BAB III
METODOLOGI PENELITIAN

3.1. Tempat Penelitian

Penelitian ini dilakukan di Program Studi Ilmu Teknik Fakultas Teknik Universitas Sriwijaya. Sumber referensi yang digunakan diperoleh dari Perpustakaan Universitas Sriwijaya dan penelurusan internet.

3.2. Waktu Penelitian

Penelitian ini dilaksanakan selama 3 (tiga) tahun, yaitu dari Tahun 2020 hingga Tahun 2023.

3.3. Alat

Penelitian ini dilakukan menggunakan *software Jupyter Notebook* pada aplikasi *Anaconda* yang berbasis bahasa pemograman *Python*.

3.4. Objek Penelitian

Pada penelitian ini data citra yang digunakan yaitu Citra *Chest X-ray* (CXR) yang merupakan citra yang dihasilkan melalui proses teknik *rontgen X-ray* pada bagian dada. Penelitian ini berfokus pada penggunaan teknik *ensemble learning* untuk meningkatkan kinerja klasifikasi penyakit paru-paru. Teknik ini menggabungkan tiga model *deep learning* dengan arsitektur yang berbeda, yaitu *ResNet*, *EfficientNet*, dan *Inception-v3*. Setiap arsitektur akan memproses citra CXR dan menghasilkan klasifikasi terkait penyakit paru-paru. Selanjutnya, *output* dari ketiga model ini digabungkan menggunakan *weighted voting* melalui teknik *ensemble learning* untuk menghasilkan klasifikasi akhir yang lebih akurat.

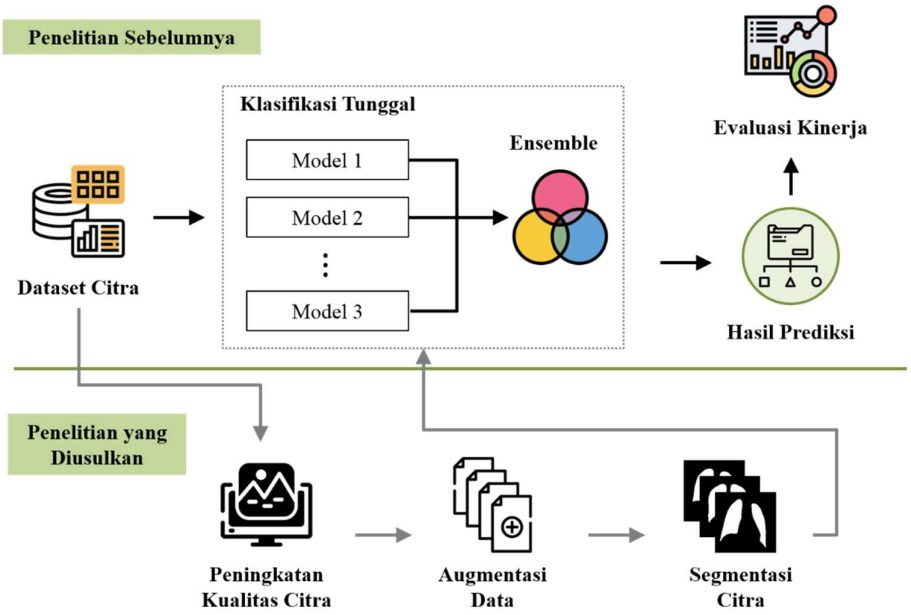
3.5. Pengumpulan Data

Data pada penelitian ini berasal dari sumber data sekunder, yaitu dataset *COVID-19 Radiography Database* yang diperoleh melalui laman https://www.kaggle.com/datasets/tawsifurrahman/covid19-radiography-database.

Dataset ini merupakan kumpulan citra *Chest X-Ray* (CXR) dari proyek penelitian Qatar University, Doha, Qatar, dan University of Dhaka, Bangladesh yang berkolaborasi bersama Pakistan dan Malaysia. Terdapat 4 kelas yang digunakan pada penelitian ini yaitu 3.616 citra *COVID-19*, 10.192 citra normal, 6.012 citra *lung opacity* (*non-COVID*), dan 1.345 citra *pneumonia*.

