BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Metode Penelitian

Pada tahapan ini, penelitian dimulai dengan studi pustaka, pengumpulan data, perancangan, implementasi, dan pengujian sistem. Pada tahapan studi pustaka, peneliti mencari referensi mengenai permasalahan yang akan diteliti. Kemudian peneliti mencari referensi penelitian sebelumnya yang membahas permasalahan yang akan diteliti. Kemudian pada tahap pengumpulan data penulis mengumpulkan dataset berupa gambar daun tomat. Selanjutnya pada tahap perancangan penulis membuat rancangan sistem untuk menyelesaikan masalah yang diteliti. Kemudian pada tahapan implementasi penulis menerapkan ilmu yang didapat pada studi pustaka dan menerapkan rancangan sistem. Pada tahapan pengujian, penulis melakukan pengujian pada sistem yang telah diterapkan.

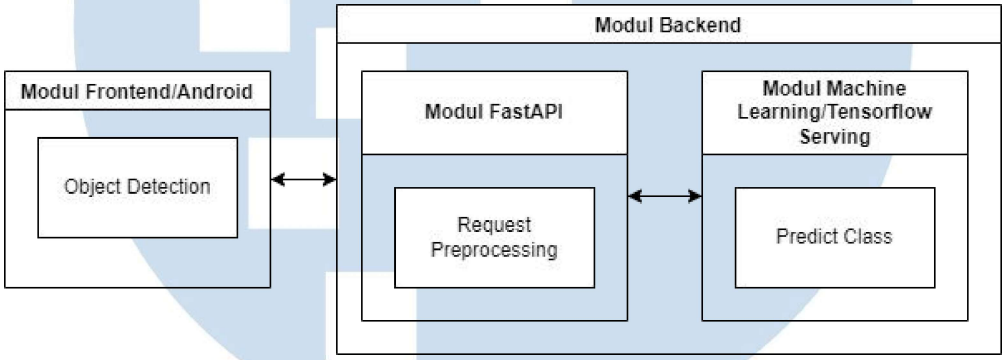
Pada tahapan studi literatur, peneliti mempelajari riset sebelumnya yang membahas tentang deteksi penyakit pada tanaman tomat. Penulis juga melakukan studi pustaka melalui website, buku, serta video yang ada di internet. Selama tahapan studi pustaka penulis mempelajari bahasa pemograman baru seperti Kotlin dan Python yang dapat diterapkan pada sistem yang akan diterapkan.

3.2 Perancangan Modul

Dalam penelitian ini sistem akan dibagi menjadi 2 modul yaitu Frontend dan Backend.

Modul Frontend berupa aplikasi Android yang dibuat menggunakan IDE Android Studio. Modul Frontend ini berfungsi untuk memberikan tampilan kepada pengguna. Dalam modul Frontend terdapat model Object Detection milik *library* ML Kit yang digunakan untuk melakukan ekstraksi objek pada sebuah gambar. Saat objek terdeteksi, gambar dari objek tersebut akan dikirimkan ke Backend untuk di prediksi. Selanjutnya modul Backend yang menggunakan *framework* FastAPI dengan bahasa pemograman *Python*. Modul FastAPI berfungsi untuk menangani gambar yang dikirim oleh Frontend. Kemudian modul

FastAPI akan melakukan prediksi dengan memproses gambar yang diterima dan mengirimkan hasil tersebut ke modul Machine Learning dalam bentuk Tensorflow Serving. Module Machine Learning menggunakan Tensorflow Serving yang mengubah bentuk model menjadi endpoint untuk melakukan prediksi sebuah gambar. Saat menerima sebuah gambar, Tensorflow Serving akan melakukan prediksi dan mengirimkan kembali hasil prediksi ke Backend dan hasil tersebut akan diteruskan ke Frontend. Berikut rancangan modul pada Gambar 3.1

