BAB IV
HASIL DAN PEMBAHASAN

4.1 Hasil *Bussines Understanding*

4.1.1 *Determine Business Objectives*

Tanaman selada merupakan salah satu komoditas sayuran holtikultura yang memiliki permintaan pasar yang tinggi. Namun pada saat tertentu permintaan pasar tidak dapat terpenuhi karena jumlah produksi yang berkurang. Keadaan seperti musim kemarau atau musim pengujan yang berkepanjangan mengakibatkan tanaman selada terkena penyakit. Penyakit yang menyerang tanaman sangat berpengaruh pada jumlah tanaman yang diproduksi.

Menurut Ali Nurmansyah dosen IPB dari departemen produksi pertanian fakultas pertanian serangan penyakit tanaman memiliki pengaruh langsung dan tidak langsung terhadap penurunan hasil panen. Menurut laporannya dalam FAO di tahun 2021 serangan penyakit pada tanaman menurunkan produksi tanaman global hingga 40%. Kerugian ekonomi global ditaksir mencapai lebih dari USD 220 miliar. Kemudian menurut sekjen Perhimpunan Fitopatologi Indonesia (PFI) Prof. Dr. Achmadi Priyatmojo penyakit yang menyerang tanaman dapat merugikan petani antara 20% hingga 35%. Selada memiliki peluang bisnis yang sangat baik. Penelitian yang dilakukan oleh Wildanur Adawiyah menyatakan bahwa permintaan pasar terhadap tanaman selada di CV Spirit Wira mencapai 1-1.5 ton perbulannya dan CV Spirit wira dapat memproduksi 200-250 kg per 500 m2 [56]. Kendala dalam budidaya selada salah satunya adalah gangguan penyakit [57].

Kesalahan dalam penanganan dapat menjadi salah satu faktor yang mengakibatkan berkurangnya hasil pertanian bahkan hingga gagal panen. Kesalahan dalam penanganan disebabkan karena kesalahan dalam mengidentifikasi jenis penyakit yang diderita oleh suatu tanaman. Oleh karena itu dibutuhkan sebuah model yang mampu membantu dalam mengidentifikasi penyakit.

4.1.2 *Asses Situation*

Penelitian yang dilakukan oleh Hindra Hara Arfi Putri Kurnia pada tahun 2018 menyatakan bahwa penyakit yang menyerang selada diantaranya adalah busuk daun yang disebabkan oleh *Bremia Lactuae* dan hawar daun oleh *Alternaria sp* [10], dan penelitian lain yang dilakukan oleh R Yang yang berjudul "*Detection of abnormal hydroponic lettuce leaves based on image processing and machine learning*", Juga menyatakan bahwa abnormalitas pada tanaman selada terlihat pada daun selada. Pada penelitian ini gejala penyakit yang menyerang tanaman selada dibagi menjadi dua yaitu bercak dan busuk. Sehingga kelas pada penelitian adalah sehat, busuk, dan bercak [27]. Dengan pertimbangan dua penelitian diatas peneliti kemudian membagi kelas menjadi 3 kelas yaitu sehat, busuk daun, dan hawar daun.

Proses *deployment* yang ideal pada penelitian ini adalah melakukan *deployment* kedalam perangkat *mobile* dengan mempertimbangkan jumlah penggunaan komputer yang masih terbatas dilinkungan masyarakat perdesaan. Berdasarkan data BPS mengenai persentase penduduk yang menggunakan computer, jumlah pengguna komputer di daerah pedesaan Jawa Barat pada tahun 2021 hanya di angka 4.70%. Tetapi pembuatan perangkat *mobile* menjadi sulit karena keterbatasan perangkat pembuatan aplikasi juga keterbatasan pengetahuan mengenai *mobile development* yang dimiliki oleh peneliti, oleh karena itu pada penelitian ini proses *deployment* akan dilakukan dengan men-*deploy* model ke dalam sebuah *API* dengan menggunakan *FastAPI* dengan harapan dapat lebih mudah dikembangkan dikemudian hari.

4.1.3 *Determine Data Mining Goals*

Proses identifikasi penyakit dilakukan dengan menggunakan algoritma *CNN* yang mana menggunakan data gambar sebagai input. Berdasarkan informasi yang diperoleh dari langkah *determine business objectives* dan *asses situation* diperoleh tujuan *data mining* yaitu membuat sebuah model yang mampu mengidentifikasi penyakit pada tanaman selada. Model diharapkan dapat membedakan jenis penyakit menjadi 3 kelas yang sudah ditentukan. model dibangun menggunakan algoritma *CNN arsitekteur DenseNet201* yang menggunakan data gambar sebagai *input* identifikasi.

