BAB 3
METODOLOGI PENELITIAN

3.1 Metodologi Penelitian

Berikut adalah beberapa tahapan yang dilakukan dalam penelitian ini:

1. Studi Literatur

   Pada tahap ini dilakukan pencarian, pembacaan, dan pemahaman terhadap literatur yang berhubungan dengan penyakit katarak, CNN, dan ResNet50V2. Literatur tersebut dapat berupa jurnal, buku, maupun sumber referensi lainnya. Tujuan dari tahap ini adalah untuk memperdalam pemahaman teoritis.

2. Pencarian Dataset

   Pada tahap ini, dibutuhkan data untuk mendukung penelitian. Dataset yang diperlukan adalah citra mata untuk keperluan pelatihan model. Dataset ini diperoleh dari situs web Kaggle yang diunggah oleh Nanda Padia. Dataset yang digunakan berjumlah 612 gambar yang terbagi dalam dua kelas, yaitu mata normal dan katarak.

3. Perancangan program

   Pada tahap ini, dilakukan perancangan yang mencakup pembuatan *flowchart* untuk menggambarkan alur proses, perancangan model klasifikasi menggunakan *pretrained* ResNet50V2 serta perancangan *website* menggunakan model yang telah dibuat.

4. Implementasi program

   Pada tahap ini, dilakukan implementasi program berdasarkan rancangan sebelumnya. Bahasa pemrograman Python digunakan untuk pembuatan model klasifikasi karena mendukung pengolahan data, pembuatan model, dan pembuatan *website* secara efisien.

5. Pengujian dan Evaluasi

   Pada tahap ini, model diuji dan dievaluasi menggunakan data tes yang berbeda dari data pelatihan dan validasi. Evaluasi dilakukan dengan

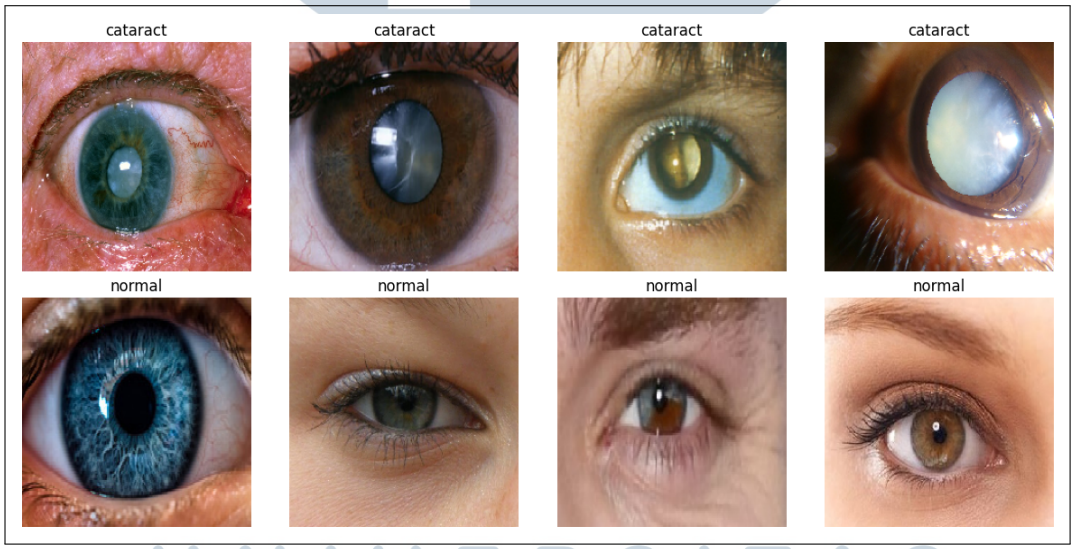
menghitung *precision*, *recall*, dan *f1-score* untuk mengukur kinerja model secara menyeluruh.

6. Penulisan Laporan

   Selama proses penelitian, penulisan laporan dilakukan secara paralel dengan pelaksanaan kegiatan penelitian. Dengan demikian, seluruh proses penelitian tercatat dengan baik dan tidak ada bagian penting yang terlewat untuk disertakan dalam laporan akhir.