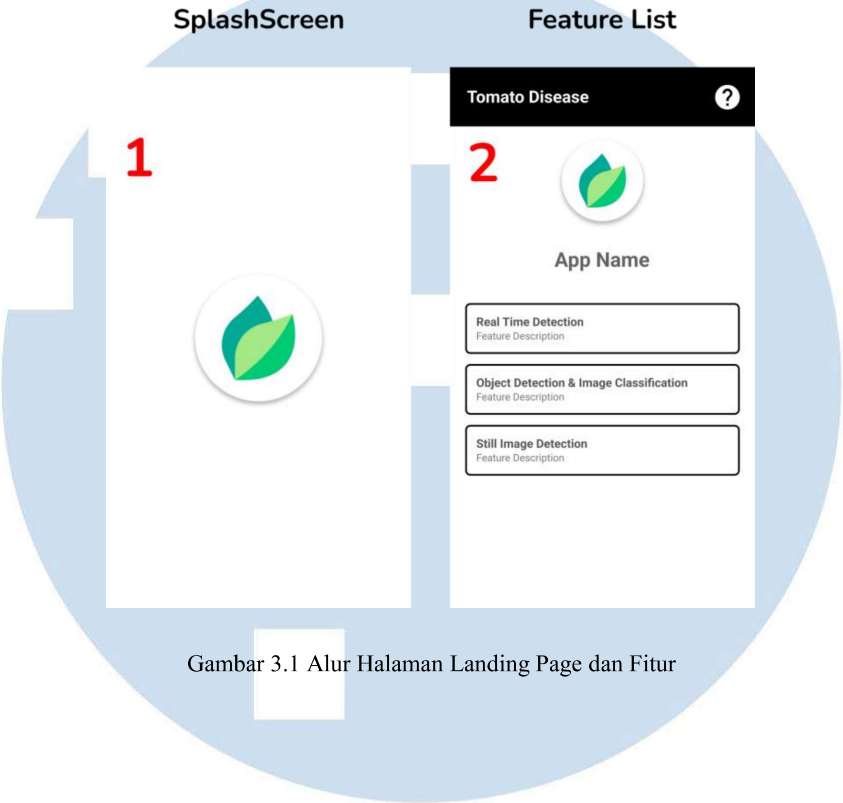


Gambar 3.1 Rancangan Modul

3.3 Perancangan Tampilan Aplikasi

Pada modul Frontend peneliti membuat *mock* tampilan menggunakan aplikasi Figma yang berbasis web. Dalam perancangan modul Frontend peneliti merancang 3 fitur yang dapat digunakan untuk mendeteksi penyakit tanaman yang terdiri dari Real Time Detection, Multiple Object Detection, dan Still Image. Ketiga fitur tersebut tidak dapat langsung melakukan klasifikasi, melainkan hanya melakukan deteksi objek dan mengirimkan gambar setiap objek untuk diklasifikasi di modul Backend. Pertama, fitur Real Time Detection digunakan untuk mendeteksi penyakit tanaman secara *realtime* menggunakan kamera ponsel dengan bantuan Object Detection. Hal ini bertujuan untuk mengetahui daun manakah yang terjangkit penyakit. Kemudian fitur kedua merupakan Multiple Object Detection yang bertujuan untuk mendeteksi beberapa daun tomat sekaligus. Terahkir fitur Still Image yang digunakan untuk mendeteksi foto dari daun

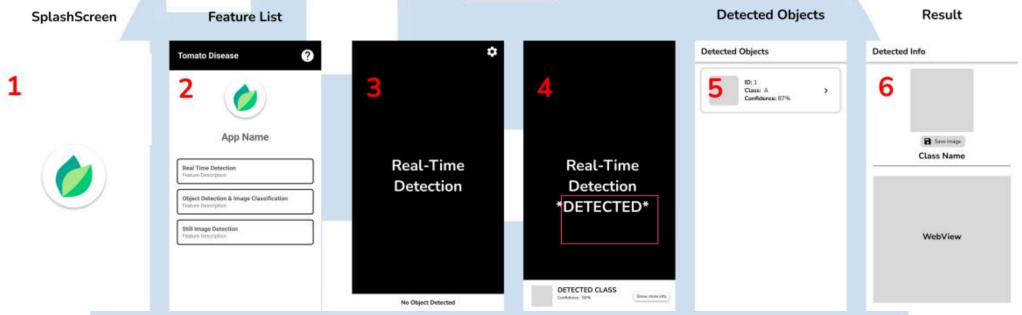
tanaman tomat. Fitur Still Image digunakan untuk mengatasi permasalahan apabila fitur Object Detection tidak dapat mendeteksi sebuah daun.



