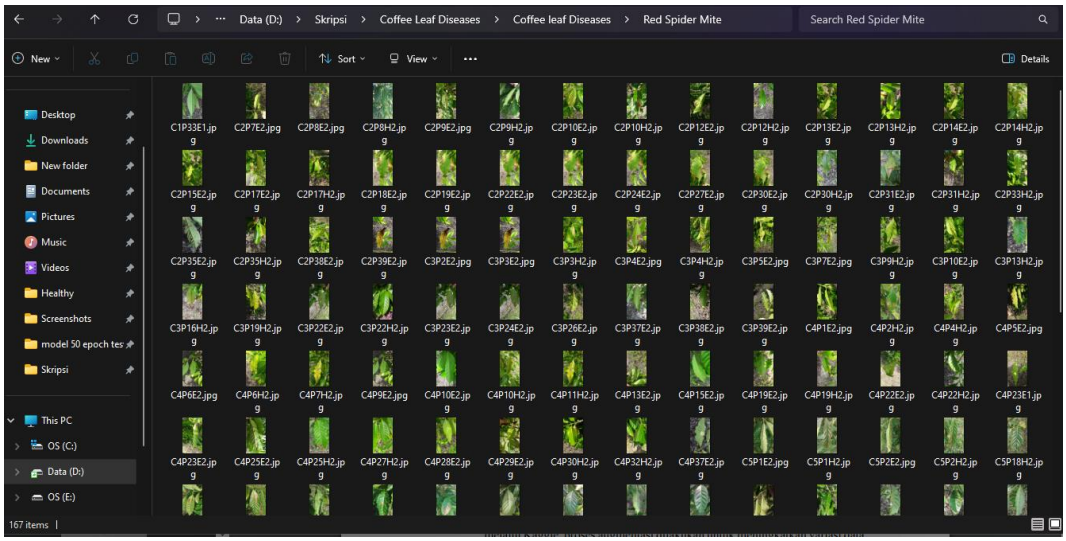
BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil

Hasil dari pengumpulan data dilakukan dengan mencari dataset terkait penyakit kopi tipe arabika yang mana didapat melalui platform Kaggle. Dataset ini berisi 3768 gambar yang terdiri dari 5 kelas, 1 kelas daun sehat dan 4 kelas daun yang terinfeksi penyakit. Dapat dilihat salah satu dari kelas penyakit pada gambar 4.1



Gambar 4.1 Hasil pengumpulan data penyakit kopi

4.2 *Labelling Data*

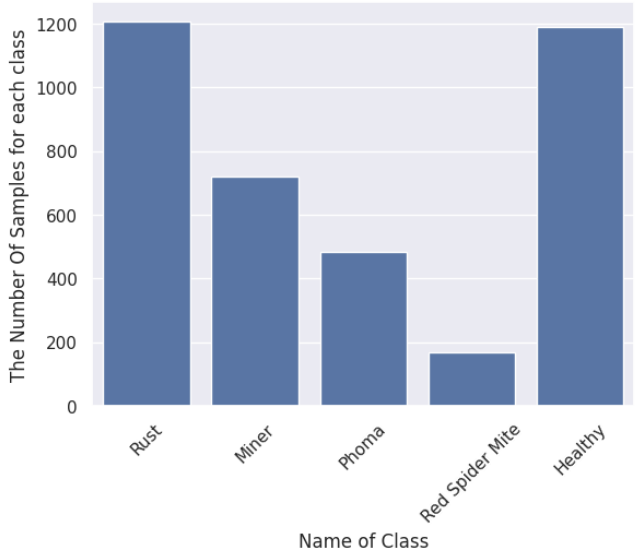
Pada proses pelabelan data yang mana bertujuan untuk memberikan nama atau identitas pada data agar dapat dikenali oleh model. Pelabelan data dilakukan untuk memberikan informasi terkait dataset dari 5 subfolder yang mana terdiri 5 kelas yaitu *healthy*, *phoma*, *red spider mite*, *miner*, dan *leaf rust*. Pelabelan berhubungan langsung dengan pembagian gambar secara keseluruhan, dengan perbandingan 80:20, 80% untuk data latih dan 20%

untuk data uji. Berikut ini kode pada *google colab* untuk proses pelabelan.

```
# Set the path to the dataset
dataset_path = '/content/drive/MyDrive/Coffee leaf Diseases'

# Initialize empty lists for storing the images and labels
images = []
labels = []

# Loop over the subfolders in the dataset
for subfolder in os.listdir(dataset_path):

    subfolder_path = os.path.join(dataset_path, subfolder)
    if not os.path.isdir(subfolder_path):
        continue

  # Loop over the images in the subfolder
    for image_filename in os.listdir(subfolder_path):
       # Load the image and store it in the images list
        image_path = os.path.join(subfolder_path, image_filename)
        images.append(image_path)

        # Store the label for the image in the labels list
        labels.append(subfolder)

 # Create a pandas DataFrame from the images and labels
df = pd.DataFrame({'image': images, 'label': labels})
df.head()
```

Gambar 4.2 code proses pelabelan data

Dan berikut ini merupakan gambar dari jumlah sample dataset serta nama-nama dari kelasnya.



Gambar 4.3 Jumlah *Sample Dataset*

4.3 *Preprocessing Data*

Pada tahap *Preprocessing Data* ini meliputi beberapa langkah yaitu pengumpulan data, pembuatan label pada citra, dan juga augmentasi citra gambar untuk seluruh data yang tersedia.

4.3.1 Augmentasi Data

Proses Augmentasi ini berfungsi untuk mengurangi *overfitting* yang terjadi pada dataset. Selain itu, proses augmentasi gambar ini dapat menambah jumlah data yang digunakan. Tetapi data yang telah dihasilkan hanya berguna didalam proses pelatihan model. Berikut ini merupakan kode yang digunakan untuk melakukan proses augmentasi gambar.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Set the image size and batch size
image_size = (256, 256)
batch_size = 32

# Create an ImageDataGenerator object with data augmentation options for image preproces
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'
)


# Create a generator for the training data
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
)
```

```
# Create a generator for the test data
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
)
```

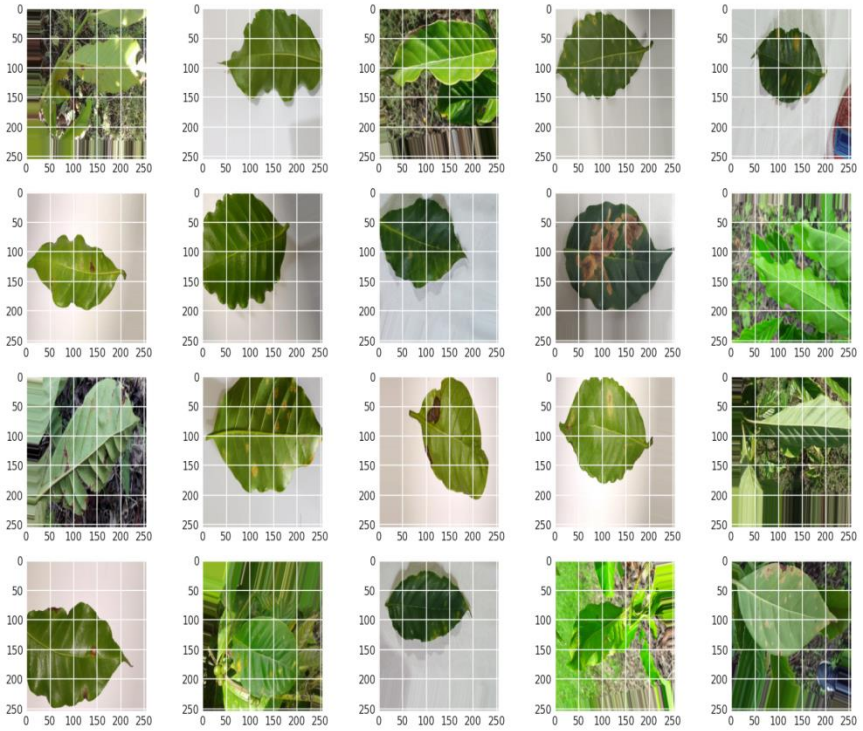
```
Found 3014 validated image filenames belonging to 5 classes.
Found 754 validated image filenames belonging to 5 classes.
```

Gambar 4.4 Proses Augmentasi Gambar

Pada gambar 4.4, fungsi *ImageDataGenerator* digunakan untuk melakukan augmentasi gambar, yang kemudian akan disimpan dalam variabel datagen. Berikut adalah parameter-parameter yang akan diterapkan untuk menghasilkan augmentasi gambar:

- *rescale* = 1.0/255. Ini berfungsi untuk mengubah ukuran pada gambar data *pixel* RGB (0-255) yang akan menjadi pada rentang angka (0-1) untuk memudahkan proses *train* data
- *rotation_range* = 20. Berfungsi untuk mengatur rentang rotasi gambar, yang mana gambar dapat diputar secara acak sampai 20 derajat baik searah atau berlawanan arah jarum jam.
- *width_shift_range* = 0.2. Berfungsi menggeser gambar secara *horizontal* (kekanan atau kiri) mencapai 20% dari lembar gambar.
- *height_shift_range* = 0.2. Berfungsi menggeser gambar secara *vertical* (keatas atau kebawah) mencapai 20% dari lembar gambar.
- *shear_range* = 0.2. Berfungsi untuk melakukan transformasi *shearing* pada gambar, yang berarti gambar diubah kedalam bentuk kemiringan dengan tingkat kemiringan mencapai 20%.

- *zoom_range* = 0.2. Berfungsi untuk melakukan *zoom* pada gambar secara *random* dalam rentang 20%, kemudian gambar bisa diperbesar atau diperkecil.
- *horizontal_flip* = *True*. Berfungsi untuk membalikkan gambar secara *horizontal* (*flip*). Ini dapat membantu model untuk belajar lebih banyak lagi varian dari gambar yang ada.
- *validation_split* = 0.2. Berfungsi untuk menentukan presentase data yang digunakan untuk memvalidasi. Pada bagian ini, 20% dari total data gambar akan digunakan untuk validasi, sementara itu sisanya 80% akan digunakan untuk pelatihan.



Gambar 4.5 Hasil Augmentasi Gambar

Pada gambar 4.5 menunjukkan hasil dari proses augmentasi gambar. Proses augmentasi akan memutar, memiringkan, dan juga memperbesar gambar secara acak dari parameter yang telah dijelaskan sebelumnya.

4.4 *Model Building*

Dalam pembuatan *model building* CNN ini terdiri dari 4 lapisan, dan juga 1 lapisan *fully connected (neural network).* Dalam proses pemodelan CNN ini dilakukan dengan menggunakan TensorFlow, seperti kode dibawah 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(5,activation='softmax'))

model.summary()
```