3.2 Pencarian Dataset

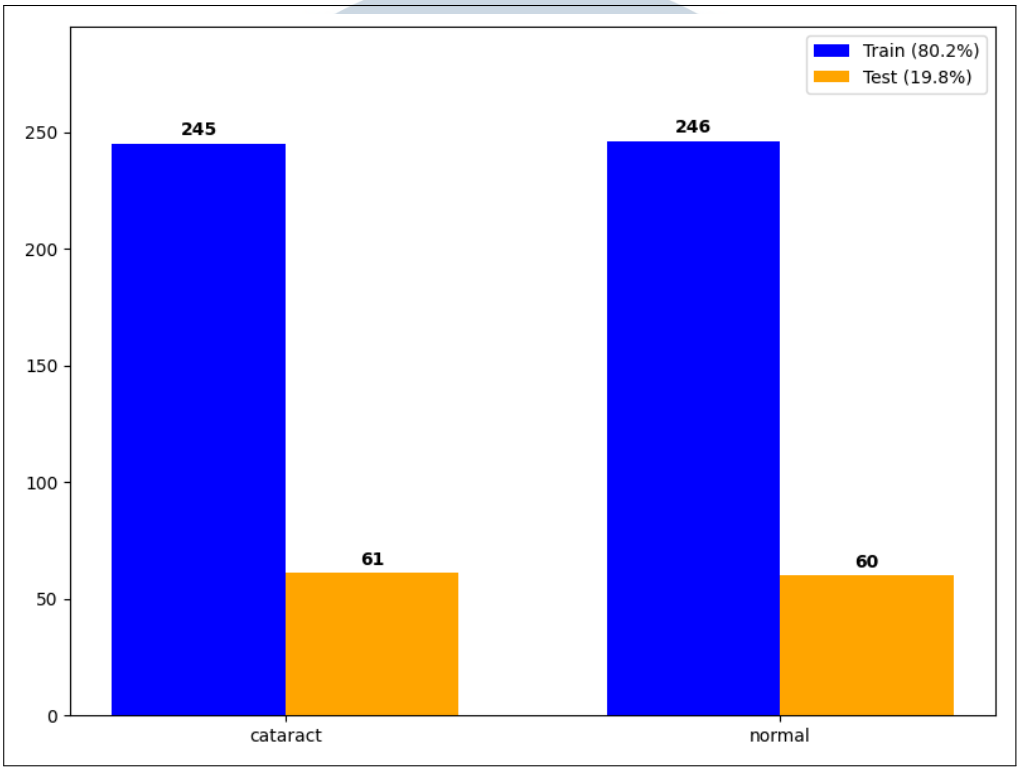
Tahap ini dilakukan pencarian dan pemilihan dataset yang sesuai untuk tugas klasifikasi citra mata dengan dua kelas, yaitu katarak dan normal. Dataset yang digunakan memiliki kualitas dan struktur yang mendukung proses pelatihan dan pengujian model. Berikut ini adalah contoh data dari setiap kelas yang digunakan dalam penelitian ini.



Gambar 3.1. Data citra katarak dan normal

Dataset ini diperoleh dari platform Kaggle yang dipublikasikan oleh Nanda Padia, berisikan gambar mata dengan dua kategori, yaitu katarak dan normal. Dataset ini terdiri atas dua bagian, yaitu data latih (*train*) yang mencakup 245 gambar mata katarak dan 246 gambar mata normal, serta data uji (*test*) dengan 61 gambar mata katarak dan 60 gambar mata normal. Sehingga secara *default* dari

struktur pembagian dan jumlah gambar tiap kelasnya, dataset ini sudah siap untuk pengembangan model. Berikut adalah histogram distribusinya.



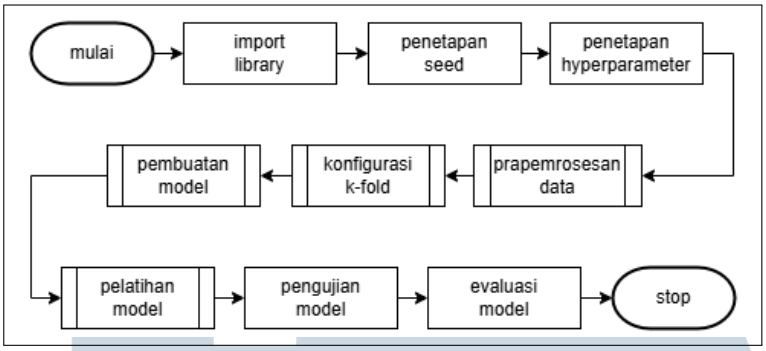
Gambar 3.2. Distribusi data gambar

3.3 Perancangan Program

Perancangan program ini dibagi menjadi dua bagian utama, yaitu perancangan model klasifikasi dan perancangan *website*. Dalam perancangan ini digunakan *flowchart* sebagai representasi alur kerja sistem.

3.3.1 Perancangan Model

Pada perancangan model klasifikasi akan dijelaskan alur proses dari awal hingga akhir dalam model yang dibangun, dimulai dari tahap *import library*, dilanjutkan dengan penetapan *seed*, penetapan *hyperparameter*, prapemrosesan data, pembuatan model, pelatihan model, hingga pengujian akhir pada data tes. *Flowchart* pada Gambar 3.3 bertujuan untuk memberikan gambaran menyeluruh mengenai langkah-langkah teknis yang dilakukan dalam model deteksi secara sistematis dan terstruktur.



Gambar 3.3. *Flowchart* perancangan model

A Import Library

Pada tahap awal pengembangan model, dilakukan proses impor terhadap sejumlah pustaka (*library*) yang diperlukan untuk menunjang seluruh alur kerja dalam penelitian ini. *Library* utama yang digunakan meliputi TensorFlow dan Keras untuk membangun dan melatih model klasifikasi, termasuk pemanggilan *pretrained* model ResNet50V2.

Selain itu, digunakan juga pustaka pendukung seperti NumPy dan Pandas untuk manipulasi data, Matplotlib dan Seaborn untuk visualisasi hasil, serta Scikit-learn untuk fungsi evaluasi seperti *confusion matrix*, *classification report*, dan *Stratified K-Fold Cross Validation*. *Library* tambahan seperti PIL dan glob digunakan untuk proses pemuatan serta manajemen file gambar. Penggunaan berbagai pustaka ini memungkinkan implementasi pengolahan data dan pelatihan model secara efisien.

B Penetapan Seed

Untuk memastikan hasil pelatihan model yang konsisten dan dapat direproduksi, penelitian ini menetapkan nilai *seed* secara eksplisit. *Seed* berfungsi mengendalikan elemen-elemen acak, seperti pembagian data dan inisialisasi bobot awal, sehingga model dengan konfigurasi yang sama akan menghasilkan performa yang sama setiap kali dijalankan. Hal ini sangat penting terutama dalam konteks *hyperparameter tuning*, di mana berbagai kombinasi seperti *batch size*, *learning rate*, jenis *optimizer*, dan *hyperparameter* lainnya yang memengaruhi performa model.