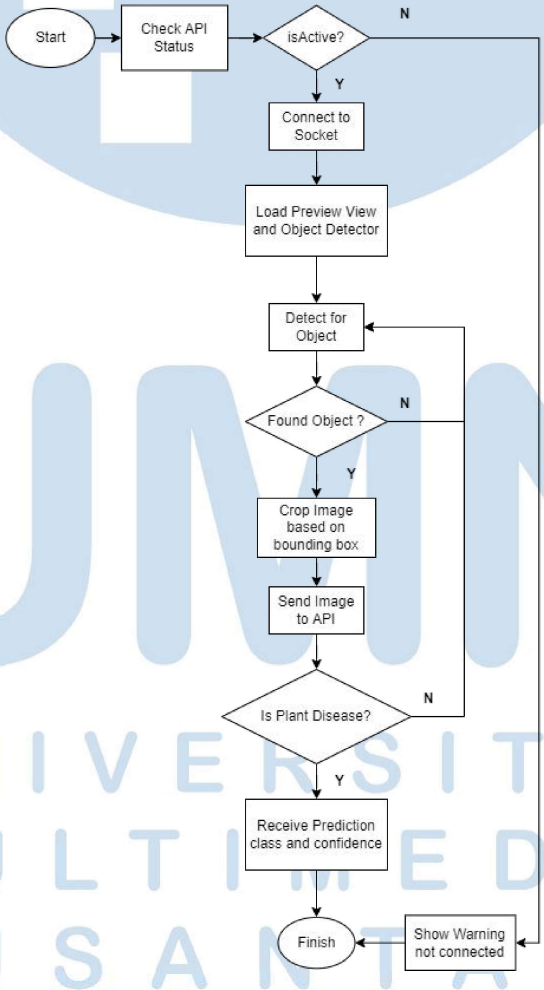
Gambar 3.1 Alur Halaman Landing Page dan Fitur

Alur aplikasi dimulai dari *landing page* atau kemudian dilanjutkan dengan halaman fitur seperti pada Gambar 3.2. Dalam halaman fitur, aplikasi akan menampilkan ketiga fitur yang digunakan untuk mendeteksi penyakit pada tanaman tomat. Saat fitur Real-Time dipilih, pertama kali laman akan melakukan pengecekan apabila aplikasi terhubung pada internet. Apabila terhubung, maka aplikasi akan melanjutkan, sedangkan saat tidak terhubung aplikasi akan memberikan dialog untuk mencoba kembali atau kembali ke laman fitur. Setelah terkoneksi, halaman akan menampilkan tampilan kamera belakang ponsel atau *preview view*. Tampilan tersebut telah dilengkapi dengan fitur *object detection* milik *library* ML Kit. Apabila model *object detection* mendeteksi sebuah objek, maka gambar objek tersebut akan di potong berdasarkan *bounding box* objek tersebut dan hasil potongan gambar tersebut dikirim ke API melalui protokol Websocket untuk mengetahui jenis penyakit tomat. Apabila terdeteksi, maka API akan mengirimkan respons dan tampilan akan menggambarkan *bounding box*

seperti pada Gambar 3.3 nomor 4. Kemudian informasi mengenai objek yang terdeteksi akan dimunculkan dibawah halaman. Dalam tampilan informasi tersebut terdapat sebuah tombol untuk membuka objek yang terdeteksi dan hasil deteksi apabila memilih salah satu objek yang terdeteksi. Terdapat Gambar 3.4 yang merupakan alur aplikasi dalam bentuk *flowchart*.