Gambar 4.6 Model CNN

Pada model CNN ini terdiri dari 4 layer konvolusi dengan ukuran kernel 3x3 dengan jumlah filter yang dimulai dari 256, 128, 128, dan 64. Pada masing-masing layer *MaxPooling* 2x2 untuk ukuran filter. Lapisan *Flatten* mengubah fitur menjadi bentuk satu dimensi, lapisan *Dense* dengan 32 *neuron* dan aktivasi ReLU memprosesnya sebelum masuk ke lapisan *output* dengan 4 *neuron* untuk klasifikasi multi-kelas, *neuron* dan aktivasi *Softmax*. Model ini dapat menerima gambar dengan dimensi 256, 256, dan 3. Di bawah ini pada gambar 4.7 adalah hasil dari pengembangan model CNN:

Model: "sequential"

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, 5)	165

Total params: 925,189 (3.53 MB)
Trainable params: 925,189 (3.53 MB)
Non-trainable params: 0 (0.00 B)

Gambar 4.7 Hasil Model CNN

4.5 Training Model

Sebelum memulai pelatihan, ada beberapa pengaturan yang harus dilakukan agar model yang dibuat sempurna dan proses pelatihan berjalan dengan lebih efisien. Pada kode berikut menunjukkan pengaturan tersebut:

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

Gambar 4.8 Kode Untuk Pengaturan Model

Kode diatas merupakan fungsi untuk mengatur optimasi model dengan menggunakan *Adam Optimizer*, *categorical crossentropy* digunakan untuk fungsi *loss* pada klasifikasi multi-kelas, dan untuk evaluasi kinerja model digunakan *metrics accuracy.*

- *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.

- *loss* = *'categorical_crossentropy'* = Fungsi *loss* ini digunakan untuk masalah klasifikasi multi-kelas dengan *one-hot encoding*.
- *metrics* = ['*accuracy*'] = Metrik akurasi digunakan untuk mengukur seberapa sering prediksi model benar dibandingkan dengan label asli.

Langkah selanjutnya ketika sudah selesai melakukan pengaturan yaitu melatih model CNN. Berikut merupakan kode untuk melakukan pelatian model CNN:

```
from sklearn.utils.class_weight import compute_class_weight

# Menghitung bobot kelas
class_weights = compute_class_weight(
    'balanced',
    classes=np.unique(y_train),
    y=y_train
)

# Menyusun class_weights menjadi dictionary
class_weight_dict = dict(enumerate(class_weights))

# Menyusun training dengan weighted loss
history = model.fit(
    train_generator,
    validation_data=test_generator,
    epochs=50,
    class_weight=class_weight_dict
)
```

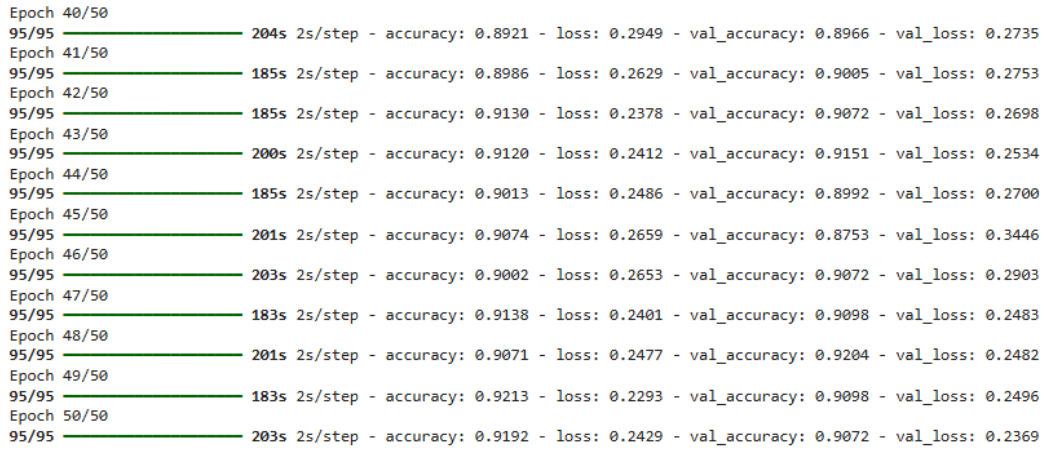
Gambar 4.9 Kode untuk melatih model

Untuk data latih dan validasi, proses pelatihan akan menghasilkan catatan yang berisi metrik performa model seperti *loss* dan *accuracy*.

- Parameter '*balanced*': Menginstruksikan fungsi untuk menghitung bobot kelas secara otomatis sehingga bobot untuk setiap kelas sebanding dengan jumlah sampel di kelas tersebut.
- *classes* = *np*.unique(y_*train*): Mengambil daftar kelas unik dari label target (*y_train*).

- *y=y_train*: Label target untuk menghitung distribusi kelas.
- *enumerate*(*class_weights*): memberikan indeks untuk setiap bobot kelas (dengan asumsi indeks mewakili kelas).
- *dict*(): mengubah hasil *enumerate* menjadi *dictionary* dengan format [kelas: bobot].
- *train_generator*: *Generator* atau data yang digunakan untuk melatih model.
- *validation_data=test_generator*: Data validasi untuk mengevaluasi performa model selama pelatihan.
- *Epochs*=50: Model akan dilatih sebanyak 50 *epoch*. Setiap 1 *epoch* adalah satu siklus iterasi penuh melalui dataset.
- *class_weight=class_weight_dict*: Menyediakan bobot kelas yang dihitung sebelumnya kepada metode fit. Ini menginstruksikan model untuk mempertimbangkan bobot saat menghitung kerugian (*loss*) selama pelatihan, sehingga memperhatikan kelas minoritas.

Hasil dari proses training dapat dilihat pada gambar dibawah ini:



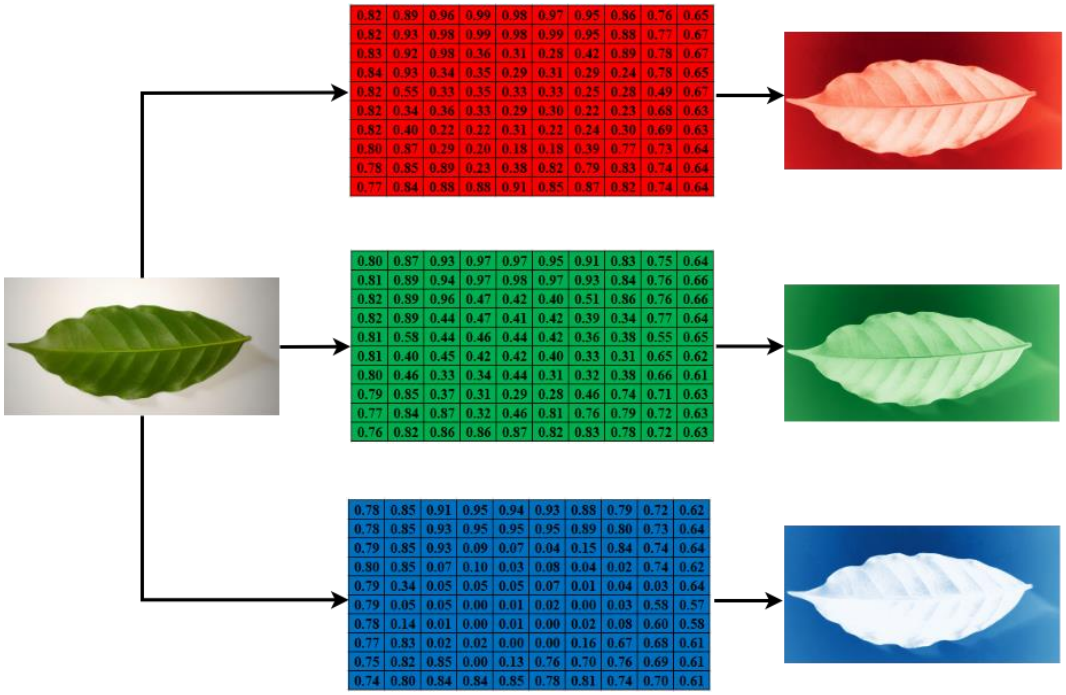
Gambar 4.10 Hasil Proses *Training* Model

Hasil akhir dari proses pelatihan model CNN yaitu di *epoch* 50 sudah menunjukan keakurasian hingga diatas 90%.

Selama proses pelatihan model, gambar akan melalui empat lapisan CNN: lapisan *input*, lapisan konvolusi, lapisan *pooling*, dan lapisan *fully connected*, seperti yang ditunjukkan di bawah ini.

4.5.1 Lapisan Input / *Input Layer*

Data dari gambar daun kopi akan dimasukkan ke lapisan pertama. Pada lapisan ini, gambar diubah menjadi matriks tiga dimensi yang memiliki warna (*RGB channel*), panjang, dan lebar. Contoh berikut menunjukkan hasil input gambar.



Gambar 4.11 *Hasil input layer*

Pada gambar 4.11 proses *input layer* citra akan diekstrak kedalam tiga warna, merah, hijau, dan biru.

4.5.2 Lapisan Konvolusi

Pada lapisan / *layer* ini sebuah input akan dimasukkan kedalam lapisan konvolusi yang mana nilai *RGB* akan diekstrak ke setiap *pixel* yang berada pada citra sehingga menghasilkan *output (feature map)* berfungsi untuk mengolah citra pada lapisan konvolusi maka diperlukan sebuah *filter*. Berikut ini salah satu contoh perhitungan pada *channel* merah. Seperti pada gambar 4.12

0.82	0.89	0.96	0.99	0.98	0.97	0.95	0.86	0.76	0.65
0.82	0.93	0.98	0.99	0.98	0.99	0.95	0.88	0.77	0.67
0.83	0.92	0.98	0.36	0.31	0.28	0.42	0.89	0.78	0.67
0.84	0.93	0.34	0.35	0.29	0.31	0.29	0.24	0.78	0.65
0.82	0.55	0.33	0.35	0.33	0.33	0.25	0.28	0.49	0.67
0.82	0.34	0.36	0.33	0.29	0.30	0.22	0.23	0.68	0.63
0.82	0.40	0.22	0.22	0.31	0.22	0.24	0.30	0.69	0.63
0.80	0.87	0.29	0.20	0.18	0.18	0.39	0.77	0.73	0.64
0.78	0.85	0.89	0.23	0.38	0.82	0.79	0.83	0.74	0.64
0.77	0.84	0.88	0.88	0.91	0.85	0.87	0.82	0.74	0.64

Gambar 4.12 *Input* Citra Merah

Tahap ini dilakukan perhitungan pada sebuah input yang mana bertujuan untuk mendapatkan *feature map*. Perhitungan ini dibantu dengan menggunakan *filter* nilai acak berukuran 3x3 pada gambar 4.13.

