BAB III

METODE DAN PERANCANGAN PROGRAM

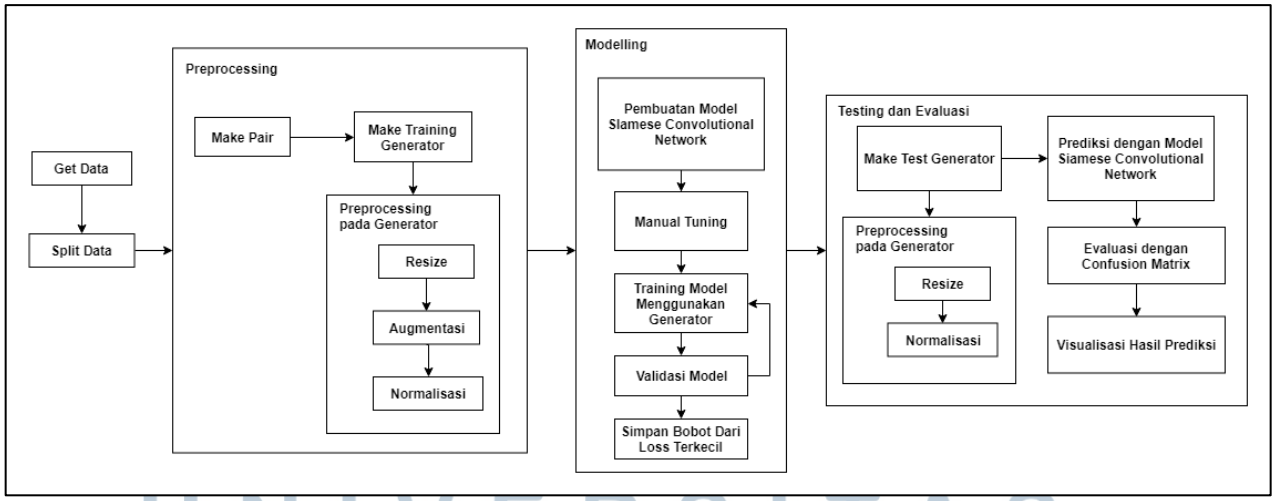
3.1. Metode Penelitian

Metode penelitian yang digunakan dalam mengimplementasikan *siamese convolutional network* pada citra *chest x-ray* untuk mengklasifikasikan penyakit pneumonia adalah sebagai berikut.

1. Studi Literatur

   Pada studi literatur, dilakukan pencarian dan pembelajaran terhadap konsep-konsep pengklasifikasian citra, teori-teori yang berkaitan dengan *siamese convolutional network*, bahasa pemrograman yang akan digunakan, dan juga seputar penyakit pneumonia melalui *e-book*, *e-journal*, dan sumber referensi lain.

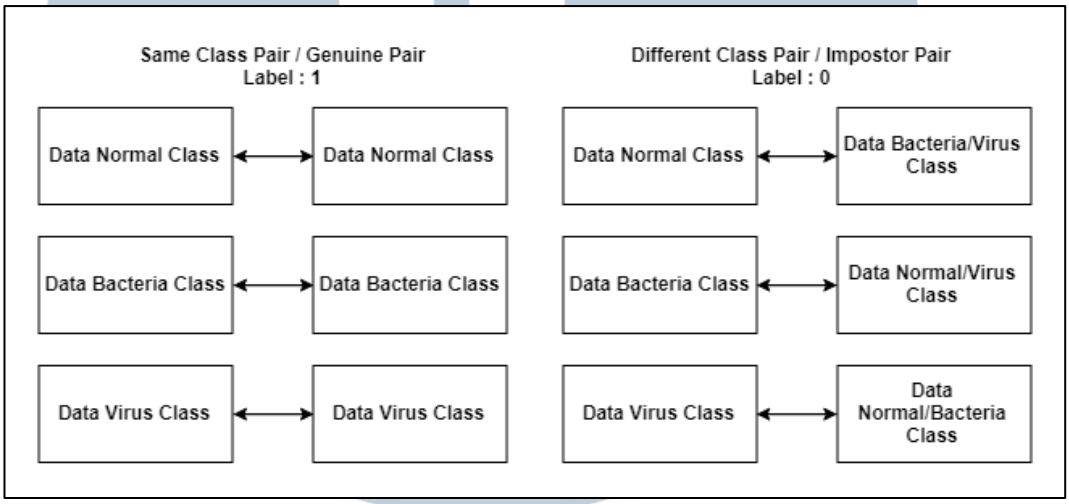
2. Perancangan Program



Gambar 3.1. Kerangka Metode Penelitian Untuk Perancangan Program

Pada tahapan perancangan program, perancangan akan dilakukan sesuai dengan kerangka penelitian pada Gambar 3.1. Pertama-tama akan dilakukan proses *get data* untuk

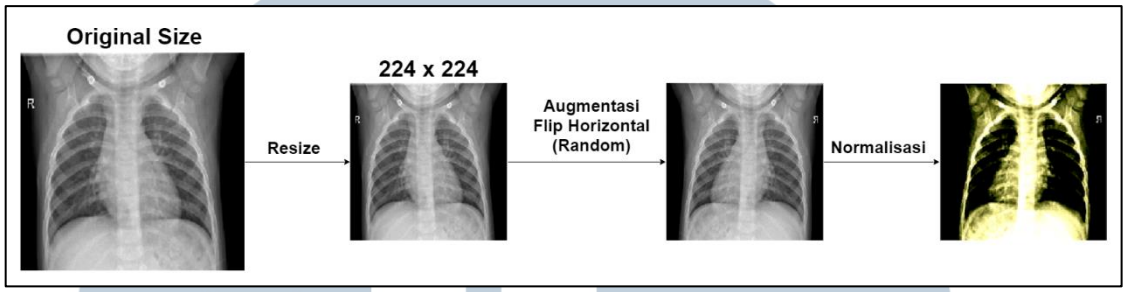
mengambil data dari *directory* yang dipisahkan sesuai dengan kelas masing-masing. Data yang didapatkan masih dalam bentuk *path* dan data tersebut akan melalui proses *split* data untuk dibagi sebesar 72% untuk *training*, 8% untuk validasi, dan 20% untuk *testing* melalui teknik data *split*. Hasil yang didapat berupa 4214 data untuk *training*, 470 data untuk validasi, dan 1172 data untuk *testing*.



Gambar 3.2. Ilustrasi Pembentukan *Pair* Untuk Masing-masing Kelas

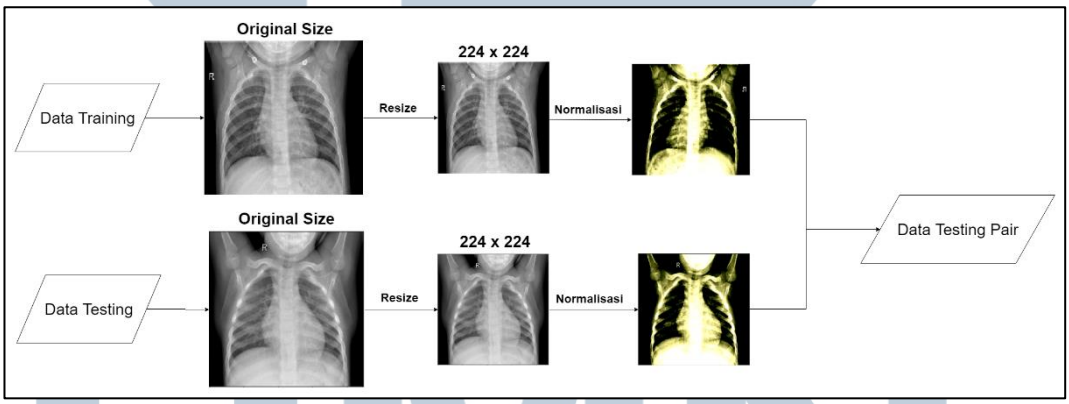
Proses dilanjutkan dengan *preprocessing* data yang akan mengubah bentuk data menjadi bentuk *pair* sesuai dengan bentuk *input* dari *siamese convolutional network* dengan memasangkan 2 buah citra yang memiliki kelas yang sama atau berbeda dengan rasio 1:1. Gambar 3.2. menunjukkan pembentukan pair untuk masing-masing kelas. Kemudian, dilanjutkan dengan pembuatan generator yang dapat menghasilkan citra yang siap untuk proses *training* dan *validasi* dengan beberapa *preprocessing* tambahan seperti *resize* dan augmentasi. Setiap citra akan di-*resize* menjadi 224 x 224 mengacu pada ukuran *input* yang digunakan oleh model VGG (Simonyan dan Zisserman, 2014). Beberapa citra akan dibalik secara horizontal (*flip horizontal*) secara acak. Kemudian, citra juga akan dinormalisasi

menggunakan sampel dari 1000 data *training*. Gambar 3.3. menunjukkan ilustrasi dari proses *preprocessing* yang dijalankan pada generator.



Gambar 3.3. Ilustrasi *Preprocessing* Pada *Training* Generator

Setelah generator telah dibangun, dilakukan perancangan arsitektur *siamese convolutional network* yang akan digunakan. Generator yang dibangun akan digunakan untuk proses tuning secara manual dan juga *training*. Pemilihan bobot akan berdasarkan nilai *loss validation* terkecil.



Gambar 3.4. Ilustrasi *Preprocessing* Pada *Testing* Generator

Bobot yang sudah disimpan akan digunakan untuk proses *testing* dan *evaluasi*. Pertama-tama, akan dibangung generator untuk memasangkan data *testing* dengan sampel dari data *training* secara acak. *Preprocessing* seperti *resize* dan normalisasi juga dilakukan di generator. Gambar 3.4. menunjukkan ilustrasi dari penggunaan generator pada proses *testing*.

Proses dilanjutkan dengan prediksi dari model yang telah dibangun. Hasil prediksi akan dievaluasi menggunakan *confusion matrix*. Gambar 3.5.

menunjukkan ilustrasi dari tabel *confusion matrix* yang akan digunakan untuk menghitung *precision, recall, f1 score*, dan *accuracy*. Selain itu, perancangan untuk masing-masing proses secara lebih detail dan alur *website* yang dikembangkan akan dijelaskan dengan menggunakan *flowchart*.