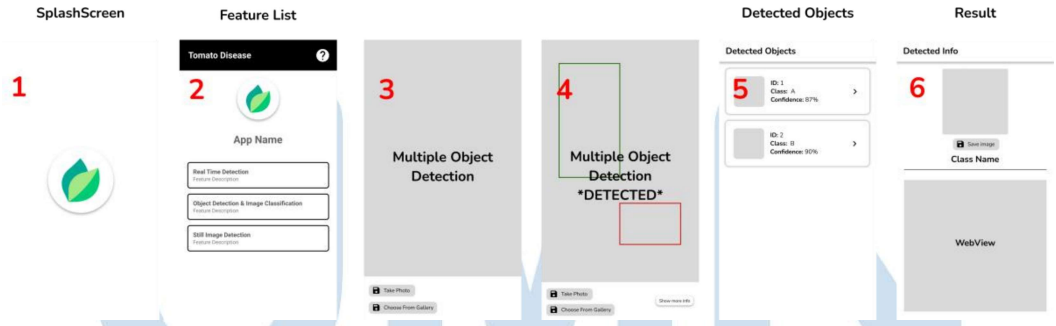


Gambar 3.2 Alur Tampilan Fitur Real Time Detection

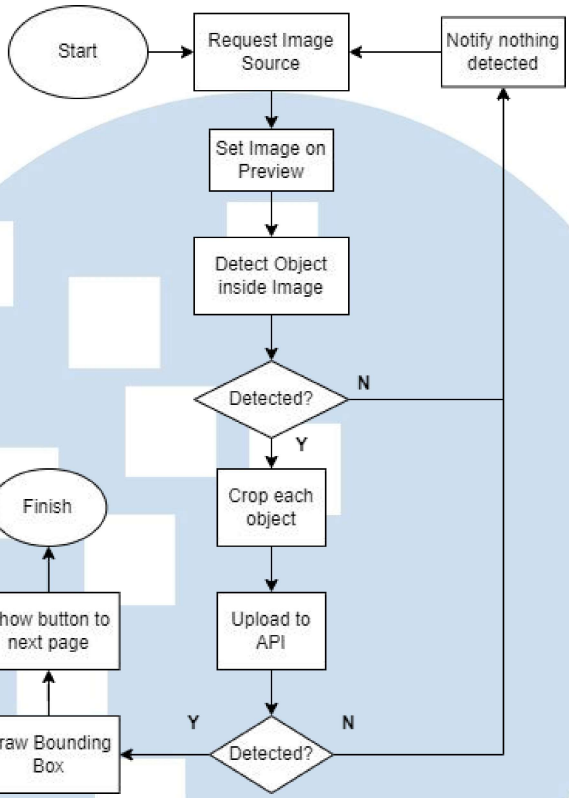


Gambar 3.3 Alur Kerja Fitur Real Time Detection

Selanjutnya saat fitur Multiple Object Detection dipilih, halaman akan menampilkan 2 tombol untuk memilih sumber gambar dari kamera ponsel atau media penyimpanan ponsel seperti pada Gambar 3.5. Saat menekan tombol untuk mengambil gambar dari kamera, aplikasi akan membuka tampilan kamera ponsel dan meminta untuk mengambil gambar. Apabila tombol mengambil dari media penyimpanan, aplikasi akan membuka media penyimpanan dan meminta untuk memilih gambar. Setelah mengambil gambar, gambar tersebut akan ditampilkan pada laman Multiple Object Detection dan *object detector* akan mendeteksi objek yang ada pada gambar. Apabila *object detector* tidak menemukan objek, aplikasi akan memberikan peringatan bahwa tidak menemukan objek. Saat *object detector* menemukan objek, aplikasi akan melakukan *cropping* untuk seluruh objek yang terdeteksi dan mengirimkan hasil *crop* ke API. Apabila API dapat mendeteksi gambar tersebut, aplikasi menggambar *bounding box* pada objek yang terdeteksi dan menampilkan tombol untuk kehalaman selanjutnya yang dapat menampilkan objek apa saja yang terdeteksi. Terdapat alur fitur Multiple Object Detection dalam bentuk *flowchart* pada Gambar 3.6.