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

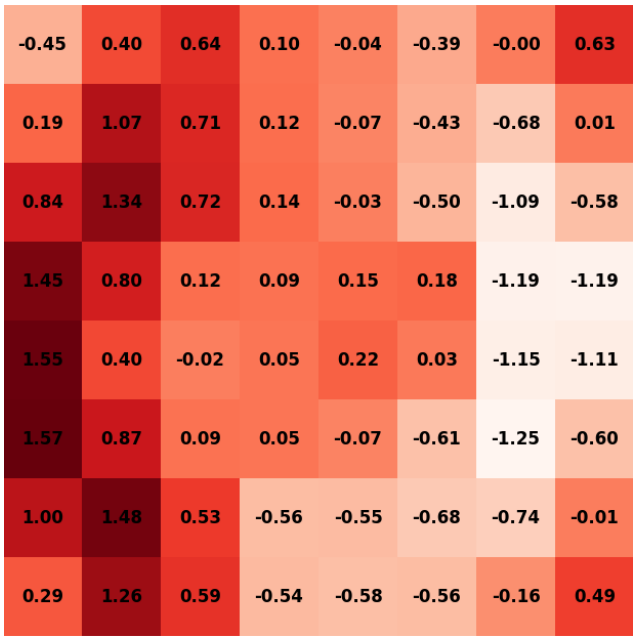
Gambar 4.13 Contoh *filter*

Persamaan 1.1 akan digunakan dalam proses perhitungan untuk mencari *feature map* dari gambar. Dengan demikian, perhitungan dapat dilakukan.

$y_{1,1}$ = (0.82*1) + (0.89*0) + (0.96*(-1)) + (0.82*1) + (0.93*0) + (0.98*(-1)) + (0.83*1) + (0.92*0) + (0.98*(-1))

$y_{1,1}$ = -0.45

Sehingga hasil yang di berikan pada kolom 1.1 adalah -0.45, dan pada gambar 4.14 merupakan hasil dari keseluruhan perhitungan sebuah citra.



Gambar 4.14 Hasil perhitungan citra merah

Saat perhitungan pada lapisan konvolusi telah selesai, proses selanjutnya adalah menambahkan fungsi aktivasi. Penggunaan *ReLU* untuk fungsi aktivasi pada lapisan konvolusi, yaitu untuk mengubah nilai x menjadi nol, jika nilai x lebih kecil dari nol. Berikut ini merupakan contoh proses penambahan fungsi aktivasi *ReLU*.

$$y_{1,1} = \begin{cases} 0.40 \; if \;\; 0.40 > 0 \\ 0 \;\; if \; 0.40 < 0 \end{cases}$$

Bisa dilihat pada perhitungan $y_{1,1}$ diatas nilai 0.40 lebih besar dari nol yang mana hasilnya akan tetap 0.40. Yang sehingganya hasil penambahan fungsi aktivasi untuk menghasilkan *feature map* dapat diliat pada gambar 4.15.

0	0.40	0.64	0.10	0	0	0	0.63
0.19	1.07	0.71	0.12	0	0	0	0.01
0.84	1.34	0.72	0.14	0	0	0	0
1.45	0.80	0.12	0.09	0.15	0.18	0	0
1.55	0.40	0	0.05	0.22	0.03	0	0
1.57	0.87	0.09	0.05	0	0	0	0
1.00	1.48	0.53	0	0	0	0	0
0.29	1.26	0.59	0	0	0	0	0.49

Gambar 4.15 Hasil *Feature Map*

4.5.3 Lapisan *Pooling* / *Pooling Layer*

Pada titik ini, digunakan untuk mengurangi jumlah perhitungan dan parameter *neural network*. Dalam penelitian ini, *Max Pooling* adalah jenis *pooling layer* yang digunakan. Yang mana pengambilan nilai terbesar hasil kovolusi merupakan fungsi dari *max pooling*. *Max pooling* matriks berukuran 2x2 dengan *stride* 2 diproses oleh *pooling layer* sehingga menghasilkan *feature map* yang didapatkan pada layer kovolusi, ditunjukkan pada gambar di bawah ini.

0	0.40	0.64	0.10	0	0	0	0.63
0.19	1.07	0.71	0.12	0	0	0	0.01
0.84	1.34	0.72	0.14	0	0	0	0
1.45	0.80	0.12	0.09	0.15	0.18	0	0
1.55	0.40	0	0.05	0.22	0.03	0	0
1.57	0.87	0.09	0.05	0	0	0	0
1.00	1.48	0.53	0	0	0	0	0
0.29	1.26	0.59	0	0	0	0	0.49

→

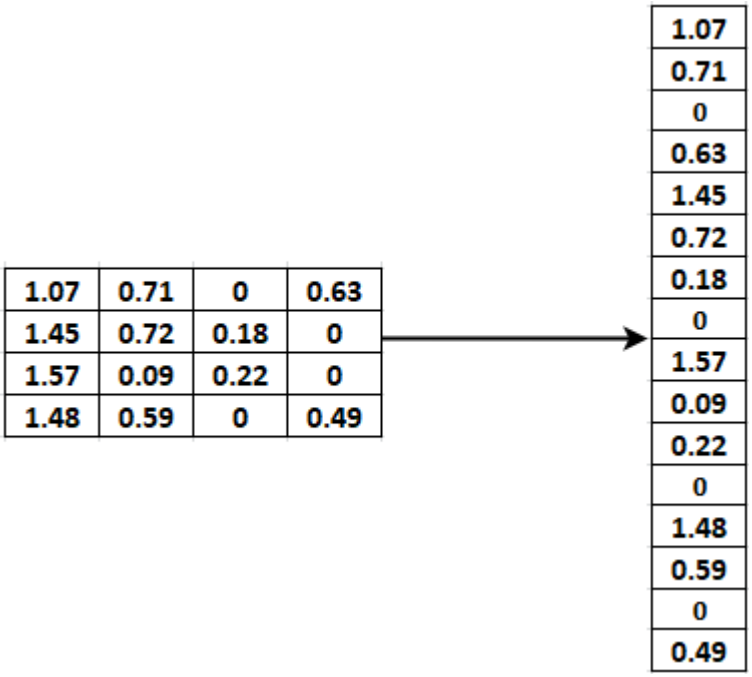
1.07	0.71	0	0.63
1.45	0.72	0.18	0
1.57	0.09	0.22	0
1.48	0.59	0	0.49

Gambar 4.16 Proses *input* dan *output pooling layer*

Input digambarkan sebagai *feature map*, dan *output* digambarkan sebagai hasil dari *pooling layer*. Pada proses ini akan menghasilkan sebuah *output* yang memiliki nilai terbesar dari semua *input*. Didalam kasus ini nilai *input* terbesar adalah 1.07, yang akan dimasukkan ke dalam *output pooling layer*.

4.5.4 Lapisan *Flatten*

Flatten merupakan tahapan mengubah *matrix* kebentuk *vector/matrix* 1 dimensi yang di dapat dari hasil *output pooling layer*, dapat dilihat pada gambar berikut ini.

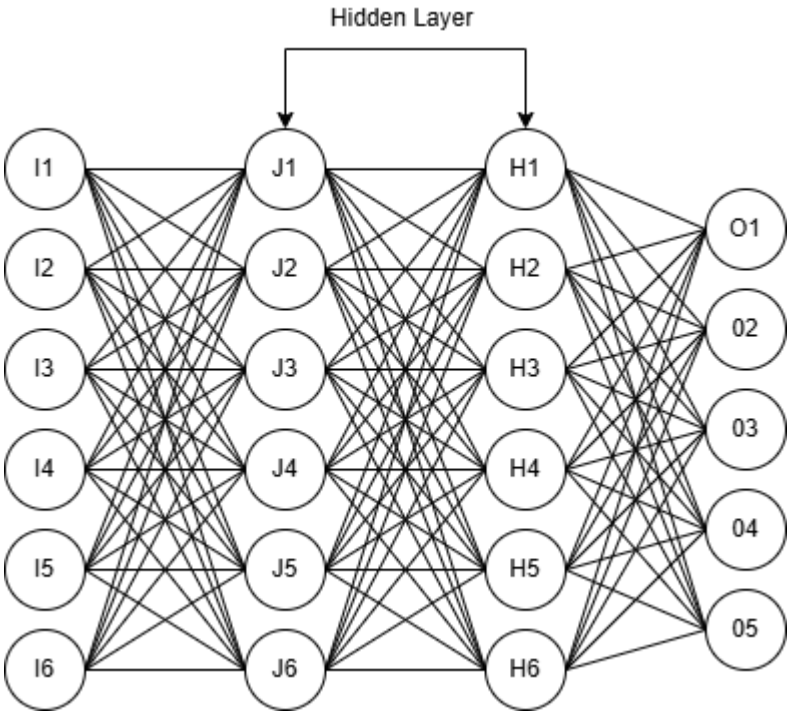


Gambar 4.17 Ilustrasi proses *Flatten*

Gamabr 4.17 adalah proses *flatten* dimana *matrix* 4x4 diubah menjadi 16x1 atau *vector*. *Output* pada tahapan ini adalah *vector* 1 dimensi yang akan digunakan sebagai input di *fully connected layer*.

4.5.5 Lapisan *Fully Connected*

Output dari tahapan *flatten layer* yaitu *vector* 1 dimensi, yang akan menjadi inputan pada tahapan ini, berikut ini merupakan gambar ilustrasi *fully connected layer*.



Gambar 4.18 Ilustrasi proses *dense + softmax*

Gambar diatas merupakan contoh proses *dense + softmax* yang mana dengan jumlah *vector* 1 dimensi berjumlah 16 yang akan di hitung dari setiap *hidden layer* nya yang berjumlah 16. Berikut ini merupakan perhitungannya.

$$\sum_{i=1}^{N} \mathrm{I}_i * V_{ij} = J_i$$

$$\sum_{i=1}^{N} \mathrm{J}_i * W_{ij} = H_i$$

$$\sum_{i=1}^{N} \mathrm{H}_i * X_{ij} = O_i$$

I1,I2,I3,I4,I5,I6 dan seterusnya hingga I16 adalah nilai *output* pada proses *flatten* yang kemudian dilakukan *fully connected layer* dengan perhitungan pada setiap *hidden layer* sebagai berikut (untuk nilai bobot pada setiap *layer* harus berbeda, dan untuk nilai bobot itu bebas menggunakan berapapun).