3.6. Kerangka Kerja Penelitian

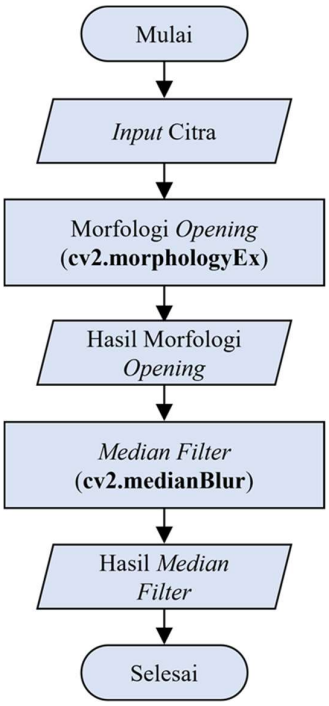
Kerangka kerja pada penelitian *multiclass classification* untuk penyakit berdasarkan citra CXR paru-paru dengan menerapkan *ensemble learning* dapat dilihat pada Gambar 3.1. Pada bagian atas menunjukkan penerapan *ensemble learning* yang umum digunakan berdasarkan literatur dalam beberapa tahun terakhir, sedangkan pada bagian bawah menunjukkan penerapan *ensemble learning* yang diusulkan dengan mengimplementasikan beberapa tahap sebelum memasuki proses klasifikasi citra. Tahap-tahap tersebut yaitu perbaikan kualitas citra untuk memperoleh kualitas citra yang lebih baik sehingga mempermudah proses pengklasifikasian citra, augmentasi citra untuk memperbanyak data latih dengan mentransformasi citra asli pada orientasi yang berbeda, dan segmentasi citra CXR untuk memisahkan objek paru-paru dari objek lainnya.



Gambar 3.1. Kerangka kerja penelitian

3.5.1. Peningkatan Kualitas Citra

Proses peningkatan kualitas citra merupakan tahapan awal dalam proses pengolahan data setelah pengumpulan data yang bertujuan untuk meningkatkan kualitas citra. Tahapan peningkatan kualitas citra pada penelitian ini disajikan dalam bentuk *flowchart* yang dapat dilihat pada Gambar 3.2.



Gambar 3.2. Tahapan Peningkatan Kualitas Citra

Proses peningkatan kualitas citra yang terlihat pada Gambar 3.2 dilakukan dengan langkah-langkah berikut:

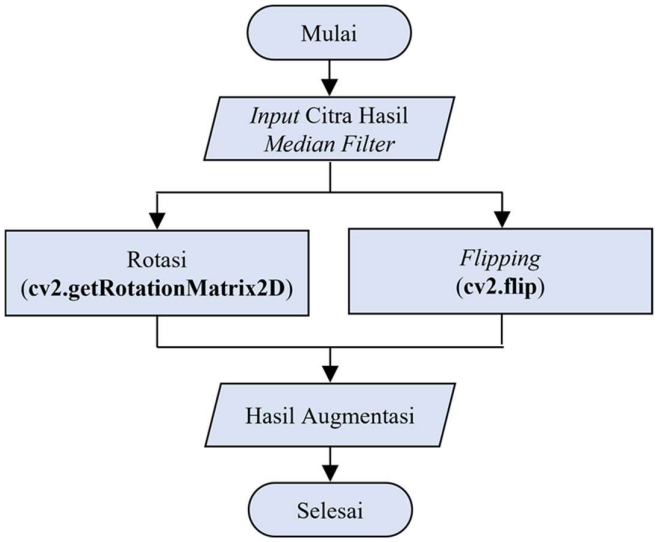
1. Citra CXR yang diperoleh dari dataset *COVID-19 Radiography* dijadikan sebagai *input* citra.
2. Citra yang telah di *input* kemudian dilakukan operasi Morfologi *Opening* dengan menggunakan fungsi cv2.morphologyEx. Operasi ini bertujuan untuk memperbaiki kontur dan menghilangkan objek-objek tipis yang ada pada citra, sehingga menghasilkan citra yang lebih halus dan bersih dengan tepi yang lebih tajam.
3. Citra hasil dari proses Morfologi *Opening* selanjutnya dilakukan proses *Median Filter* menggunakan fungsi cv2.medianBlur. *Median Filter* bertujuan untuk

mengurangi *noise* pada citra dengan memperhatikan distribusi intensitas piksel dalam citra, khususnya piksel-piksel yang memiliki nilai ekstrem.

