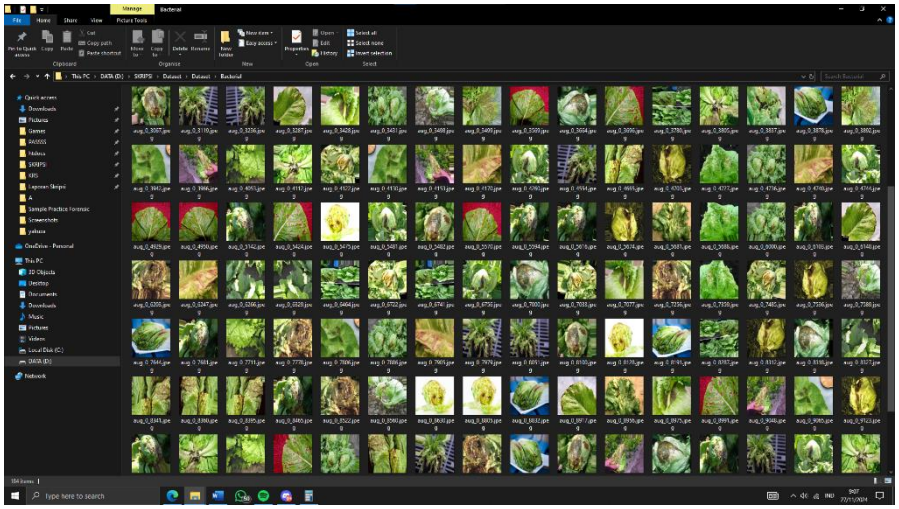
BAB IV
HASIL DAN PEMBAHASAN

4.1 Pengumpulan Data

Pengumpulan data dilakukan dengan menggunakan dataset yang tersedia di platform *Kaggle*. Dataset ini berisi gambar daun selada yang telah diklasifikasikan ke dalam berbagai kategori, seperti sehat atau terinfeksi penyakit tertentu. Total gambar yang digunakan adalah sebanyak 2320 gambar dan Gambar 4.1. menunjukkan hasil dari pengumpulan data melalui *Kaggle*

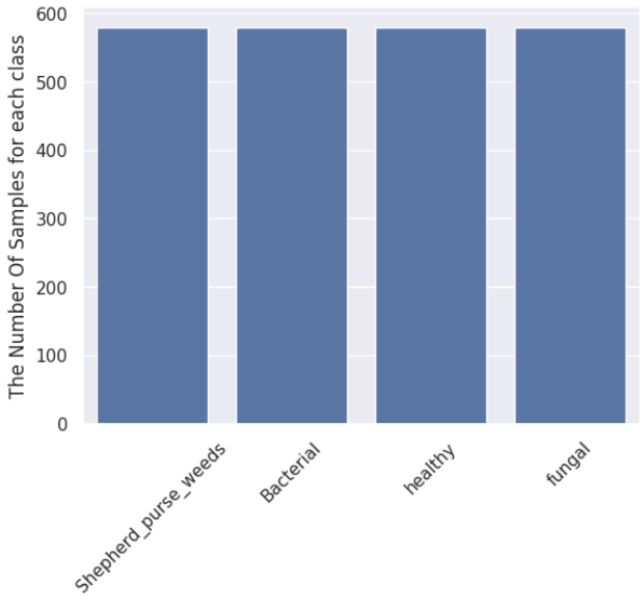


Gambar 4.21 gambar pengumpulan data

Dari hasil pengumpulan data gambar daun selada sebanyak 2320 gambar melalui *Kaggle*, proses augmentasi dilakukan untuk meningkatkan variasi data, sehingga model dapat lebih baik dalam mengenali berbagai kondisi daun selada

4.1.1 Labelling Data

Proses pelabelan data bertujuan untuk memberikan identitas pada data sehingga dapat dikenali oleh model. Setiap folder diberi nama sesuai dengan kategori yang sesuai, seperti *bacterial*, *fungal*, *healthy*, dan *shepherd purse weeds*. Setiap folder diberi nama sesuai dengan kategori yang sesuai dan diisi dengan data yang sesuai dengan namanya, seperti yang ditunjukkan pada Gambar 4.2.



Gambar 4.22 Label Data

4.2 Preproccessing Data

Tahapan *preprocessing data* citra meliputi pengumpulan data (*gathering*), pemberian label pada citra, serta augmentasi citra terhadap seluruh data yang tersedia.

4.2.1 Augmentasi Data

Augmentasi data bertujuan untuk mengurangi overfitting pada dataset yang tersedia. Selain itu, teknik augmentasi gambar ini juga dapat menambah jumlah data yang digunakan. Namun, data yang dihasilkan hanya dapat digunakan dalam proses pelatihan model. Berikut adalah kode yang digunakan untuk melakukan augmentasi pada gambar.

```
from tensorflow.keras.preprocessing.image
import ImageDataGenerator

image_size = (256, 256)
batch_size = 32

datagen = ImageDataGenerator(
    rescale=1.0/255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
```

```
    horizontal_flip=True,
    validation_split=0.2,
    fill_mode='nearest'
)


train_generator = datagen.flow_from_dataframe(
    df_train,
    x_col='image',
    y_col='label',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True
)

test_generator = datagen.flow_from_dataframe(
    df_test,
    x_col='image',
    y_col='label',
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False
)
```

Fungsi *ImageDataGenerator* digunakan dalam program di atas untuk mengubah gambar, yang kemudian disimpan dalam variabel datagen. Parameter yang digunakan untuk menghasilkan peningkatan gambar adalah sebagai berikut.

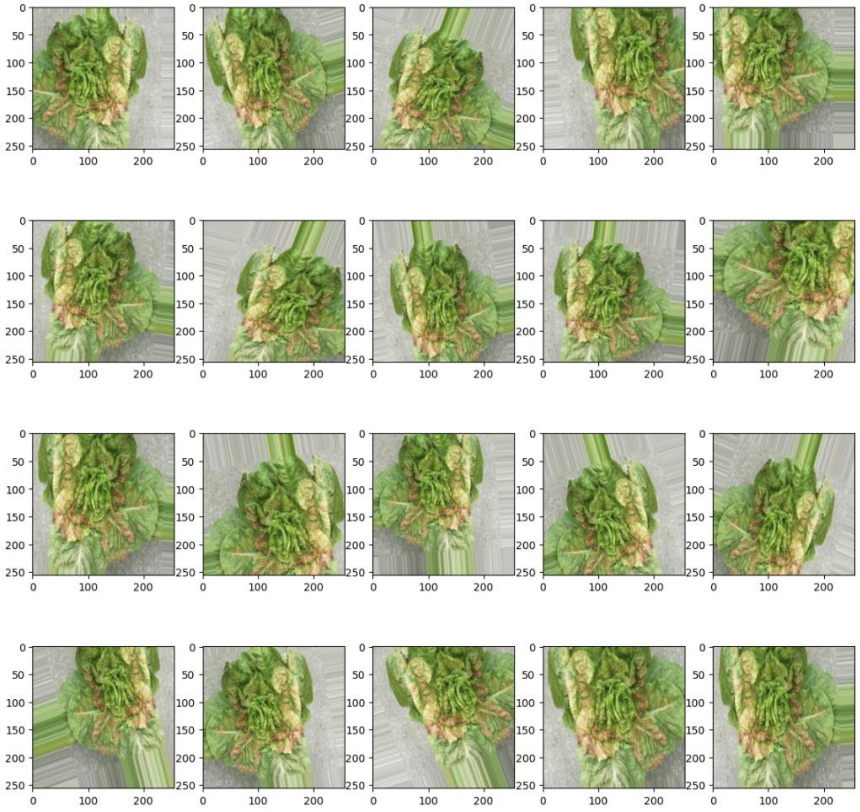
a. *rescale*=1.0/255. Melakukan normalisasi nilai piksel gambar, mengubah rentang piksel dari 0-255 menjadi 0-1 dengan membagi setiap nilai piksel dengan 255.
b. *rotation_range*=20. Mengatur rentang rotasi gambar. Gambar dapat diputar secara acak hingga 20 derajat, baik searah atau berlawanan arah jarum jam.
c. *width_shift_range*=0.2. Mengubah foto ke arah horizontal (kanan atau kiri) hingga 20% dari lebar gambar.
d. *height_shift_range*=0.2. Mengubah foto ke arah vertikal (atas atau bawah) hingga 20% dari tinggi gambar.

e. *shear_range*=0.2. Melakukan transformasi *shearing* pada gambar, yang berarti gambar akan diubah dalam bentuk kemiringan dengan tingkat kemiringan sebesar 20%.

f. *zoom_range*=0.2. Melakukan zoom pada gambar secara acak dalam rentang 20%. Gambar bisa diperbesar atau diperkecil.

g. *horizontal_flip*=True. Membalik gambar secara horizontal (flip). Ini membantu model untuk belajar lebih banyak variasi dari gambar yang tersedia.

h. *validation_split*=0.2. Menentukan persentase data yang akan digunakan untuk validasi. Di sini, 20% dari total data akan digunakan untuk validasi, sementara sisanya (80%) akan digunakan untuk pelatihan.



Gambar 4.23 Hasil Augmentasi Data

Gambar 4.3 menunjukkan hasil dari augmentasi gambar. Proses augmentasi ini akan memutar, memiringkan, dan memperbesar gambar secara acak berdasarkan parameter yang telah dijelaskan sebelumnya.

4.3 Building Model

Pemodelan CNN akan terdiri dari 4 lapisan *convolutional*, dan 1 lapisan *fully connected* (*neural network*). Proses pemodelan CNN ini akan dilakukan menggunakan TensorFlow, seperti yang dijelaskan berikut ini.