J_1 =
(1.07*0.2)+(0.71*0.2)+(0*0.2)+(0.63*0.2)+(1.45*0.2)+(0.72*0.2)+(0.18*0.2)+(0*0.2)+(1.57*0.2)+(0.09*0.2)+(0.22*0.2)+(0*0.2)+(1.48*0.2)+(0.59*0.2)+(0*0.2)+(0.49*0.2) = 1.84

J_2 =
(1.07*0.1)+(0.71*0.1)+(0*0.1)+(0.63*0.1)+(1.45*0.1)+(0.72*0.1)+(0.18*0.1)+(0*0.1)+(1.57*0.1)+(0.09*0.1)+(0.22*0.1)+(0*0.1)+(1.48*0.1)+(0.59*0.1)+(0*0.1)+(0.49*0.1) = 0.92

J_3 =
(1.07*0.3)+(0.71*0.3)+(0*0.3)+(0.63*0.3)+(1.45*0.3)+(0.72*0.3)+(0.18*0.3)+(0*0.3)+(1.57*0.3)+(0.09*0.3)+(0.22*0.3)+(0*0.3)+(1.48*0.3)+(0.59*0.3)+(0*0.3)+(0.49*0.3) = 2.76

J_4 =
(1.07*0.5)+(0.71*0.5)+(0*0.5)+(0.63*0.5)+(1.45*0.5)+(0.72*0.5)+(0.18*0.5)+(0*0.5)+(1.57*0.5)+(0.09*0.5)+(0.22*0.5)+(0*0.5)+(1.48*0.5)+(0.59*0.5)+(0*0.5)+(0.49*0.5) = 4.6

J_5 =
(1.07*0.4)+(0.71*0.4)+(0*0.4)+(0.63*0.4)+(1.45*0.4)+(0.72*0.4)+(0.18*0.4)+(0*0.4)+(1.57*0.4)+(0.09*0.4)+(0.22*0.4)+(0*0.4)+(1.48*0.4)+(0.59*0.4)+(0*0.4)+(0.49*0.4) = 3.68

J_6 =
(1.07*0.7)+(0.71*0.7)+(0*0.7)+(0.63*0.7)+(1.45*0.7)+(0.72*0.7)+(0.18*0.7)+(0*0.7)+(1.57*0.7)+(0.09*0.7)+(0.22*0.7)+(0*0.7)+(1.48*0.7)+(0.59*0.7)+(0*0.7)+(0.49*0.7) = 6.44

J_7 =
(1.07*0.6)+(0.71*0.6)+(0*0.6)+(0.63*0.6)+(1.45*0.6)+(0.72*0.6)+(0.18*0.6)+(0*0.6)+(1.57*0.6)+(0.09*0.6)+(0.22*0.6)+(0*0.6)+(1.48*0.6)+(0.59*0.6)+(0*0.6)+(0.49*0.6) = 5.52

J_8 =
(1.07*0.9)+(0.71*0.9)+(0*0.9)+(0.63*0.9)+(1.45*0.9)+(0.72*0.9)

+(0.18*0.9)+(0*0.9)+(1.57*0.9)+(0.09*0.9)+(0.22*0.9)+(0*0.9)+(
1.48*0.9)+(0.59*0.9)+(0*0.9)+(0.49*0.9) = 8.28

J_9 =
(1.07*0.8)+(0.71*0.8)+(0*0.8)+(0.63*0.8)+(1.45*0.8)+(0.72*0.8)
+(0.18*0.8)+(0*0.8)+(1.57*0.8)+(0.09*0.8)+(0.22*0.8)+(0*0.8)+(
1.48*0.8)+(0.59*0.8)+(0*0.8)+(0.49*0.8) = 7.36

J_{10} =
(1.07*0.11)+(0.71*0.11)+(0*0.11)+(0.63*0.11)+(1.45*0.11)+(0.72
*0.11)+(0.18*0.11)+(0*0.11)+(1.57*0.11)+(0.09*0.11)+(0.22*0.11
)+(0*0.11)+(1.48*0.11)+(0.59*0.11)+(0*0.11)+(0.49*0.11)= 1.012

J_{11}=
(1.07*0.10)+(0.71*0.10)+(0*0.10)+(0.63*0.10)+(1.45*0.10)+(0.72
*0.10)+(0.18*0.10)+(0*0.10)+(1.57*0.10)+(0.09*0.10)+(0.22*0.10
)+(0*0.10)+(1.48*0.10)+(0.59*0.10)+(0*0.10)+(0.49*0.10)= 0.92

J_{12}=
(1.07*0.13)+(0.71*0.13)+(0*0.13)+(0.63*0.13)+(1.45*0.13)+(0.72
*0.13)+(0.18*0.13)+(0*0.13)+(1.57*0.13)+(0.09*0.13)+(0.22*0.13
)+(0*0.13)+(1.48*0.13)+(0.59*0.13)+(0*0.13)+(0.49*0.13)= 1.196

J_{13}=
(1.07*0.12)+(0.71*0.12)+(0*0.12)+(0.63*0.12)+(1.45*0.12)+(0.72
*0.12)+(0.18*0.12)+(0*0.12)+(1.57*0.12)+(0.09*0.12)+(0.22*0.12
)+(0*0.12)+(1.48*0.12)+(0.59*0.12)+(0*0.12)+(0.49*0.12)= 1.104

J_{14}=
(1.07*0.15)+(0.71*0.15)+(0*0.15)+(0.63*0.15)+(1.45*0.15)+(0.72
*0.15)+(0.18*0.15)+(0*0.15)+(1.57*0.15)+(0.09*0.15)+(0.22*0.15
)+(0*0.15)+(1.48*0.15)+(0.59*0.15)+(0*0.15)+(0.49*0.15)=1.38

J_{15} =
(1.07*0.14)+(0.71*0.14)+(0*0.14)+(0.63*0.14)+(1.45*0.14)+(0.72
*0.14)+(0.18*0.14)+(0*0.14)+(1.57*0.14)+(0.09*0.14)+(0.22*0.14
)+(0*0.14)+(1.48*0.14)+(0.59*0.14)+(0*0.14)+(0.49*0.14)=1.288

J_{16}=
(1.07*0.16)+(0.71*0.16)+(0*0.16)+(0.63*0.16)+(1.45*0.16)+(0.72*0.16)+(0.18*0.16)+(0*0.16)+(1.57*0.16)+(0.09*0.16)+(0.22*0.16)+(0*0.16)+(1.48*0.16)+(0.59*0.16)+(0*0.16)+(0.49*0.16)=1.472

Hasil J_1 sampai dengan J_{16} ini akan dikalikan dengan bobot baru yang berbeda-beda untuk mencari nilai H_1 sampai H_{16} sebagai *hidden layer* selanjutnya.

H_1=
(1.84*0.1)+(0.92*0.1)+(2.76*0.1)+(4.6*0.1)+3.68*0.1)+(6.44*0.1)+(5.52*0.1)+(8.28*0.1)+(7.36*0.1)+(1.012*0.1)+(0.92*0.1)+(1.196*0.1)+(1.104*0.1)+(1.38*0.1)+(1.288*0.1)+(1.472*0.1) = 4.9772

H_2=
(1.84*0.2)+(0.92*0.2)+(2.76*0.2)+(4.6*0.2)+3.68*0.2)+(6.44*0.2)+(5.52*0.2)+(8.28*0.2)+(7.36*0.2)+(1.012*0.2)+(0.92*0.2)+(1.196*0.2)+(1.104*0.2)+(1.38*0.2)+(1.288*0.2)+(1.472*0.2) = 9.9544

H_3=
(1.84*0.3)+(0.92*0.3)+(2.76*0.3)+(4.6*0.3)+3.68*0.3)+(6.44*0.3)+(5.52*0.3)+(8.28*0.3)+(7.36*0.3)+(1.012*0.3)+(0.92*0.3)+(1.196*0.3)+(1.104*0.3)+(1.38*0.3)+(1.288*0.3)+(1.472*0.3)= 14.9316

H_4=
(1.84*0.4)+(0.92*0.4)+(2.76*0.4)+(4.6*0.4)+3.68*0.4)+(6.44*0.4)+(5.52*0.4)+(8.28*0.4)+(7.36*0.4)+(1.012*0.4)+(0.92*0.4)+(1.196*0.4)+(1.104*0.4)+(1.38*0.4)+(1.288*0.4)+(1.472*0.4)= 19.9088

H_5=
(1.84*0.5)+(0.92*0.5)+(2.76*0.5)+(4.6*0.5)+3.68*0.5)+(6.44*0.5)+(5.52*0.5)+(8.28*0.5)+(7.36*0.5)+(1.012*0.5)+(0.92*0.5)+(1.196*0.5)+(1.104*0.5)+(1.38*0.5)+(1.288*0.5)+(1.472*0.5) = 24.886

H_6=
(1.84*0.6)+(0.92*0.6)+(2.76*0.6)+(4.6*0.6)+3.68*0.6)+(6.44*0.6)+(5.52*0.6)+(8.28*0.6)+(7.36*0.6)+(1.012*0.6)+(0.92*0.6)+(1.196*0.6)+(1.104*0.6)+(1.38*0.6)+(1.288*0.6)+(1.472*0.6)= 29.8632

H_7=
(1.84*0.7)+(0.92*0.7)+(2.76*0.7)+(4.6*0.7)+3.68*0.7)+(6.44*0.7)+(5.52*0.7)+(8.28*0.7)+(7.36*0.7)+(1.012*0.7)+(0.92*0.7)+(1.196*0.7)+(1.104*0.7)+(1.38*0.7)+(1.288*0.7)+(1.472*0.7)= 34.8404

H_8=
(1.84*0.8)+(0.92*0.8)+(2.76*0.8)+(4.6*0.8)+3.68*0.8)+(6.44*0.8)+(5.52*0.8)+(8.28*0.8)+(7.36*0.8)+(1.012*0.8)+(0.92*0.8)+(1.196*0.8)+(1.104*0.8)+(1.38*0.8)+(1.288*0.8)+(1.472*0.8)= 39.8176

H_9=
(1.84*0.9)+(0.92*0.9)+(2.76*0.9)+(4.6*0.9)+3.68*0.9)+(6.44*0.9)+(5.52*0.9)+(8.28*0.9)+(7.36*0.9)+(1.012*0.9)+(0.92*0.9)+(1.196*0.9)+(1.104*0.9)+(1.38*0.9)+(1.288*0.9)+(1.472*0.9)= 44.7948