4. Citra hasil *Median Filter* dijadikan sebagai *Output* sekaligus dapat digunakan untuk proses selanjutnya.

3.5.2. Augmentasi Data

Tahapan augmentasi data bertujuan untuk memperbanyak data latih sehingga setiap kelas akan memiliki jumlah data yang seimbang. Data yang telah dilakukan perbaikan kualitas citra diperbanyak menggunakan teknik transformasi rotasi dan *flipping* yang disajikan dalam bentuk *flowchart* yang dapat dilihat pada Gambar 3.3.



Gambar 3.3. Tahapan Augmentasi Data

Tahapan augmentasi data yang terlihat pada Gambar 3.3 dilakukan dengan langkah-langkah berikut:

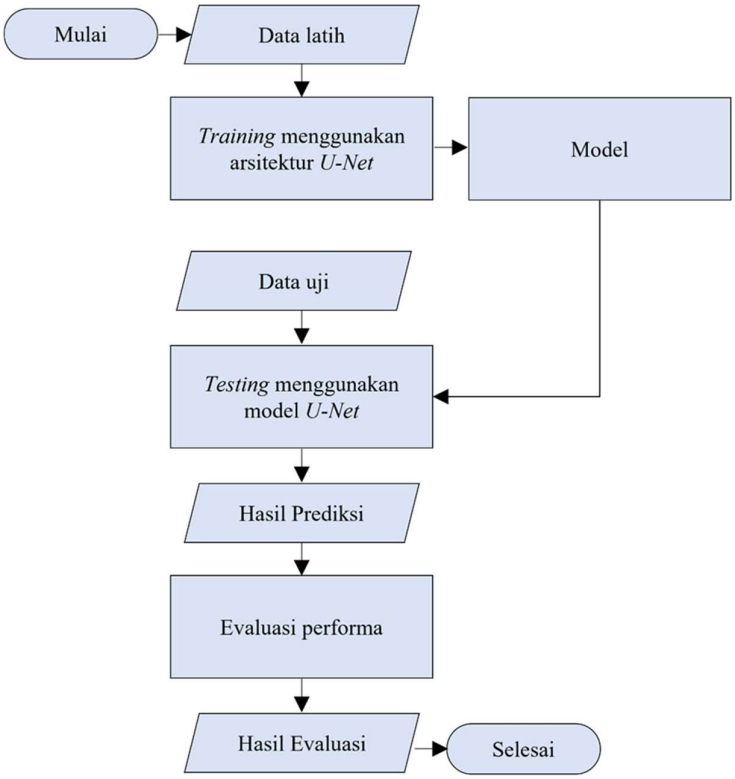
1. Citra hasil *Median Filter* dijadikan sebagai *input* pada proses augmentasi data.
2. Citra yang telah di *input* kemudian dilakukan proses augmentasi dengan menggunakan dua teknik transformasi yaitu rotasi dan *flipping*. Teknik transformasi rotasi menggunakan fungsi cv2.getRotationMatrix2D dengan mengubah sudut antara 1° hingga 359° pada citra. Sedangkan teknik transformasi *flipping* dilakukan dengan memutar gambar secara horizontal dan

vertikal menggunakan fungsi cv2.flip. Kedua teknik ini memberikan variasi baru pada dataset CXR dengan mengubah arah dan sudut orientasi citra sehingga menciptakan variasi yang lebih beragam.

3. Citra hasil augmentasi data dijadikan sebagai *Output* sekaligus dapat digunakan untuk proses selanjutnya.

3.5.3. Segmentasi Citra

Pada tahapan ini, citra hasil augmentasi dilakukan proses segmentasi yang bertujuan untuk membagi citra menjadi dua bagian, yaitu paru-paru (*foreground*) sebagai piksel berwarna putih dan bagian bukan paru-paru (*background*) sebagai piksel berwarna hitam. Segmentasi citra paru-paru dapat membantu mendeteksi penyakit atau kelainan pada paru-paru. Proses segmentasi citra paru-paru terdiri dari dua tahap utama, yaitu *training* data dan *testing* data dengan menggunakan arsitektur *U-Net*. Tahapan segmentasi citra CXR pada penelitian ini disajikan dalam bentuk *flowchart* seperti pada Gambar 3.4.



Gambar 3.4. Tahapan Segmentasi Citra

Pada Gambar 3.4 terlihat bahwa data yang dihasilkan dari proses augmentasi dibagi menjadi dua proses yaitu data latih dan data uji. Data latih digunakan untuk proses *training*, sedangkan data uji digunakan untuk proses *testing*.