Predicted	Normal	Bacteria	Virus	Total
Actual				
Normal	a	b	c	a+b+c
Bacteria	d	e	f	d+e+f
Virus	g	h	i	g+h+i
Total	a+d+g	b+e+h	c+f+i	1172

Predicted	Bacteria	Not Bacteria
Actual		
Bacteria	e (True Positive)	d+f (False Negative)
Not Bacteria	b+h (False Positive)	a+c+g+i (True Negative)

Predicted	Normal	Not Normal
Actual		
Normal	a (True Positive)	b+c (False Negative)
Not Normal	d+g (False Positive)	e+f+h+i (True Negative)

Predicted	Virus	Not Virus
Actual		
Virus	i (True Positive)	g+h (False Negative)
Not Virus	c+f (False Positive)	a+b+d+e (True Negative)

Gambar 3.5. Ilustrasi *Confusion Matrix* Untuk Masing-masing Kelas

3. Pembuatan Program

Pada tahapan pembuatan program, dilakukan implementasi berdasarkan perancangan yang telah dilakukan sebelumnya. Implementasi untuk *preprocessing* hingga proses evaluasi model dibuat dengan menggunakan Jupyter Notebook dengan bahasa pemrograman Python. Penggunaan *library TensorFlow* dan beberapa *library* lainnya digunakan untuk meningkatkan performa dalam komputasi dan pembuatan *neural network.* Selanjutnya, dilakukan pengembangan *Application Programming Interface*

(API) dari proses prediksi model tersebut sehingga dapat digunakan untuk pembuatan *website* dalam bahasa pemrograman Javascript dengan *framework* React.js.

4. Proses Training

   Pada tahapan ini, maka akan dilakukan proses *training* terhadap data yang sudah diproses menjadi bentuk berpasangan dengan pembagian data sebesar 10.000 pasang citra untuk *input* dan 2.000 pasang citra untuk validasi.

5. Testing dan Evaluasi

   Pada tahapan ini, dilakukan *testing* dan evaluasi pada model yang telah dibuat. Proses *testing* akan menggunakan data yang berbeda dari proses *training* yang kemudian akan dievaluasi melalui *confusion matrix* dengan dihitung nilai akurasinya. Penghitungan *precision, recall*, dan *f1 score* juga akan dilakukan untuk mendukung validitas tingkat akurasi.

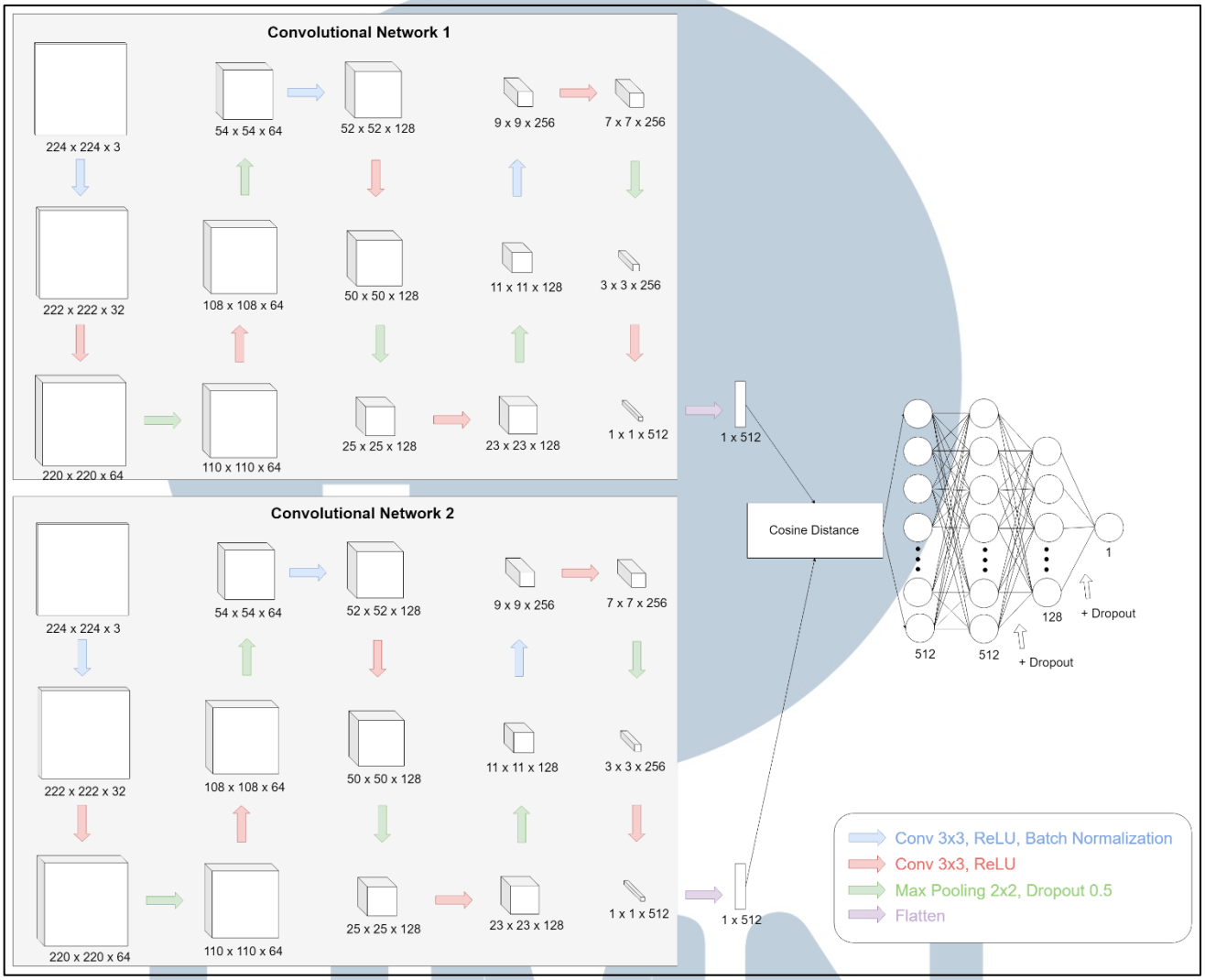
6. Konsultasi dan Penulisan

   Tahapan ini berupa konsultasi secara berkala dengan dosen pembimbing dan menulis laporan berdasarkan penelitian yang dilakukan. Tahap ini akan dijalankan bersamaan dengan tahapan-tahapan lain dalam penelitian.

3.2. Perancangan Program

Perancangan program yang dibuat akan dibagi menjadi dua, yaitu perancangan arsitektur *siamese convolutional network* yang akan digunakan dan perancangan alur keseluruhan program dan *website* dengan menggunakan *flowchart*.

3.2.1. Perancangan Arsitektur Siamese Convolutional Network



Gambar 3.6. Arsitektur *Siamese Convolutional Network*

Model *siamese convolutional network* yang digunakan memiliki arsitektur seperti yang ditunjukkan pada Gambar 3.1. Arsitektur ini dapat dibagi menjadi beberapa bagian sebagai berikut.

1. 2 buah *Convolutional Network*

   Pada setiap *convolutional network*, *input* yang diterima berupa citra dengan ukuran 224 x 224 *pixel*. *Convolutional network* yang digunakan merupakan pengembangan dari arsitektur *convolutional neural network* yang digunakan oleh Saraiva dkk. (2019) dengan penambahan regularisasi,

convolutional layer, dan *pooling layer*. *Network* ini memiliki 14 *layer* yang terdiri dari 9 *convolutional layer*, 5 *max-pooling layer*, *batch normalization regularization*, dan *dropout regularization.* Detail dari masing-masing *layer* dapat dilihat pada Gambar 3.1. Untuk setiap *convolutional layer*, nilai dari stride yang digunakan adalah dua *pixel*, nilai dari *padding* adalah nol, dan ukuran konvolusi adalah 3x3. Spesifikasi tersebut digunakan untuk mengurangi komputasi yang dilakukan, tetapi tidak mengurangi informasi fitur-fitur yang ada pada *image*. Pada *layer* terakhir, akan ditambahkan *flatten layer* untuk mengubah *output* menjadi 2 dimensi yaitu 1 x 512.

2. *Connection Function*

   Connection function yang digunakan pada arsitektur ini adalah *cosine distance*. Fungsi ini akan menghubungkan dua buah *convolutional network* dengan mengevaluasi *similarity* dari kedua *network*.

3. *Fully Connected Network*

   Pada bagian *fully connected network*, *input* yang digunakan merupakan hasil dari *connection function* dengan jumlah neuron sebesar 512 *node*. *Network* ini menggunakan 2 *hidden layer* dengan jumlah neuron tiap *layer* sebesar 512 dan 128 *node*. *Dropout regularization* ditambahkan pada *output* dari setiap *hidden layer*. *Output layer* berupa 1 buah neuron yang merepresentasikan hasil dari *similarity* antara dua buah citra.

4. *Cost Function*

   Cost function yang digunakan adalah *cross-entropy loss function* yang dihitung berdasarkan *output* dari *fully connected network*.

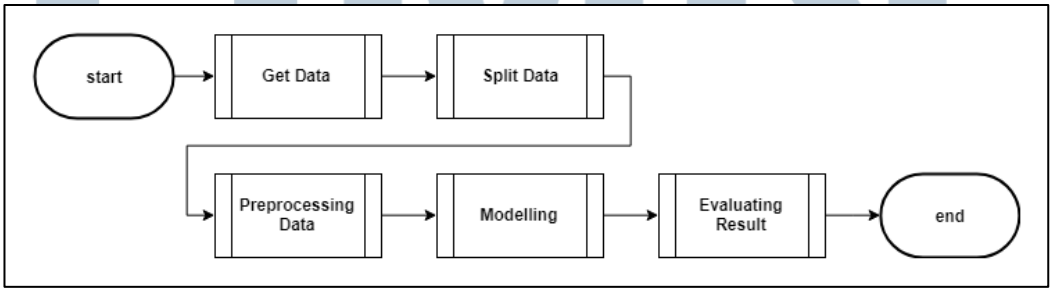
Arsitektur ini akan melalui proses *training* menggunakan *mini-batch gradient descent* dengan masing-masing *batch* berukuran 16 data dan *Adam optimizer* dengan parameter *default* ($\beta_1 = 0.9$ dan $\beta_2 = 0.999$). *Hyperparameter* lain seperti *learning rates* dan *dropout rates* pada *dropout regularization* ditentukan melalui proses *tuning* secara manual. Pemilihan model akan ditentukan berdasarkan nilai *validation loss* yang paling kecil.

3.2.2. Flowchart

Perancangan dengan *flowchart* dibagi menjadi 2 bagian, yaitu *flowchart* untuk *all pipeline* atau keseluruhan alur program yang mencakup proses filter data hingga proses evaluasi data *testing* dan *flowchart* untuk aplikasi *website* yang dikembangkan.