4.1.4 *Plan Activities*

Dalam membuat sebuah aplikasi yang identifikasi penyakit secara garis besar dapat dibagi kedalam 3 aktivitas yaitu pengumpulan dan pengolahan data, pembuatan model *machine learning*, dan *deployment model*. Tahapan pertama dilakukan dengan mengumpulkan data gambar penyakit tanaman selada dengan mengunjungi kawasan pertanian dan mengambil gambar penyakit secara mandiri kemudian melakukan *data preprocessing* hingga data dinilai dapat digunakan pada tahapan berikutnya. Tahapan 2 adalah membuat model menggunakan algoritma *CNN* arsitektur DenseNet201 dengan beberapa konfigurasi yang berbeda. Hasil terbaik dari model kemudian akan digunakan pada tahapan selanjutnya yaitu *deployment* kedalam sebuah *API* menggunakan *FastAPI*.

4.2 Hasil *Data Understanding*

4.2.1 *Collect Initial Data*

Berdasarkan informasi yang diperoleh dari tahapan sebelumnya bahwa data yang digunakan pada penelitian ini adalah data gambar tanaman selada baik yang sehat maupun yang terkena penyakit. karena terbatasnya data yang tersedia di internet menyebabkan data yang digunakan adalah data yang dikumpulkan secara manual dengan mengambil gambar tanaman selada secara langsung di perkebunan tanaman selada. Data diambil sebanyak 1.400 gambar menggunakan kamera *handphone* Xiaomi Redmi 9C dengan spesifikasi kamera 13 MP, f/2.2, 28mm (wide), 1.0µm, PDAF. Kemudian data dibagi menjadi 3 kelas berdasarkan ciri ciri penyakit yang terdapat pada tanaman selada. Data yang dikumpulkan tidak sesuai dengan target awal jumlah data yang sudah ditentukan karena terbatasnya tanaman selada yang mengalami salah satu kelas penyakit yaitu hawar daun. Hal ini menyebabkan berkurangnya data yang diperoleh pada kelas hawar daun yang tadinya 500 menjadi 400. Sehingga total data yang diperoleh adalah 1.400 gambar.

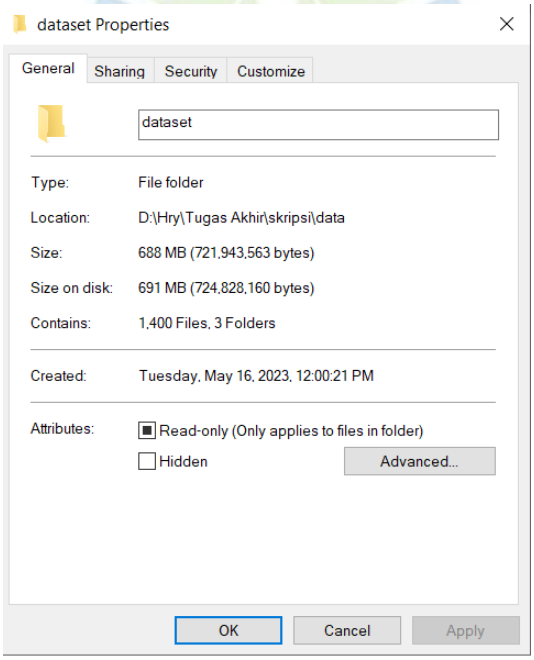
4.2.2 *Describe Data*

Setelah data terkumpul kemudian dilakukan pendeskripsian data agar lebih memahami karakteristik data pada setiap kelas yang ada. Yang pertama kelas sehat. Kelas sehat memiliki ciri berwarna hijau cerah dan tidak terbapat noda penyakit pada daun. Kelas yang kedua adalah hawar daun atau bercak daun. Hawar daun

