BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

AI & Machine Learning Research and Development Intern berada dibawah divisi IT perusahaan. Seluruh hasil kerja dilaporkan kepada supervisor selaku IT Lead. Seluruh tugas dan koordinasi dilakukan melalui *email* serta pesan *whatsapp*.

3.2 Tugas dan Uraian Kerja Magang

Pada awal periode magang diberikan pengantar mengenai *Javascript* serta implementasi *restful API*. Pengantar meliputi beberapa *challenge coding* serta materi yang berhubungan dengan projek yang akan dikerjakan. Setelah mendapatkan pengantar, dilanjutkan dengan pengembangan sistem *face recognition* yang dapat digunakan melalui *API.* Setelah project selesai, jobdesk yang diberikan adalah melakukan riset serta pengembangan model face recognition.

Berikut adalah tabel *timeline* pengerjaan proyek *Face recognition API*.

Tabel 3.1. Detail Pekerjaan

No.	Pekerjaan	Mulai	Selesai	Keterangan
1	Orientasi Karyawan	26 Agustus, 2024	29 Agustus, 2024	Penjelasan jobdesk, *coding challenge*, penjelasan projek yang akan dikerjakan
2	Brainstorming Project	30 Agustus, 2024	04 September, 024	Melakukan diskusi, riset, menentukan endpoints

3	Implementasi model deteksi, *setup database*	05 September, 2024	09 September, 2024	Menambahkan *library* model deteksi, memasukkan data yang diperlukan kedalam *database*, melakukan uji coba.
4	Menambahkan *authentication, encryption*	10 September, 2024	11 September, 2024	Menambahkan fitur *authentication* dan token. Membuat serta melakukan tes pada tiap *endpoints*
5	*Testing & Debugging*	12 September, 2024	20 September, 2024	Melakukan tes dan mencari *bug*.
6	*Tuning & Bug Fixing*	23 September, 2024	27 September, 2024	Melakukan tuning model deteksi, serta melakukan *bug fixing*.

Kegiatan magang diawali dengan orientasi Karyawan pada minggu 1-2. Pada masa ini, lingkungan bekerja diperkenalkan bersama dengan supervisor serta penjelasan jobdesk. Diberikan juga pengantar yang bertujuan untuk memberikan gambaran atas project yang akan dikerjakan. Setelah diberikan pengantar, dilanjutkan dengan pengerjaan project *face recognition API* pada minggu 3-6. Selanjutnya diberikan tugas untuk melakukan riset serta *tuning model face recognition* pada minggu 7-12.

3.2.1 Orientasi Karyawan

Kegiatan magang dimulai pada hari Senin, 26 Agustus 2024 secara secara daring pada jam 08.00 WIB. Pada kesempatan ini diperkenalkan dengan supervisor serta penjabaran tugas yang akan diberikan. *Coding challenge* seputar implementasi *javascript* serta hubungannya dengan restful API. *Challenge* terdiri dari 3 bagian yang berisikan materi – materi yang banyak digunakan dalam implementasi pada project nantinya. Supervisor juga memberikan bahan belajar untuk dipelajari secara mandiri.

3.2.2 Brainstorming project

Tujuan project ini adalah membuat sistem *face recognition* berbasis *API.* Pembuatan *API* ini ditujukan untuk melakukan verifikasi terhadap *alert evidence* dari *device monitoring driver*. Pada lokasi pertambangan, tiap mobil operasional memiliki *device* yang melakukan pengawasan terhadap pengemudi. *Device* ini mendeteksi kelainan yang terjadi pada pengemudi. Beberapa kejadian yang dapat dideteksi oleh alat ini antara lain: keadaan *fatigue* dimana ciri – ciri seseorang kelelahan dimulai dari mata tertutup, serta pandangan menghadap kebawah. Selanjutnya keadaan pengemudi yang terganggu seperti menggunakan alat komunikasi, serta mampu mendeteksi kondisi penggunaan sabuk pengaman. Adapula kondisi yang belum bisa dideteksi adalah pengemudi merupakan pengemudi yang terjadwal pada saat itu. Dari semua kejadian ini, alat akan mengambil foto *evidence* yang kemudian dikirimkan ke *control room*.