A. Flowchart All Pipeline

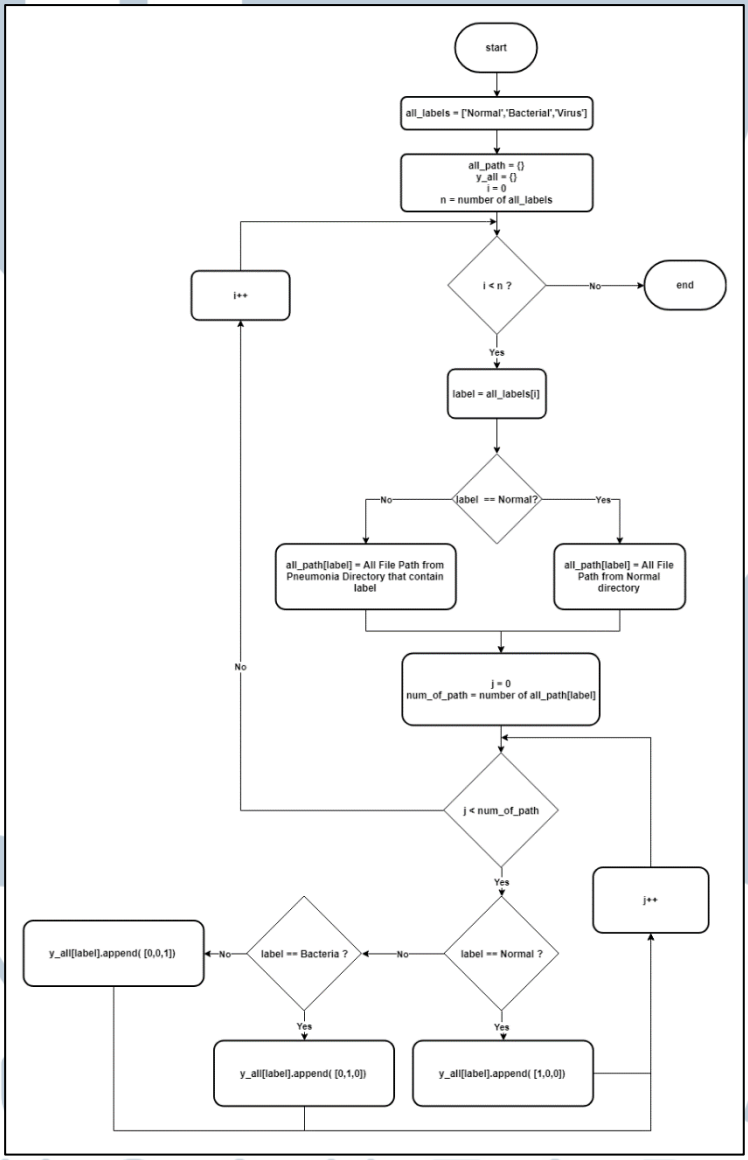
Alur program secara keseluruhan dijelaskan melalui perancangan diagram alur atau *flowchart* yang ditunjukkan pada Gambar 3.2. Program akan dibagi menjadi 5 proses utama yaitu *get data, split data, preprocessing data, modelling,* dan *evaluating result.*



Gambar 3.7. *Flowchart* Alur Program Secara Keseluruhan.

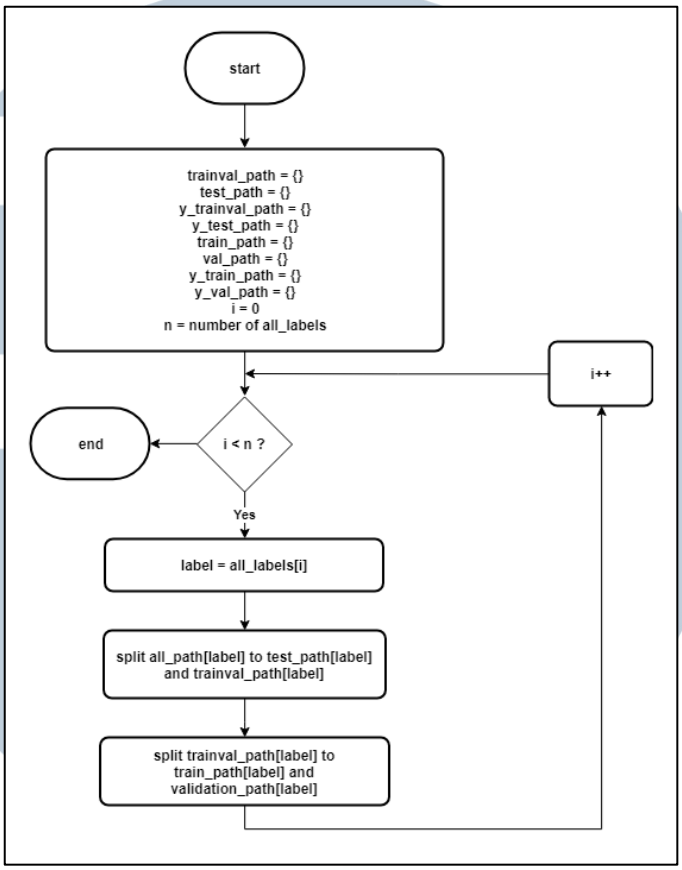
A.1. Flowchart Get Data

Pengembangan *flowchart* untuk proses *get data* dapat dilihat pada Gambar 3.3. Pada proses ini, dilakukan pengambilan data yang akan digunakan untuk proses *training* hingga *testing*. Data-data tersebut akan dipisahkan berdasarkan label atau target kelasnya dan disimpan pada variabel *all_path*. Data yang diambil hanya berupa *path* dari tempat penyimpanan data tersebut. Label pada setiap data juga akan ditampung pada variabel *y_all* dalam bentuk *one-hot encoder*.



Gambar 3.8. *Flowchart* Proses *Get Data*.

A.2. Flowchart Split Data



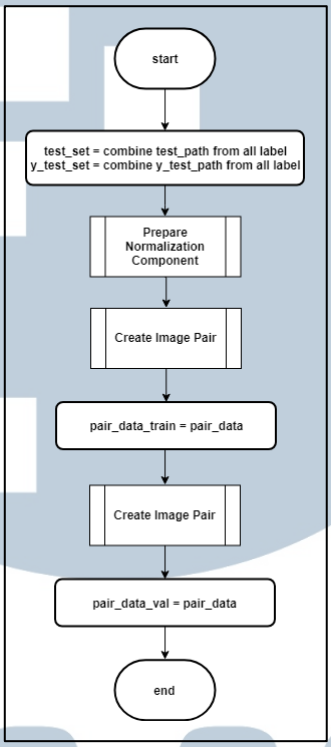
Gambar 3.9. *Flowchart* Proses *Split Data*.

Pada proses *split data*, *flowchart* dapat dilihat pada Gambar 3.4. Proses dimulai dengan membagi data *all_path* dan *y_all* pada masing-masing label menjadi data *training* (*train_path* dan *y_train_path*), data *validation (val_path* dan *y_val_path)*, dan data *testing (test_path* dan *y_test_path)* dengan pembagian sebesar 72%, 8%, dan 20%.

A.3. Flowchart Preprocessing Data

Pada proses *preprocessing* data, *flowchart* akan dijabarkan pada Gambar 3.4. Untuk keperluan klasifikasi, data *testing* pada seluruh label akan digabungkan menjadi satu terlebih dahulu. Kemudian, subproses *prepare normalization component* akan dijalankan untuk mendapatkan nilai *mean* dan *standard deviation*

yang nantinya akan digunakan untuk proses normalisasi. Proses dilanjutkan dengan fungsi *create image pair* untuk mendapatkan bentuk *input* yang sesuai dengan arsitektur *siamese convolutional network*. Bentuk *input* yang didapatkan adalah bentuk citra secara berpasangan. Fungsi ini akan dijalankan dua kali yaitu untuk data *training* sebagai *pair_data_train* dan data *validation* sebagai *pair_data_val*.

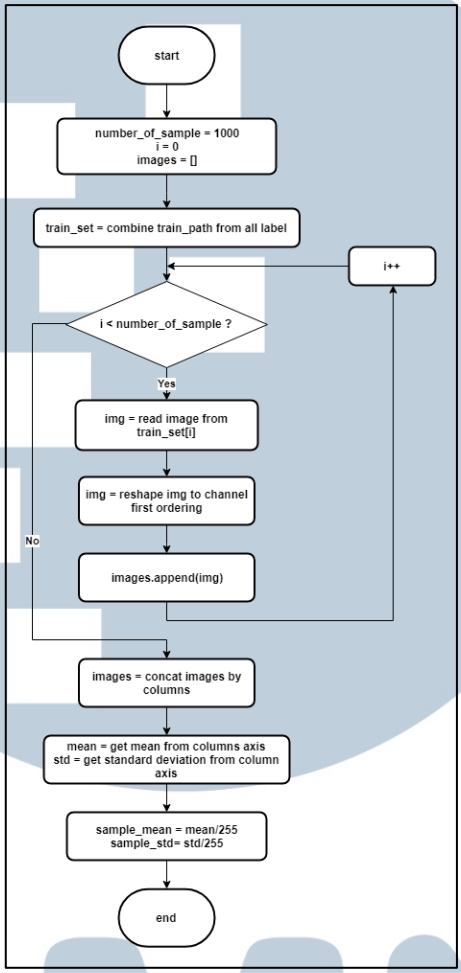


Gambar 3.10. *Flowchart* Proses *Preprocessing Data*.

A.3.1. Flowchart Prepare Normalization Component

Flowchart pada subproses *prepare normalization component* dapat dilihat pada Gambar 3.5. Subproses ini akan menyiapkan komponen untuk normalisasi citra dengan menggunakan *mean* dan *standard deviation* dari 1000 data sampel. Pertama-tama, *train_set* yang berisi data *training* masing-masing label akan digabungkan menjadi satu. Pada setiap data *train_set*, *image* akan dibaca dan di-*reshape* menjadi bentuk 2 dimensi dengan *channel* warna pada posisi depan.

Keseluruhan *image* akan digabungkan dan dicari nilai *mean* dan *standard deviation*-nya sebagai variabel *sample_mean* dan *sample_std*.