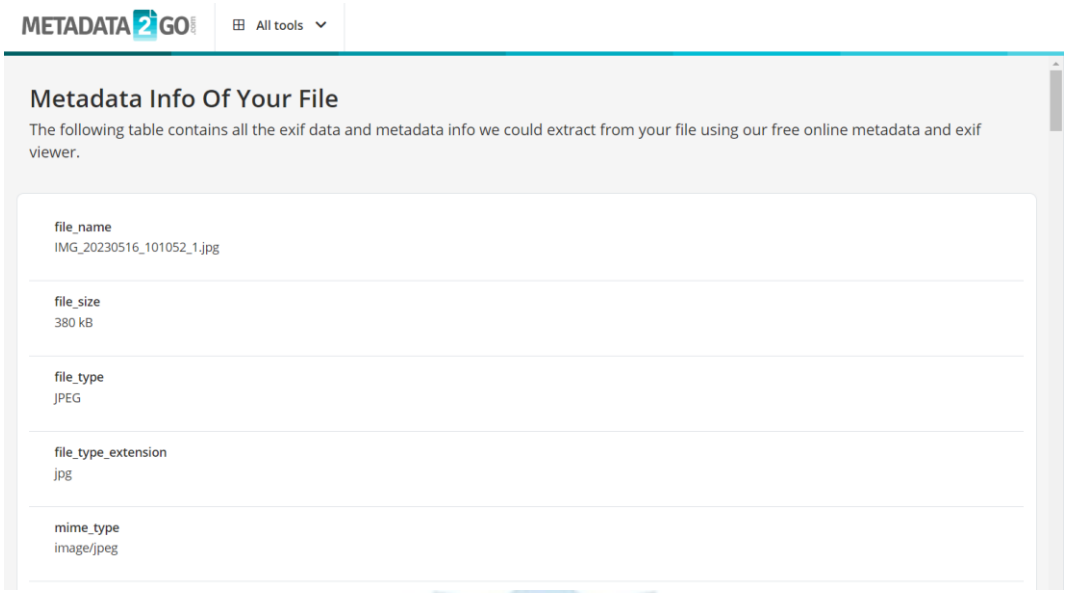
terlihat ada bercak-bercak coklat tua sampai hampir hitam. Bercak berbentuk hitam berukuran kecil namun jika dibiarkan akan membesar dan bergabung menjadi satu serangan dimulai dari daun bawah kemudian naik keatas. Terkadang daunnya akan berlubang karena bercak bercak itu mengering lalu jatuh, dan daun menggulung kriting. Yang terakhir busuk daun (*downy mildew*). Busuk daun memiliki ciri ciri pada tulang-tulang daun terjadi bercak bersudut, bewarna hijau muda pucat sampai kuning menghitam. Pada permukaan bawah daun dapat berbentuk kapang (jamur) putih. Bagian daun yang terinfeksi saling berhubungan dan berubah menjadi bercak yang besar.

4.2.3 *Explore Data*

Pada tahapan *explore data* data yang sudah dikumpulkan sebelumnya akan explore untuk mencari informasi yang ada pada data. Pada tahapan *explore data* peneliti menggunakan *file manager* dan metadata2go sebagai alat eksplorasi. Penggunaan *tools file manager* untuk eksplorasi data digambarkan pada gambar 4.1 dan proses eksplorasi dengan *tools* metadata2go digambarkan oleh gambar 4.2



Gambar 4.1 Eksplorasi *dataset property*



Gambar 4.2 Pemanfaatan metadata2go untuk explorasi data

Hasil eksplorasi diperoleh sebagai berikut:

1. Total data 1.400 gambar
2. Data terbagi menjadi 3 kelas 400 hawar daun, 500 busuk daun, dan 500 sehat.
3. *File Type*: JPEG
4. *File_type*_extension: jpg
5. *Mime_type*: image/jpeg
6. *X_resolution*: 72
7. *Y_resolution*: 72
8. *Image_width*: 3120
9. *Image_height*: 3120
10. *Bits_per_sample*: 8
11. *Color_components*: 3
12. *Mega_pixel*: 8.2

4.3 Hasil *Data Preparation*

4.3.1 *Select data*

Berdasarkan informasi yang diperoleh pada tahapan selanjutnya maka dilanjutkan pada tahap pemilihan data. Data diseleksi dan dipilih agar sesuai dengan apa yang dibutuhkan peneliti. Gambar yang tidak memiliki keterkaitan dan tidak

jelas dieliminasi. Proses pemilihan data dilakukan dengan cara melihat gambar secara manual dan menyesuaikan gambar dengan deskripsi data yang ada sebelumnya. total data yang sudah diseleksi pada tahap ini berjumlah 1.381 gambar dengan pembagian 500 kelas sehat, 490 kelas busuk daun, dan 391 kelas hawar daun.