Tanpa penetapan *seed*, proses acak seperti inisialisasi bobot dapat menyebabkan perbedaan jalur pelatihan dan hasil akhir, bahkan ketika konfigurasi

hyperparameter-nya sama. Akibatnya, peningkatan akurasi yang terlihat bisa jadi berasal dari keberuntungan proses acak, bukan dari pengaruh nyata *hyperparameter* yang diuji. Dengan menggunakan *seed* yang tetap, penelitian ini memastikan bahwa setiap perbedaan performa model benar-benar disebabkan oleh variasi nilai *hyperparameter*, bukan oleh faktor acak yang tak terkendali.

C Penetapan Hyperparameter

Hyperparameter adalah parameter yang ditentukan sebelum proses pelatihan dimulai dan memiliki peran dalam mengontrol perilaku model selama proses pembelajaran. Pada tahap ini, dilakukan penetapan nilai awal untuk beberapa *hyperparameter* utama seperti jumlah *epoch*, ukuran *batch*, tingkat pembelajaran (*learning rate*), jenis *optimizer*, dan *early stopping*. Penetapan nilai-nilai tersebut tidak bersifat tetap, melainkan digunakan sebagai dasar awal untuk eksperimen. Selanjutnya, dilakukan pencarian kombinasi *hyperparameter* terbaik melalui serangkaian pengujian dan evaluasi model. Pendekatan ini bertujuan untuk memperoleh konfigurasi yang menghasilkan performa model paling optimal terhadap data yang digunakan.

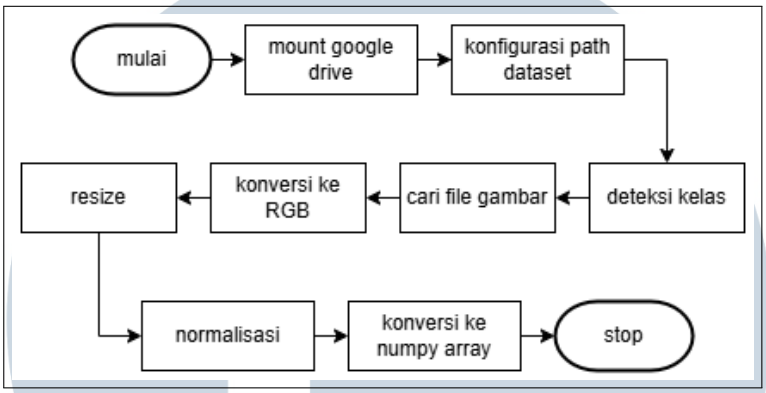
D Prapemrosesan Data

Tahap ini merupakan proses awal untuk mempersiapkan citra sebelum digunakan dalam pelatihan model. Berdasarkan alur pada Gambar 3.4, proses dimulai dengan *mount* Google Drive yang berfungsi untuk mengakses dataset dari direktori yang telah ditentukan. Selanjutnya dilakukan konfigurasi *path* menuju folder `train` dan `test`.

Proses pelabelan dilakukan secara otomatis berdasarkan nama subdirektori dari dataset yang merepresentasikan kelas. Setelah kelas terdeteksi, sistem akan mencari semua *file* gambar dengan ekstensi umum seperti JPG, JPEG, PNG, dan BMP. Setiap gambar yang ditemukan kemudian dikonversi ke format RGB untuk memastikan konsistensi kanal warna. Setelah itu, gambar diubah ukurannya (*resize*) ke dimensi yang telah ditentukan (224x224 piksel) sesuai dengan kebutuhan model.

Langkah selanjutnya adalah normalisasi, yaitu membagi nilai piksel dengan 255 agar berada dalam rentang nol hingga satu. Gambar yang telah dinormalisasi dikumpulkan ke dalam *list*, lalu seluruhnya dikonversi menjadi *array* NumPy untuk mempercepat pemrosesan dalam TensorFlow. Pada tahap ini, dataset juga dianalisis

untuk melihat distribusi jumlah data per kelas, serta untuk memastikan tidak ada ketidakseimbangan data yang signifikan sebelum memasuki proses pelatihan model.

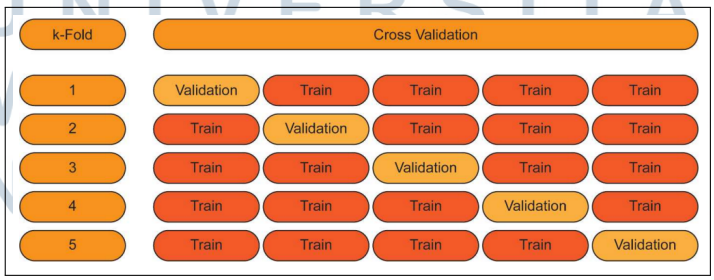


Gambar 3.4. *Flowchart* prapemrosesan data

E Konfigurasi Stratified K-Fold

Pada tahap ini, dilakukan konfigurasi validasi silang menggunakan teknik *Stratified K-Fold Cross Validation* untuk memastikan bahwa distribusi kelas pada setiap lipatan (*fold*) tetap seimbang. Teknik ini membagi data latih ke dalam k lipatan (*fold*) secara acak namun tetap mempertahankan proporsi kelas pada setiap *fold*, sehingga setiap *fold* merepresentasikan distribusi kelas secara keseluruhan.