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
BatchNormalization

model=Sequential()
model.add(Conv2D(256,(3,3),activation='relu',input_shape=(256,256,3))
)
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(2,2))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(4,activation='softmax'))

model.summary()
```

Model CNN ini terdiri dari empat lapisan konvolusi dengan kernel 3x3 dan jumlah filter bertahap 256, 128, 128, dan 64, masing-masing diikuti oleh *MaxPooling* 2x2 untuk mereduksi dimensi fitur. Setelah fitur diubah menjadi bentuk 1 dimensi melalui lapisan *Flatten*, lapisan *Dense* dengan 32 neuron dan aktivasi ReLU memprosesnya sebelum masuk ke lapisan output dengan 4 neuron dan aktivasi Softmax untuk klasifikasi multi-kelas. Model ini dirancang untuk menerima input gambar dengan dimensi (256, 256, 3). Hasil pembuatan model CNN berada di bawah berikut :

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 256)	7,168
max_pooling2d (MaxPooling2D)	(None, 127, 127, 256)	0
conv2d_1 (Conv2D)	(None, 125, 125, 128)	295,040
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 128)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	147,584
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0

conv2d_3 (Conv2D)	(None, 28, 28, 64)	73,792
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 32)	401,440
dense_1 (Dense)	(None, 4)	132

Total params: 925,156 (3.53 MB)
Trainable params: 925,156 (3.53 MB)
Non-trainable params: 0 (0.00 B)

4.4 Training Model

Sebelum melakukan pelatihan, diperlukan beberapa pengaturan agar model yang dihasilkan optimal dan proses pelatihan lebih efisien. Pengaturan tersebut dapat dilihat pada source code berikut ini:

```
from tensorflow.keras.optimizers import Adam
model.compile(
    optimizer = Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
  )
```

Kode ini mengatur optimasi model dengan *Adam optimizer*, menggunakan *categorical crossentropy* sebagai fungsi *loss* untuk klasifikasi multi-kelas, dan mengevaluasi kinerja model menggunakan metrik akurasi.

a. *optimizer=Adam(learning_rate=0.001)*: Adam adalah algoritma optimasi yang adaptif dan efisien, cocok untuk memperbarui bobot model selama pelatihan. Parameter *learning_rate=0.001* mengontrol kecepatan pembelajaran untuk mencapai hasil optimal.
b. *loss='categorical_crossentropy'* = Fungsi *loss* ini digunakan untuk masalah klasifikasi multi-kelas dengan *one-hot encoding*.
c. *metrics=['accuracy']* = Metrik akurasi digunakan untuk mengukur seberapa sering prediksi model benar dibandingkan dengan label asli.

Setelah selesai melakukan pengaturan, langkah berikutnya adalah melatih model CNN. Berikut adalah *source code* untuk proses pelatihan:

```
history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=50)
```

Proses pelatihan akan menghasilkan *history* yang berisi metrik performa model seperti *loss* dan *accuracy* untuk data latih dan validasi.

a. *train_generator* = *Generator* yang menyediakan data latih dalam bentuk *batch* secara bertahap. Biasanya digunakan dengan *data augmentation* untuk efisiensi memori.
b. *epochs=50* = Model akan dilatih selama 50 epoch, di mana satu *epoch* adalah satu siklus penuh iterasi terhadap seluruh data latih.
c. *validation_data=validation_generator* = Data validasi yang juga disediakan oleh generator untuk mengevaluasi performa model pada setiap *epoch*.

Hasil dari proses training dapat dilihat pada gambar dibawah ini