- *Training* data

Training data dilakukan untuk membangun model dengan melatih arsitektur terhadap data latih agar dapat mengenali pola dan fitur-fitur yang tersedia pada data. Proses *training* data menggunakan arsitektur *U-Net* dilakukan dengan langkah-langkah sebagai berikut:

1. Inisialisasi parameter: jumlah *epoch* dan ukuran *batch size* untuk satu *epoch*.
2. Lakukan inisialisasi nilai bobot pada *epoch* pertama.
3. *Split* data: Bagi data hasil augmentasi menjadi dua bagian yaitu data latih dan data validasi. Selanjutnya data tersebut diproses ke dalam arsitektur *U-Net*.
4. Jalur *encoder*:
   a. *Input* citra dilakukan operasi *convolutional layer* dengan kernel berukuran 3×3 yang diikuti dengan fungsi aktivasi ReLU.
   b. Hasil dari operasi konvolusi (*feature maps*) selanjutnya dilakukan normalisasi menggunakan *batch normalization*.
   c. Kemudian lakukan operasi *max pooling* dengan ukuran 2×2 untuk mengurangi dimensi *feature maps*.
   d. Ulangi langkah (a) sampai (c) sebanyak empat kali.
5. Jalur *decoder*:
   a. Lakukan peningkatan ukuran *feature maps* menggunakan *transposed convolution* berukuran 2×2.
   b. Gabungkan *feature maps* hasil proses konvolusi pada jalur *encoder* dengan hasi *transposed convolution* pada jalur *decode* menggunakan *concatenate*.
   c. Ulangi langkah (a) dan (b) sebanyak tiga kali.
6. Lakukan kembali proses *convolution layer* dengan kernel berukuran 1×1 dan dilanjutkan dengan fungsi aktivasi *sigmoid*.
7. Hitung nilai *loss function binary cross entrophy*. Jika nilai *loss function* untuk data validasi kurang dari nilai validasi *loss* pada *epoch* sebelumnya maka

bobot akan disimpan dan digunakan untuk *epoch* selanjutnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.

8. Ulangi langkah (2) sampai (7) hingga mencapai *epoch* terakhir.
9. Simpan bobot terakhir ke dalam model *U-Net* untuk digunakan pada saat *testing* data.

- *Testing* data

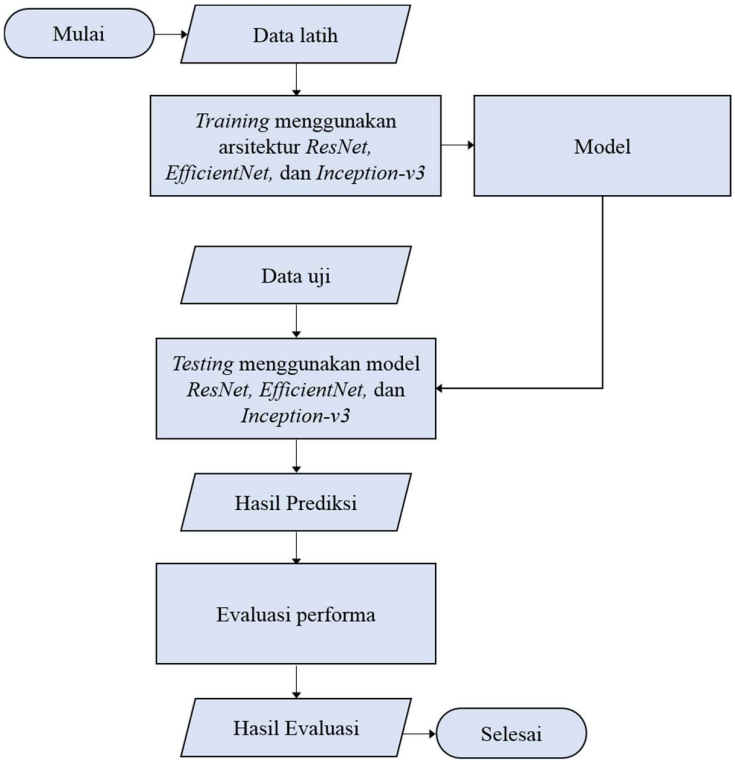
Testing data dilakukan untuk menguji model *U-Net* yang didapatkan dari proses *training* dengan menggunakan data *testing*. Hasil segmentasi menggunakan model *U-Net* menunjukkan bahwa piksel dengan nilai 1 (berwarna putih) merepresentasikan paru-paru, sementara piksel dengan nilai 0 (berwarna hitam) merepresentasikan *background*. Selanjutnya, nilai-nilai ini akan termuat dalam *confusion matrix* yang dapat digunakan untuk menghitung kinerja model *U-Net*.