API ini dirancang untuk melakukan verifikasi terhadap pemilik wajah yang ada pada *evidence* tersebut. Karena *evidence* yang masuk terlalu banyak, sehingga akan menyulitkan apabila harus diperiksa secara *manual*.

Application Programming Interface (API) merupakan mekanisme untuk 2 aplikasi berkomunikasi satu sama lain dalam dengan menggunakan protokol yang telah ditentukan. *API* menghubungkan antara server dan client (Iqbal, 2023).

Gambar 3.1. Ilustrasi cara kerja API

Aplikasi yang mengirimkan *request* disebut sebagai *client*, sedangkan yang mengirim *response* merupakan server. Sedangkan fungsi dari *API* sebagai perantara

komunikasi antar kedua aplikasi. Sedangkan *RESTful* API didasarkan prinsip *Representational State Transfer* (REST) yang menekankan *statelessness*, arsitektur *client – server* serta metode standar manipulasi data yaitu *Create, Read, Update, Delete* (CRUD), yang memungkinkan pengguna untuk melakukan operasi dasar pada aplikasi. Dalam *RESTful API*, operasi *CRUD* diimplementasikan menggunakan *HTTP Methods* yaitu *GET, POST, PUT, DELETE*.

Berikut adalah daftar *endpoint* pada sistem bersama dengan *HTTP Methods* yang digunakan.

Tabel 3.2. Daftar *Endpoints*

Endpoint	HTTP Methods	Keterangan
/register	POST	Mendaftarkan username
/login	POST	Login
/enroll-face	POST	Mendaftarkan wajah
/identify-face	POST	Mengirim gambar yang akan dideteksi
/create-facegallery	POST	Membuat galeri wajah
/list-faces	GET	List wajah pada galeri
/delete-facegallery	DELETE	Menghapus galeri
/my-facegallery	GET	List galeri yang dimiliki user
/delete-face	DELETE	Menghapus wajah

Endpoint /register dengan metode POST digunakan untuk mendaftarkan pengguna baru ke dalam sistem, data yang diperlukan adalah *username* dan kata sandi. Selanjutnya, *endpoint /login* dengan metode *POST* digunakan untuk autentikasi pengguna yang sudah terdaftar. Pengguna harus memasukkan kata sandi yang valid untuk masuk ke sistem. Setelah melakukan *login*, pengguna akan mendapatkan *token* untuk dapat mengakses fitur – fitur lainnya.

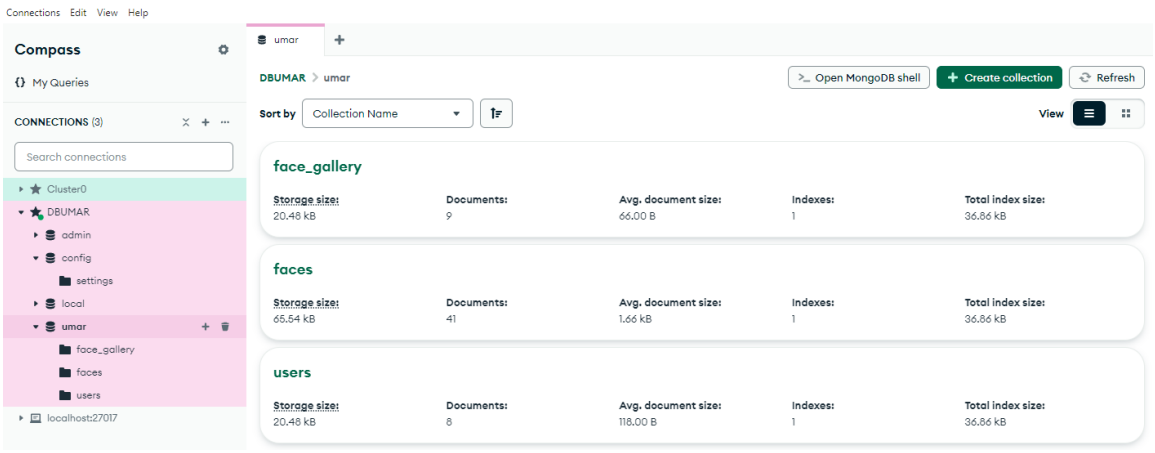
Endpoint /enroll-face memungkinkan pengguna untuk mendaftarkan data wajah mereka ke sistem *menggunakan metode POST*. Data wajah ini kemudian dapat digunakan untuk keperluan identifikasi. Untuk proses identifikasi, *endpoint /identify-face* digunakan dengan mengirimkan gambar melalui metode *POST* agar sistem dapat mendeteksi atau mencocokkan wajah dengan data yang sudah terdaftar sebelumnya.