```
Epoch 45/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 71s 960ms/step - accuracy: 0.9121 - loss: 0.2210 - val_accuracy: 0.8685 - val_loss: 0.3678
Epoch 46/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 82s 957ms/step - accuracy: 0.9185 - loss: 0.2500 - val_accuracy: 0.8944 - val_loss: 0.2924
Epoch 47/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 83s 977ms/step - accuracy: 0.9348 - loss: 0.1689 - val_accuracy: 0.9116 - val_loss: 0.2951
Epoch 48/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 59s 933ms/step - accuracy: 0.9331 - loss: 0.1781 - val_accuracy: 0.8944 - val_loss: 0.3087
Epoch 49/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9216 - loss: 0.2021 - val_accuracy: 0.9116 - val_loss: 0.2910
Epoch 50/50
58/58 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9233 - loss: 0.1928 - val_accuracy: 0.9267 - val_loss: 0.2278
```

Hasil akhir dari proses pelatihan model CNN di *epoch* 50 menujukkan keakurasian diatas 90%. Selain itu, peneliti juga telah melakukan beberapa proses pelatihan (*training*) dengan variasi parameter tertentu. Hasil dari pelatihan tersebut akan ditampilkan dalam bentuk tabel di bawah ini.

Epoch	Training Accuracy	Training Loss	Validation Accuracy	Validation Loss	Learning Rate	Deskripsi
50	93.67%	0.1596	93.99%	0.2625	0.001	Akurasi validasi tertinggi (93.99%) dengan gap kecil antara akurasi training dan validasi. Cocok untuk model dengan generalisasi baik.
60	93.76%	0.1494	90.63%	0.2571	0.001	Akurasi validasi turun signifikan, menunjukkan kemungkinan overfitting meskipun training loss kecil.
50	92.01%	0.2234	92.54%	0.2569	0.0001	Akurasi stabil dengan selisih kecil antara training dan validation, menunjukkan keseimbangan model.
60	91.48%	0.2092	92.25%	0.2587	0.0001	Model cukup stabil, tetapi training accuracy lebih rendah dibandingkan opsi lainnya.

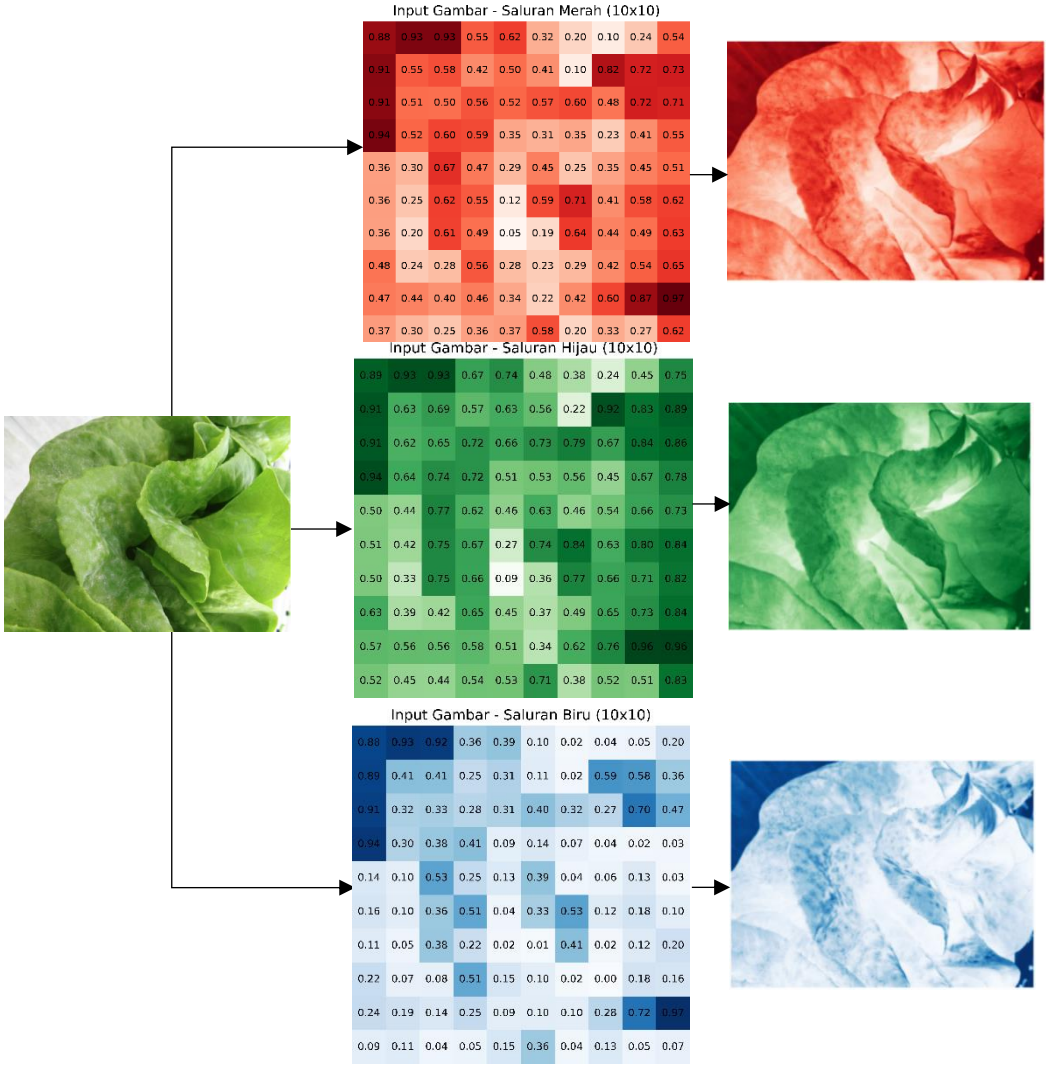
Tabel 4.3 Tabel hasil *training* model

Berdasarkan hasil pengujian, model dengan konfigurasi *epoch* 50 dan *learning rate* 0.001 menunjukkan akurasi validasi tertinggi sebesar 93.99% dengan gap yang kecil antara akurasi training (93.67%) dan validasi, serta loss yang seimbang. Hal ini mengindikasikan bahwa model memiliki kemampuan generalisasi yang baik tanpa tanda-tanda *overfitting*. Dibandingkan dengan konfigurasi lainnya, pilihan ini memberikan kinerja yang optimal untuk akurasi validasi sekaligus menjaga stabilitas model, sehingga direkomendasikan untuk digunakan.

Selama proses pelatihan model, gambar akan melalui empat lapisan CNN berupa *input layer*, *convolution layer*, *pooling layer* dan *fully connected layer*, seperti yang ditunjukkan dibawah ini.

4.4.1 *Input Layer*

Gambar daun selada akan dimasukkan ke lapisan awal. Di lapisan ini, gambar tersebut diolah menjadi matriks tiga dimensi yang terdiri dari panjang, lebar, dan saluran warna (RGB). Contoh berikut menggambarkan hasil input dari gambar tersebut.

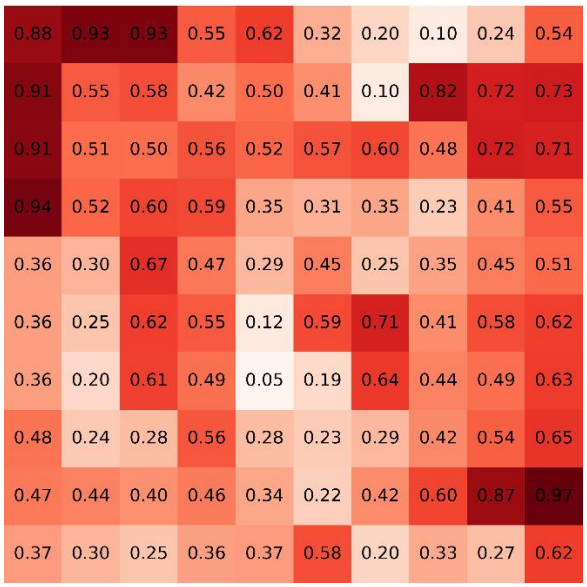


Gambar 4.24 hasil *input layer*

Pada gambar diatas, terdapat proses *input layer*, gambar diekstrak mengubahnya menjadi tiga warna: merah, hijau, dan biru.

4.4.2 Convolution Layer

Pada lapisan ini, sebuah input akan dimasukkan ke dalam *convolution layer,* di mana nilai RGB dari setiap piksel pada citra akan diekstraksi. Proses ini menghasilkan output berupa *feature map*. untuk mengolah gambar pada *convolution layer*, digunakan filter. Sebagai ilustrasi, dapat dilakukan perhitungan yang dilakukan melalui saluran merah yang diambil dari *input layer.*



Gambar 4.25 *red channel*

Pada tahap ini, dilakukan perhitungan untuk menghasilkan peta fitur dengan bantuan filter berukuran 3x3 yang memiliki nilai acak, yang terlihat seperti gambar berikut.

1	0	-1
1	0	-1
1	0	-1

Gambar 4.26 contoh filter

Perhitungan untuk menentukan nilai sebuah *feature map* gambar dimulai dengan menggunakan koordinat awal matriks (1,1). Oleh karena itu, perhitungan dilakukan berdasarkan submatriks yang sesuai dengan posisi tersebut sebagai berikut

$$y[1,1] = (0.88 \cdot 1) + (0.93 \cdot 0) + (0.93 \cdot -1) + (0.91 \cdot 1) + (0.55 \cdot 0) + (0.58 \cdot -1) + (0.91 \cdot 1) + (0.51 \cdot 0) + (0.50 \cdot -1)$$

$$y[1,1] = (0.88) + (0) + (-0.93) + (0.91) + (0) + (-0.58) + (0.91) + (0) + (-0.50)$$

$$y[1,1] = 0.88 - 0.93 + 0.91 - 0.58 + 0.91 - 0.50$$

$$y[1,1] = 0.69$$

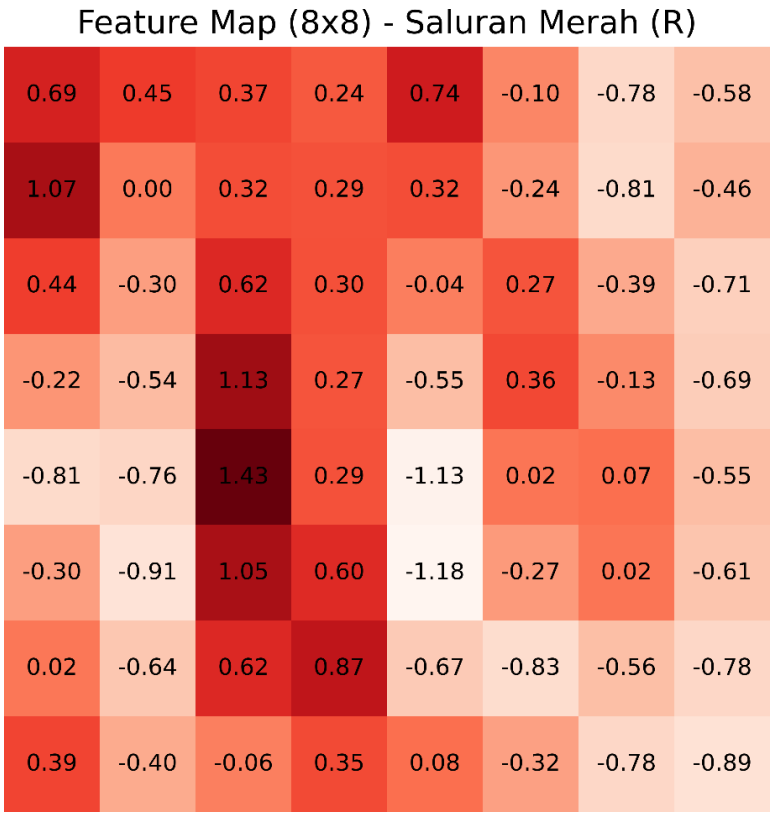
Rumus untuk menghitung ukuran feature map setelah konvolusi adalah:

- *Ukuran Feature Map = (Ukuran Gambar−Ukuran Kernel) + 1*

Dengan menggantikan nilai-nilai yang diberikan:

- *Ukuran Feature Map = (10−3) + 1 = 8*