- Evaluasi kinerja segmentasi

Evaluasi kinerja ini bertujuan untuk mengevaluasi sejauh mana model segmentasi yang diusulkan mampu melakukan segmentasi dengan baik. Pada penelitian ini, evaluasi kinerja yang dihitung yaitu nilai akurasi, presisi, *recall*, dan *F1-Score*.

3.5.4. Klasifikasi Citra

Pada tahapan ini dilakukan klasifikasi penyakit paru-paru menggunakan citra CXR yang telah disegmentasi. Penerapan metode *ensemble learning* dengan teknik *weighted voting* menggunakan arsitektur *ResNet*, *EfficientNet*, dan *Inception-v3* diterapkan untuk mengklasifikasikan citra ke dalam 4 kelas yaitu *COVID-19*, normal, *lung opacity* (*non-COVID*), dan *pneumonia*. Tahapan klasifikasi citra pada penelitian ini disajikan dalam bentuk *flowchart* yang dapat dilihat pada Gambar 3.5.



Gambar 3.5. Tahapan Klasifikasi Citra

Pada Gambar 3.5 terlihat bahwa data yang dihasilkan dari proses segmentasi dibagi menjadi dua yaitu data latih dan data uji. Data latih digunakan untuk proses *training*, sedangkan data uji digunakan untuk proses *testing*. Selanjutnya dilakukan proses *training* menggunakan arsitektur *ResNet, EfficientNet*, dan *Inception-v3*. Hasil *training* berupa model yang akan diuji pada tahap *testing* menggunakan data uji. Proses *testing* akan menghasilkan nilai evaluasi dan klasifikasi kedalam 4 kelas yaitu *COVID-19*, normal, *lung opacity*, dan *pneumonia*.

3.5.4.1. *Training* Menggunakan Arsitektur *ResNet*

Proses *training* data menggunakan arsitektur *ResNet* dilakukan dengan langkah-langkah sebagai berikut:

1. Inisialisasi parameter: jumlah *epoch* dan ukuran *batch size* untuk satu *epoch*.
2. Inisialisasi nilai bobot pada *epoch* pertama.

3. *Split* data: Bagi data hasil segmentasi menjadi dua bagian yaitu data latih dan data validasi. Selanjutnya data tersebut diproses ke dalam arsitektur *ResNet*.
4. *Input* citra dilakukan operasi *convolutional layer* dengan kernel berukuran 7×7 yang diikuti dengan fungsi aktivasi ReLU.
5. Hasil dari operasi konvolusi (*feature maps*) selanjutnya dilakukan normalisasi menggunakan *batch normalization*.
6. Kemudian lakukan operasi *max pooling* dengan ukuran 3×3 untuk mengurangi dimensi *feature maps*.
7. Selanjutnya, hasil operasi *max pooling* diproses ke dalam *conv_block.* Lakukan seperti langkah (4) sampai (6) sebanyak tiga kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 3×3, dan 1×1. Namun, pada saat proses kovolusi ketiga tidak disertai dengan fungsi aktivasi *ReLU*.
8. Hasil operasi *max pooling* juga diproses ke dalam jalur *residual connection*, dimana dilakukan proses *convolutional layer* dengan kernel berukuran 1×1 yang diikuti dengan operasi *batch normalization*.
9. Lakukan penambahan hasil *feature maps* dari langkah (7) dan (8) menggunakan operasi *add*. Selanjutnya hasil tersebut dilakukan operasi fungsi aktivasi ReLU.
10. Hasil dari fungsi aktivasi ReLU diproses ke dalam *identity_block,* dimana proses ini dilakukan seperti pada langkah (7).
11. Lakukan penambahan hasil *feature maps* dari langkah (9) dan (10) menggunakan operasi *add*.
12. Ulangi langkah (10) dan (11) dengan menggunakan hasil *feature maps* dari langkah (11).
13. Ulangi langkah (6) sampai (11) sebanyak 3 kali, namun pada *identity_blok* diulang secara berturut-turut sebanyak tiga kali, lima kali, dan dua kali.
14. Lakukan pengurangan ukuran *feature maps* menggunakan *global average pooling.* Selanjutnya hasil tersebut diproses ke dalam *dense layer*, kemudian disubstitusikan ke dalam fungsi aktivasi *sigmoid*.