Endpoint /create-facegallery dengan metode *POST* digunakan untuk membuat galeri wajah baru, sedangkan */list-fac*es dengan *metode GET* memungkinkan pengguna melihat daftar wajah yang ada dalam galeri tertentu. Pengguna dapat melihat daftar galeri yang mereka miliki melalui *endpoint /my-facegallery* dengan metode *GET*. Jika ingin menghapus galeri, pengguna dapat menggunakan *endpoint /delete-facegallery* dengan metode *DELETE*. Selain itu, *endpoint /delete-face* dengan metode *DELETE* digunakan untuk menghapus data wajah tertentu dari galeri. Setiap proses yang dilakukan user, memerlukan token sebagai autentikasi sehingga user hanya dapat melakukan perubahan kepada miliknya sendiri.

3.2.3 Implementasi model deteksi, *setup database*

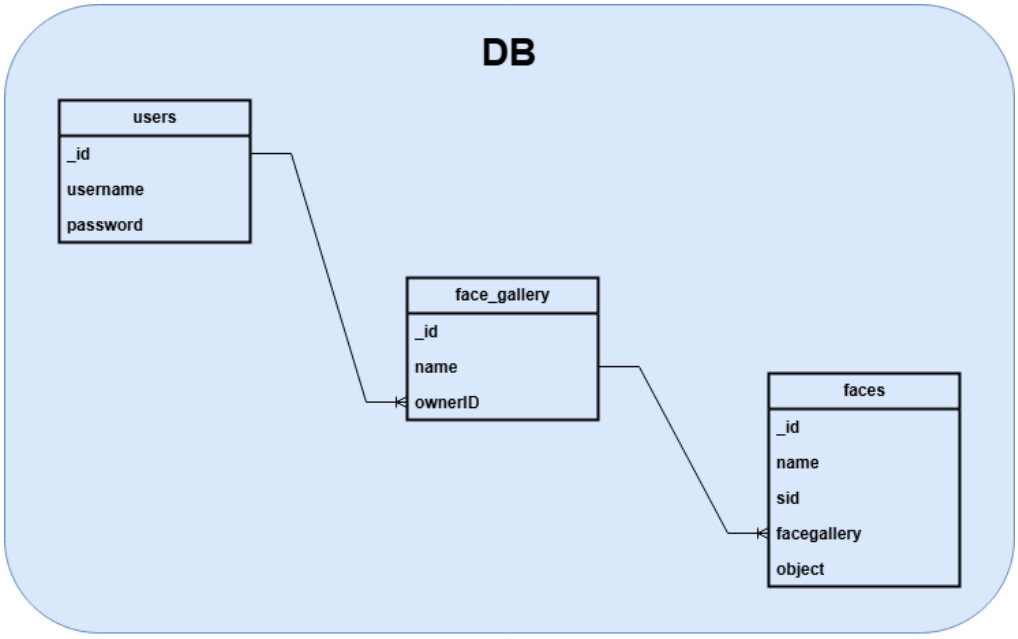
Selanjutnya proyek ini menggunakan service dari *MongoDB* sebagai database. Karena proyek yang dilakukan masih dalam tahap *proof of concept,* maka dari itu belum ada *database* yang dapat digunakan untuk mendukung proyek ini. *Database* ini diperlukan untuk menyimpan *username* serta *password* dari user, dan juga sebagai penyimpanan *dataset* wajah dari driver. *MongoDB* dipilih karena merupakan *database* kategori *NoSQL*. Pemilihan *NoSQL* dari *MongoDB* karena pengaplikasiannya yang mudah serta cocok untuk tahap pengembangan karena membutuhkan fleksibilitas sehingga tidak perlu melakukan banyak konfigurasi terhadap struktur *databasenya.*