Artinya, gambar *input* yang berukuran 10x10 akan menghasilkan *feature map* dengan ukuran 8x8. Ini terjadi karena filter 3x3 akan dipindahkan sepanjang gambar 10x10, dan dengan *stride* 1, filter dapat diterapkan 8 kali baik secara horizontal maupun vertikal. Sehingga hasil yang diberikan pada koordinat [1,1] adalah 0.69 dan juga ukuran *feature map* adalah 8



Gambar 4. 27 *feature map* merah

Setelah proses perhitungan pada *convolution layer* selesai, langkah berikutnya adalah menerapkan fungsi aktivasi. Dalam penelitian ini, fungsi aktivasi yang digunakan pada lapisan konvolusi adalah ReLU. Fungsi aktivasi ReLU (*Rectified Linear Unit*) bekerja dengan cara mengubah semua nilai negatif menjadi nol dan mempertahankan nilai positif sebagaimana adanya. Hal ini dilakukan secara elemen demi elemen pada matriks input (*feature map*). Rumus matematis untuk ReLU adalah:

$f(x) = max\ (0,x)$

Di mana:

- x adalah nilai elemen input pada matriks.

- Jika $x>0$, maka $f(x)=x$
- Jika $x\leq 0$, maka $f(x)=0$

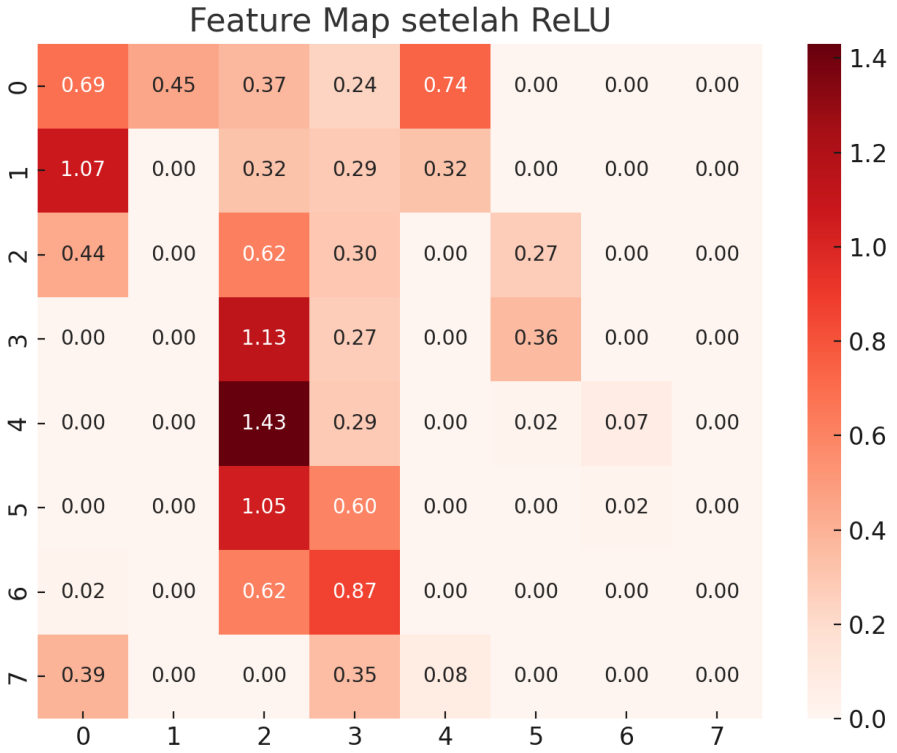
Sebagai contoh, jika diterapkan pada elemen x = −0.78 maka:

- $f(-0.78)=max(0,-0.78)=0$

Sedangkan untuk elemen x=0.74, maka:

- $f(0.74)=max(0,0.74)=0.74$

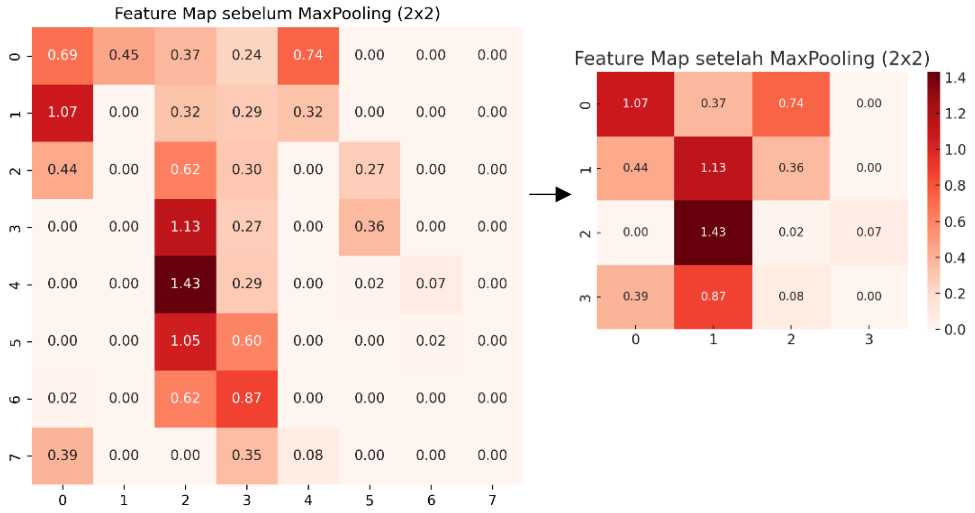
Proses ini dilakukan secara elemen per elemen hingga menghasilkan matriks baru yang hanya berisi nilai nol dan positif.



Gambar 4.28 *feature map* setelah ReLu

4.4.3 Pooling Layer

Pada tahap ini, tujuan utamanya adalah mengurangi jumlah parameter dan kompleksitas perhitungan dalam *neural network*. Dalam penelitian ini, *pooling layer* yang digunakan adalah *MaxPooling*, yang berfungsi untuk mengambil nilai maksimum dari hasil *convolution*. Ukuran matriks untuk *Maxpooling* yang diterapkan adalah 2x2, dengan *stride* sebesar 2. Proses ini dilakukan pada *feature map* yang dihasilkan oleh *convolution layer*, seperti yang terlihat pada gambar di bawah ini, sebelum diteruskan ke langkah berikutnya.

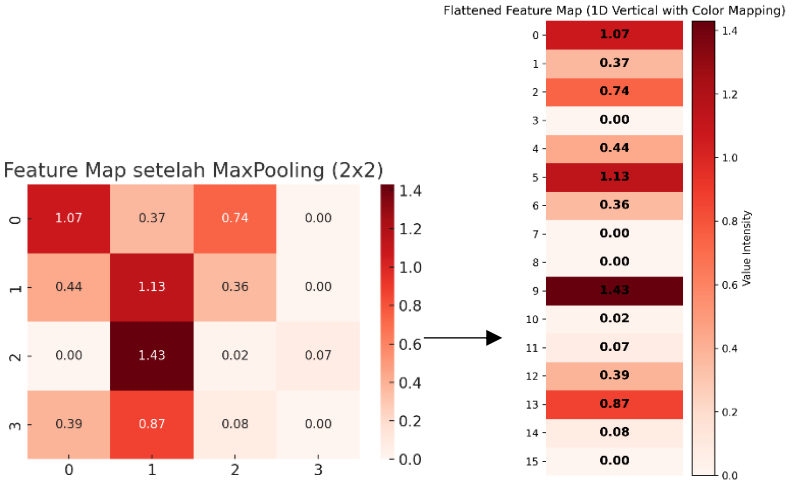


Gambar 4.29 Proses *input* dan *output pooling layer*

Pada gambar di atas, input berupa *feature map* yang dihasilkan dari konvolusi, sementara *output* merupakan hasil *dari pooling layer*. Proses ini akan menghasilkan nilai output yang merupakan nilai terbesar dari setiap blok 2x2 pada *input*. Dalam kasus ini, nilai terbesar dalam setiap blok 2x2 akan dipilih dan dimasukkan ke dalam *output* dari *pooling layer*.

4.4.4 Flatten Layer

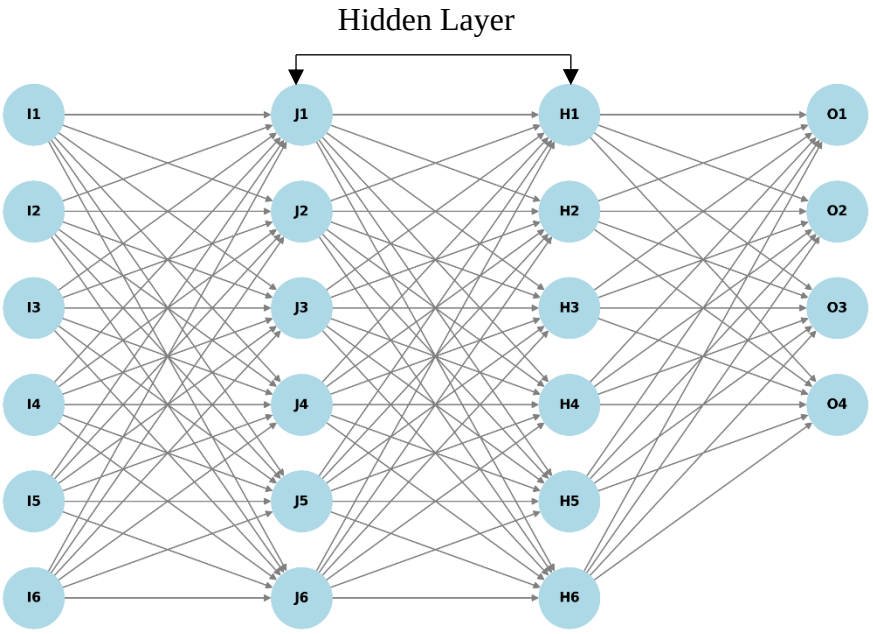
Flatten adalah proses mengubah matriks menjadi vektor atau matriks satu dimensi yang diperoleh dari output lapisan pooling. Proses ini dapat dilihat pada gambar berikut.



Gambar 4.30 Proses Flatten Layer

Gambar 4.30 menunjukkan proses flatten, di mana matriks berukuran 4x4 dikonversi menjadi vektor berukuran 16x1. Hasil dari tahap ini adalah vektor satu dimensi yang akan digunakan sebagai input untuk lapisan fully connected.

4.4.5 Fully Connected Layer



Gambar 4.31 Ilustrasi Dense + Softmax Layer

Gambar di atas menunjukkan proses Dense + Softmax, di mana vektor satu dimensi berjumlah 16 digunakan sebagai input. Setiap nilai dalam vektor tersebut akan dihitung melalui hidden layer yang juga memiliki 16 neuron. Berikut adalah proses perhitungannya.

$$\sum_{I=1}^{N} \mathrm{I}i \; x \; Vij = Ji$$

$$\sum_{I=1}^{N} \mathrm{I}i \; x \; Wij = Hi$$

$$\sum_{I=1}^{N} \mathrm{I}i \; x \; Xij = Oi$$

I1, I2, I3, I4, I5, I6, hingga I16 merupakan nilai output dari proses Flatten, yang kemudian diproses melalui Fully Connected Layer. Setiap nilai akan dihitung dalam hidden layer dengan bobot yang berbeda untuk setiap koneksi. Nilai bobot dapat ditentukan secara bebas sesuai kebutuhan. Berikut adalah perhitungan pada setiap hidden layer.

J_1=(1.07×0.2)+(0.37×0.2)+(0.74×0.2)+(0.00×0.2)+(0.44×0.2)+(1.13×0.2)+(0.36×0.2)+(0.00×0.2)+(0.00×0.2)+(1.43×0.2)+(0.02×0.2)+(0.07×0.2)+(0.39×0.2)+(0.87×0.2)+(0.08×0.2)+(0.00×0.2)

Lalu

J_1=(0.214)+(0.074)+(0.148)+(0.000)+(0.088)+(0.226)+(0.072)+(0.000)+(0.000)+(0.286)+(0.004)+(0.014)+(0.078)+(0.174)+(0.016)+(0.000)J_1 = (0.214) + (0.074) + (0.148) + (0.000) + (0.088) + (0.226) + (0.072) + (0.000) + (0.000) + (0.286) + (0.004) + (0.014) + (0.078) + (0.174) + (0.016) + (0.000)

J1=(0.214)+(0.074)+(0.148)+(0.000)+(0.088)+(0.226)+(0.072)+(0.000)+(0.000)+(0.286)+(0.004)+(0.014)+(0.078)+(0.174)+(0.016)+(0.000)

Lalu jumlahkan semuanya:

J_1=1.394

Sehingga dari perhitungan diatas didapatkan J_1 sampai J_{16} berupa tabel dibawah ini

Indeks J	Hasil J	Nilai Acak yang digunakan
J_1	1.3940	(1.07×0.2),(0.37×0.2),…
J_2	0,6970	(1.07×0.1),(0. 37×0.1),…