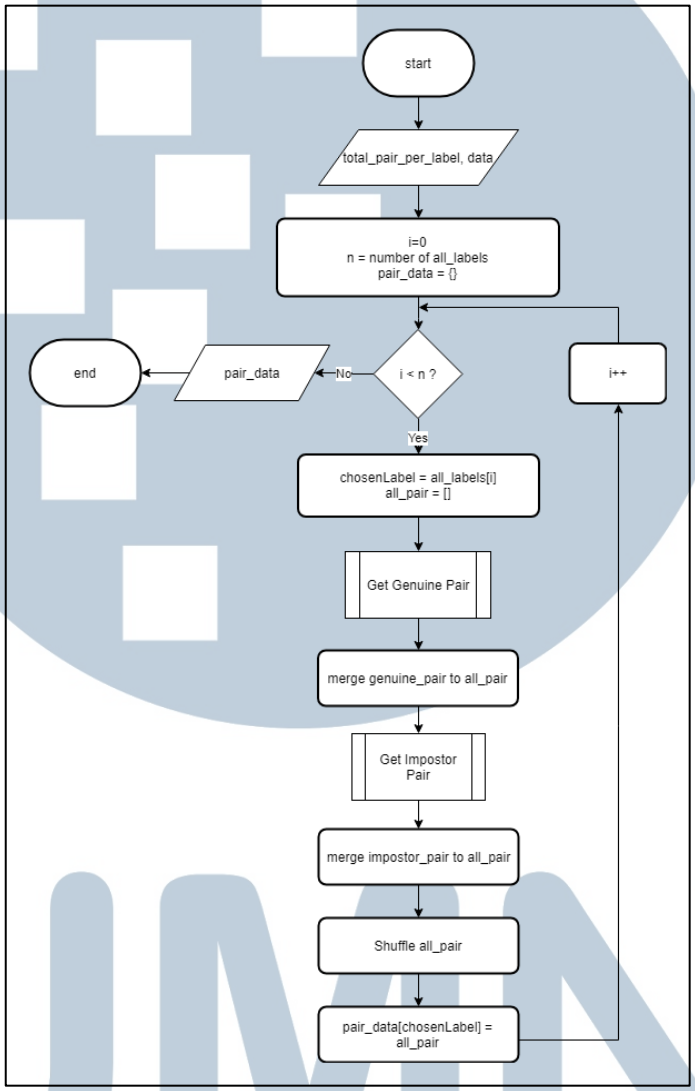


Gambar 3.11. *Flowchart* Subproses *Prepare Normalization Component*

A.3.2. Flowchart Create Image Pair

Flowchart pada fungsi *create image pair* dapat ditunjukkan pada Gambar 3.6. Fungsi ini digunakan untuk menghasilkan *input* citra dalam bentuk berpasangan dengan label 0 untuk pasangan citra yang memiliki kelas yang berbeda dan label 1 untuk pasangan citra yang memiliki kelas yang sama. Fungsi akan menerima parameter *input* berupa jumlah pasangan untuk tiap label dan data yang berisi *path* dari citra. Berdasarkan data yang diterima, fungsi akan memanggil fungsi *get genuine pair* untuk mendapatkan pasangan data dengan kelas yang sama dan fungsi

get impostor pair untuk mendapatkan pasangan data dengan kelas yang berbeda. *Output* dari kedua subproses tersebut akan digabungkan sebagai variabel *all_pair* dengan posisi dari tiap data akan di-*shuffle*.

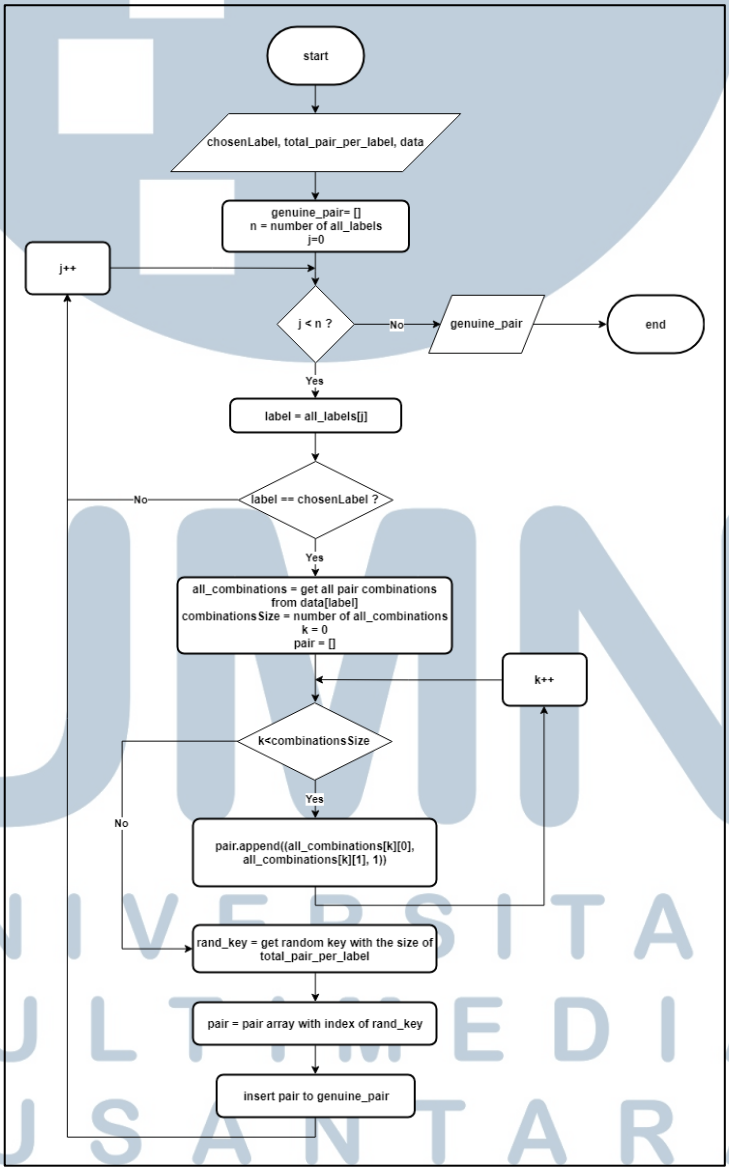


Gambar 3.12. *Flowchart* Fungsi *Create Pair*

A.3.2.1. Flowchart Get Genuine Pair

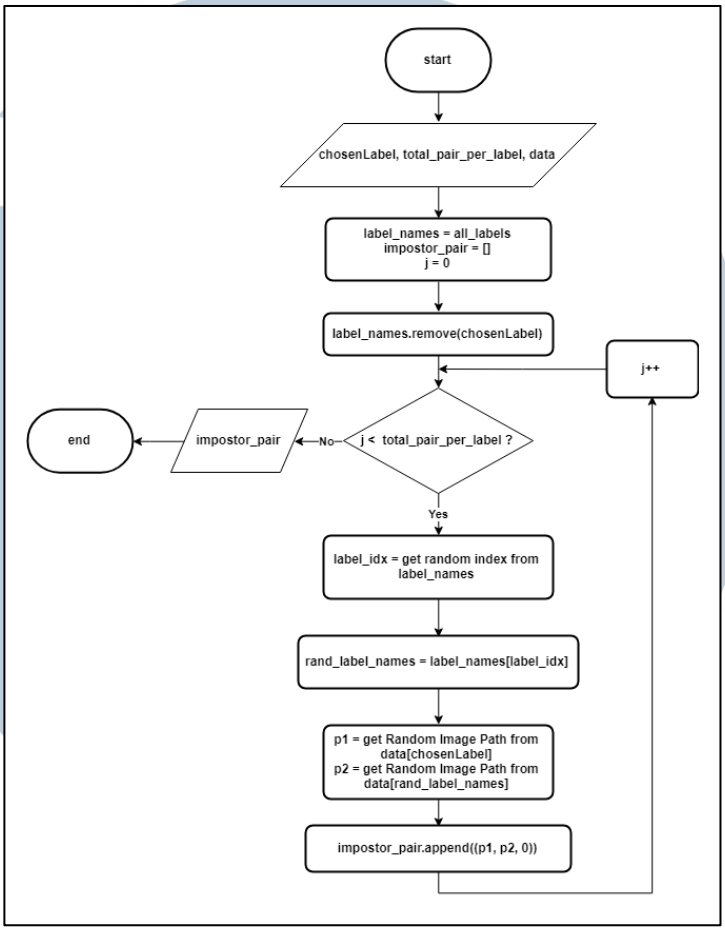
Fungsi *get genuine pair* merupakan fungsi yang dijalankan pada fungsi *create image pair* untuk mendapatkan citra berpasangan dengan kelas yang sama. Fungsi ini akan menerima parameter *choosenLabel* sebagai label yang dipilih, *total_per_pair_label* sebagai jumlah pasangan *genuine input* yang diinginkan, dan

data sebagai data yang akan diproses. Subproses akan dimulai dengan mencari seluruh kombinasi pasangan dari data dengan label yang dipilih dan disimpan di *array all_combinations*. Pada setiap data kombinasi, data dipasangkan dengan angka label 1 yang menandakan kedua citra memiliki kelas yang sama. Untuk setiap data kombinasi ke-*k*, bentuk data yang akan dihasilkan yaitu (all_combinations[k][0], all_combinations[k][1], 1). Fungsi akan mengembalikan data yang dipilih secara random sebanyak *total_per_pair_label*. *Flowchart* fungsi *get genuine pair* dapat dilihat pada Gambar 3.7.



Gambar 3.13. *Flowchart* Fungsi *Get Genuine Pair*

A.3.2.2. Flowchart Get Impostor Pair



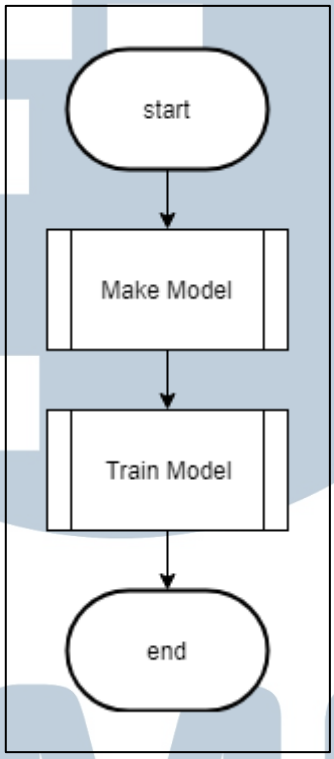
Gambar 3.14. *Flowchart* Fungsi *Get Impostor Pair*

Fungsi *get impostor pair* digunakan untuk menghasilkan pasangan citra yang memiliki kelas yang berbeda. *Flowchart* untuk fungsi *get impostor pair* dapat digambarkan oleh Gambar 3.8. Fungsi ini juga menerima parameter *chosenLabel* sebagai label yg dipilih, *total_per_pair_label* sebagai jumlah pasangan *impostor pair* yang diinginkan, dan *data* sebagai data yang akan diproses. Pertama-tama fungsi akan menghapus *chosenLabel* dari *array label_names* yang berisi label secara keseluruhan. Iterasi akan dilakukan sebanyak jumlah dari *total_per_pair_label*. Untuk setiap iterasi, bentuk data yang akan dihasilkan yaitu ((p1, p2), 0) dengan *p1* merupakan *image path* dari data yang memiliki kelas *choosenLabel*, *p2* merupakan *image path* dari data dengan kelas yang telah di-

random berdasarkan array *label_names*, dan angka 0 menandakan bahwa kedua *input image* memiliki kelas yang berbeda.

A.4. Flowchart Modelling

Flowchart pada proses *modelling* dapat ditunjukkan pada Gambar 3.9. Proses *modelling* akan dibagi menjadi dua subproses yaitu *make model* dan *train model*.