Berikut adalah tampilan isi database yang dilihat melalui antarmuka yang disediakan oleh *MongoDB.*



Gambar 3.2. Tampilan database

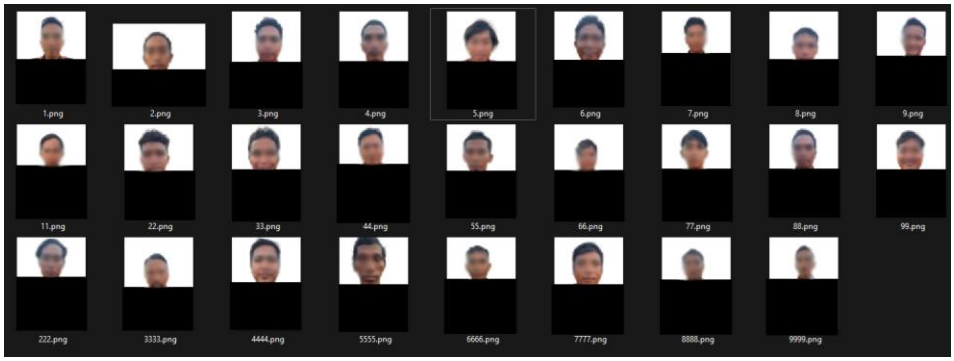
Dapat dilihat pada gambar, *database* memiliki 3 *collections* yaitu *face_gallery*, *faces*, dan users. *MongoDB* menyimpan data dalam bentuk *documents* yang merupakan tipe data yang digunakan pada *MongoDB*. *Collections* merupakan kumpulan dari *documents* yang tersimpan.



Gambar 3.3. Database schema

Database terdiri dari tiga *collection*, yaitu *users*, *face_gallery*, dan *faces*. *Collections users* digunakan untuk menyimpan data pengguna, yaitu *_id*, *username*, dan *password*. Selanjutnya, *collections face_gallery* berfungsi untuk menyimpan informasi galeri wajah, dengan *_id* sebagai identitas galeri, *name* untuk nama galeri, dan *ownerID* sebagai referensi ke *_id* pengguna dalam *collections users*. Collections faces, digunakan untuk menyimpan data wajah dalam setiap galeri, dengan *_id* sebagai identitas unik wajah, *name* sebagai nama pemilik wajah, *sid* sebagai *id operator*, *facegallery* yang mereferensikan *_id* pada *collections face_gallery*, dan object yang menyimpan data wajah. Hubungan antar collections bersifat *one-to-many*, di mana satu pengguna dapat memiliki banyak galeri, dan setiap galeri dapat menyimpan banyak wajah.

Selanjutnya *dataset training* juga dimasukkan kedalam database. Gambar dimasukkan kedalam database melalui *endpoint /enroll-face*, karena gambar yang masuk kedalam *database* perlu melewati algoritma *face descriptor* terlebih dahulu. *Face descriptor* disini merupakan representasi numerik dari fitur wajah. Nantinya *value* ini akan disimpan bersamaan pada *database* bersama gambar.



Gambar 3.4. Dataset training wajah

Dataset ini terdiri dari 41 gambar wajah, masing – masing berasal dari individu yang berbeda. Selanjutnya untuk digunakan sebagai perbandingan dengan *dataset* pengujian nantinya. Gambar merupakan foto *close-up* dengan format *png* berukuran 433x577 pixel. Gambar digunakan tanpa dilakukan *pre-processing*.

Dataset *testing* diambil dari *alert evidence* yang berasal dari *device* yang terpasang pada kendaraan operasional, berikut adalah beberapa sampel *data testing* yang digunakan.

Gambar 3.5. Dataset testing wajah

Total sebanyak 119 gambar digunakan sebagai *dataset testing*. Gambar berukuran 704x576 *pixel* dengan format *jpeg.* Gambar ini akan dikomparasikan dengan *dataset training*. *Dataset training* terdiri dari gambar-gambar wajah yang dimiliki oleh individu yang sama dengan yang terdapat dalam *dataset training.*

Selanjutnya pada bagian model deteksi, digunakan *library faceapi.js*. *Library* ini menawarkan berbagai macam fitur deteksi seperti *face detection, face recognition, emotion recognition, facial attribute,* dan juga *real-time processing. Library* ini memiliki 3 *pre-trained model* yang tersedia untuk melakukan *face recognition* yaitu *SSDMobileNetV1, TinyFaceDetector, Multi-task Cascaded Convolutional Neural*

Network (MTCNN). Sehingga dalam pengaplikasian *library* ini tidak perlu melakukan *training* model kembali.