Dalam penelitian ini digunakan nilai $k = 5$ (lima-fold), dengan pengacakan data (*shuffle*) yang dikontrol menggunakan parameter *random state* agar eksperimen dapat direproduksi. Setiap iterasi menggunakan satu *fold* sebagai data validasi dan sisanya sebagai data pelatihan, sehingga seluruh data dapat dimanfaatkan secara optimal baik untuk pelatihan maupun validasi. Hasil dari setiap *fold*, seperti akurasi dan *loss* pada data pelatihan serta validasi, dicatat untuk dianalisis dan dihitung rata-ratanya guna mendapatkan gambaran kinerja model yang lebih baik.



Gambar 3.5. Visualisasi 5 fold cross validation

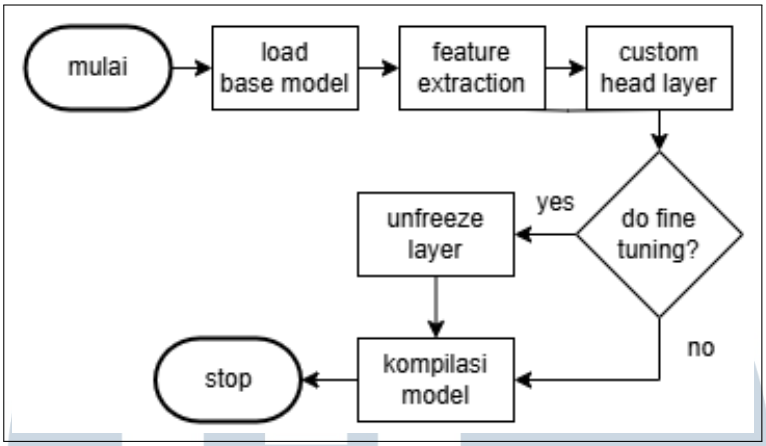
Sumber: [48]

5-fold cross validation sebagaimana divisualisasikan pada Gambar 3.5. Metode ini digunakan untuk mengevaluasi performa model secara menyeluruh dan mengurangi risiko *overfitting*. Jumlah keseluruhan data yang digunakan adalah 612 data, yang terdiri atas 491 data pelatihan (*training*) dan 121 data pengujian (*testing*). Data pelatihan tersebut kemudian dibagi menjadi lima bagian (*fold*) yang masing-masing berisi sekitar 98 data. Pada setiap iterasi, satu *fold* digunakan sebagai data validasi (98 data), sedangkan empat *fold* lainnya digunakan untuk melatih model (392 data). Proses ini diulang sebanyak lima kali sehingga setiap *fold* berkesempatan menjadi data validasi satu kali. Hasil evaluasi dari kelima iterasi ini kemudian dirata-ratakan untuk mendapatkan estimasi performa model yang lebih akurat.

F Pembuatan Model

Gambar 3.6 menjelaskan proses pembuatan model. Proses diawali dengan memuat *base* model pretrained yaitu ResNet50V2. Model dasar kemudian digunakan untuk ekstraksi fitur dengan membekukan semua lapisan (*feature extraction*), dan ditambahkan *custom head layer* sesuai dengan kebutuhan klasifikasi. Lapisan kustom yang ditambahkan terdiri dari *Global Average Pooling* untuk mengubah *output* spasial dari layer konvolusional terakhir (dalam hal ini dari ResNet50V2) menjadi vektor satu dimensi yang lebih kecil dan siap diproses oleh *fully connected layer*. Penambahan *Global Average Pooling* dipilih berdasarkan penelitian [49, 50].

Satu lapisan lainnya yaitu *Dense* dengan fungsi aktivasi Sigmoid untuk klasifikasi biner (katarak dan normal) yang dipilih berdasarkan penelitian [49, 50]. Setelah itu, model dikompilasi dan proses selesai. Selanjutnya, jika strategi *fine tuning* diaktifkan, beberapa lapisan terakhir dari *base model* akan di-*unfreeze* agar dapat disesuaikan lebih lanjut dengan dataset. Model kemudian dikompilasi dan proses selesai.



Gambar 3.6. *Flowchart* pembuatan model

G Pelatihan Model

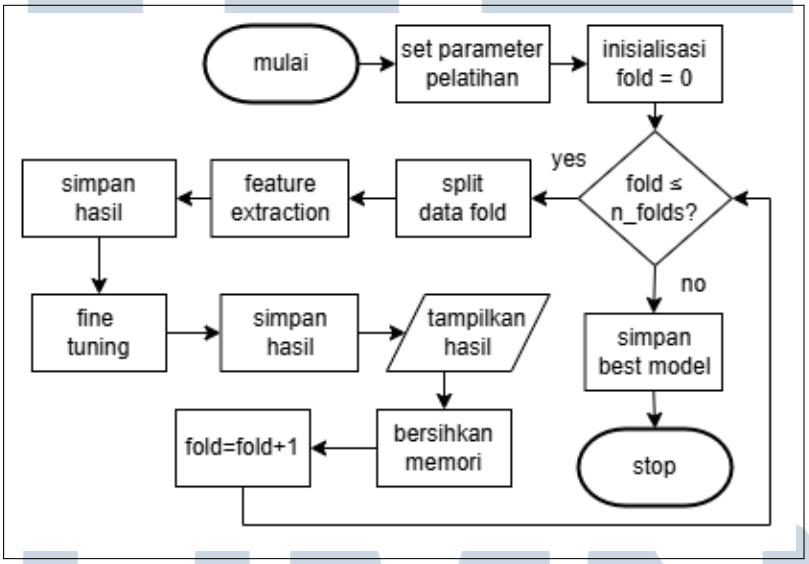
Berdasarkan penelitian [51, 52], pelatihan model dilakukan dalam dua fase, yaitu dengan menggunakan model *pretrained* sebagai ekstraktor fitur (*frozen*) dengan tambahan *custom layer* pada *head* model dan dilanjutkan *fine tuning* dengan membuka beberapa layer terakhir (*unfreeze*) pada model *pretrained* untuk penyesuaian. Pada fase *Feature Extraction* dilakukan pelatihan awal dengan bobot model dasar dibekukan, serta penambahan beberapa layer kustom.