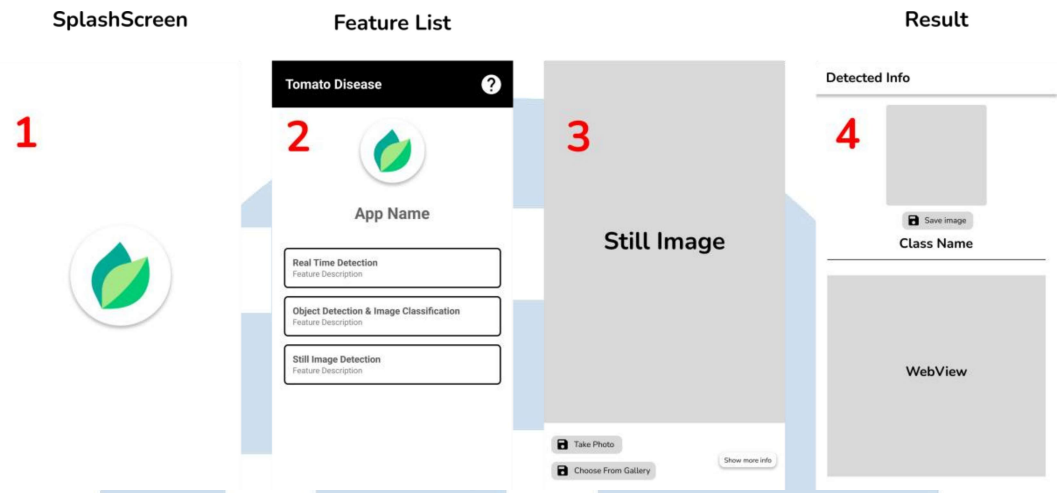


Gambar 3.4 Alur Tampilan Fitur Multiple Object Detection

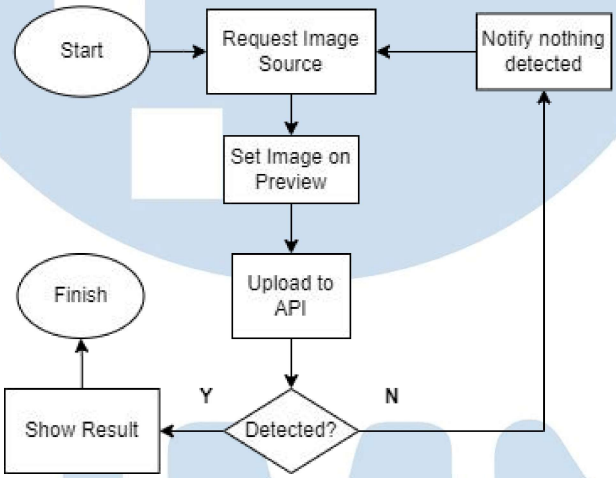


Gambar 3.5 Alur Kerja Fitur Multiple Object Detection

Pada fitur Still Image, terdapat 2 tombol seperti Multiple Object Detection. Dalam fitur Still Image dan Multiple Object terdapat pembeda yaitu fitur Still Image hanya mengirim satu gambar utuh, sedangkan Multiple Object dilakukan processing terlebih dahulu. Fitur ini akan langsung memberikan hasil ahkir prediksi tanpa menggambar *bounding box* pada gambar. Terdapat alur tampilan fitur Still Image pada Gambar 3.7 dan alur dalam bentuk *flowchart* pada Gambar 3.8.