SSDMobileNetV1 merupakan model deteksi hasil gabungan dari Single *Shot MultiBox Detector* (SSD) dengan *MobileNetV1* sebagai *backbone*. Model ini bekerja dengan pendekatan *single shot* yang deteksi dilakukan dalam satu proses.

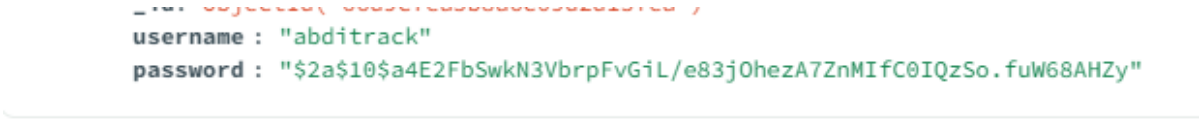
Selanjutnya *Tiny Face Detector* yang merupakan model deteksi wajah, model ini dapat melakukan pengenalan wajah berdasarkan fitur wajah. Setelah melakukan deteksi, model akan mengekstrak fitur – fitur tersebut untuk meningkatkan akurasi dalam pengenalan wajah.

Multi-task Cascaded Convolutional Neural Network bekerja dengan tiga jaringan *Convolutional Neural Network* yaitu *P-Net, R-Net,* dan O-Net. *P-Net* menghasilkan kotak pembatas serta letak wajah. *R-Net* melakukan penyempurnaan dengan meningkatkan akurasi kotak pembatas. Lalu *O-Net* mendeteksi fitur wajah seperti mata dan mulut.

Ketiga model deteksi tersebut, telah melalui tahap *training* untuk melakukan deteksi wajah. Untuk *SSDMobileNetV1,* telah dilatih menggunakan *WIDERFACE dataset. WIDERFACE dataset* merupakan dataset acuan yang digunakan untuk evaluasi serta pengembangan algoritma deteksi wajah, gambarnya dipilih dari sumber yang tersedia secara umum. Untuk *model Tiny Face Detectior* telah dilatih menggunakan *dataset* yang terdiri dari ±14.000 gambar wajah yang telah dilabeli menggunakan kotak pembatas

3.2.4 Menambahkan *authentication, encryption*

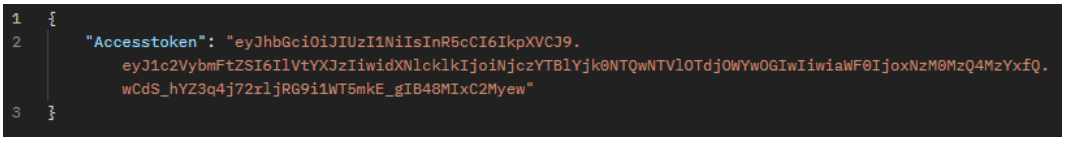
Pada sistem ini, digunakan *library bcrypt.js* untuk melakukan *hashing password*. *Hashing password* dilakukan agar *password* yang disimpan pada *database* tidak dalam bentuk teks asli. *Bcrypt.js* juga menangani *authentication* pada *password* user disaat melakukan *login.*



Gambar 3.6. Hashing password

Dapat dilihat pada kolom *password* berbentuk *random string* sehingga *password* yang asli tidak terlihat.

Apabila pengguna berhasil melakukan login, server akan memberikan respon dengan mengirimkan *token*, token ini yang akan digunakan oleh pengguna agar dapat menggunakan fitur yang tersedia. Untuk melakukan *generate token* digunakan library *Json Web Token* (JWT). Dapat dilihat pada gambar berikut pada saat user telah berhasil melakukan *login*.



Gambar 3.7. Response server

Server memberikan respon *token* yang dapat digunakan oleh pengguna. Struktur token pada umumnya terdiri dari *Header, payload,* dan *signature* yang membuat tiap token unik, sehingga tiap user akan memiliki token yang berbeda (Rana, dkk, 2023).