H_{10}=
(1.84*0.10)+(0.92*0.10)+(2.76*0.10)+(4.6*0.10)+3.68*0.10)+(6.44*0.10)+(5.52*0.10)+(8.28*0.10)+(7.36*0.10)+(1.012*0.10)+(0.92*0.10)+(1.196*0.10)+(1.104*0.10)+(1.38*0.10)+(1.288*0.10)+(1.472*0.10) = 4.9772

H_{11}=
(1.84*0.11)+(0.92*0.11)+(2.76*0.11)+(4.6*0.11)+3.68*0.11)+(6.44*0.11)+(5.52*0.11)+(8.28*0.11)+(7.36*0.11)+(1.012*0.11)+(0.92*0.11)+(1.196*0.11)+(1.104*0.11)+(1.38*0.11)+(1.288*0.11)+(1.472*0.11) = 5.47492

H_{12}=
(1.84*0.12)+(0.92*0.12)+(2.76*0.12)+(4.6*0.12)+3.68*0.12)+(6.44*0.12)+(5.52*0.12)+(8.28*0.12)+(7.36*0.12)+(1.012*0.12)+(0.92*0.12)+(1.196*0.12)+(1.104*0.12)+(1.38*0.12)+(1.288*0.12)+(1.472*0.12) = 5.97264

H_{13}=
(1.84*0.13)+(0.92*0.13)+(2.76*0.13)+(4.6*0.13)+3.68*0.13)+(6.44*0.13)+(5.52*0.13)+(8.28*0.13)+(7.36*0.13)+(1.012*0.13)+(0.92

*0.13)+(1.196*0.13)+(1.104*0.13)+(1.38*0.13)+(1.288*0.13)+(1.4
72*0.13) = 6.47036

H_{14}=

(1.84*0.14)+(0.92*0.14)+(2.76*0.14)+(4.6*0.14)+3.68*0.14)+(6.4
4*0.14)+(5.52*0.14)+(8.28*0.14)+(7.36*0.14)+(1.012*0.14)+(0.92
*0.14)+(1.196*0.14)+(1.104*0.14)+(1.38*0.14)+(1.288*0.14)+(1.4
72*0.14) = 6.96808

H_{15}=

(1.84*0.15)+(0.92*0.15)+(2.76*0.15)+(4.6*0.15)+3.68*0.15)+(6.4
4*0.15)+(5.52*0.15)+(8.28*0.15)+(7.36*0.15)+(1.012*0.15)+(0.92
*0.15)+(1.196*0.15)+(1.104*0.15)+(1.38*0.15)+(1.288*0.15)+(1.4
72*0.15) = 7.4658

H_{16}=

(1.84*0.16)+(0.92*0.16)+(2.76*0.16)+(4.6*0.16)+3.68*0.16)+(6.4
4*0.16)+(5.52*0.16)+(8.28*0.16)+(7.36*0.16)+(1.012*0.16)+(0.92
*0.16)+(1.196*0.16)+(1.104*0.16)+(1.38*0.16)+(1.288*0.16)+(1.4
72*0.16) = 7.96352

Hasil dari setiap *neuron* H_1 sampai H_{16} ini akan dikalikan lagi dengan *weight* yang berbeda-beda untuk menghasilkan nilai O_1 sampai O_5 yang mana itu adalah kelas yang ada di dataset.

O_1=

(4.9772*0.4)+(9.9544*0.4)+(14.9316*0.4)+(19.9088*0.4)+(24.886
*0.4)+(29.8632*0.4)+(34.8404*0.4)+(39.8176*0.4)+(44.7948*0.4)
+(4.9772*0.4)+(5.47492*0.4)+(5.97264*0.4)+(6.47036*0.4)+(6.96
808*0.4)+(7.4658*0.4)+(7.96352*0.4)= 107.7066

O_2=

(4.9772*0.5)+(9.9544*0.5)+(14.9316*0.5)+(19.9088*0.5)+(24.886
*0.5)+(29.8632*0.5)+(34.8404*0.5)+(39.8176*0.5)+(44.7948*0.5)
+(4.9772*0.5)+(5.47492*0.5)+(5.97264*0.5)+(6.47036*0.5)+(6.96
808*0.5)+(7.4658*0.5)+(7.96352*0.5)= 134.63326

O_3=
(4.9772*0.2)+(9.9544*0.2)+(14.9316*0.2)+(19.9088*0.2)+(24.886*0.2)+(29.8632*0.2)+(34.8404*0.2)+(39.8176*0.2)+(44.7948*0.2)+(4.9772*0.2)+(5.47492*0.2)+(5.97264*0.2)+(6.47036*0.2)+(6.96808*0.2)+(7.4658*0.2)+(7.96352*0.2)= 53.8533

O_4=
(4.9772*0.3)+(9.9544*0.3)+(14.9316*0.3)+(19.9088*0.3)+(24.886*0.3)+(29.8632*0.3)+(34.8404*0.3)+(39.8176*0.3)+(44.7948*0.3)+(4.9772*0.3)+(5.47492*0.3)+(5.97264*0.3)+(6.47036*0.3)+(6.96808*0.3)+(7.4658*0.3)+(7.96352*0.3)= 80.78

O_5=
(4.9772*0.1)+(9.9544*0.1)+(14.9316*0.1)+(19.9088*0.1)+(24.886*0.1)+(29.8632*0.1)+(34.8404*0.1)+(39.8176*0.1)+(44.7948*0.1)+(4.9772*0.1)+(5.47492*0.1)+(5.97264*0.1)+(6.47036*0.1)+(6.96808*0.1)+(7.4658*0.1)+(7.96352*0.1)= 26.926652

Langkah selanjutnya adalah perhitungan *softmax* dengan rumus eksponensial O_1 dibagi dengan total eksponensial O_1 sampai eksponensial O_5. Tahapan ini dilakukan sebanyak 5 kali baik pada O_1 sampai O_5 dengan rumus dan ilustrasi berikut ini:

$$s(O_i)s(O_i)\frac{eO_i}{\sum e\ O_j}$$

Dimana:

- eO_i adalah eksponensial dari masing-masing nilai $O_{i.}$
- $\sum e\ O_i$ adalah jumlah dari semua eksponensial nilai *O.*

$$s(O_1)\frac{eO_i}{\sum e\ O_j} = \frac{107.7066}{107.7066} = 1$$

$$s(O_2)\frac{eO_i}{\sum e\ O_j} = \frac{134.63326}{107.7066} = 0.8002$$

$$s(O_3)\frac{eO_i}{\sum e\,O_j} = \frac{53.8533}{107.7066} = 2.00$$

$$s(O_4)\frac{eO_i}{\sum e\,O_j} = \frac{80.78}{107.7066} = 1.33$$

$$s(O_5)\frac{eO_i}{\sum e\,O_j} = \frac{26.926652}{107.7066} = 4$$

Sehingga hasil dari perhitungan *softmax* pada contoh citra warna merah yang tertinggi yaitu pada O_5 dengan nilai 4. Seperti itulah contoh untuk perhitungan mencari nilai *fully connected layer*.

4.6 *Evaluation Model*

Pada tahap evaluasi model, dari hasil proses *training model* akan dievaluasi untuk melihat seberapa besar keakurasian dan seberapa besar model dapat meminimalkan kesalahan dalam deteksi.

Berikut ini adalah hasil dari *accuracy* dan *loss*. Hasil ini akan digunakan untuk melihat perbandingan proses berjalan dengan baik atau tidak.

Gambar 4.19 Hasil dari Train dan Validation Accuracy

Dari hasil pelatihan yang sudah dilakukan dapat dilihat pada gambar 4.17. Bahwa hasil dari proses training dengan 50*epoch* menghasilkan *train*

accuracy sebesar 0.9192 atau 91.92% dan *validation accuracy* sebesar 0.9072 atau 90.72%.

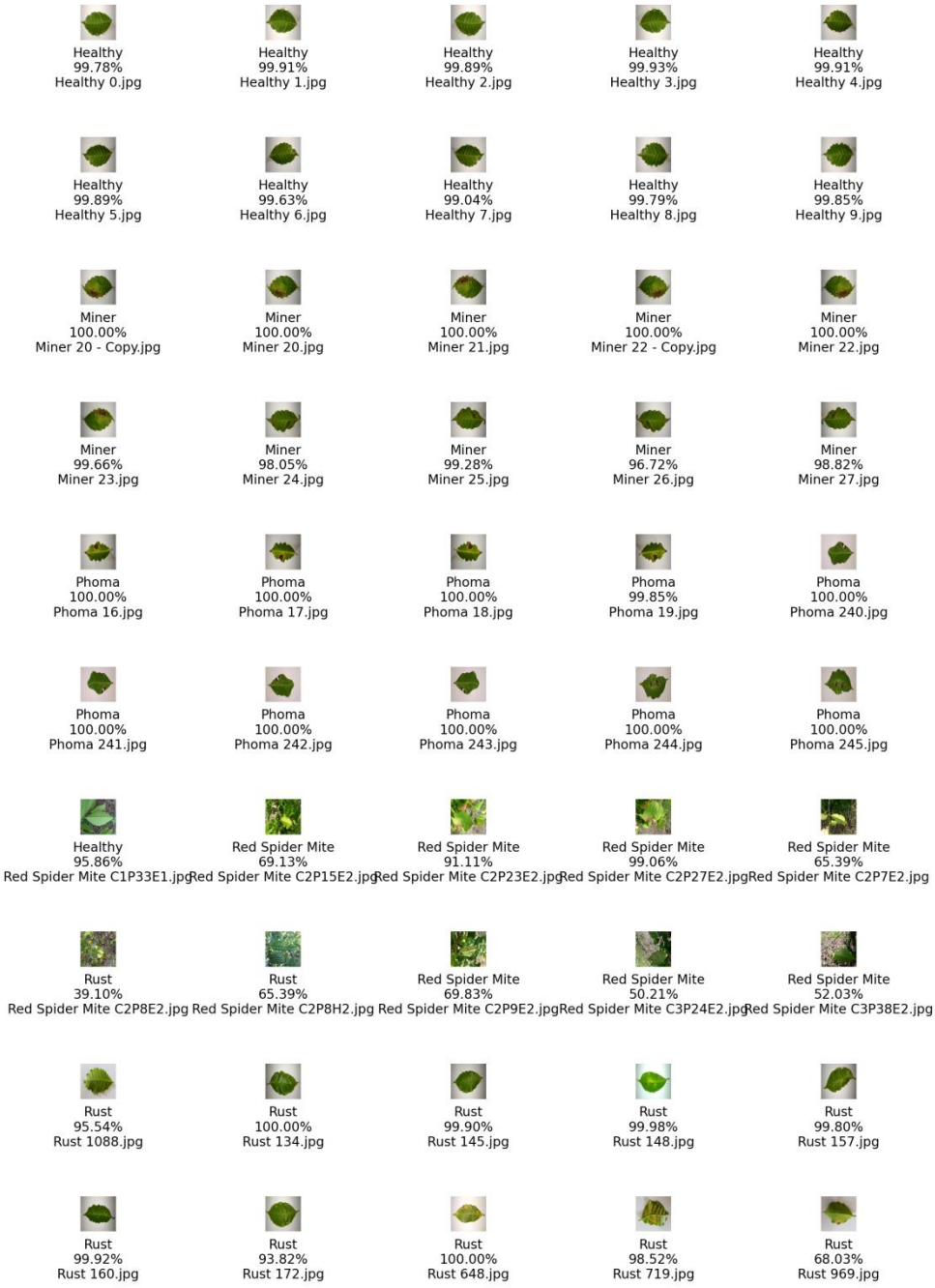
Gambar 4.20 Hasil dari Train dan Validation Loss

Dari hasil pelatihan yang sudah dilakukan dapat dilihat pada gambar 4.18. Bahwa hasil dari proses training dengan 50*epoch* menghasilkan *train loss* sebesar 0.2429 dan *validation loss* sebesar 0.2369.