Pelatihan akan dilakukan dengan konfigurasi berbeda secara manual untuk menemukan kombinasi hiperparameter terbaik seperti `batch size`, `learning rate`, serta struktur layer tambahan dalam *custom head* model. Pada fase kedua, yaitu *Fine tuning*, beberapa *layer* dari model dasar dibuka untuk dilatih ulang guna meningkatkan kemampuan generalisasi model. Nilai input berupa jumlah layer yang di-*unfreeze* juga dikonfigurasi dengan berbeda untuk menemukan konfigurasi optimal.

G.1 Tahapan Pelatihan

Flowchart pada Gambar 3.7 menunjukkan tahapan pelatihan model menggunakan pendekatan *Stratified K-Fold* dengan dua fase utama, yaitu *feature extraction* dan *fine tuning*. Proses dimulai dengan inisialisasi parameter pelatihan. Dalam hal ini konfigurasi *hyperparameter* dan *callbacks*. Nilai *hyperparameter* dan *callback* ini ditetapkan diawal untuk digunakan dalam pelatihan. Selanjutnya inisialisasi *fold* dengan nilai nol dan akan melakukan *looping* selama nilai *fold* masih lebih kecil dari jumlah total *fold* (n_folds).

Setelah itu, data dibagi menjadi subset pelatihan dan validasi berdasarkan indeks *fold* saat ini. Model baru dibuat dan dilatih pada fase *feature extraction*. Setelah pelatihan, model dievaluasi dan hasilnya disimpan. Selanjutnya proses *fine tuning* dilakukan dengan membuka beberapa layer akhir dari model pra-latih dan menyesuaikan bobotnya menggunakan *learning rate* yang lebih kecil. Setelah *fine tuning* selesai, model kembali dievaluasi dan hasilnya disimpan. Proses dilanjutkan ke *fold* berikutnya hingga seluruh *fold* selesai diproses. Setelah semua *fold* selesai, dilakukan analisis hasil pelatihan, dan model dengan akurasi validasi terbaik disimpan sebagai model akhir (klasifikasi-katarak.h5).



Gambar 3.7. *Flowchart* pelatihan model

G.2 Konfigurasi Pelatihan

Berdasarkan penelitian [53] yang membandingkan jumlah *epoch* 10, 15, 20, 30, dan 50 terhadap performa model, didapatkan jumlah yang paling optimal dalam hal akurasi validasi dan indikasi *overfitting* (gap *train* - *val* lebih dari lima) adalah 20. Dari tabel hasil percobaan tersebut terlihat semakin banyak *epoch* semakin tinggi akurasi, namun dilihat dari *val acc* maupun *val loss*, menunjukkan gap yang besar. Kebalikannya, semakin kecil jumlah *epoch* indikasi *overfitting* cukup terkontrol namun akurasi lebih rendah karena model belum cukup untuk belajar. Oleh karena itu, dalam proses pelatihan, jumlah *epoch* akan di-set 20.

Selain jumlah *epoch*, untuk mengatasi *overfitting* akan diterapkan *callback* yaitu *early stopping* untuk menghentikan pelatihan apabila tidak ada peningkatan

meskipun belum mencapai total *epoch*. Berdasarkan penelitian [54] yang membandingkan *early stopping patience* dua, tiga, empat, lima, 10, 15, dan 20, didapatkan *patience* lima yang paling optimal terhadap performa model dalam *val acc* dan *val loss*. Oleh karena itu, dalam proses pelatihan jumlah *patience early stopping* akan di set dengan *patience* = lima, sehingga apabila selama lima *epoch* berturut-turut stagnan, maka pelatihan akan dihentikan.

Callback lainnya adalah model *checkpoint* untuk menyimpan model terbaik selama pelatihan, *reduce learning rate* (*ReduceLROnPlateau*) untuk menyesuaikan (*adjust*) nilai *learning rate* jika tidak terjadi peningkatan pada *validation loss* selama sejumlah *epoch* tertentu. Berdasarkan penelitian [55] *patience* akan di set = tiga, dengan *factor* = 0.5 sehingga apabila validation loss tidak ada peningkatan selama tiga *epoch* maka *initial learning rate* akan dikurangi sebanyak 50%.

G.3 Hyperparameter Tuning

Berdasarkan penelitian [56, 57, 58], dalam tahap pelatihan ini dilakukan proses *hyperparameter tuning* (pencarian kombinasi parameter terbaik) secara manual, yaitu mengubah nilai parameter yang mempengaruhi akurasi validasi model. Setiap satu hasil konfigurasi pelatihan akan dicatat untuk dibandingkan hasilnya dengan konfigurasi lainnya dalam menemukan *hyperparameter* yang paling optimal. *Hyperparameter tuning* akan dilakukan secara bertahap dimulai dari pencarian *batch size* terbaik, *learning rate*, jenis *optimizer*, penambahan *custom layer* pada *head model* dasar (*feature extraction*), dan terakhir yaitu membuka beberapa layer pada model dasar untuk dilatih ulang (*fine tuning*).