3.2.5 *Testing & Debugging*

Pengujian dibagi menjadi 2 bagian, yaitu untuk menguji fitur keseluruhan dari *API* dan menguji performa serta akurasi dari deteksi. Untuk menguji fitur, semua *endpoints* dicoba untuk melihat hasil respon yang diberikan apakah sudah sesuai dengan keinginan. Pengujian keseluruhan fitur menggunakan aplikasi *Postman* untuk mengirimkan *http request* serta menerima respon dari server

Untuk pengujian pertama, dilakukan dengan mengirimkan *http request* melalui *postman* dan melihat respon yang diberikan oleh *server* apakah sudah sesuai dengan

keinginan. Untuk respon yang diharapkan pada masing- masing endpoint, disajikan dalam tabel berikut ini.

Tabel 3.3 Daftar respon *server*

Fungsi		Input		Response
		Header	Body	

Post	Register		{
			"name": "abditest",
			"username": "abditest",
			"password": "abditest"
			}
				{
				"message": "User created successfully"
				}

Post	Login		{
			"username": "abditrack",
			"password": "abditrack"
			}
				{
				"Accesstoken": ""
				}

Get	My Facegallery	accesstoken	["Gallery Abditrack", "Gallery Abditest 2", "abdi456", "abdi789"]	[{ "_id": "", "name": "Username", "ownerId": "" }, { "_id": "", "name": "Username", "ownerId": "" }] response code : **200**

Post	Create Facegallery	Accesstoken	{
			facegallery_name: "apaaja"
			}
				{
				"facegallery_name": "apaaja",
				"status": "201",
				"status_message": "Success"
				}
				
				response code : **201**

Del	Delete Facegallery	Accesstoken	{
			"name": "Abditestzs"
			}
				{
				"facegallery_id": "teshapus",
				"status": "200",
				"status_message": "Success"
				}

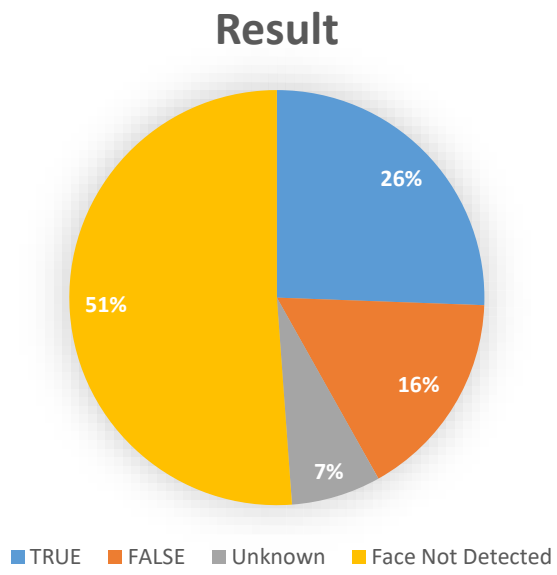
Post	Enroll Face	Accesstoken	face (foto)
			{
			user_id: "SID operator",
			user_name: "nama operator",
			facegallery_name: "nama face gallery"
			}
				{
				"status": "201",
				"status_message": "Success - Face Successfully Enrolled"
				}
				
				response code: **201**

Get	List Faces	Accesstoken	{
			"facegallery_name": "Test gallery"
			}
				{
				"faces": [
				{
				"user_id": "IDUSER1",
				"user_name": "Nama User Satu"
				},
				{
				"user_id": "IDUSER2",
				"user_name": "Nama User Dua"
				},
				...
				],
				"status": "200",
				"status_message": "Success"
				}

Pada tabel 3.3 dapat dilihat bahwa respon yang diharapkan dari *server* untuk masing – masing *request* ada pada kolom *response*. Kolom *body* dan *header* merupakan bentuk *request* yang harus dikirimkan pada *server*.