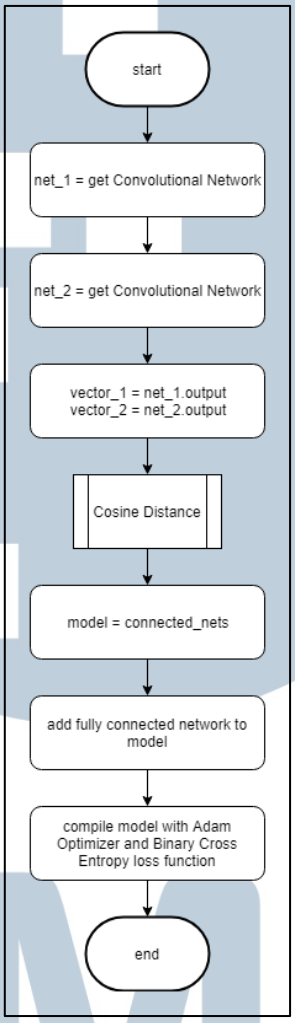


Gambar 3.15. *Flowchart* Proses *Modelling*

A.4.1. Flowchart Make Model

Subproses *make model* digunakan untuk menjabarkan proses pembuatan *model* sesuai dengan arsitektur pada Gambar 3.1. Pengembangan *flowchart* pada subproses ini dapat dilihat pada Gambar 3.10. Variabel *net_1* dan *net_2* akan menampung 2 buah model *convolutional network* berdasarkan arsitektur yang dirancang. Hasil *output* dari *net_1* dan *net_2* akan digabungkan pada subproses *cosine distance*. Proses akan dilanjutkan dengan penambahan *fully connected layer*

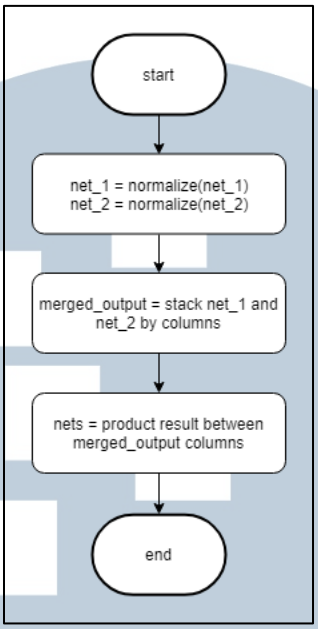
pada hasil *output* dari subproses *cosine distance*. Setelah model selesai dibuat, *model* akan di-*compile* dengan *Adam optimizer* dan *binary cross-entropy loss function* dalam penggunaanya saat proses *training*



Gambar 3.16. *Flowchart* Subproses *Make Model*

A.4.1.1. Flowchart Cosine Distance

Flowchart dari subproses *cosine distance* dapat digambarkan pada Gambar 3.11. Subproses ini bertujuan untuk menggabungkan dua buah *convolutional network* yaitu *net_1* dan *net_2*. Pertama-tama hasil *output* dari *net_1* dan *net_2* akan dinormalisasi. Kedua hasil normalisasi tersebut akan digabungkan dan dihitung nilai perkalian matriksnya antar kolom.

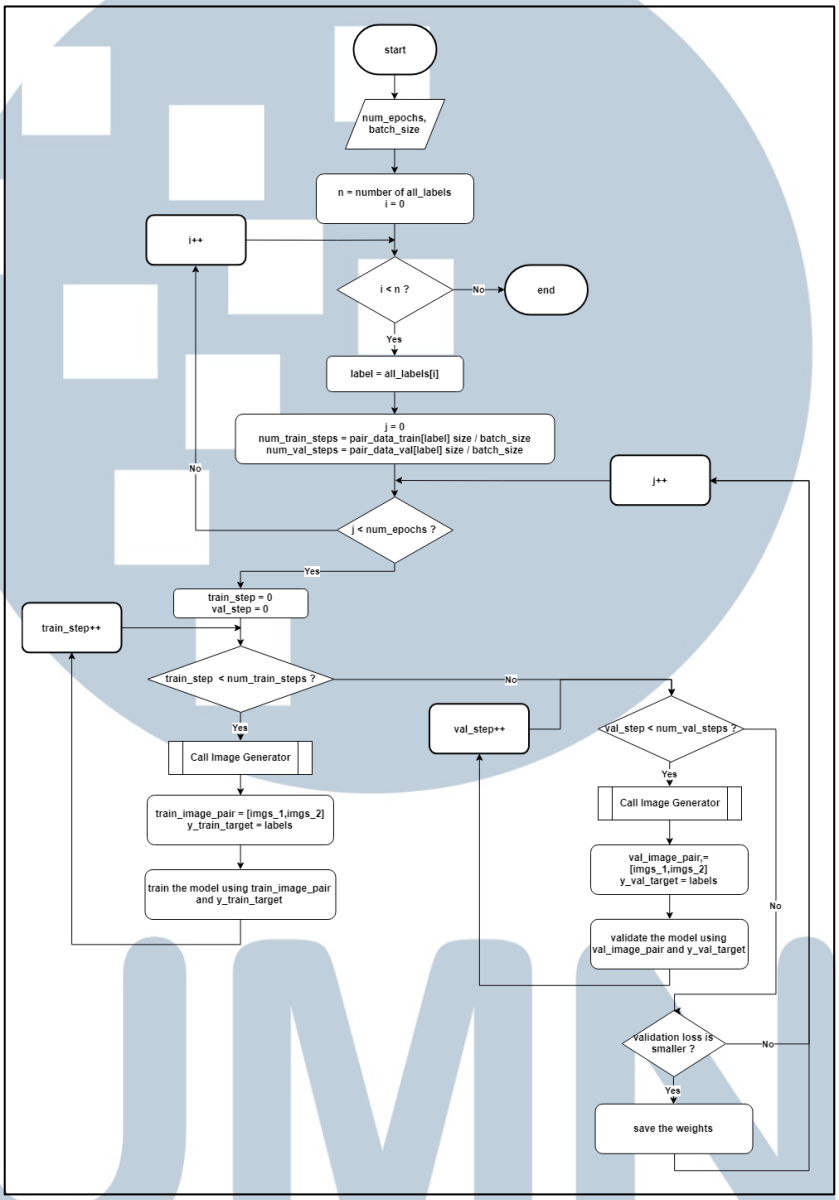


Gambar 3.17. *Flowchart* Subproses *Cosine Distance*

A.4.2. Flowchart Train Model

Pada subproses *train model*, model yang sudah di-*compile* pada subproses *make model* akan masuk ke fase *training*. Subproses ini akan menerima parameter berupa *num_epoch* sebagai jumlah *epoch* dan *batch_size* sebagai ukuran data pada tiap *batch*. Masing-masing label akan memiliki modelnya sendiri-sendiri. Pada model setiap label, *num_train_steps* dan *num_val_steps* yang merupakan jumlah *step* pada tiap *epoch* akan dihitung terlebih dahulu dengan membagi ukuran data dengan ukuran *batch*. Untuk setiap *epoch*, *training* dan *validation* akan dilakukan sejumlah *num_train_steps* dan *num_val_steps*. Pada setiap *step*, data yang dipakai untuk *training* dan *validation* akan didapatkan dengan memanggil fungsi *call image generator*. Data yang didapatkan berbeda-beda sesuai dengan urutan *batch*. Kemudian, dilanjutkan dengan proses *training* dan *validation* untuk data yang sudah didapatkan melalui generator. Untuk pemilihan model akan menggunakan *validation loss* sebagai metrik. Setiap kali model mendapatkan *validation loss* yang

lebih kecil dari sebelumnya, maka bobot dari model tersebut akan disimpan. *Flowchart* untuk subproses ini dapat ditunjukkan pada Gambar 3.12.

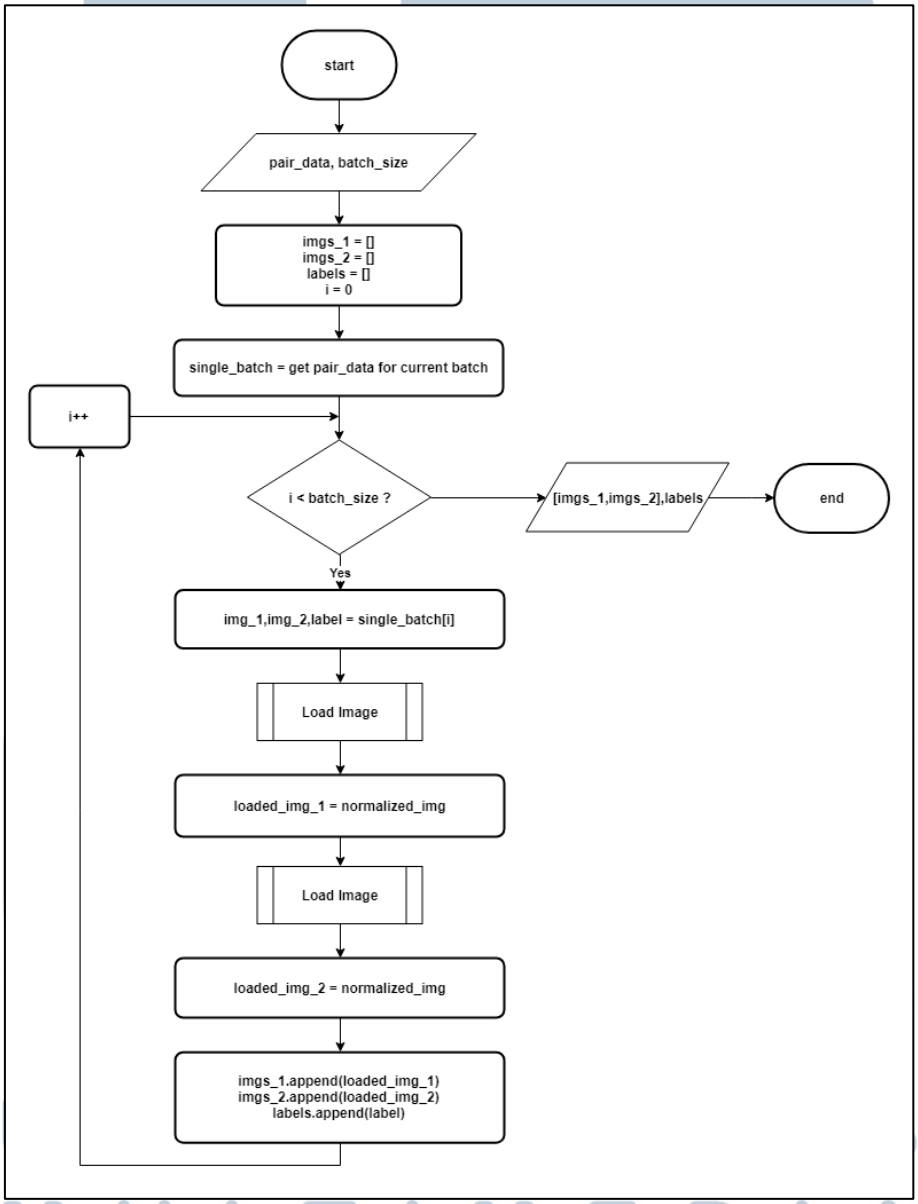


Gambar 3.18. *Flowchart* Subproses *Train* Model

A.4.2.1. Flowchart Call Image Generator

Fungsi *call image generator* digunakan untuk mengembalikan data yang sudah siap untuk proses *training*. *Flowchart* untuk *call image generator* dapat dilihat pada Gambar 3.13. Fungsi ini akan menerima parameter berupa *pair_data* sebagai data *path* dalam bentuk *pair* dan *batch_size* sebagai ukuran dari *batch*.