J_3	2.0910	(1.07×0.3),(0. 37×0.3),…
J_4	3.4850	(1.07×0.5),(0. 37×0.5),…
J_5	2.7880	(1.07×0.4),(0. 37×0.4),…
J_6	4.8790	(1.07×0.7),(0. 37×0.7),…
J_7	4.1820	(1.07×0.6),(0. 37×0.6),…
J_8	6.2730	(1.07×0.9),(0. 37×0.9),…
J_9	5.5760	(1.07×0.8),(0. 37×0.8),…
J_{10}	0.7667	(1.07×0.11),(0. 37×0.11),…
J_{11}	0.6970	(1.07×0.10),(0. 37×0.10),…
J_{12}	0.9061	(1.07×0.13),(0. 37×0.13),…
J_{13}	0.8364	(1.07×0.12),(0. 37×0.12),…
J_{14}	1.0455	(1.07×0.15),(0. 37×0.15),…
J_{15}	0.9758	(1.07×0.14),(0. 37×0.14),…
J_{16}	1.1152	(1.07×0.16),(0. 37×0.16),…

Tabel 4.4 Hasil J_1 sampai J_{16}

Nilai J_1 hingga J_{16} yang telah dihitung akan dikalikan dengan bobot baru yang berbeda untuk masing-masing koneksi, guna memperoleh nilai H_1

hingga H_{16} sebagai output dari hidden layer berikutnya.

H_1= (1. 3940×0.1)+(0. 6970×0.1)+(2. 0910×0.1)+(3.4850×0.1)(2. 7880×0.1)+(4. 8790×0.1)+(4. 1820×0.1) +(6. 2730×0.1)+(5. 5760×0.1)+(0. 7667×0.1)+(0. 6970×0.1)+(0.9061×0.1)+(0.8364×0.1)+(1. 0455×0.1)+(0. 9758×0.1)+(1. 1152×0.1) = 3.7708

Sehingga dari perhitungan diatas didapatkan H_1 sampai H_{16} berupa tabel dibawah ini

Indeks H	Hasil H	Bobot Baru
H_1	3.7708	0.1
H_2	7.5415	0.2
H_3	11.3123	0.3
H_4	15.0831	0.4
H_5	18.8538	0.5
H_6	22.6246	0.6
H_7	26.3954	0.7
H_8	30.1662	0.8
H_9	33.9369	0.9
H_{10}	3.7708	0.10
H_{11}	4.1478	0.11
H_{12}	4.5249	0.12
H_{13}	4.9020	0.13
H_{14}	5.2791	0.14
H_{15}	5.6562	0.15
H_{16}	6.0332	0.16

Tabel 4.5 Hasil H_1 sampai H_{16}

Setiap nilai output dari neuron H_1 hingga H_{16} akan dikalikan dengan bobot yang berbeda untuk menghasilkan nilai O_1 hingga O_4**,** yang mewakili kelas dalam dataset.

O_1= (3.7708×0. 4)+(7.5415×0. 4)+(11.3123×0. 4)+(15.0831×0. 4)+(18.8538×0.4)+(22.6246×0. 4)+(26.3954× 0. 4)+(30.1662×0. 4) +(33.9369×0. 4)+(3.7708×0. 4)+(4.1478×0. 4)+(4.5249×0. 4)+(4.9020×0. 4)+(5.2791×0. 4)+(5.6562×0. 4)+(6.0332×0. 4)

O_1 = 81.59944

Sehingga perhitungan dari O_1 sampai O_4 adalah sebagai berikut

Indeks O	Hasil O	Bobot Baru
O_1	81.5994	0.4
O_2	101.9993	0.5
O_3	40.79972	0.2
O_4	61.19958	0.3

Tabel 4.6 Hasil O_1 sampai O_4

Langkah selanjutnya adalah perhitungan softmax. Softmax digunakan untuk mengubah sekumpulan nilai menjadi probabilitas. Rumus untuk setiap nilai O_i dalam himpunan $\{O_1,O_2,O_3,...,O_n\}$ adalah:

$$S(O_i) = \frac{e^{O_i}}{\sum_{j=1}^{n} e^{O_j}}$$

Di Mana:

- $S(O_i)$ adalah nilai softmax untuk O_i
- e^{O_i} adalah eksponensial dari nilai O_i
- $\sum_{j=1}^{n} e^{O_i}$ adalah jumlah dari semua eksponensial nilai O_j

Dan dimasukkan nilai dengan diketahui

- O1 = 81.5994
- O2 = 101.9993
- O3 = 40.79972
- O4 = 61.19958

Maka, dihitung eksponensial dari masing-masing nilai :

$$e^{O_1}, \quad e^{O_2}, \quad e^{O_3}, \quad e^{O_4}$$

Lalu di jumlahkan semua hasil eksponensial

$$\sum e^{O_j} = e^{O_1} + e^{O_2} + e^{O_3} + e^{O_4}$$

Kemudian, di bagi masing-masing eksponensial dengan jumlah total tersebut

$$S(O_1) = \frac{e^{O_1}}{\sum e^{O_j}}, \quad S(O_2) = \frac{e^{O_2}}{\sum e^{O_j}}, \quad S(O_3) = \frac{e^{O_3}}{\sum e^{O_j}}, \quad S(O_4) = \frac{e^{O_4}}{\sum e^{O_j}}$$

Sehingga berdasarkan perhitungan sebelumnya :
$S(O_1) \approx 1.38 \times 10^{-9}$
$S(O_2) \approx 0.999999999$
$S(O_3) \approx 2.64 \times 10^{-27}$
$S(O_4) \approx 1.91 \times 10^{-18}$

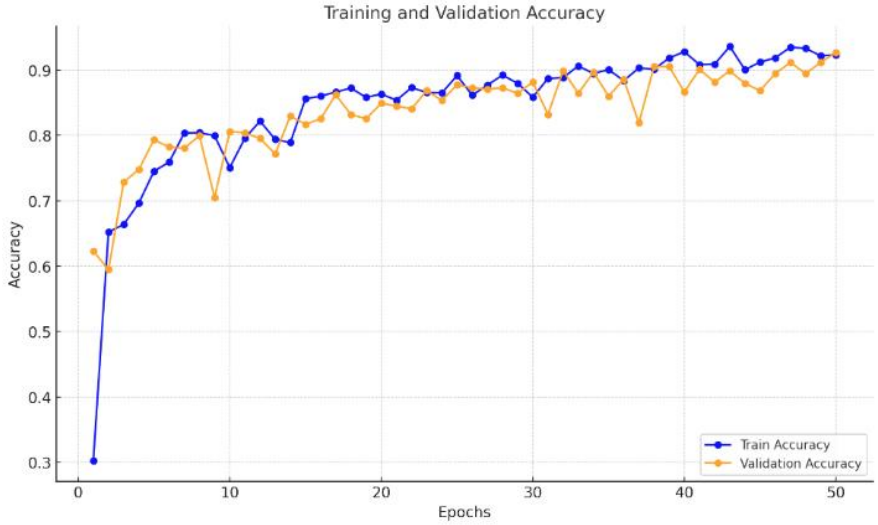
Dari sini, terlihat bahwa nilai softmax tertinggi adalah untuk O_2 dengan probabilitas mendekati 1 yang O_2 merupakan *Fungal.*

Kesimpulan:

- O_2 = 101.9993 memiliki probabilitas tertinggi, sehingga jika ini adalah data citra warna, maka warna yang terkait dengan nilai ini akan mendominasi hasil klasifikasi.
- Dalam kasus ini, kita bisa menyimpulkan bahwa citra warna merah memiliki nilai tertinggi jika O2 merepresentasikan warna merah.

4.5 Evaluation Model

Pada tahap evaluasi model, hasil dari proses pelatihan akan dianalisis untuk menilai tingkat akurasi dan kemampuan model dalam meminimalkan kesalahan deteksi. Akurasi dan loss yang diperoleh digunakan sebagai indikator untuk menentukan apakah pelatihan model telah berjalan dengan baik atau memerlukan perbaikan lebih lanjut.



Gambar 4.32 Grafik *train and validation accuraccy*

Grafik akurasi menunjukkan peningkatan kinerja model dari epoch awal hingga akhir pelatihan, dengan akurasi pelatihan mencapai lebih dari 92% dan akurasi validasi mencapai 93.67%. Hal ini menunjukkan model belajar dengan baik

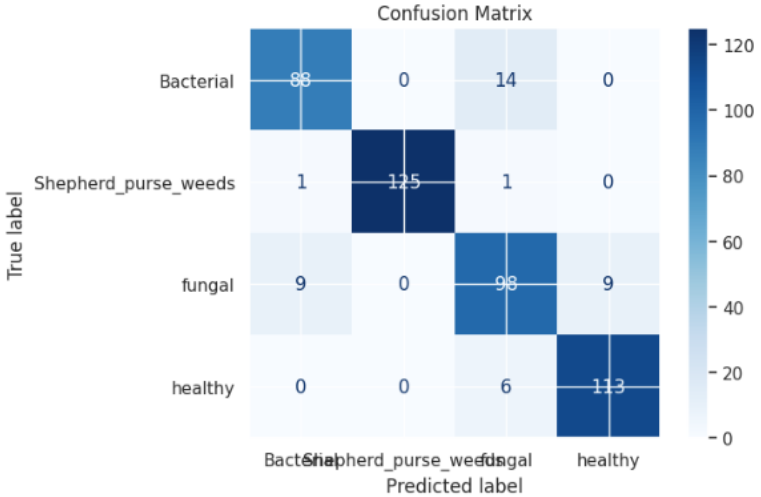
dan dapat menggeneralisasi data dengan baik.

Gambar 4.33 Grafik *train and validation loss*