15. Hitung *loss function categorical cross* entrophy. Jika nilai *loss function* untuk data validasi kurang dari nilai validasi *loss* pada *epoch* sebelumnya maka bobot akan disimpan dan digunakan untuk *epoch* selanjutnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
16. Ulangi langkah (2) sampai (15) hingga mencapai *epoch* terakhir.
17. Simpan bobot terakhir ke dalam model *ResNet* untuk digunakan pada saat *testing* data.

3.5.4.2. *Training* Menggunakan Arsitektur *EfficientNet*

Proses *training* data menggunakan arsitektur *EfficientNet* dilakukan dengan langkah-langkah sebagai berikut:

1. Inisialisasi parameter: jumlah *epoch* dan ukuran *batch size* untuk satu *epoch*.
2. Inisialisasi nilai bobot pada *epoch* pertama.
3. *Split* data: Bagi data hasil segmentasi menjadi dua bagian yaitu data latih dan data validasi. Selanjutnya data tersebut diproses ke dalam arsitektur *EfficientNet*.
4. *Input* citra dilakukan operasi *convolutional layer* dengan kernel berukuran 3×3 yang diikuti dengan *batch normalization* dan fungsi aktivasi ReLU.
5. Selanjutnya hasil dari langkah (4) diproses ke dalam blok *MBConv1* sebanyak satu kali. Pada blok *MBConv1* ini, *feature maps* diproses ke dalam *depthwise covnolution* dengan kernel 3×3, kemudian dilakukan normalisasi menggunakan *batch normalization* yang diikuti dengan fungsi aktivasi ReLU.
6. Selanjutnya, lakukan pengurangan ukuran *feature maps* menggunakan *global average pooling*. Kemudian, dilanjutkan kembali proses *convolution layer* dengan kernel 1×1 sebanyak dua kali.
7. Lakukan proses perkalian antara hasil dari langkah (5) dan (6). Kemudian, dilanjutkan ke dalam proses *convolution layer* dengan kernel 1×1 yang diikuti proses *batch normalization*.

8. Selanjutnya hasil dari langkah (7) diproses ke dalam blok *MBConv6* sebanyak dua kali. Pada blok *MBConv6* ini, dilakukan seperti pada langkah (5) sampai (7) dimana pada proses *depthwise convolution* digunakan kernel berukuran 3×3.
9. Hasil pada langkah (8) diproses ke dalam lapisan *dropout* dan dilanjutkan penambahan *feature maps* hasil pada langkah (7) dan (8).
10. Lakukan kembali proses blok *MBConv6* seperti langkah (8) dan (9) dengan kernel *depthwise convolution* berukuran 5×5 sebanyak dua kali.
11. Lakukan kembali proses blok *MBConv6* seperti langkah (8) dan (9) dengan kernel *depthwise convolution* berukuran 3×3 sebanyak tiga kali.
12. Lakukan kembali proses blok *MBConv6* seperti langkah (8) dan (9) dengan kernel *depthwise convolution* berukuran 5×5 sebanyak tujuh kali.
13. Lakukan kembali proses blok *MBConv6* seperti langkah (8) dan (9) dengan kernel *depthwise convolution* berukuran 3×3 sebanyak satu kali.
14. Lakukan kembali proses *convolutional layer* dengan kernel berukuran 1×1, kemudian dilanjutkan dengan operasi *batch normalization* yang diikuti fungsi aktivasi ReLU.
15. Lakukan pengurangan ukuran *feature maps* menggunaan *global average pooling.*
16. Hasil dari langkah (14) diproses ke dalam *dense layer*, kemudian disubstitusikan ke dalam fungsi aktivasi *softmax*.
17. Hitung *loss function categorical cross entrophy*. Jika nilai *loss function* untuk data validasi kurang dari nilai validasi *loss* pada *epoch* sebelumnya maka bobot akan disimpan dan digunakan untuk *epoch* selanjutnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
18. Ulangi langkah (2) sampai (17) hingga mencapai *epoch* terakhir.
19. Simpan bobot terakhir ke dalam model *EfficientNet* untuk digunakan pada saat *testing* data.