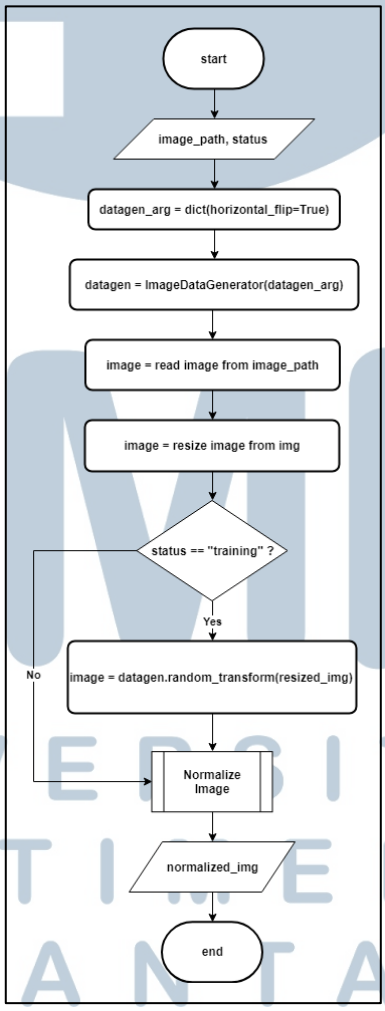
Pertama-tama, variabel *single_batch* akan menampung *pair_data* sebanyak *batch_size*. Untuk setiap *image path* di *single_batch*, akan dipanggil fungsi *load image* untuk membaca dan menambahkan augmentasi pada *image* tersebut. Fungsi ini akan mengembalikan data dalam bentuk *pair* dan *similarity* label data tersebut sejumlah ukuran *batch_size*.



Gambar 3.19. *Flowchart* Fungsi *Call Image Generator*

A.4.2.1.1. Flowchart Load Image

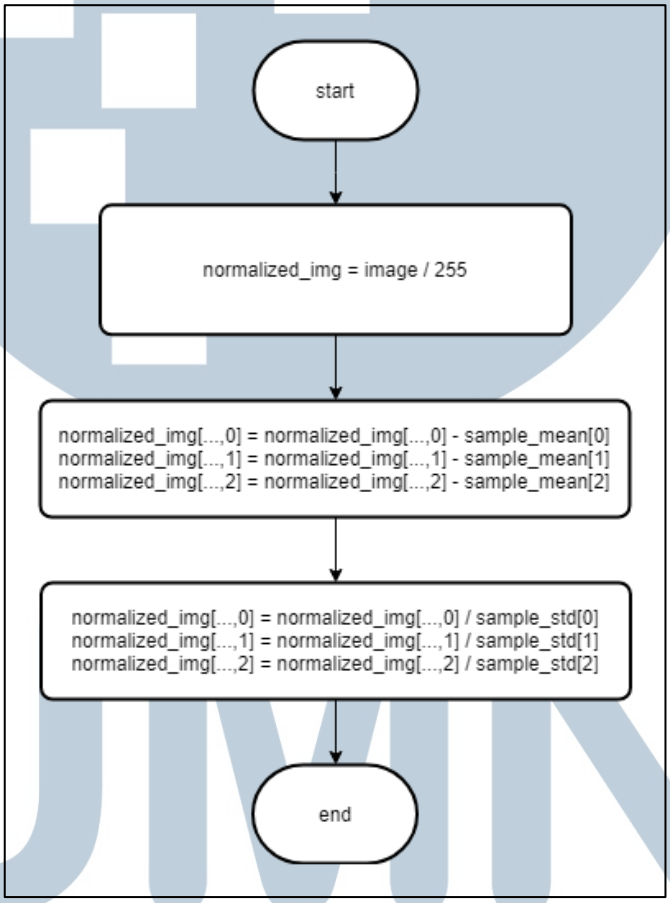
Fungsi *load image* akan digunakan untuk membaca *image* dari *image path* dan menambahkan augmentasi pada *image* tersebut. *Flowchart* fungsi *load image* dapat dijabarkan pada Gambar 3.14. Fungsi ini menerima parameter *image_path* sebagai *path* dari *image* yang akan dibaca dan *status* sebagai penanda dari proses *training* atau *testing*. Pertama-tama, variabel *datagen* akan menampung *generator* yang melakukan augmentasi berupa *horizontal flip* pada *image*. Kemudian, *image* dari *image_path* akan dibaca dan di-*resize* menjadi ukuran 224 x 224 *pixel*. Apabila status adalah *training*, maka akan dilakukan augmentasi dengan memanggil *datagen*. Kemudian, *image* akan dinormalisasi melalui subproses *normalize image*.



Gambar 3.20. *Flowchart* Fungsi *Load Image*

A.4.2.1.1.1. Flowchart Normalize Image

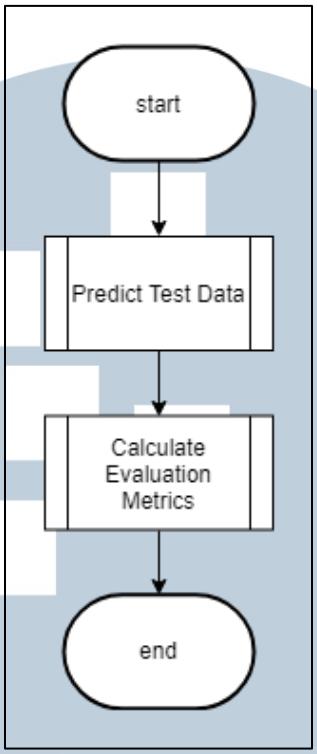
Subproses *normalize image* digunakan untuk melakukan normalisasi pada *image* menggunakan *mean* dan *standard deviation* yang sudah disiapkan pada subproses *prepare normalization component* yaitu *sample_mean* dan *sample_std*. Pada subproses ini, tiap *channel* dari *image* akan dikurangi dengan *sample_mean* dan dibagi dengan *sample_std*. *Flowchart* subproses ini dapat dilihat pada Gambar 3.15.



Gambar 3.21. *Flowchart* Subproses *Normalize Image*

A.5. Flowchart Evaluating Result

Flowchart untuk proses *evaluating result* dapat ditunjukkan pada Gambar 3.16. Proses *evaluating result* dibagi menjadi dua subproses yaitu *predict test data* dan *calculate evaluation metrics*.

