proses optimasi selama iterasi serangan serta memastikan bahwa adversarial examples yang dihasilkan tetap berada dalam batas perturbasi maksimum ε. Dengan pengaturan ini, perbandingan performa model pada berbagai tingkat iterasi PGD dapat dilakukan secara adil dan konsisten.

Hasil evaluasi dari setiap skenario kemudian digunakan untuk menghitung metrik utama yang menjadi fokus penelitian, yaitu *clean accuracy*, *PGD accuracy* pada berbagai tingkat serangan PGD dan *attack success rate*. *Attack success rate* merepresentasikan proporsi sampel yang berhasil disalahklasifikasikan akibat serangan adversarial.

Analisis terhadap keempat metrik tersebut dilakukan secara komparatif untuk mengamati degradasi performa model dari kondisi bersih hingga kondisi adversarial dengan serangan PGD yang lebih kuat. Dengan demikian, evaluasi ini memberikan gambaran yang jelas mengenai trade-off antara akurasi dasar dan ketahanan model terhadap serangan adversarial sesuai dengan tujuan penelitian.

3.2.9 Pustaka yang Digunakan

Implementasi penelitian ini memanfaatkan ekosistem pustaka Python untuk mendukung seluruh tahapan pengembangan dan evaluasi model, mulai dari pemrosesan citra, pelatihan model, hingga evaluasi ketahanan terhadap serangan adversarial. Pemilihan pustaka didasarkan pada stabilitas, efisiensi komputasi, serta kesesuaian dengan kebutuhan evaluasi robust model deep learning.

Adapun pustaka utama yang digunakan dalam penelitian ini adalah sebagai berikut:

1. PyTorch [64], digunakan sebagai framework utama untuk implementasi arsitektur EfficientNet-B0, pelatihan model, serta perancangan dan penerapan skema *adversarial training* dan evaluasi robust.

2. Torchvision [78], digunakan untuk menyediakan model pra-latih EfficientNet-B0 serta berbagai transformasi citra yang diperlukan pada tahap pra-pemrosesan data.

3. NumPy [79], digunakan untuk komputasi numerik dan perhitungan metrik evaluasi seperti *clean accuracy*, *PGD accuracy*, dan *attack success rate*.

4. Pandas [80], dimanfaatkan untuk pengelolaan dan penyusunan hasil eksperimen dalam bentuk tabel guna mendukung analisis komparatif antar skenario pelatihan.

5. Matplotlib [81], digunakan untuk visualisasi hasil evaluasi model, khususnya perbandingan performa antara kondisi bersih dan kondisi adversarial.

6. Tqdm [82], digunakan untuk menampilkan *progress bar* selama proses pelatihan dan evaluasi model.

7. Pillow (PIL) [83], digunakan untuk pembacaan dan pemrosesan citra dalam berbagai format.

8. Google Colaboratory [84], digunakan sebagai lingkungan komputasi berbasis cloud yang menyediakan akselerasi GPU untuk mendukung pelaksanaan eksperimen.