3.5.4.3. *Training* Menggunakan Arsitektur *Inception-v3*

Proses *training* data menggunakan arsitektur *Inception-v3* dilakukan dengan langkah-langkah sebagai berikut:

1. Inisialisasi parameter: jumlah *epoch* dan ukuran *batch size* untuk satu *epoch*.
2. Inisialisasi nilai bobot pada *epoch* pertama.
3. *Split* data: Bagi data hasil segmentasi menjadi dua bagian yaitu data latih dan data validasi. Selanjutnya data tersebut diproses ke dalam arsitektur *Inception-v3*.
4. *Input* citra dilakukan operasi *convolutional layer* dengan kernel berukuran 3×3 yang diikuti dengan operasi *batch normalization* dan fungsi aktivasi ReLU.
5. Lakukan langkah (4) sebanyak dua kali.
6. Lakukan pengurangan ukuran *feature maps* menggunaan *max pooling*.
7. Ulangi langkah (5) dan (6) dengan kernel konvolusi berukuran 1×1.
8. Selanjutnya hasil *max pooling* dari langkah (7) diproses ke dalam blok *Inception A* sebanyak tiga kali. Pada blok *Inception A* terdapat empat jalur, dimana pada jalur pertama dilakukan seperti langkah (iv) sebanyak tiga kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 3×3, dan 3×3. Pada jalur kedua dilakukan seperti langkah (iv) sebanyak dua kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1 dan 5×5. Pada jalur ketiga dilakukan proses pengurangan dimensi *feature maps* menggunakan *average pooling*, kemudian dilanjutkan seperti langkah (4) dengan kernel konvolusi berukuran 1×1. Pada jalur keempat dilakukan seperti langkah (4) dengan kernel konvolusi berukuran 1×1. Selanjutnya, hasil pada setiap jalur digabungkan menggunakan *concatenate.*
9. Hasildari langkah (8) diproses ke dalam blok *Inception B*. Pada blok *Inception B* terdapat tiga jalur, dimana pada jalur pertama dilakukan seperti langkah (4) sebanyak tiga kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 3×3, dan 3×3. Pada jalur kedua

dilakukan proses pengurangan dimensi *feature maps* menggunkan *max pooling*. Pada jalur ketiga, dilakukan seperti langkah (4) sebanyak satu kali. Selanjutnya, hasil pada setiap jalur digabungkan menggunakan *concatenate.*

10. Hasil dari langkah (9) diproses ke dalam blok *Inception C* sebanyak empat kali. Pada blok *Inception C* terdapat empat jalur, dimana pada jalur pertama dilakukan seperti langkah (4) sebanyak lima kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 7×1, 1×7, 7×1, dan 1×7. Pada jalur kedua dilakukan seperti langkah (4) sebanyak tiga kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 1×7, dan 7×1. Pada jalur ketiga dilakukan proses pengurangan dimensi *feature maps* menggunakan *average pooling*, kemudian dilanjutkan seperti langkah (4) dengan kernel konvolusi berukuran 1×1. Pada jalur keempat dilakukan seperti langkah (4) dengan kernel konvolusi berukuran 1×1. Selanjutnya, hasil pada setiap jalur digabungkan menggunakan *concatenate.*
11. Hasildari langkah (10) diproses ke dalam blok *Inception D*. Pada blok *Inception D* terdapat tiga jalur, dimana pada jalur pertama dilakukan seperti langkah (4) sebanyak empat kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, 1×7, 7×1, dan 3×3. Pada jalur kedua dilakukan seperti langkah (4) sebanyak dua kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, dan 3×3. Pada jalur ketiga dilakukan proses pengurangan dimensi *feature maps* menggunkan *max pooling*, Selanjutnya, hasil pada setiap jalur digabungkan menggunakan *concatenate.*
12. Hasildari langkah (11) diproses ke dalam blok *Inception E* sebanyak dua kali. Pada blok *Inception E* terdapat empat jalur, dimana pada jalur pertama dilakukan seperti langkah (4) sebanyak dua kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×1, dan 3×3. Selanjutnya, dipecah menjadi dua jalur dimana masing-masing jalur dilakukan seperti langkah (4) sebanyak satu kali dengan ukuran kernel