Gambar 3.6 Alur tampilan fitur Still Image

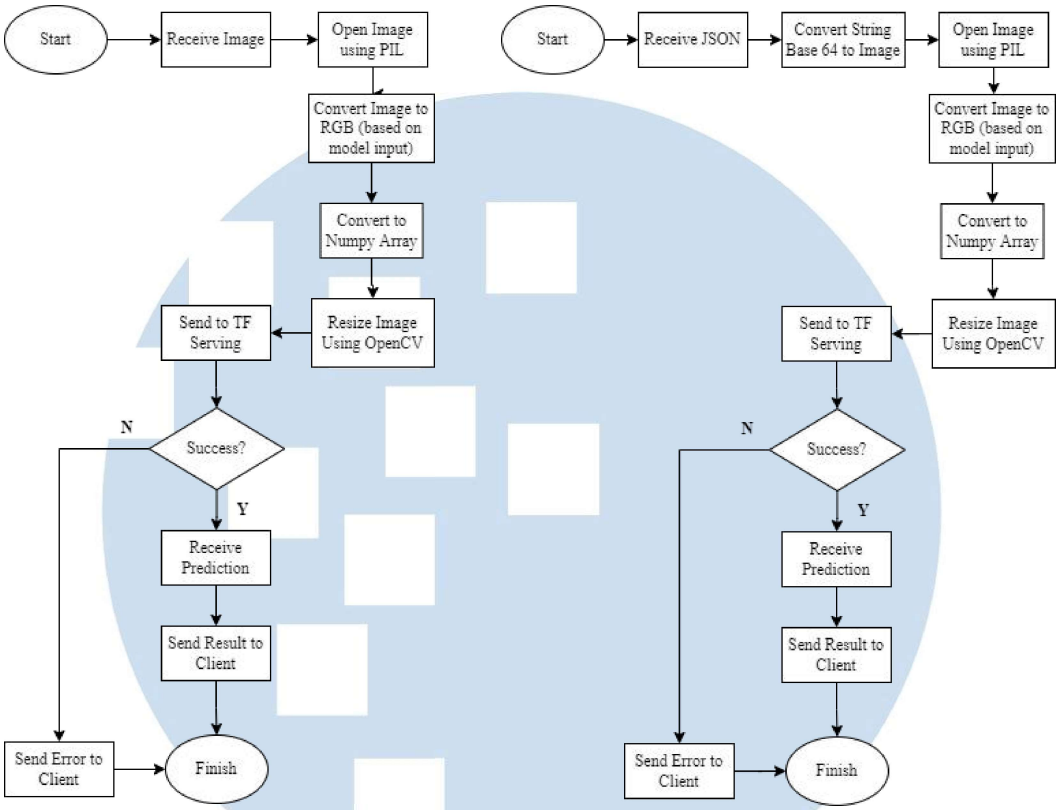


Gambar 3.7 Alur kerja fitur Still Image

Dalam aplikasi Android setiap gambar yang diproses masih dalam bentuk format Bitmap. Dimana format Bitmap merupakan format yang memiliki ukuran jauh lebih besar dibandingkan format umum seperti JPEG, JPG, dan PNG. Untuk mengatasi ukuran yang cukup besar, setiap gambar yang akan dikirim ke API akan di *rescale* menjadi 50% lebih kecil dari ukuran sebelumnya. Hal ini bertujuan untuk memperkecil ukuran gambar. Kemudian gambar tersebut akan diubah menjadi format JPEG untuk mencapai ukuran gambar terkecil. Setelah itu, gambar tersebut akan di *encode* menjadi String Base64 sebelum dikirim ke API Endpoint.

3.4 Rancangan Backend

Pada modul Backend ini, terdapat *endpoint* yang menerima *request* berubah gambar dari modul Frontend aplikasi. Gambar tersebut merupakan hasil dari proses Object Detection dalam modul Frontend yang telah di *crop*. Modul backend dibagi menjadi dua yaitu modul FastAPI dan Tensorflow Serving. Dalam modul FastAPI terdapat endpoint yang dapat diakses untuk mengirimkan gambar dalam bentuk Base64 dan File. Endpoint terbagi menjadi yaitu */still_image*, */still_image_base64*, */still_images_base64*, dan */ws/live_detection*. Seluruh endpoint menggunakan metode POST kecuali untuk */ws/live_detection* yang menggunakan Websocket. Saat menerima gambar untuk diprediksi, modul Backend akan memproses gambar terlebih dahulu. Pada setiap *endpoint* data yang terima akan dibaca menggunakan Python Image Library (PIL). Khusus endpoint yang menerima *request* String Base64 akan di-*decode* terlebih dahulu menjadi gambar. Kemudian data tersebut akan di konversi ke 'RGB' dan Numpy Array yang kemudian akan di-*resize* menjadi ukuran 224 x 224 menyesuaikan ukuran input model *machine learning*. Kemudian setelah proses pembacaan gambar selesai, modul FastAPI akan melakukan *request* ke modul Tensorflow Serving untuk mendapatkan hasil prediksi. Dalam modul FastAPI, server akan dijalankan menggunakan Asynchronous Server Gateway Interface (ASGI) dengan *library* Hypercorn. Setelah dijalankan server akan dibuka ke internet menggunakan ngrok. Hal ini dikarenakan aplikasi android pada ponsel tidak dapat melakukan *request* secara langsung ke *localhost*. Terdapat alur cara kerja modul Backend pada gambar 3.9.