Ini menunjukkan bahwa model cukup baik dalam mempelajari pola dari data latih dan model tidak terlalu *overfitting*. Juga model berhasil meminimalkan kesalahan pada data latih serta model memiliki performa yang stabil diluar data latih.

4.7 Pengujian Model

Setelah selesai pada proses training serta model berhasil dibuat, selanjutnya yaitu tahap melakukan pengujian pada model. Pengujian ini tidak lain bertujuan untuk mengevaluasi kemampuan dari model CNN dalam memprediksi data gambar dengan akurat. Berikut ini merupakan hasil pengujian model:



Gambar 4.21 Hasil pengujian model dengan 50 gambar

Dapat dilihat pada gambar 4.19 pengujian model menggunakan 50 data gambar menunjukan akurasi berbeda beda. Berikut ini penjelasan lebih rinci, dapat dilihat pada table berikut:

Tabel 4.1 Hasil Pengujian Model 50 Gambar

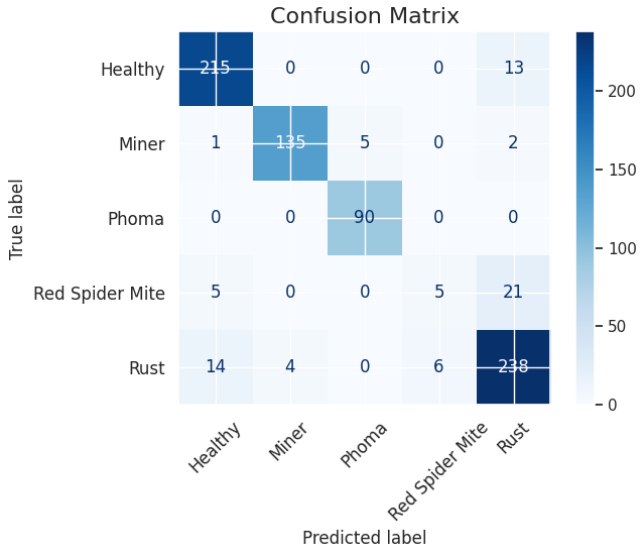
No	Prediction	Actual	Confidence
1	Healthy	Healthy	99.78%
2	Healthy	Healthy	99.91%
3	Healthy	Healthy	99.98%
4	Healthy	Healthy	99.93%
5	Healthy	Healthy	99.91%
6	Healthy	Healthy	99.89%
7	Healthy	Healthy	99.93%
8	Healthy	Healthy	99.04%
9	Healthy	Healthy	99.79%
10	Healthy	Healthy	99.85%
11	Miner	Miner	100.00%
12	Miner	Miner	100.00%
13	Miner	Miner	100.00%
14	Miner	Miner	100.00%
15	Miner	Miner	100.00%
16	Miner	Miner	99.66%
17	Miner	Miner	98.05%
18	Miner	Miner	99.28%
19	Miner	Miner	96.72%
20	Miner	Miner	98.82%
21	Phoma	Phoma	100.00%
22	Phoma	Phoma	100.00%
23	Phoma	Phoma	100.00%
24	Phoma	Phoma	100.00%
25	Phoma	Phoma	100.00%
26	Phoma	Phoma	100.00%
27	Phoma	Phoma	100.00%
28	Phoma	Phoma	100.00%
29	Phoma	Phoma	100.00%
30	Phoma	Phoma	100.00%
31	Healthy	Red Spider Mite	95.86%
32	Red Spider Mite	Red Spider Mite	69.13%
33	Red Spider Mite	Red Spider Mite	91.11%
34	Red Spider Mite	Red Spider Mite	99.06%

35	Red Spider Mite	Red Spider Mite	65.39%
36	Rust	Red Spider Mite	39.10%
37	Rust	Red Spider Mite	65.39%
38	Red Spider Mite	Red Spider Mite	69.83%
39	Red Spider Mite	Red Spider Mite	50.21%
40	Red Spider Mite	Red Spider Mite	52.03%
41	Rust	Rust	95.54%
42	Rust	Rust	100.00%
43	Rust	Rust	99.90%
44	Rust	Rust	99.98%
45	Rust	Rust	99.80%
46	Rust	Rust	99.92%
47	Rust	Rust	93.82%
48	Rust	Rust	100.00%
49	Rust	Rust	98.52%
50	Rust	Rust	68.03%

Dari hasil pengujian model menggunakan 50 data gambar, terdapat tiga kesalahan model dalam memprediksi yang mana telah diberi warna kuning untuk membedakan hasil dari ketepatan akurasi. Tetapi dalam hal ini model sudah cukup baik dalam memprediksi gambar dikarenakan rata-rata nilai presentase yaitu 90% keatas.

4.8 *Confusion Matrix*

Berikut ini merupakan hasil evaluasi matriks menggunakan *confusion matrix*.



Gambar 4.22 Hasil *confusion matrix*

Pada gambar 4.20 dapat dilihat bahwa *index* ke 0 (daun tanaman kopi *Healthy*) berhasil mengklasifikasikan 215 data daun *Healthy* dari total jumlah 228 data daun *Health*, pada *index* ke 1 (daun tanaman kopi *Miner*) berhasil mengklasifikasikan 135 data daun *Miner* dari total jumlah 143 data daun *Miner*, pada *index* ke 2 (daun tanaman kopi *Phoma*) berhasil mengklasifikasikan 90 data daun *Phoma* dari total jumlah 90 data, pada *index* ke 3 (daun tanaman kopi *Red Spider Mite*) hanya berhasil mengklasifikasikan 21 data daun *Red Spider Mite* dari total jumlah 31 data daun sehingga masih ada beberapa salah deteksi, lalu yang terakhir yaitu pada *index* ke 4 (daun tanaman kopi *Rust*) berhasil mengklasifikasikan 238 data daun *Rust* dari total jumlah 262 data daun *Rust.*

Dari hasil *confusion matrix* tersebut dapat dilihat *precission*, *recall*, dan *f1 score* dapat dilihat pada table berikut:

Tabel 4.2 Hasil *Confusion Matrix*

Classification Repot	Precission	Recall	F1-Score	Support
Healthy	0.91	0.94	0.93	228
Miner	0.97	0.94	0.96	143
Phoma	0.95	1.00	0.97	90
Red Spider Mite	0.45	0.16	0.24	31
Rust	0.87	0.91	0.89	262
Accuracy			0.91	754
Macro average	0.83	0.79	0.80	754
Weighted avarage	0.89	0.91	0.90	754

Kategori Healthy memiliki *precission* 0.91 dan *F1-Score* 93%, kategori Miner memiliki *precission* yang paling besar diantara ke 4 kelas yang ada yaitu sekitar 0.97 serta *F1-Score* 96%, kategori Phoma memiliki *precission* yang cukup tinggi dari kelas Healthy dan memiliki *F1-Score* yang paling besar diantara ke 4 kelas yang ada yaitu 97%, sedangkan kategori Red Spider Mite memiliki nilai *precission* dan *F1-Score* terendah yaitu sekitara 24%, dan Kategori Rust memiliki *precission* dan *F1-Score* yang sudah cukup baik diantara 89%.