Grafik loss menunjukkan penurunan loss pelatihan dan validasi yang signifikan, dengan nilai akhir loss validasi sekitar 0.2278. Tren yang serupa antara keduanya menunjukkan bahwa model tidak mengalami overfitting dan memiliki performa yang stabil.

4.6 Confusion Matrix

Dengan pembagian *split data* 0,2, 20% digunakan untuk data uji dan 80% untuk pelatihan. *Confusion matrix* pada data uji menunjukkan kinerja model dalam memprediksi label yang belum pernah dilihat sebelumnya. Matriks ini membantu menilai akurasi model dan mengidentifikasi masalah seperti *overfitting* atau *underfitting*, yang mencerminkan kemampuan generalisasi model. Berikut adalah hasil dari evaluasi matrix menggunakan *confusion matrix.*



Gambar 4.34 *Confusion matrix*

Pada gambar di atas dapat dilihat bahwa pada indeks ke-0 (*Bacterial*), model dapat mengklasifikasikan 88 data dengan benar dari total 102 data pada kategori ini. Pada indeks ke-1 (*Shepherd purse weeds*), model memprediksi 115 data dengan tepat dari total 117 data. Pada indeks ke-2 (*Fungal*), model hanya dapat mengklasifikasikan 98 data dengan benar dari total 116 data. Sementara itu, pada

indeks ke-3 (*Healthy*), model berhasil mengklasifikasikan 113 data dengan benar dari total 119 data. *Precission, recall*, dan *f1-score* dapat dilihat dari hasil *confusion matrix* di atas, yang dapat dilihat pada tabel di bawah ini.

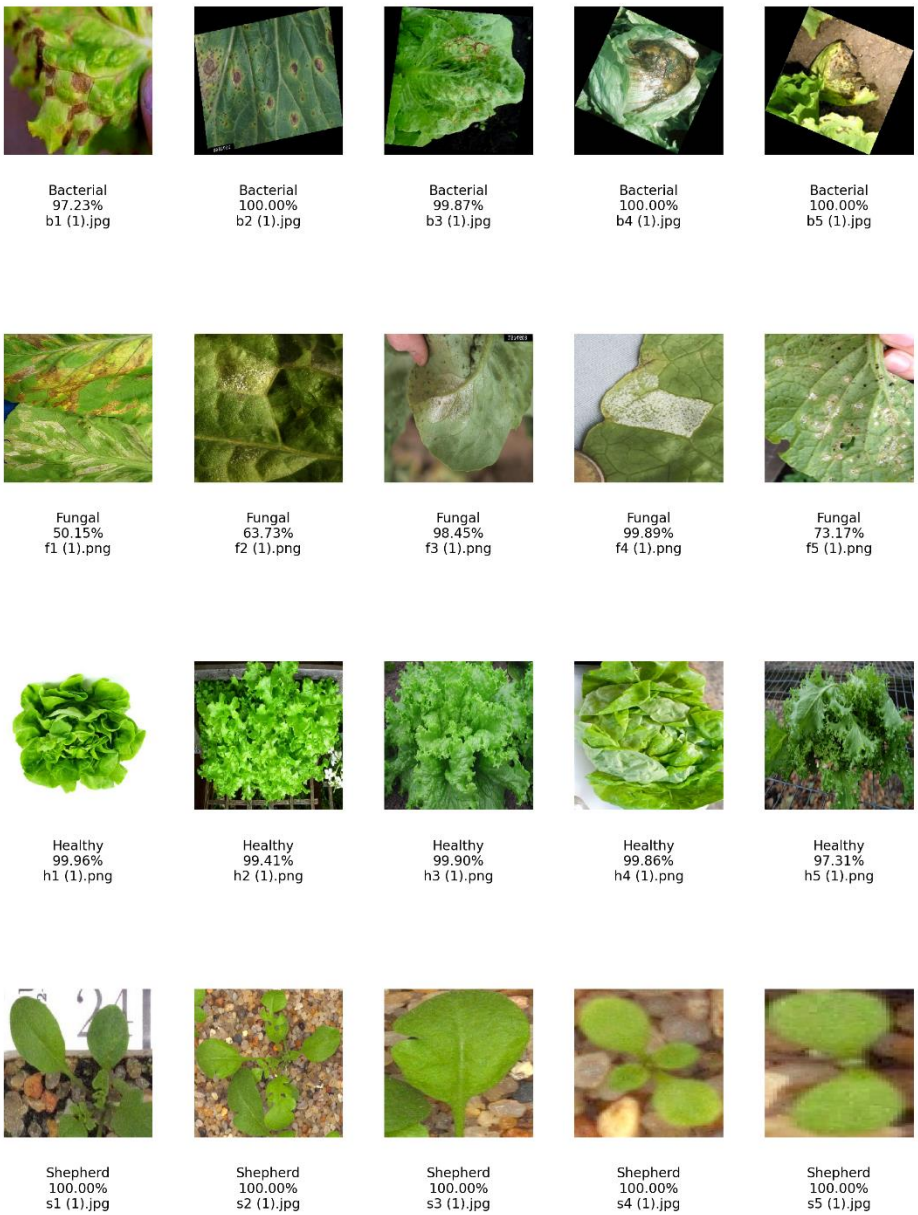
Classification Report	*Precission*	*Recall*	*F1-score*	*Support*
Bacterial	0.90	0.86	0.88	102
Fungal	0.82	0.84	0.83	116
Healthy	0.93	0.95	0.94	119
Shepherd purse weeds	1.00	0.98	0.99	127
Accuracy			0.91	464
Macro average	0.91	0.91	0.91	464
Weighted average	0.91	0.91	0.91	464

Tabel 4.7 Table *confusion matrix*

Kategori *Bacterial* memiliki precision 0,90 dan *f1-score* 88%, kategori *Fungal* lebih rendah dengan *f1-score* 83%. *Healthy* menunjukkan performa tinggi dengan *f1-score* 94%, sementara *Shepherd purse weeds* memiliki kinerja hampir sempurna dengan *f1-score* 99%.

4.7 Uji Model

Setelah proses training selesai dan model berhasil dibuat, tahap selanjutnya adalah melakukan pengujian. Pengujian ini bertujuan untuk mengevaluasi kemampuan model dalam memprediksi data dengan akurat. Berikut beberapa hasil uji dataset sebagai berikut:



Gambar 4.35 Uji model

Hasil pengujian model menggunakan 60 dataset gambar menunjukkan variasi akurasi yang berbeda-beda. Untuk penjelasan lebih rinci, dapat dilihat pada Tabel berikut

No	Prediction	Actual	Confidence
1	Bacterial	Bacterial	97.23%
2	Bacterial	Bacterial	100.00%
3	Bacterial	Bacterial	99.87%
4	Bacterial	Bacterial	100.00%
5	Bacterial	Bacterial	100.00%
6	Bacterial	Bacterial	96.28%
7	Bacterial	Bacterial	99.42%
8	Bacterial	Bacterial	100.00%
9	Bacterial	Bacterial	99.87%
10	Bacterial	Bacterial	100.00%
11	Bacterial	Bacterial	100.00%
12	Bacterial	Bacterial	100.00%
13	Bacterial	Bacterial	84.99%
14	Bacterial	Bacterial	67.86%
15	Bacterial	Bacterial	98.34%
16	Fungal	Fungal	50.15%
17	Fungal	Fungal	99.77%
18	Fungal	Fungal	98.92%
19	Fungal	Fungal	99.96%
20	Fungal	Fungal	98.64%
21	Fungal	Fungal	90.56%
22	Fungal	Fungal	59.85%
23	Fungal	Fungal	63.73%
24	Fungal	Fungal	98.45%
25	Fungal	Fungal	99.89%
26	Fungal	Fungal	73.17%
27	Fungal	Fungal	98.84%
28	Fungal	Fungal	72.02%
29	Fungal	Fungal	95.30%
30	Fungal	Fungal	98.18%

31	Healthy	Healthy	99.96%
32	Healthy	Healthy	99.62%
33	Healthy	Healthy	99.41%
34	Healthy	Healthy	99.97%
35	Healthy	Healthy	99.83%
36	Healthy	Healthy	99.99%
37	Healthy	Healthy	99.97%
38	Healthy	Healthy	99.41%
39	Healthy	Healthy	99.90%
40	Healthy	Healthy	99.86%
41	Healthy	Healthy	97.31%
42	Healthy	Healthy	99.99%
43	Healthy	Healthy	99.98%
44	Healthy	Healthy	99.77%
45	Healthy	Healthy	99.51%
46	Shepherd purse weeds	Shepherd purse weeds	100.00%
47	Shepherd purse weeds	Shepherd purse weeds	100.00%
48	Shepherd purse weeds	Shepherd purse weeds	100.00%
49	Shepherd purse weeds	Shepherd purse weeds	100.00%
50	Shepherd purse weeds	Shepherd purse weeds	100.00%
51	Shepherd purse weeds	Shepherd purse weeds	100.00%
52	Shepherd purse weeds	Shepherd purse weeds	100.00%
53	Shepherd purse weeds	Shepherd purse weeds	100.00%
54	Shepherd purse weeds	Shepherd purse weeds	100.00%
55	Shepherd purse weeds	Shepherd purse weeds	100.00%
56	Shepherd purse weeds	Shepherd purse weeds	100.00%
57	Shepherd purse weeds	Shepherd purse weeds	100.00%
58	Shepherd purse weeds	Shepherd purse weeds	100.00%
59	Shepherd purse weeds	Shepherd purse weeds	100.00%
60	Shepherd purse weeds	Shepherd purse weeds	100.00%

Tabel 4.8 Tabel uji model

4.8 Implementasi Deteksi Penyakit Pada Aplikasi

Dalam implementasi deteksi penyakit pada aplikasi, model terlebih dahulu diubah ke format TensorFlow Lite agar dapat disematkan ke dalam aplikasi.