Penentuan parameter terbaik didasarkan atas rata-rata akurasi validasi dari lima *fold*. Selain itu dinilai dari selisih (gap) antara akurasi latih (*train acc*) dan akurasi validasi (*val acc*), apabila *train acc* lebih tinggi dari *val acc* dan gap antara keduanya jauh maka termasuk indikasi *overfitting*. Berdasarkan penelitian [59] gap lebih dari 5% mengindikasikan model *overfitting*. Jika *generalization gap* melebihi 5%, hal ini sering menjadi indikasi bahwa model tidak mampu melakukan generalisasi dengan baik terhadap data baru [60, 61].

G.4 Batch Size

Dalam menentukan *batch size* yang sesuai dengan model dan dataset yang digunakan, dilakukan pelatihan dengan nilai *batch size* yang berbeda. Berdasarkan

penelitian [56], nilai yang akan dibandingkan yaitu 64, 32, dan 16. Ketiganya akan dikonfigurasi dengan parameter pelatihan yang sama yaitu *learning rate* 0.0001 dan *optimizer* Adam. Kombinasi *learning rate* 0.0001 dan *optimizer* Adam ini adalah yang paling optimal menurut penelitian [62] untuk base model ResNet50V2.

G.5 Learning Rate

Dalam menentukan *learning rate* yang sesuai dengan model yang digunakan, dilakukan pelatihan dengan nilai *learning rate* yang berbeda. Berdasarkan penelitian [57], nilai yang akan dibandingkan yaitu 0.001, 0.0001, dan 0.00001. Ketiganya akan dikonfigurasi dengan parameter pelatihan yang sama yaitu *optimizer* adam dan *batch size* terbaik dari pemilihan sebelumnya.

G.6 Optimizer

Dalam menentukan *optimizer* yang optimal dengan model dan dataset yang digunakan, dilakukan pelatihan dengan jenis *optimizer* yang berbeda. Berdasarkan penelitian [58], jenis *optimizer* yang akan dicoba dalam pelatihan yaitu Adam, RMSProp, dan SGD. Ketiganya akan dikonfigurasi dengan parameter yang sama yaitu *learning rate* terbaik dan *batch size* terbaik pada pemilihan sebelumnya.

G.7 Jumlah Neuron Dense Layer

Eksperimen ini bertujuan untuk memilih konfigurasi jumlah neuron yang paling optimal dalam meningkatkan performa klasifikasi pada dataset target dengan memanfaatkan fitur-fitur umum yang telah dipelajari oleh model *pretrained*. Berdasarkan penelitian [63], tiga konfigurasi *Dense layer* dengan 1024, 512, dan 128 neuron dengan aktivasi ReLU untuk menambahkan kapasitas *non-linearitas* menghasilkan akurasi yang paling tinggi. Oleh karena itu ketiganya akan dibandingkan untuk menemukan konfigurasi yang paling optimal. Seluruh pelatihan dilakukan dengan parameter yang sama yaitu *batch size* terbaik, *learning rate* terbaik, dan *optimizer* terbaik dari pemilihan sebelumnya.

G.8 Fine Tuning

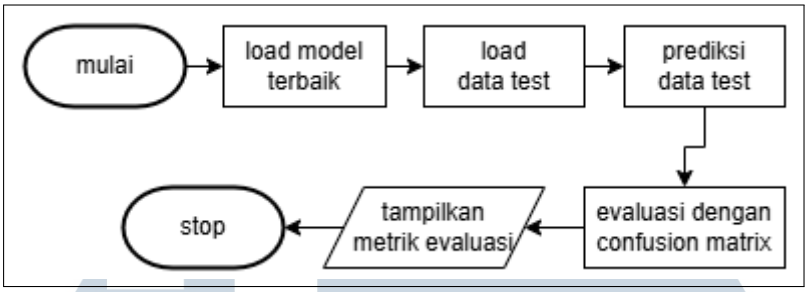
Penyesuaian layer (*fine tuning*) pada model *pretrained* yang telah dilatih sebelumnya adalah salah satu pendekatan efektif dalam melakukan *transfer*

learning [64]. Berdasarkan artikel [65], untuk menghindari perubahan drastis selama *fine-tuning*, *learning rate* sering dikurangi agar pembaruan bobot lebih halus dan tidak menyebabkan hilangnya pengetahuan yang diperoleh saat *pre-training*. Learning rate yang terlalu besar dapat merusak bobot yang sudah baik [52]. Oleh karena itu, *learning rate* akan dikurangi menjadi lebih kecil menjadi 0.00001, berbeda dengan fase pelatihan pertama yaitu 0.0001.

Pada pelatihan ini, dilakukan *fine tuning* terhadap model *pretrained* dengan membuka (*unfreeze*) sejumlah layer terakhir dari model dasar agar dapat dilatih ulang bersama dengan arsitektur tambahan (*head*). Berdasarkan penelitian [66], *fine tuning* secara selektif pada lapisan tertentu dapat menghasilkan performa lebih baik, menyetel terlalu banyak berisiko menghapus pengetahuan awal. Penelitian [67] melakukan *unfreeze* sebanyak 50 layer. Oleh karena itu konfigurasi layer yang di *unfreeze* dilakukan secara bertahap dimulai dari 40, 50, dan 60 *layer* terakhir dari model *pretrained* dengan parameter pelatihan yang sama yaitu *batch size* terbaik, *learning rate awal* terbaik, *learning rate fine tuning* 0.00001 (lebih kecil), *optimizer* terbaik, dan konfigurasi *layer Dense* dengan jumlah neuron paling optimal pada pemilihan sebelumnya.