4.9 Implementasi Model Deteksi Penyakit Pada *Aplikasi Coffee Health*

Pada proses implementasi model deteksi penyakit di aplikasi Coffee Health, sebelum itu model terlebih dahulu diubah kedalam format TensorFlow Lite agar dapat dimasukkan ke dalam aplikasi. Berikut ini kode untuk mengubah model kedalam format TensorFlow Lite.

```
import tensorflow as tf

# Konversi model ke format TFLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

Gambar 4.23 Kode convert model ke *TensorFlow Lite*

Sebelum melakukan pembuatan aplikasi, peneliti membuat sebuah rancangan, yang mana terdiri dari *use case diagram, activity diagram*, *class diagram*, dan desain aplikasi, yang dapat dilihat pada bab 3 sub bab 3.4. Tujuan dari implementasi ini adalah untuk menguji kemampuan model dalam mengkategorikan jenis penyakit pada tanaman kopi. Perlunya perangkat keras untuk menguji deteksi penyakit pada aplikasi tercantum dalam tabel berikut ini.

Tabel 4.3 Spesifikasi *Hardware* Uji Coba

Spesifikasi Laptop	
Processor	Intel Core i5-8250U
RAM	12 GB
HDD	1 TB
SSD	240 GB
OS	Windows 11 Home

Spesifikasi Smartphone	
Processor	Mediatek Dimensity 700 (7 nm) Octa-core (2x2.2 GHz Cortex-A76 & 6x2.0 GHz Cortex-A55)
RAM	6 GB
ROM	128 GB
Camera	48 MP, f/1.8, 26mm (wide), 1/2.0", 0.8µm, PDAF 2 MP, f/2.4, (macro) 2 MP, f/2.4, (depth)

4.10 Tampilan Aplikasi

Berikut ini merupakan hasil dari aplikasi yang telah siap untuk digunakan.

4.10.1 Tampilan *Splash Screen*

Halaman splash screen adalah tampilan awal aplikasi, yang menampilkan nama dan logo aplikasi selama 3 detik sebelum masuk ke tampilan berikutnya, dapat di lihat pada gambar 4.22.

Gambar 4.24 Tampilan Splash Screen

4.10.2 Tampilan *Home*

Tampilan Home aplikasi dirancang sebagai pusat navigasi utama dengan antarmuka yang sederhana terdapat beberapa fitur yang dapat diakses pengguna yaitu:

1 Fitur cara melakukan scan di aplikasi Coffee Health

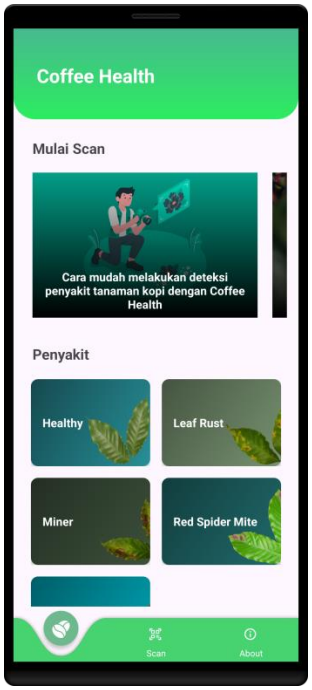
Pengguna dapat melihat cara-cara untuk melakukan scan daun kopi dengan baik dan benar agar hasil deteksi maksimal.

2 Fitur informasi jenis kopi

Pengguna dapat melihat informasi jenis-jenis kopi dan juga manfaat dari kopi.

3 Fitur informasi Penyakit

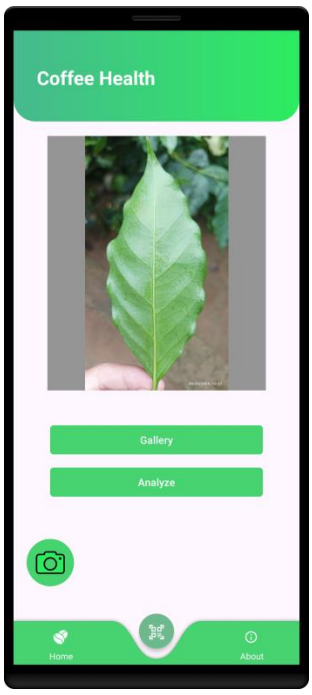
Pengguna dapat melihat informasi setiap penyakit antara lain *Healthy*, *Leaf Rust*, *Miner*, *Phoma*, dan *Red Spider Mite*.



Gambar 4.25 Tampilan *Home*

4.10.3 Tampilan *Scan*

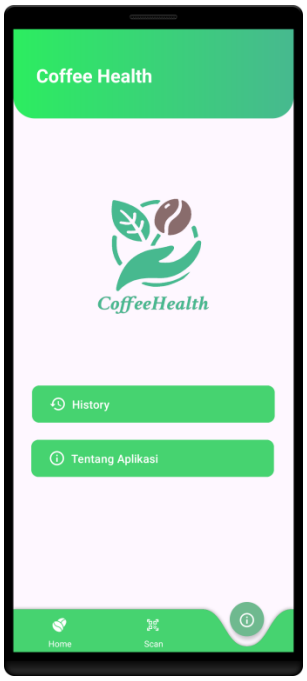
Pada tampilan scan pengguna dapat mengakses fitur kamera dan juga galeri, serta deteksi.



Gambar 4.26 Tampilan *Scan*

4.10.4 Tampilan *About*

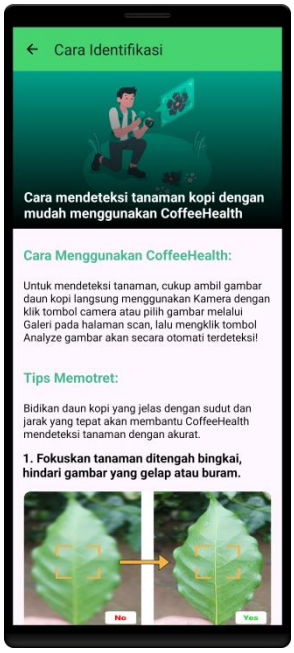
Pada tampilan *About*, pengguna dapat mengakses fitur informasi lihat histori hasil deteksi dan informasi tentang aplikasi Coffee Health.



Gambar 4.27 Tampilan *About*

4.10.5 Tampilan Informasi cara penggunan *scan*

Pada tampilan ini pengguna dapat melihat cara-cara untuk menggunakan aplikasi secara baik dan benar.



Gambar 4.28 Tampilan Informasi cara identifikasi

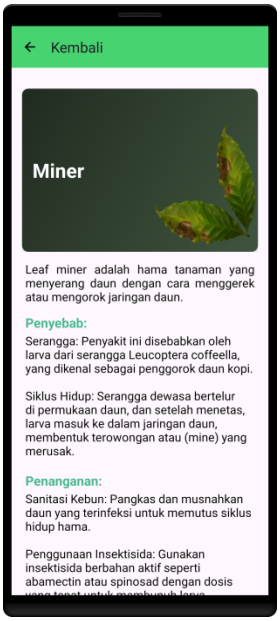
4.10.6 Tampilan Informasi Tipe Kopi dan Manfaatnya

Pada tampilan ini pengguna dapat melihat informasi-informasi terkait tipe dan juga manfaat dari kopi.

Gambar 4.29 Tampilan Informasi Tipe kopi

4.10.7 Tampilan informasi terkait penyakit

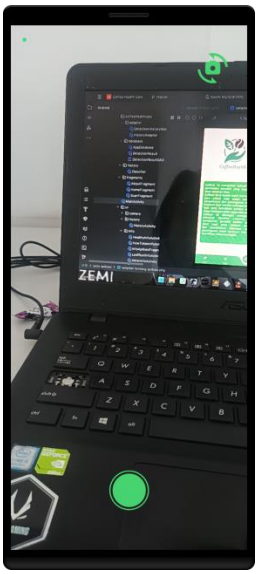
Pada tampilan ini pengguna bisa melihat informasi-informasi penyakit penyebab serta saran penanganannya yang diambil dari sumber-sumber terpercaya, gambar dibawah merupakan salah satu tampilan dari informasi penyakit.



Gambar 4.30 Tampilan Informasi Penyakit

4.10.8 Tampilan Akses ke *Camera*

Berikut ini merupakan tampilan ketika pengguna mengakses ke kamera.



Gambar 4.31 Akses ke Kamera

4.10.9 Tampilan Akses ke *Galerry*

Berikut ini merupakan tampilan ketika pengguna mengakses ke galeri.

Gambar 4.32 Akses ke Galeri

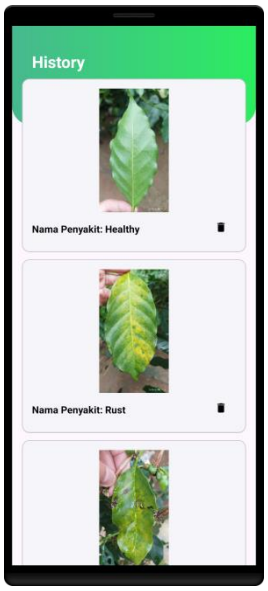
4.10.10 Tampilan Hasil Deteksi

Pada tampilan ini merupakan hasil dari deteksi gambar, pengguna dapat melihat informasi terkait penyebab dan penanganan penyakit kopi. Dan pengguna dapat menyimpan hasil deteksi ini ke histori.

Gambar 4.33 Tampilan Hasil Deteksi

4.10.11 Tampilan Histori

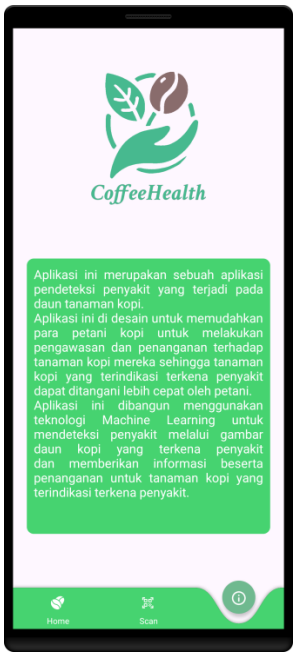
Tampilan ini merupakan hasil dari ketika pengguna menyimpan hasil deteksi penyakit.



Gambar 4.34 Tampilan Histori

4.10.12 Tampilan Informasi Aplikasi

Pada tampilan ini pengguna dapat melihat informasi terkait aplikasi Coffee Health.



Gambar 4.35 Tampilan informasi aplikasi

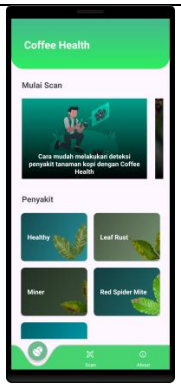
4.11 Uji Coba

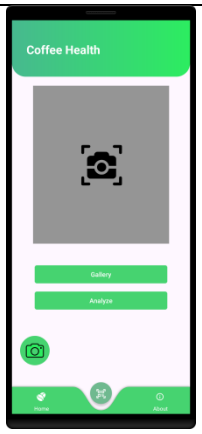
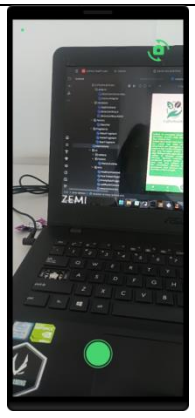
4.11.1 Pengujian Aplikasi

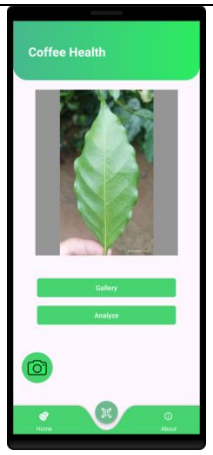
Pengujian aplikasi dilakukan dengan menginstal aplikasi dan menjalankannya pada perangkat dengan sistem operasi Android. Pengujian dilakukan menggunakan perangkat smartphone dengan spesifikasi sebagai berikut:

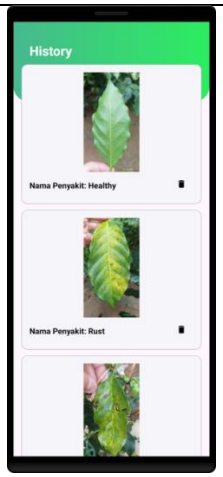
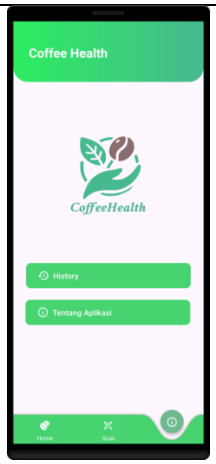
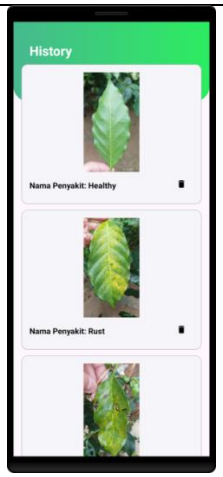
- Nama Smartphone : POCO M3 Pro 5G
- Versi Android : Android 12
- Resolusi Kamera : 48MP

Tabel 4.4 Hasil Pengujian Aplikasi

No	Uji Coba	Deskripsi	Hasil yang diharapkan	Status
1	Pengguna masuk ke Aplikasi	Tampilan Home akan tampil		Sukses