```
import tensorflow as tf

model_path = "/content/model.keras"

tflite_model_path = "/content/model.tflite"

model = tf.keras.models.load_model(model_path)

converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open(tflite_model_path, 'wb') as f:
    f.write(tflite_model)
```

Sebelum memulai pembuatan aplikasi, peneliti terlebih dahulu melakukan perancangan, yang mencakup diagram aktivitas, diagram urutan, dan desain tampilan, yang dapat ditemukan di bab 3, sub bab 3.3. Tujuan dari implementasi deteksi penyakit ini adalah untuk menguji ketepatan model dalam mengklasifikasikan berbagai jenis penyakit pada tanaman selada. Selain itu, aplikasi yang bertujuan untuk mendeteksi penyakit tanaman ini membutuhkan perangkat keras yang tercantum dalam tabel berikut.

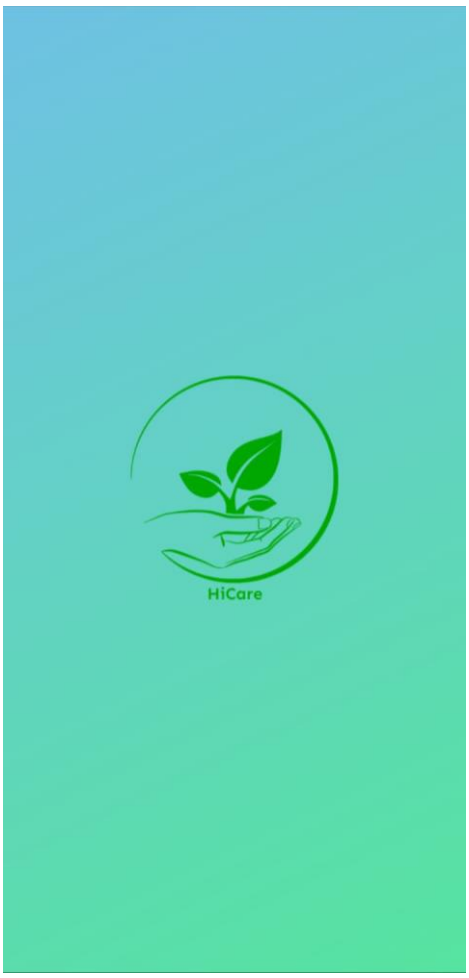
Spesifikasi PC	
Prosesor	Ryzen 5 5500U
RAM	16 GB
SSD	512 GB
OS	Windows 10 2H22

Spesifikasi Smartphone	
Prosesor	Mediatek Dimensity 1100 (6 nm) Octa-core (4x2.6 GHz Cortex-A78 & 4x2.0 GHz Cortex-A55)
RAM	8 GB
ROM	512 GB
Camera	64 MP, f/1.8, 26mm (wide), 1/1.97", 0.7µm, PDAF 8 MP, f/2.2, 120˚ (ultrawide), 1/4.0", 1.12µm 2 MP, f/2.4, (macro)

4.9 Tampilan Aplikasi

4.9.1 Tampilan Splash Screen

Tampilan splash screen adalah layar awal yang muncul saat aplikasi dibuka, menampilkan nama dan logo aplikasi. Layar ini ditampilkan selama 3 detik sebelum melanjutkan ke tampilan berikutnya. Contoh halaman splash screen dapat dilihat pada Gambar dibawah



Gambar 4.36 Tampilan *splashscreen* aplikasi

4.9.2 Tampilan Home

Tampilan Home aplikasi dirancang sebagai pusat navigasi utama dengan antarmuka yang sederhana. Pada halaman ini, pengguna dapat mengakses tiga fitur utama, yaitu:

1. Identifikasi Penyakit

Tombol ini berfungsi untuk mengakses galeri atau kamera untuk memungkinkan pengguna untuk menganalisis kondisi tanaman melalui gambar

2. Chat dengan AI

Fitur ini membuat user untuk dapat berkonsultasi dengan AI yang peneliti buat

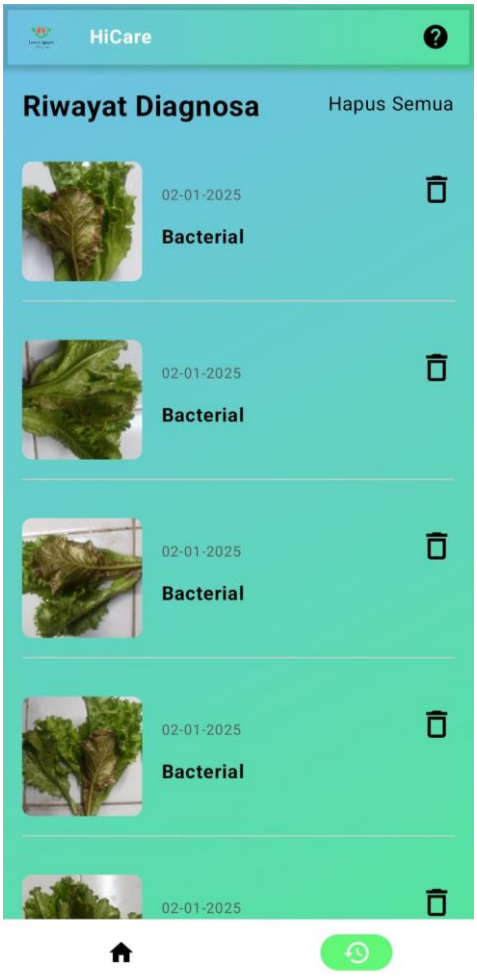
3. Detail Penyakit

Detail Penyakit memberikan informasi lengkap mengenai berbagai jenis penyakit tanaman, termasuk gejala, penyebab, dan langkah pencegahan.

Gambar 4.37 Tampilan home aplikasi

4.9.3 Tampilan History

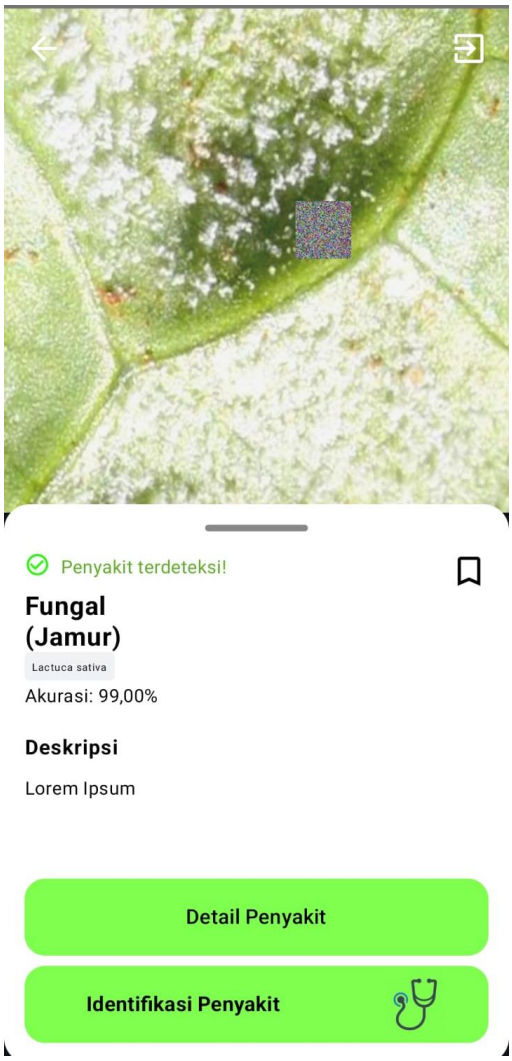
Tampilan History dirancang untuk membantu pengguna melacak riwayat identifikasi penyakit tanaman yang telah dilakukan. Halaman ini menampilkan daftar hasil analisis lengkap dengan tanggal, jenis penyakit yang terdeteksi, dan gambar tanaman yang diunggah. Dengan antarmuka yang rapi, pengguna dapat dengan mudah mengakses informasi sebelumnya untuk referensi atau keperluan dokumentasi.



Gambar 4.38 Tampilan history aplikasi

4.9.4 Tampilan Hasil Deteksi

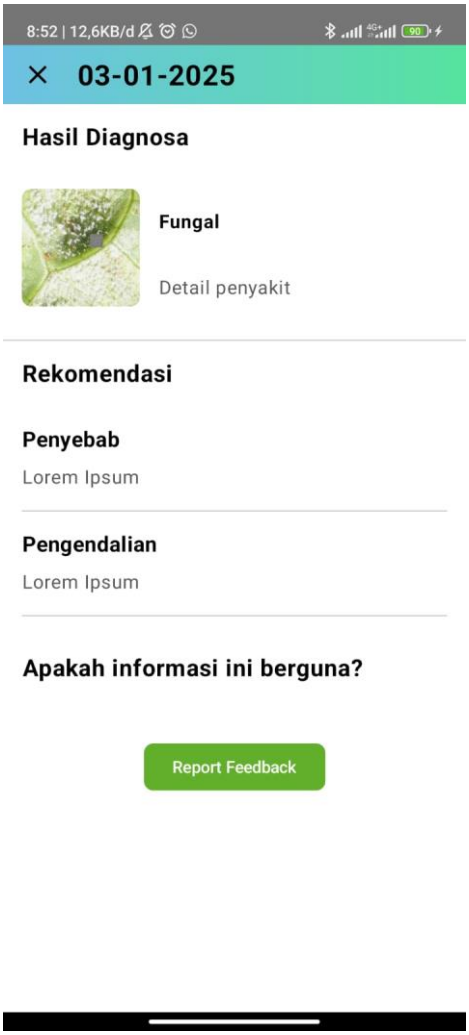
Tampilan Hasil Deteksi dirancang untuk menampilkan informasi lengkap setelah proses identifikasi penyakit tanaman selesai. Halaman ini mencakup gambar tanaman yang dianalisis, jenis penyakit yang terdeteksi, dan tingkat akurasi prediksi



Gambar 4.39 Tampilan hasil deteksi penyakit

4.9.5 Tampilan Detail Penyakit

Detail penyakit ini mencakup informasi tentang gejala, penyebab, dan cara penanganannya.



Gambar 4.40 Tampilan detail penyakit

4.10 Uji Coba

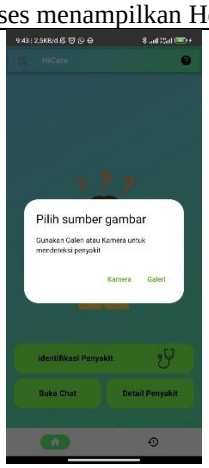
4.10.1 Pengujian aplikasi

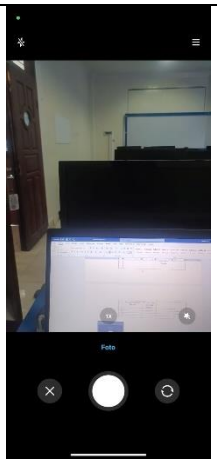
Pengujian aplikasi dilakukan dengan menginstal aplikasi dan menjalankannya pada perangkat dengan sistem operasi Android. Pengujian dilakukan menggunakan perangkat smartphone dengan spesifikasi sebagai berikut:

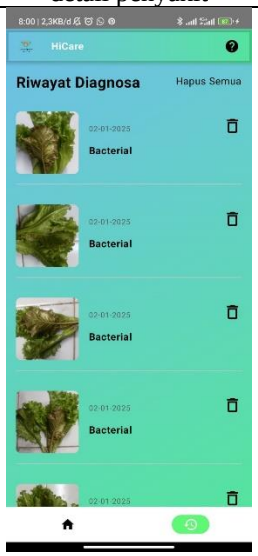
- Nama Smartphone : POCO X3 GT

-Versi Android : Android 13

- Resolusi Kamera : 64MP

No	Uji Coba	Deskripsi	Hasil yang diharapkan	Status
1	Pengguna masuk ke aplikasi	Tampilan Home akan muncul	 Sukses menampilkan Home	Sukses
2	Pengguna menekan tombol identifikasi penyakit	Tampilan opsi pilih galeri atau kamera muncul	 Sukses menampilkan opsi Ambil Gambar	Sukses

3	Pengguna menekan tombol kamera.	Program akan beralih ke Kamera	 Sistem sukses membuka kamera	Sukses
4	Pengguna menekan galeri	Program akan membuka galeri	 Sistem sukses membuka galeri	Sukses
5	Pengguna berhasil memilih gambar baik dari galeri maupun kamera.	Sistem akan memprediksi input berupa gambar dan memberikan hasil kepada pengguna berupa hasil deteksi dan akurasi	 Sistem berhasil memprediksi gambar dan menyampaikan hasilnya kepada pengguna.	Sukses