H Pengujian dan Evaluasi

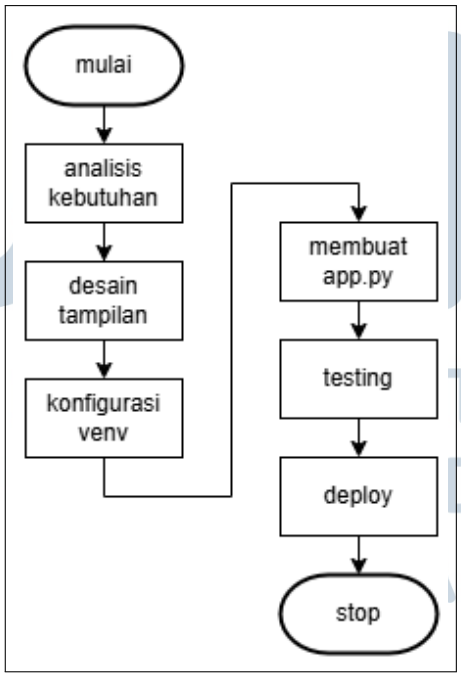
Gambar 3.8 menunjukkan *flowchart* proses pengujian dan evaluasi. Setelah proses pelatihan selesai dan model terbaik disimpan dalam format .h5, langkah pertama dalam pengujian adalah memuat model tersebut. Selanjutnya, data uji yang belum pernah digunakan selama pelatihan dimuat untuk memastikan evaluasi yang objektif. Model kemudian digunakan untuk memprediksi label data uji. Hasil prediksi dibandingkan dengan label sebenarnya menggunakan *confusion matrix* untuk memvisualisasikan akurasi klasifikasi per kelas. Setelah itu, dilakukan perhitungan metrik evaluasi seperti *precision*, *recall*, *F1-score*, dan akurasi secara keseluruhan melalui *classification report*. Proses ini bertujuan untuk memberikan gambaran menyeluruh mengenai performa model dalam menangani data uji.



Gambar 3.8. *Flowchart* pengujian dan evaluasi

3.3.2 Perancangan Website

Perancangan *website* dimulai dari proses identifikasi dan analisis kebutuhan untuk menentukan fitur serta spesifikasi sistem yang dibutuhkan. Setelah itu, dilakukan desain tampilan untuk merancang antarmuka pengguna yang sesuai dengan kebutuhan. Tahap berikutnya adalah konfigurasi virtual *environment* (venv) sebagai lingkungan kerja terisolasi untuk pengembangan aplikasi. Setelah lingkungan siap, dilakukan pembuatan *file* utama web, yaitu app.py, yang berisi logika program utama. Tahapan selanjutnya adalah pengujian (*testing*) untuk memastikan bahwa semua fitur berjalan sesuai harapan, dan diakhiri dengan proses *deploy* agar *website* dapat diakses oleh pengguna.



Gambar 3.9. *Flowchart* perancangan website

A Analisis Kebutuhan

Web ini bertujuan untuk mendeteksi penyakit katarak pada mata menggunakan model yang telah dilatih yang diintegrasikan agar bisa diakses secara daring. Pengguna dapat mengunggah gambar mata, dan web akan memproses serta memberikan hasil prediksi apakah mata tersebut terkena katarak atau tidak.

A.1 Kebutuhan Fungsional

Kebutuhan fungsional adalah fitur-fitur yang harus dimiliki oleh web agar dapat berjalan sesuai tujuan.

1. Input gambar mata

   Web harus dapat menerima input berupa gambar mata dari pengguna.

2. Prapemrosesan gambar

   Web harus melakukan prapemrosesan pada gambar (resize, normalisasi, dan lainnya) sebelum dikirim ke model.

3. Prediksi Katarak

   Web harus dapat memproses gambar menggunakan model yang sudah dilatih untuk mendeteksi katarak.

4. Menampilkan hasil prediksi

   Web harus menampilkan hasil prediksi kepada pengguna, misalnya: ”Katarak” atau ”Normal”.

5. Antarmuka pengguna

   Web harus menyediakan antarmuka yang mudah digunakan, baik berbasis *mobile* maupun desktop.

A.2 Kebutuhan Fungsional

Kebutuhan non-fungsional adalah syarat-syarat yang berkaitan dengan kualitas sistem.

1. Keamanan

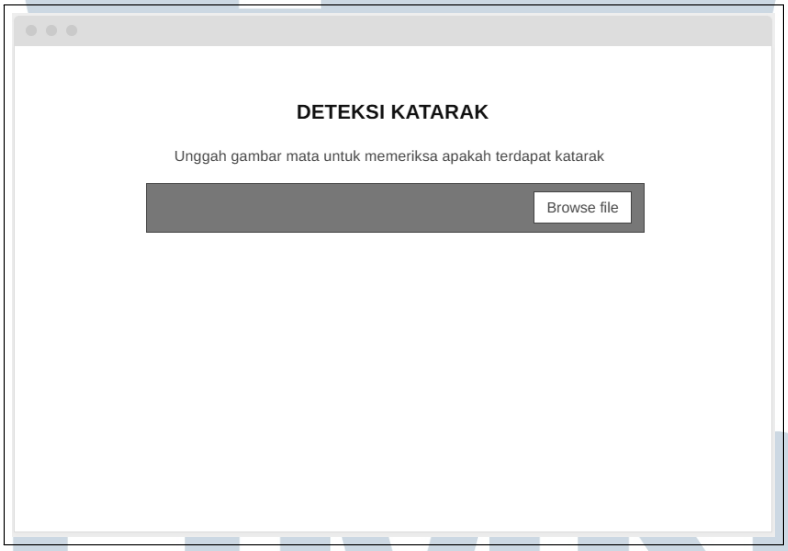
   Web harus menjaga kerahasiaan data pengguna, dengan tidak menyimpan gambar mata secara permanen dan hanya pada memori sementara.