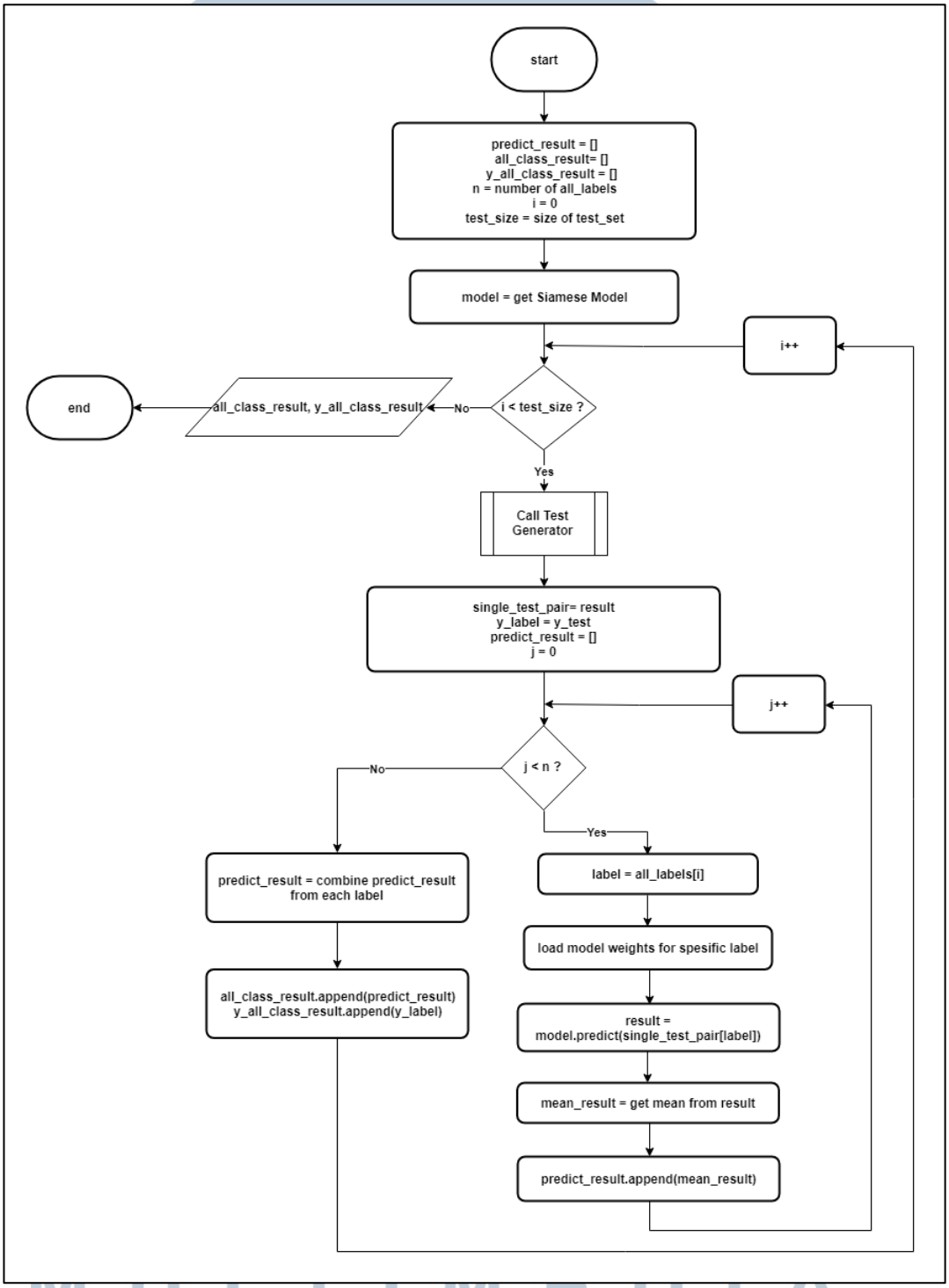


Gambar 3.22. *Flowchart* Proses *Evaluating Result*

A.5.1. Flowchart Predict Test Data

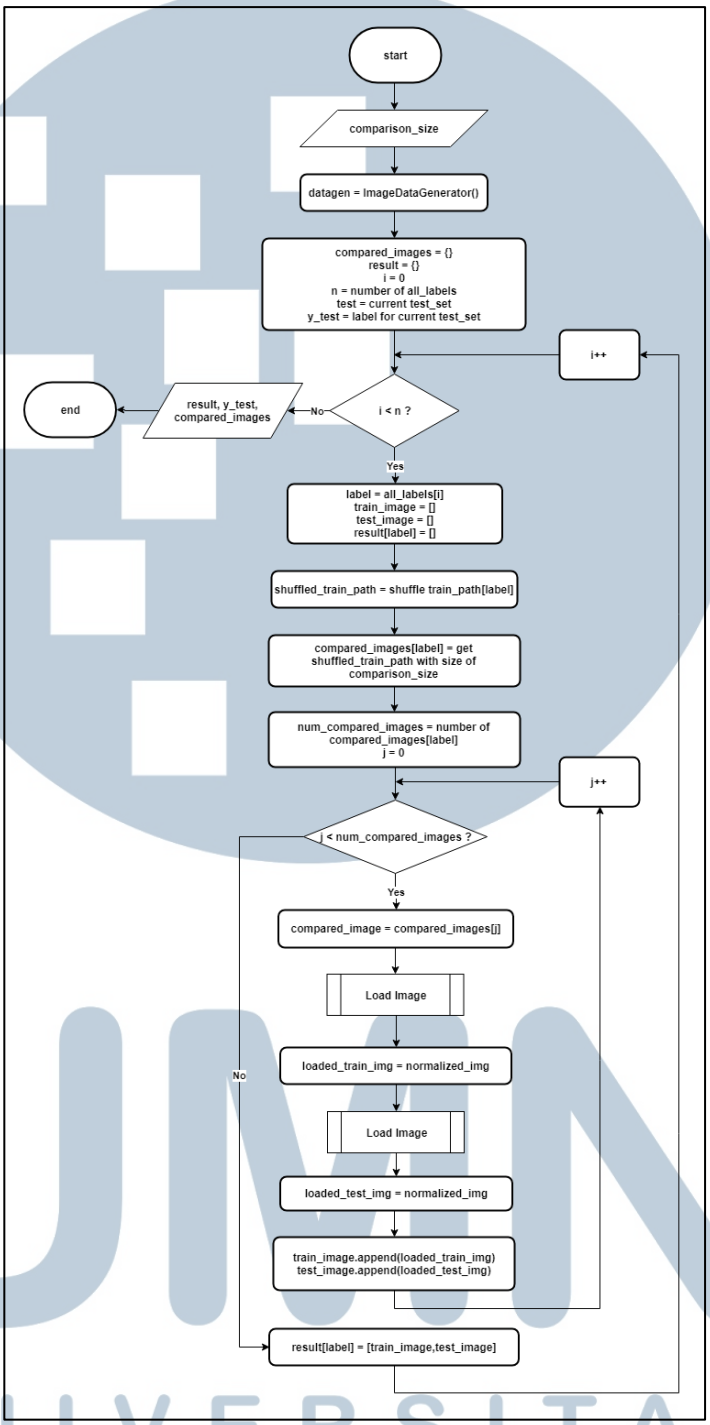
Pada Gambar 3.17, ditunjukkan *flowchart* dari subproses *predict test* data. Subproses ini akan memprediksi data *testing* yang disimpan di variabel *test_set* berdasarkan model yang telah di-*training*. Fungsi *call test generator* digunakan untuk menghasilkan data *testing* yang sudah disesuaikan dengan bentuk *input* dari *siamese convolutional network* dan *similarity label* dari data tersebut. Data yang didapat dari fungsi ini berupa data yang dipasangkan dengan data *training* dari masing-masing label sebagai data pembanding. Untuk setiap label, model akan diinisialisasi terlebih dahulu dengan bobot yang disimpan pada subproses *train model*. Baru kemudian, data yang sudah berpasangan akan diprediksi oleh model tersebut. Hasil prediksi dari keseluruhan data pembanding akan dihitung nilai

mean-nya. Nilai *mean* dari masing-masing label akan digabungkan pada variabel *all_class_result.*



Gambar 3.23. *Flowchart* Subproses *Predict Test Data*

A.5.1.1. Flowchart Call Test Generator

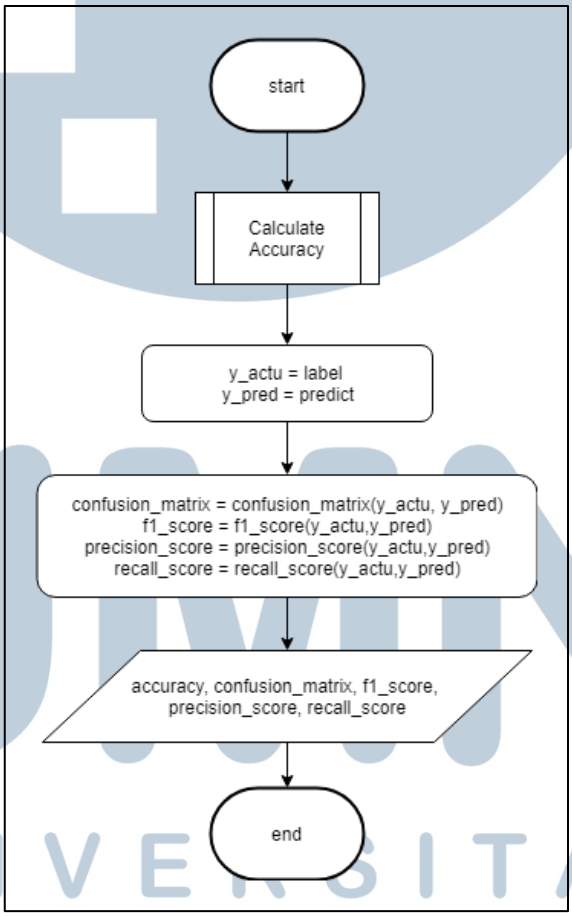


Gambar 3.24. *Flowchart* Fungsi *Call Test Generator*

Fungsi *call test generator* digunakan untuk mengembalikan data yang siap untuk proses *testing*. *Flowchart* fungsi *call test generator* akan ditunjukkan oleh Gambar 3.18. Fungsi ini akan menerima parameter *input* berupa *comparison_size*

sebagai jumlah dari data *training* pembanding yang dipasangkan dengan data *testing*. Untuk setiap label, pertama-tama data *training path* akan di-*shuffle* terlebih dahulu. Kemudian data tersebut akan digunakan sejumlah *comparison_size*. Setiap data pembanding dan data *testing* akan dibaca dan dinormalisasi melalui fungsi *load image*. Variabel *result* akan menampung data berpasangan antara data pembanding dan data *testing* untuk masing-masing label. Fungsi ini akan mengembalikan variabel *result* dan label dari data *testing* tersebut.

A.5.2. Flowchart Calculate Evaluation Metrics

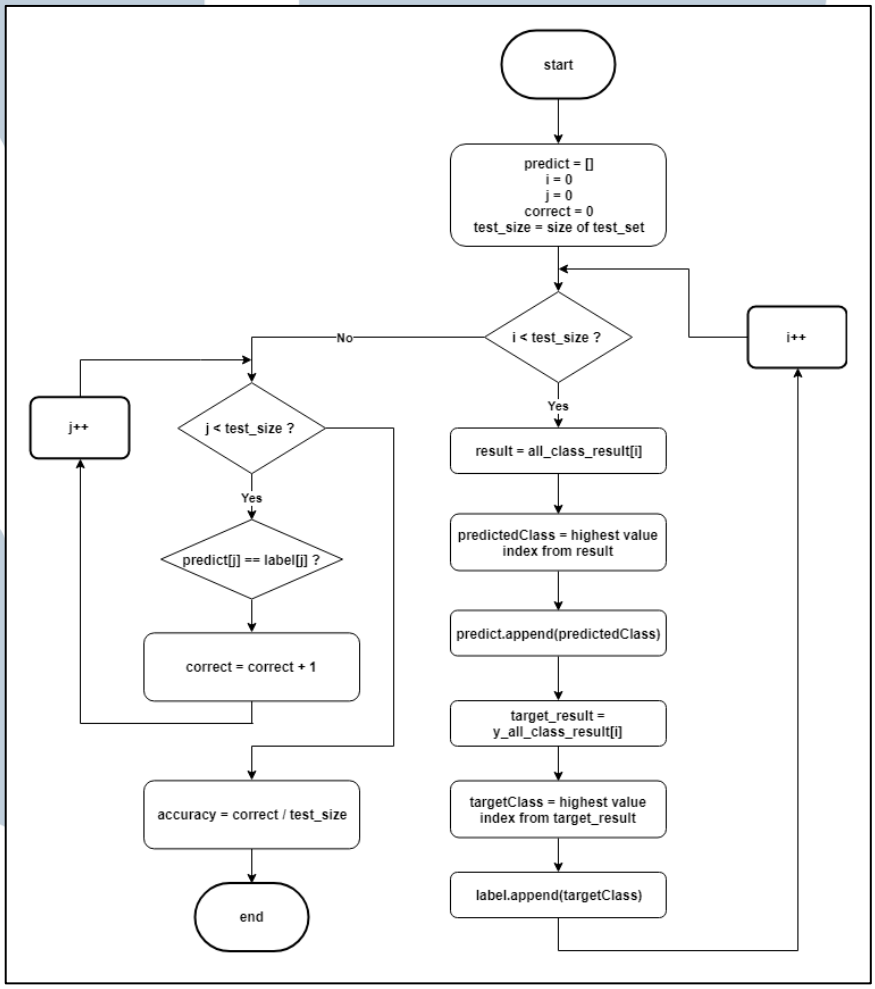


Gambar 3.25. *Flowchart* Subproses *Calculation Evaluation Metrics*

Subproses *calculate evaluation metrics* akan menghitung metrik-metrik yang digunakan untuk mengevaluasi model yang telah di-*training*. Gambar 3.19.

menjabarkan *flowchart* dari subproses *calculate evaluation metrics*. Pertama-tama, subproses *calculate accuracy* akan dipanggil untuk menghitung tingkat akurasi dari hasil prediksi model. Label dari setiap data *testing* disimpan dalam variabel *y_actu* dan hasil dari prediksi model akan disimpan dalam variabel *y_pred*. Kedua variabel itu akan digunakan untuk menampilkan *confusion matrix, f1 score, precision score, dan recall score.*