Gambar 3.8 Alur Kerja Pemrosesan Gambar pada Modul Backend

Tensorflow Serving berfungsi sebagai *endpoint* model yang telah di *deploy* untuk melakukan prediksi pada suatu gambar. Untuk melakukan proses *deployment* pada model, dapat digunakan aplikasi Docker untuk melakukan proses kontainerisasi pada model yang telah dibuat. Sebelum melakukan kontainerisasi, dapat digunakan *file .config* untuk mengatur lokasi *folder* model serta memberikan pengaturan *versioning* pada setiap model. Setelah melakukan kontainerisasi, model dapat diakses melalui endpoint dengan format *http://localhost:8051/v1/models/[nama_model]/versions/[nama_versi]:predict*. Data yang diterima oleh endpoint tersebut berupa Input Tensor yang berbentuk Array.

3.5 Rancangan Modul Machine Learning

Modul Machine Learning berfungsi untuk melakukan klasifikasi pada sebuah gambar yang diterima dari Backend FastAPI. Tahap perancangan modul Machine Learning dimulai dari proses pengumpulan data berupa gambar daun tanaman tomat. Dataset yang digunakan peneliti adalah PlantVillage [12] yang terdiri dari 54,309 gambar dengan 14 jenis tanaman. Dari dataset tersebut akan hanya diambil jenis tanaman tomat yang memiliki 9 kelas daun terjangkit penyakit dan 1 kelas daun sehat. Total gambar penyakit tomat pada dataset PlantVillage adalah 18,162 gambar. Pembagian dataset tersebut terdapat pada Tabel 3.1.

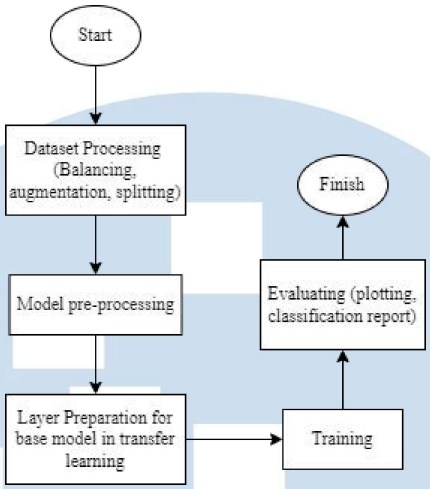
Tabel 3.1 Pembagian Kelas Tanaman Tomat pada dataset PlantVillage

Nama Kelas	**Jumlah Gambar**
Yellow Leaf Curl Virus	5357
Bacterial Spot	2127
Late Blight	1910
Septoria Leaf Spot	1771
Spider Mites	1676
Healthy	1592
Target Spot	1404
Early Blight	1000
Leaf Mold	952
Mosaic Virus	373

Kemudian peneliti juga mengumpulkan dataset berupa gambar acak diluar tanaman tomat. Hal ini bertujuan untuk mengatasi masalah CNN yang hanya dapat mendeteksi gambar pada kelas yang tersedia. Apabila model CNN

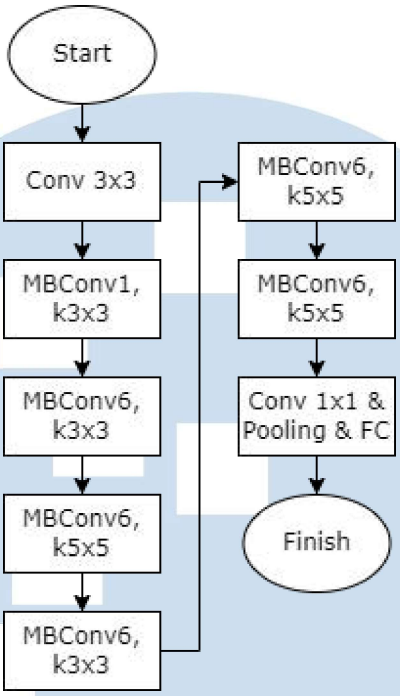
diberikan sebuah gambar acak, maka model akan berusaha melakukan prediksi menggunakan kelas yang ada dan tidak dapat mengembalikan hasil prediksi tidak ditemukan. Untuk dataset acak, peneliti menggunakan dataset CIFAR100 yang terdiri dari 100 kelas yang memiliki 600 gambar.