2	Pengguna megklik gambar cara deteksi	Tampilan informasi cara deteksi akan muncul		Sukses

3	Pengguna mengklik gambar tipe kopi	Tampilan informasi tipe dan manfaat kopi akan muncul		Sukses
4	Pengguna mengklik gambar informasi penyakit	Tampilan infromasi penyakit akan muncul.		Sukses
5	Pengguna pergi ke halaman Scan	Tampilan halaman Scan akan muncul		Sukses
6	Pengguna mengklik tombol kamera	Tampilan akses ke kamera akan muncul		Sukses

7	Pengguna mengklik tombol akses galeri	Tampilan akses ke galeri akan muncul		Sukses
8	Tampilan ketika gambar dipilih atau di tangkap oleh kamera	Hasil tangkapan kamera atau dipilih dari galeri akan muncul		Sukses
9	Pengguna mengklik tombol Analyze	Tampilan hasil deteksi akan muncul		Sukses

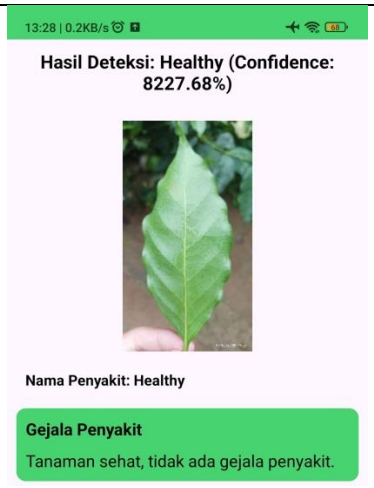
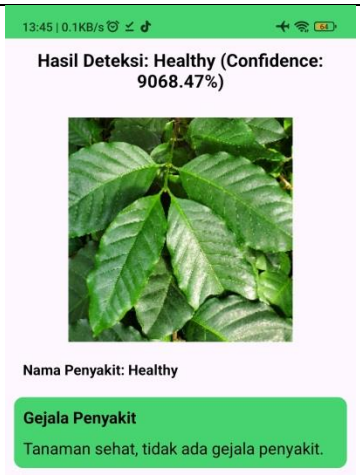
10	Pengguna mengklik tombol save yang berada disebelah kiri tombol kembali	Hasil deteksi akan disimpan di histori, dan ketika pengguna mengakses ke histori tampilan histori akan muncul		Sukses
11	Pengguna mengklik ke halaman About	Tampilan halaman about akan muncul		Sukses
12	Pengguna mengklik tombol histori	Tampilan histori akan muncul		Sukses

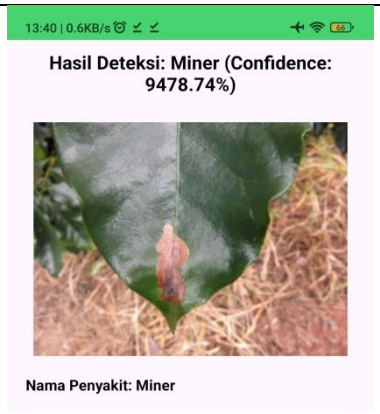
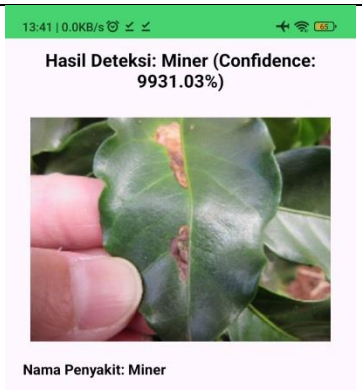
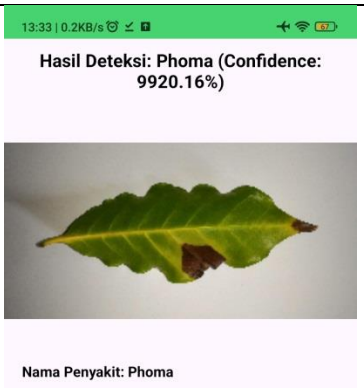
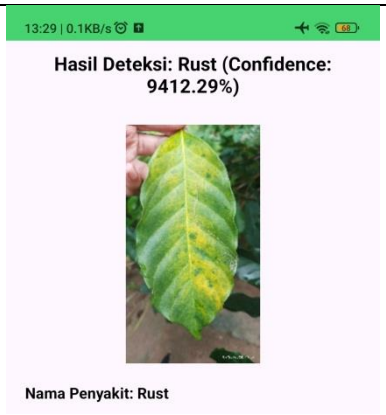
13	Pengguna mengklik tombol tentang aplikasi	Tampilan tentang aplikasi coffee health akan muncul		Sukses

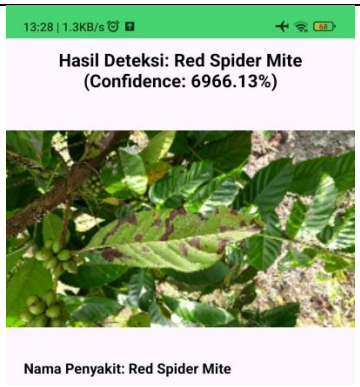
Bisa dilihat pada tabel diatas *user interface* pada aplikasi Coffee Health berjalan dengan baik dan sesuai.

4.11.2 Pengujian Fitur Deteksi Tanaman Pada Aplikasi

Tabel 4.5 Pengujian Fitur Deteksi

Test Cases	Aktual	Hasil
Validasi tanaman kopi dengan daun kopi yang sehat		Daun kopi yang sehat berhasil diklasifikasikan dengan benar. Akurasi: 82.68%
Validasi tanaman kopi dengan daun yang sehat		Daun kopi yang sehat berhasil diklasifikasikan dengan benar. Akurasi: 90.47%

Validasi tanaman kopi dengan daun yang terkana penyakit Miner		Daun kopi yang terkena penyakit Miner berhasil diklasifikasikan dengan benar. Akurasi: 94.74%
Validasi tanaman kopi dengan daun yang terkana penyakit Miner		Daun kopi yang terkena penyakit Miner berhasil diklasifikasikan dengan benar. Akurasi: 99.03%
Validasi tanaman kopi dengan daun yang terkena penyakit Phoma		Daun kopi yang terkena penyakit Phoma berhasil diklasifikasikan dengan benar. Akurasi: 99.16%
Validasi tanaman kopi dengan daun yang terkena penyakit Rust		Daun kopi yang terkena penyakit Rust berhasil diklasifikasikan dengan benar. Akurasi: 94.29%

Tanaman kopi dengan daun yang terkena penyakit Red Spider Mite		Daun kopi yang terkena penyakit Red Spider Mite berhasil di klasifikasikan dengan benar. Akurasi: 69.13%

Berdasarkan dari hasil tabel uji coba, bisa diperhatikan bahwa fitur pendekteksian penyakit yang dialami tanaman kopi bisa bekerja dengan baik, dari setiap data telah berhasil diklasifikasikan dengan benar oleh Model CNN. Sehingga dapat diperoleh tingkat hasil akurasi tertinggi sebesar 99.16%, sedangkan tingkat hasil akurasi terendah sebesar 69.13%. Dari tingkat akurasi sebesar 99.16% menunjukkan bahwa model sudah cukup baik dalam mendeteksi penyakit pada tanaman kopi, berdasarkan uji coba yang telah dilakukan.

4.12 Pembahasan

Berdasarkan dari penjelasan penyakit pada daun kopi tipe arabika yang dibahas dalam bab sebelumnya, yang mana telah ditemukan 4 jenis penyakit yang berasal dari hama maupun virus yaitu, *Miner*, *Phoma*, *Rust*, dan *Red Spider Mite.* Data yang digunakan oleh model CNN pada penelitian ini diambil dari *platform* Kaggle agar model dapat melakukan klasifikasi dengan akurat dan baik.

Pada penelitian ini dataset yang digunakan berjumlah 3768 gambar yang mencakup beberapa kelas yaitu kelas *healthy*, *miner*, *rust*, *phoma*, dan *red spider mite.* Data yang digunakan dibagi menjadi dua bagian, 80% untuk pelatihan dan 20% untuk pengujian.. Penelitian ini menggunakan model CNN serta TensorFlow yang diintegerasikan pada aplikasi *Android*.

Dari hasil akurasi yang didapatkan melalui *training* data pada Sub bab 4.5 didapatkan nilai sebesar 91%. Setelah model berhasil dibuat serta sudah diuji, langkah selanjutnya yaitu mengimplementasikan model CNN kedalam aplikasi *Android* Coffee Health. Berikut hasil implementasi yang dapat diberikan:

1. Terdapat fitur untuk mendeteksi penyakit pada daun kopi di aplikasi Coffee Health.
2. Terdapat fitur riwayat hasil deteksi yang memungkinkan petani dapat melacak pekembangan penyakit tanaman kopi secara berkala.
3. Masing-masing dari setiap penyakit terdapat penyebab dan juga cara penanganan untuk tanaman kopi yang terkena penyakit.
4. Terdapat fitur cara atau tips memotret daun kopi agar hasil deteksi lebih maksimal.
5. Terdapat juga fitur informasi penyakit beserta sumbernya dan fitur informasi jenis-jenis kopi dan manfaatnya.

Namun masih ada beberapa kelemahan pada aplikasi yaitu:

1. Penggunaan label umum seperti "*red spider mite*" dan "*phoma*" yang tidak terlalu spesifik dikarenakan kurangnya data gambar dapat mengurangi keakuratan aplikasi dalam mendeteksi penyakit tersebut.
2. Penggunaan Model CNN yang lebih akurat dapat dibangung menggunakan arsitektur seperti, *VGG16, InceptionV3,* atau *RestNet50,* yang mana mempunyai kemampuan lebih baik dalam meningkatkan kinerja model.