A.5.2.1. Flowchart Calculate Accuracy



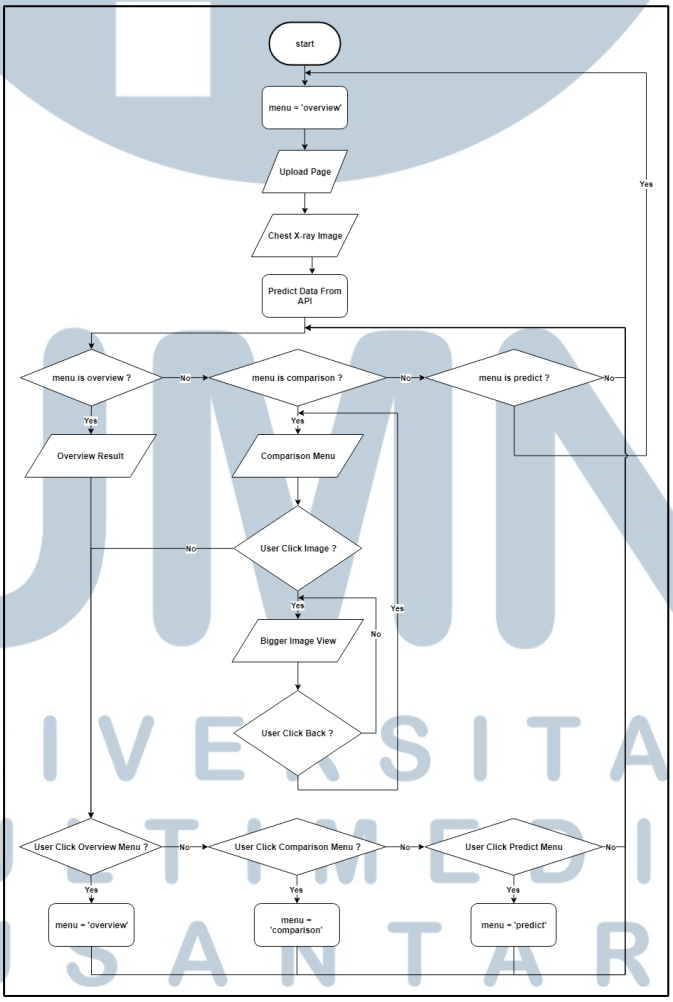
Gambar 3.26. *Flowchart* Subproses *Calculate Accuracy*

Flowchart subproses *calculate accuracy* ditunjukkan pada Gambar 3.20. Subproses ini akan menghitung tingkat akurasi dari hasil prediksi model. Untuk

setiap data *testing*, indeks *value* tertinggi dari hasil prediksi keseluruhan model akan disimpan di variabel *predict*, sedangkan indeks *value* tertinggi dari label aktual dari data akan disimpan di variabel *label*. Berdasarkan variabel *predict* dan *label*, maka akan dihitung jumlah prediksi yang benar dengan mengecek tiap *value* dari kedua variabel tersebut. Tingkat akurasi dihitung dengan membagi jumlah prediksi yang benar dengan jumlah data *testing*.

B. Flowchart Website

Website yang dikembangkan digunakan untuk memprediksi data berdasarkan *input* dari pengguna. *Flowchart* keseluruhan alur dari *website* dapat dilihat pada Gambar 3.21.



Gambar 3.27. *Flowchart* Keseluruhan Alur *Website*

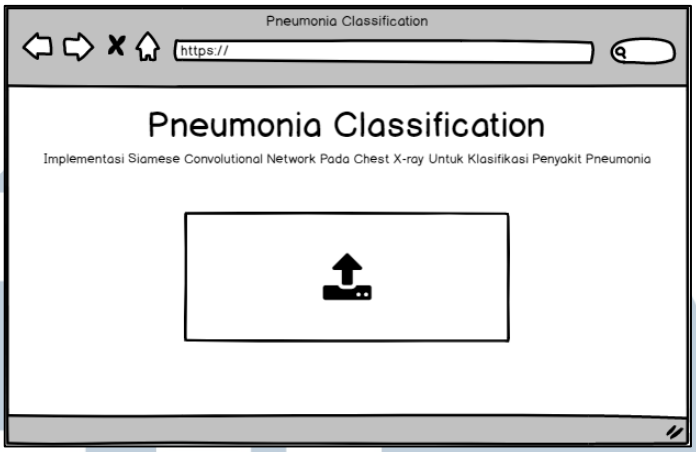
Website ini mempunyai beberapa halaman seperti halaman *upload*, halaman *overview result*, halaman *comparison menu*, dan halaman *bigger image view*. Pertama-tama *website* akan menerima *input* berupa *chest x-ray image* dari *user*. *Website* akan melakukan *request* ke API dengan proses yang hampir sama dengan proses *predict test data* yang ditunjukkan melalui *flowchart* pada Gambar 3.17. Halaman yang ditampilkan pertama kali yaitu *overview result page*. Perbedaan antara kedua proses ini terdapat pada jumlah input *test data* yang hanya berjumlah 1 data. Kemudian navigasi antar halaman mengikuti *action* yang diberikan oleh *user*. Pada halaman *comparison menu*, *user* dapat memilih salah satu dari *image* pembanding untuk ditampilkan dalam halaman *bigger image view.*

3.2.3. Rancangan Antarmuka Website

Website yang dibangun merupakan *single page website* yang menggunakan transisi untuk seolah-olah membagi *website* menjadi beberapa halaman seperti halaman *upload*, halaman *overview result*, halaman *comparison menu*, dan halaman *bigger image view*. Rancangan antarmuka untuk masing-masing halaman dapat dilihat sebagai berikut.

A. Halaman Upload

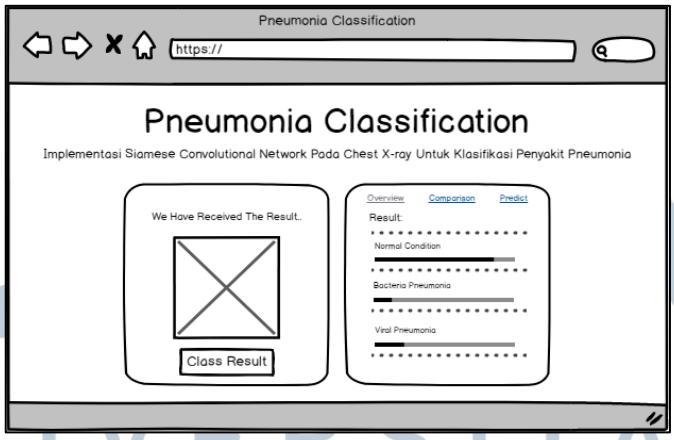
Rancangan antarmuka untuk halaman *upload* dapat dilihat pada Gambar 3.22. Pada halaman ini, terdapat kotak di tengah halaman sebagai penerima *input* berupa *chest x-ray image*.



Gambar 3.28. Rancangan Antarmuka Halaman *Upload*

B. Halaman Overview Result

Setelah menerima *input* dari *user*, *website* akan ditransisi menjadi halaman *overview result*. Halaman ini dibagi menjadi dua bagian dengan bagian pertama menampilkan *image* input dari *user* beserta hasil klasifikasi yang didapatkan dan bagian kedua menampilkan hasil persentase untuk klasifikasi tiap kelas. Terdapat *navigation bar* pada bagian atas untuk transisi antar halaman. Gambar 3.23. menunjukkan rancangan antarmuka dari halaman *overview result*.

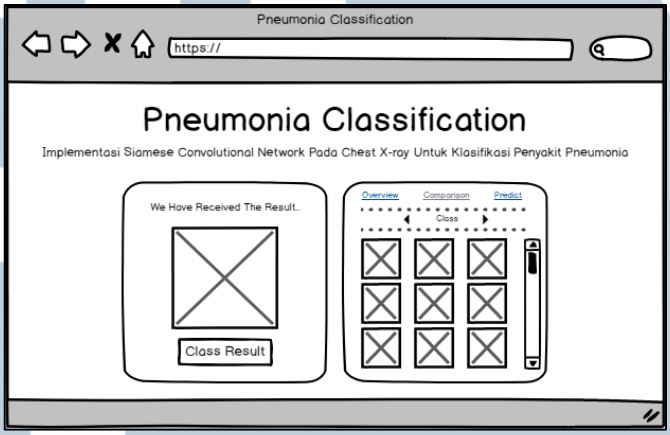


Gambar 3.29. Rancangan Antarmuka Untuk Halaman *Overview Result*

C. Halaman Comparison Menu

User dapat berpindah ke halaman *comparison menu* dengan memilih menu *comparison* pada *navigation bar*. Halaman ini berisikan *image-image* yang

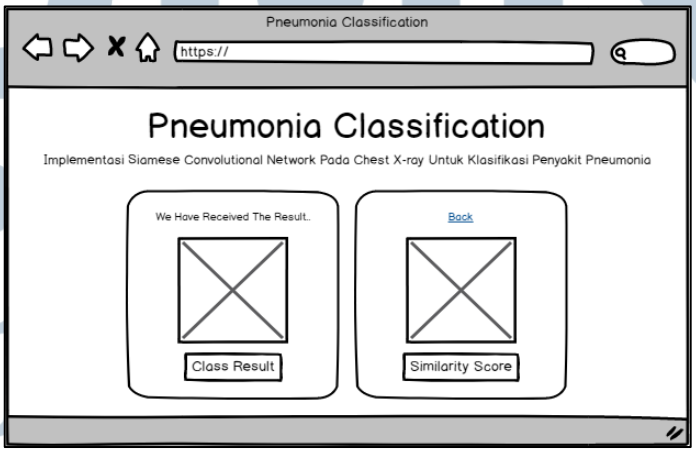
digunakan sebagai pembanding dalam menentukan klasifikasi kelas dari *input*. Terdapat menu untuk memilih kelas dari *image* pembanding pada bagian bawah dari *navigation bar*. Rancangan antarmuka dari halaman ini dapat ditunjukkan pada Gambar 3.24.



Gambar 3.30. Rancangan Antarmuka Untuk Halaman *Comparison Menu*

D. Halaman Bigger Image View

Apabila *user* memilih salah satu dari *image* pada halaman *comparison menu*, *website* akan melakukan transisi ke halaman *bigger image view*. Halaman ini menampilkan *image* yang dipilih dalam ukuran yang lebih besar untuk mempermudah dalam membandingkan *image* dengan *input* dari *user*. Nilai *similarity score* antara kedua *image* ini juga ditampilkan di bagian bawah *image*. Gambar 3.25. menunjukkan rancangan antarmuka dari halaman ini.



Gambar 3.31. Rancangan Antarmuka Untuk Halaman *Bigger Image View*