konvolusi secara berurutan berukuran 1×3, dan 3×1 dan di gabungkan menggunakan *concatenate*. Pada jalur kedua dilakukan seperti langkah (4) satu kali dengan kernel berukuran 1×1. Selanjutnya, dipecah menjadi dua jalur dimana masing-masing jalur dilakukan seperti langkah (4) sebanyak satu kali dengan ukuran kernel konvolusi secara berurutan berukuran 1×3, dan 3×1 dan digabungkan menggunakan *concatenate*. Pada jalur ketiga dilakukan proses pengurangan dimensi *feature maps* menggunakan *average pooling*, kemudian dilanjutkan seperti langkah (4) dengan kernel berukuran 1×1. Pada jalur keempat dilakukan seperti langkah (4) dengan kernel berukuran 1×1. Selanjutnya, hasil pada setiap jalur digabungkan menggunakan *concatenate.*

13. Lakukan pengurangan ukuran *feature maps* menggunakan *global average pooling*.
14. Hasil dari langkah (13) diproses ke dalam *dense layer*, kemudian disubstitusikan ke dalam fungsi aktivasi *softmax*.
15. Hitung nilai *loss function categorical cross* entrophy. Jika nilai *loss function* untuk data validasi kurang dari nilai validasi *loss* pada *epoch* sebelumnya maka bobot akan disimpan dan digunakan untuk *epoch* selanjutnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
16. Ulangi langkah (2) sampai (15) hingga mencapai *epoch* terakhir.
17. Simpan bobot terakhir ke dalam model *Inception-v3* untuk digunakan pada saat *testing* data.

3.5.4.4. *Training* Menggunakan *Ensemble Learning*

Proses *training* data menggunakan *ensemble learning* dilakukan dengan langkah-langkah sebagai berikut:

1. *Input* citra akan diproses kedalam tiga arsitektur, yaitu *ResNet*, *EfficientNet*, dan *Inception-V3*. Pada proses *training* ini, ketiga arsitektur tidak akan melakukan perubahan nilai bobot, tetapi ketiga arsitektur ini akan mengeluarkan nilai probabilitas.

2. Nilai probabilitas yang dihasilkan oleh masing-masing arsitektur selanjutnya dihitung nilai *softmax*, dan selanjutnya digunakan teknik *weighted voting*.
3. Setelah itu, *Fully Connected Layer* digunakan pada tahap berikutnya untuk penentuan bobot akhir dan manangani *overfitting* selama *training*.
4. Model *ensemble* akan menghasilkan probabilitas baru, dimana *output* tersebut akan dihitung nilai *loss function categorical cross entrophy* dan nilai akurasi nya untuk setiap *epoch*.
5. Lakukan penyimpanan bobot pada *layer weighted voting*. Bobot akan disimpan jika nilai *loss* pada data validasi lebih kecil dari *epoch* sebelumnya, jika tidak maka bobot akan diperbarui untuk *epoch* selanjutnya.
6. Ulangi langkah (2) sampai (5) hingga mencapai *epoch* terakhir.
7. Simpan bobot terakhir ke dalam model *ensemble learning* untuk digunakan pada saat *testing* data.

3.5.4.5. *Testing* pada Klasifikasi Citra

Proses *testing* dilakukan untuk mengevaluasi kinerja model dari masing-masing arsitektur yang telah di *training* dan membandingkannya dengan hasil dari metode *ensemble learning*. Proses ini dilakukan dengan menggunakan data uji yang telah dibagi sebelumnya. Adapun langkah-langkah pada tahapan ini sebagai berikut:

1. Implementasikan bobot yang dihasilkan selama proses *training* pada masing-masing arsitektur.
2. Simpan hasil dari proses *testing* pada setiap arsitektur agar dapat dibandingkan dengan metode *ensemble learning*.
3. Kemudian lakukan klasifikasi menggunakan model ketiga arsitektur dan model *ensemble learning*. Hasil klasifikasi tersebut berupa nilai bobot.
4. Selanjutnya, nilai-nilai ini akan termuat dalam *confusion matrix* yang dapat digunakan untuk menghitung evaluasi kinerja model.

3.5.4.6. Evaluasi Kinerja Klasifikasi Citra

Pada tahapan ini akan membandingkan hasil kinerja klasifikasi tunggal (*ResNet*, *EfficientNet*, dan *Inception-v3*) dengan hasil kinerja metode *ensemble larning* menggunakan ukuran kinerja model berupa nilai akurasi, presisi, *recall*, dan *F1-Score*.