2. Kinerja

   Web harus mampu memberikan hasil prediksi dalam waktu yang singkat (*real-time*).

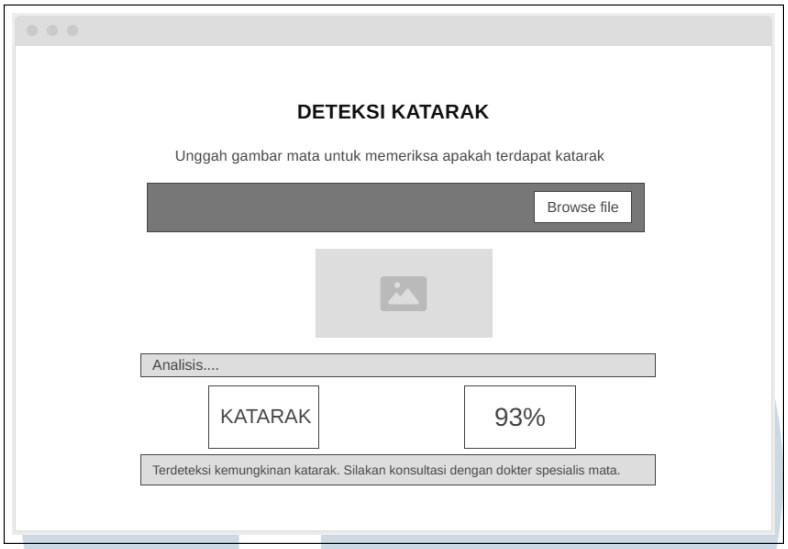
B Desain Tampilan

Pada tahap ini dilakukan perancangan *website* berupa desain tampilan antarmuka pengguna. Pada tahap ini, perancangan menentukan struktur halaman dan penempatan elemen-elemen seperti tulisan, tombol dan *form* unggah. Desain ini bisa dibuat dalam bentuk *wireframe* untuk memberikan gambaran awal sebelum masuk ke tahap implementasi.



Gambar 3.10. *Wireframe* form upload

Gambar 3.10 menunjukkan *wireframe* dari halaman antarmuka pengguna untuk fitur deteksi katarak. Tampilan ini dirancang dengan sederhana dan intuitif, dimulai dengan judul halaman ”DETEKSI KATARAK” yang diletakkan di bagian atas untuk memberikan konteks kepada pengguna. Di bawah judul, terdapat instruksi singkat yang menjelaskan bahwa pengguna perlu mengunggah gambar mata untuk memeriksa adanya indikasi katarak. Komponen utama dari halaman ini adalah sebuah *form* unggah berwarna abu-abu gelap yang dilengkapi dengan tombol ”*Browse file*” di sisi kanan, yang memungkinkan pengguna memilih *file* gambar dari perangkat mereka. Desain ini mengutamakan kejelasan dan kemudahan penggunaan untuk memastikan pengguna dapat mengakses fitur utama tanpa kebingungan.



Gambar 3.11. *Wireframe* hasil prediksi

Gambar 3.11 menampilkan *wireframe* hasil prediksi dari sistem deteksi katarak. Setelah pengguna mengunggah gambar mata, sistem akan menampilkan pratinjau gambar yang diunggah dan memulai proses analisis. Pada bagian bawah gambar terdapat indikator proses analisis yang memberikan informasi bahwa sistem sedang memproses data. Hasil klasifikasi ditampilkan secara jelas dalam dua kotak, yaitu label hasil (”KATARAK”) dan persentase tingkat kepercayaan (contohnya 93%). Di bagian paling bawah, terdapat pesan peringatan yang bersifat informatif, menyarankan pengguna untuk melakukan konsultasi dengan dokter spesialis mata apabila terdeteksi kemungkinan katarak. Desain ini dirancang untuk memberikan hasil yang mudah dipahami oleh pengguna umum serta menyampaikan informasi dengan jelas dan cepat.

C Konfigurasi Virtual Environment

Setelah desain tampilan selesai, langkah selanjutnya adalah konfigurasi virtual *environment* (venv). *Virtual environment* digunakan untuk membuat lingkungan pengembangan terisolasi agar dependensi proyek tidak bercampur dengan proyek lain atau dengan sistem Python utama. Dengan venv ini dapat mengaktifkan lingkungan ini dan menginstal semua kebutuhan pustaka melalui pip.

D Pembuatan app.py

Pada tahap ini, *file* utama aplikasi web yakni app.py mulai dibuat. *File* ini berisi logika dasar dari server web seperti *routing*, pemrosesan data dari pengguna, serta integrasi dengan desain tampilan yang telah dirancang sebelumnya.

E Testing

Setelah aplikasi dasar terbentuk, proses testing dilakukan untuk memastikan bahwa *website* berjalan sesuai harapan. Pengujian dilakukan secara manual dengan mencoba langsung fitur-fitur pada web. Tujuannya adalah untuk menemukan dan memperbaiki *bug* sedini mungkin sebelum *website* di*deploy*.

F Deployment

Tahap terakhir adalah *deployment*, yaitu proses memindahkan *website* dari lingkungan lokal ke server produksi agar dapat diakses oleh pengguna melalui internet. Proses ini dilakukan menggunakan layanan Streamlit sebagai *framework* utama pembuatan *website*.