Kemudian tahapan perancangan model yang akan menggunakan model CNN dengan metode *transfer learning* menggunakan model EfficientNetB0. Pertama kali dataset akan dibaca dan akan dilakukan *balancing* dengan menentukan batas total gambar per kelas. Apabila kelas memiliki jumlah dibawah batas total gambar per kelas, maka akan dilakukan augmentasi pada kelas tersebut untuk mengisi jumlah gambar pada kelas agar dapat mencapai batas total. Setelah itu data akan diproses ke dalam bentuk tabel menggunakan *library pandas*. Setiap data tersebut akan di dibagi menjadi set *training*, *validation*, dan *testing*. Pembagian dataset tersebut adalah 80%, 10%, dan 10%. Setiap set akan dilakukan *pre-processing* terlebih dahulu. Dalam tahap *pre-processing*, ukuran gambar yang masuk akan diubah menjadi ukuran 224 x 224. Selanjutnya akan dilakukan augmentasi pada data yang bertujuan memberikan variasi gambar pada dunia nyata serta menghindari kasus model *overfitting*. *Overfitting* merupakan sebuah kondisi dimana sebuah model hanya melakukan fitting pada set *train* sehingga mengalami kesulitan dalam melakukan deteksi pada data yang belum pernah diketahui. Proses augmentasi yang diberikan adalah rotasi gambar, pengaturan cahaya, memutarbalikkan gambar, menggeser gambar, memperbesar tampilan, dan mengganti sudut pandang gambar. Setelah melakukan *pre-processing* pada setiap set, model akan dilatih menggunakan set *train* dan *validation.* Terdapat alur rancangan pembuatan model pada Gambar 3.10.

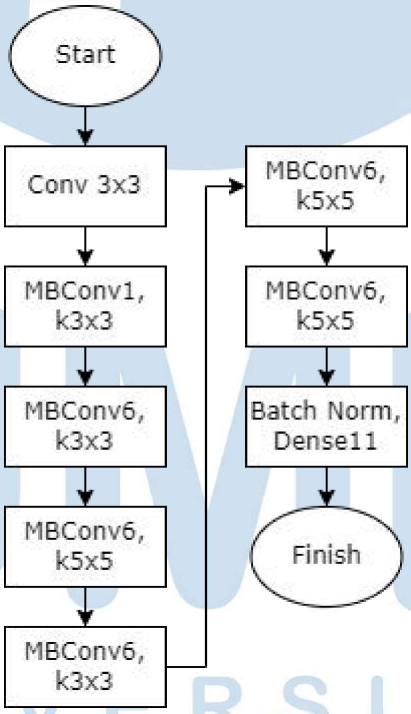


Gambar 3.9 Garis Besar Rancangan pada Model *Machine Learning*

Dalam tahap melakukan pelatihan model akan digunakan *base model* EfficientNetB0 yang memiliki arsitektur pada Gambar 3.11 dengan weight yang telah dilatih dengan dataset ImageNet yang memberikan pengetahuan awal pada model. Kemudian lapisan klasifikasi terahkir pada *base model* tidak akan diikutsertakan. Hal ini disebabkan model akan melakukan klasifikasi pada dataset baru dengan pengetahuan sebelumnya. Selanjutnya pada *base model* perlu dilakukan pembekuan pada *convolutional base*. Hal ini bertujuan untuk mencegah perubahan *weight* pada setiap lapisan *base model*. Kemudian pada *base model* akan ditambahkan *classification head* berupa GlobalAveragePooling2D. Untuk layer terahkir terdapat Dense layer untuk menerima input serta melakukan klasifikasi dengan jumlah 11 unit yang merupakan total kelas pada dataset seperti pada Gambar 3.12. Kemudian hasil pelatihan model akan di *plotting* untuk mengetahui performa model secara kesuluruhan. Hasil *training* pada model juga akan dievaluasi menggunakan set *test*.



Gambar 3.10 Arsitektur EfficientNetB0 Sebagai Base Model



Gambar 3.11 Rancangan Arsitektur dalam Penerapan *Transfer Learning*