Hasil dari pengujian performa serta akurasi akan disajikan dalam *pie chart* dibawah ini:

Tabel 3.4. Hasil deteksi



Berdasarkan hasil uji coba, didapatkan hasil sebagai berikut, hasil *true* adalah hasil deteksi yang berhasil serta dengan hasil yang benar. Hasil deteksi *true* adalah sebesar 26%. Selanjutnya *false*, *false* adalah deteksi yang berhasil tetapi hasilnya salah. Didapatkan hasil false sebesar 16%. Selanjutnya untuk *face not detected* adalah sebesar 51%, sedangkan *unknown sebesar 7%*. Untuk tiap *API call* juga memakan waktu selama 7 detik. Setelah mendapatkan hasil tersebut dapat dihitung hasil akurasi dari sistem yang telah dibuat dimana

$$Akurasi = \frac{Hasil\ True}{Total\ deteksi}$$

$$Akurasi = \frac{22}{86}$$

$$\boldsymbol{Akurasi = 25.58\%}$$

Sehingga akurasi yang didapatkan adalah sebesar 25.58%. Untuk menilai model lebih lanjut digunakan *precision, recall*, dan *F1 score* dengan rumus sebagai berikut (Yisihak, 2024)

$$Precision = \frac{True\ Postive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ positive + False\ Negative}$$

$$F1 = 2\ . \frac{Precision\ .\ Recall}{Precision + Recall}$$

Selanjutnya rumus tersebut diimplementasikan dengan data yang telah didapat pada saat pengujian sehingga hasilnya adalah sebagai berikut

$$Precision = \frac{TRUE}{TRUE + FALSE} = \frac{22}{22 + 14} = \mathbf{0,6111}\ (\mathbf{61,11}\%)$$

$$Recall = \frac{TRUE}{TRUE + UNKNOWN} = \frac{22}{22 + 6} = \mathbf{0,7857}\ (\mathbf{78,57}\%)$$

$$F1 = 2\ . \frac{Precision\ .\ Recall}{Precision + Recall} = 2. \frac{0{,}6111\ .\ 0{,}7857}{0{,}6111 + 0{,}7857} = \mathbf{0.687}\ (\mathbf{68.7}\%)$$

Berdasarkan hasil yang didapat, nilai *precision* yang rendah mengindikasikan persentase deteksi salah masih tinggi. Selanjutnya nilai *recall* yang lebih tinggi mengindikasikan sistem berhasil mendeteksi wajah yang cocok dengan benar. Untuk keseluruhan sistem dinilai dari *F1 score* sebesar 68,7% , menunjukkan hasil yang kurang baik sehingga sistem perlu dilakukan peningkatan terhadap sistem kembali

Besarnya hasil *face not detected* dapat dipengaruhi dari *dataset testing* yang memiliki kualitas gambar kurang bagus. Sehingga berdasarkan penjelasan sebelumnya dimana respon *face not detected* adalah tidak terdeteksinya wajah pada sebuah gambar.

3.2.6 *Tuning & Bug Fixing*

Pada proses tuning, dicoba beberapa model yang tersedia pada *library* yaitu *SSDMobilenetV1*, *TinyFaceDetector,* dan juga *MTCNN*. Dari proses uji coba beberapa model ini ditemuka bahwa *SSDMobileNetV1* memiliki tingkat akurasi yang paling tinggi diantara dibanding dua model lainnya. Hal ini ditunjukkan dari persentase *true* dari model mencapai angka 26%, sedangkan dua model lainnya berada dibawah itu.

Dilakukan juga penyesuaian *cut off threshold* pada *endpoint identify face* yaitu sebesar 0.55, sehingga hasil *distance* >0.55 dianggap sebagai *unknown*.

3.3 Kendala yang Ditemukan

- **Model yang kurang *robust***
  Penggunaan model deteksi *pre-trained* memberikan keterbatasan terhadap kemampuan model.
- **Kualitas *Dataset* kurang baik**
  Beberapa gambar dari *dataset* memiliki kualitas buruk seperti *blur,* kepala dengan posisi menunduk, dan kondisi lainnya yang menyebabkan wajah tertutup

3.4 Solusi atas Kendala yang Ditemukan

- **Menyusun *model* yang sesuai**
  Untuk pengembangan lanjutan, dapat melakukan *training model* sendiri sehingga cocok dengan keadaan.
- **Memilah *dataset***
  Sebelum digunakan, *dataset* yang tidak memungkinkan untuk dipakai dapat dihilangkan dahulu.