6	Pengguna menekan tombol detail penyakit	Sistem menampilkan rincian penyakit sesuai dengan prediksi dan memberikan solusi yang relevan.	 Sistem sukses menampilkan detail penyakit	sukses
7	Pengguna berpindah ke halaman history	Tampilan history muncul beserta hasil lampau	 Sistem sukses menampilkan riwayat deteksi lampau	Sukses
8	Pengguna membuka chat AI	Tampilan Chat AI muncul	 Sistem sukses menampilkan dan menjalankan chat AI	Sukses

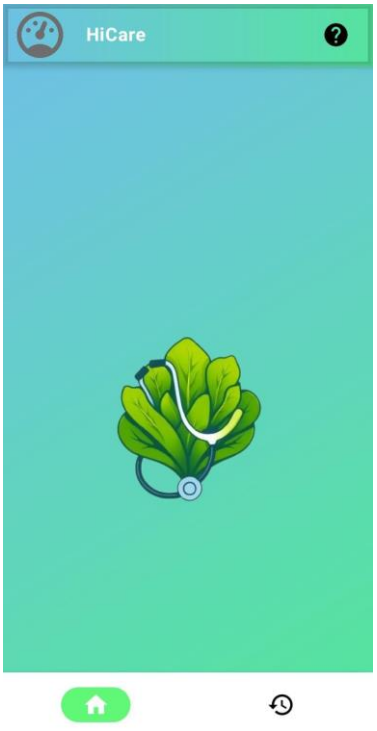
Tabel 4.9 Tabel pengujian aplikasi

Selain itu, peneliti melakukan uji aplikasi dengan menggunakan Android versi 10 dan 11 juga. Sehingga uji aplikasi didapatkan dengan hasil sebagai berikut

Uji Aplikasi	**Android 10**	**Android 11**	**Android 13**
Home	✗	✓	✓
Hasil Deteksi	✗	✓	✓
Detail Penyakit	✗	✓	✓
Riwayat	✗	✓	✓
Chat AI	✗	✓	✓

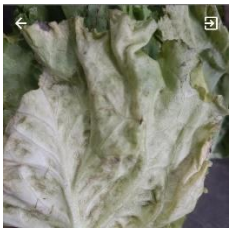
Tabel 4.10 Uji Versi Android

Pada uji aplikasi ini, untuk android 10 mengalami bug seperti tidak munculnya button intent dan lainnya. namun pada android 11 dan android 13 tidak mengalami masalah.



Gambar 4.41 Bug Home Android 11

4.10.2 Pengujian Deteksi Tanaman

No	Uji Coba	Gambar	Hasil Android 11	Hasil Android 13
1	Validasi Tanaman Sehat		Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 86,63%	Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 98,23%
2	Validasi penyakit Bacterial		Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 93,77%	Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 93,70%
3	Validasi penyakit Fungal		Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 99,13%	Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 99,99%
4	Validasi penyakit shepherd purse weeds		Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 100%	Tanaman berhasil diklasifisikasikan dengan benar Akurasi: 100%

Tabel 4.11 Tabel pengujian deteksi aplikasi

Uji coba dilakukan untuk menilai akurasi klasifikasi tanaman sehat dan penyakit pada Android 11 dan 13. Hasil menunjukkan aplikasi bekerja dengan baik pada kedua versi, dengan akurasi lebih tinggi di Android 13. Pada klasifikasi tanaman sehat, akurasi meningkat dari 86,63% (Android 11) menjadi 98,23% (Android 13). Untuk penyakit *bacterial*, akurasinya stabil di sekitar 93,7%. Penyakit *fungal* menunjukkan peningkatan dari 99,13% menjadi 99,99%. Sementara itu, klasifikasi *shepherd purse weeds* mencapai akurasi sempurna (100%) pada kedua versi. Secara keseluruhan, aplikasi lebih optimal di Android 13, terutama dalam mengidentifikasi tanaman sehat.

4.11 Pembahasan

Berdasarkan pengumpulan data penyakit selada yang dijelaskan pada Bab sebelumnya, ditemukan tiga jenis penyakit yang berasal dari hama maupun virus, yaitu bacterial, fungal, dan shepherd's purse weeds. Data ini diambil dari Kaggle untuk memastikan model dapat melakukan klasifikasi dengan akurasi yang baik.

Dataset yang digunakan terdiri dari 2320 gambar yang mencakup berbagai jenis penyakit. Data ini dibagi menjadi 80% untuk training dan 20% untuk testing. Penelitian ini mengembangkan model CNN dengan memanfaatkan TensorFlow yang akan diintegrasikan ke dalam aplikasi Android. Dari hasil training yang dijelaskan pada Sub Bab 4.4, model CNN mencapai akurasi sebesar 93.67%. Setelah model yang memadai berhasil dibuat, langkah berikutnya adalah mengimplementasikannya ke dalam aplikasi dengan menambahkan fitur deteksi tanaman. Hasil implementasi dapat diberikan sebagai berikut:

1. Aplikasi dapat mengidentifikasi penyakit selada dengan akurasi 93.67% dan dilengkapi fitur deteksi real-time melalui gambar yang diunggah atau diambil menggunakan kamera Android.
2. Disediakan fitur riwayat penyakit yang memungkinkan petani melacak perkembangan penyakit tanaman selada secara berkala.
3. Setiap penyakit selada yang berhasil diklasifikasikan akan disertai dengan solusi yang relevan.

Namun aplikasi ini memiliki berapa kelemahan yaitu :

1. Penggunaan label umum seperti "bacterial" dan "fungal" yang tidak terlalu spesifik dapat mengurangi keefektifan solusi dari penyakit yang spesifik, sehingga menyulitkan diagnosis dan penanganan yang tepat.
2. Aplikasi ini hanya tersedia di platform Android dan belum mendukung platform lain seperti iOS, sehingga membatasi aksesibilitas bagi pengguna iPhone atau iPad.
3. Aplikasi ini hanya mendukung Android 11 sampai 13 saja, sehingga penggunaan pada android 10 kebawah kemungkinan akan terjadi bug seperti yang sudah dijelaskan.