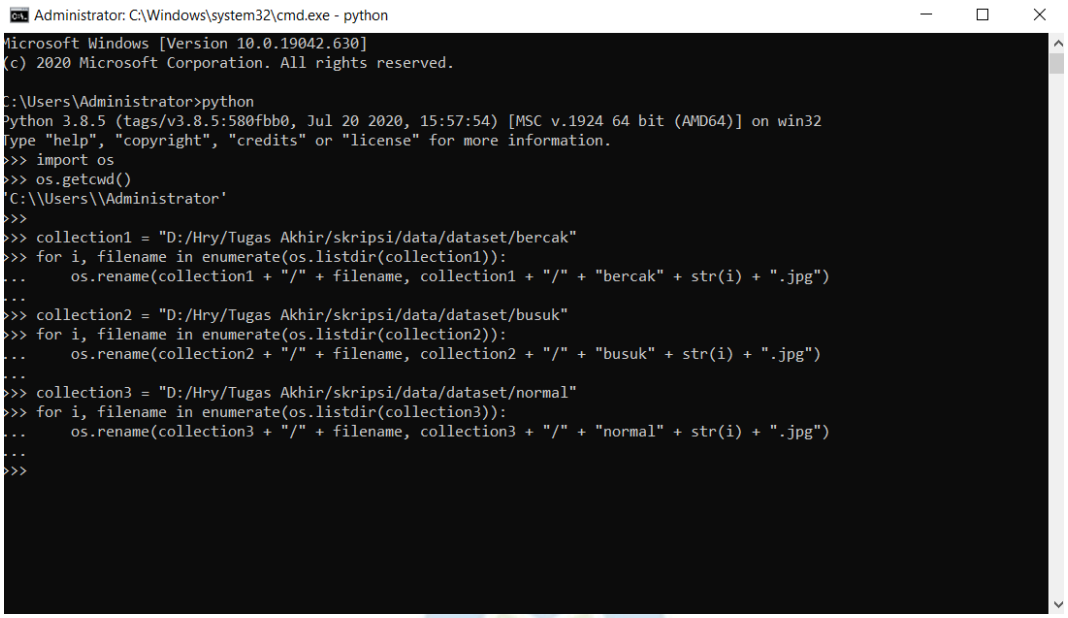
4.3.2 *Data Preprocessing*

Tahapan *data preprocessing* merupakan salah satu tahapan terpenting dalam proses *machine learning*. Dengan melakukan *data preprocessing* yang baik akan memudahkan dan meningkatkan kualitas model *machine learning*. Pada tahapan data *preprocesing* ada beberapa proses yang dilakukan. Adapun proses yang dilakukan sebagai berikut:

1. Mengubah nama
2. Import data kedalam *coding environment*
3. Melakukan *split data*
4. Melakukan *konfigurasi data tambahan*

1. Mengubah Nama

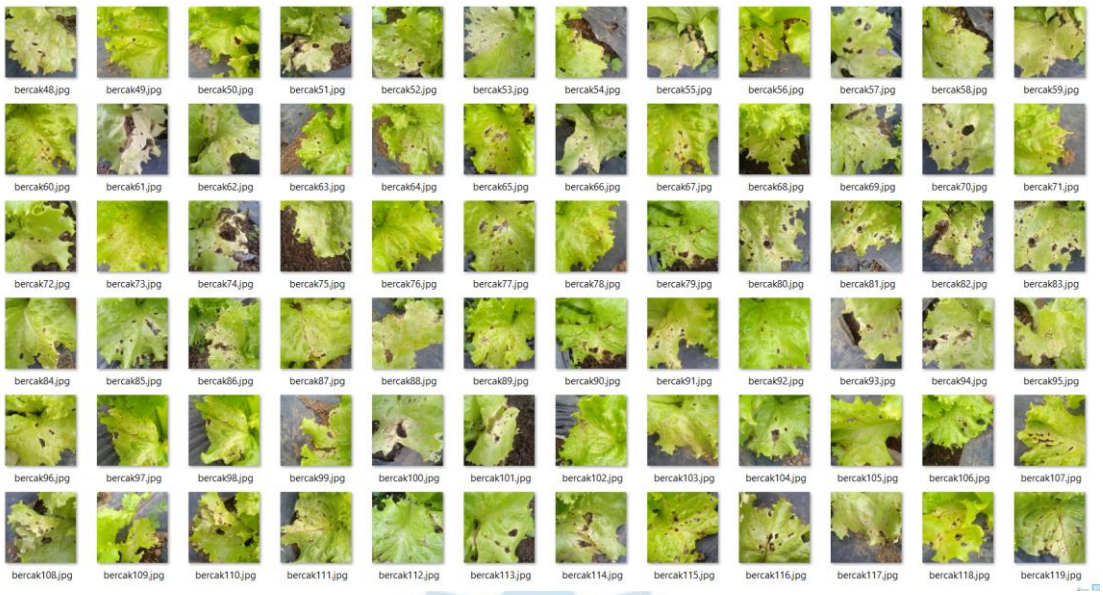
Pada penelitian ini gambar yang digunakan merupakan data yang diambil menggunakan kamera sehingga memiliki nama yang sulit diidentifikasi. Sehingga Mengubah nama data dilakukan dengan tujuan memudahkan peneliti dalam mengidentifikasi kelas data. Proses perubahan nama dilakukan dengan menggunakan bahasa pemograman *python* melalui *CMD* (*Command Prompt*).



Gambar 4.3 Proses mengubah nama seluruh data sesuai kelas

Pada gambar 4.3 menunjukan proses pengubahan nama file menggunakan bahasa pemograman pyhon melalui *CMD* dengan bantuan *library os*. Perubahan nama dilakukan pada 3 folder sesuai dengan kelas yang ditentukan sebelumnya. hasil perubahan nama gambar ditampilkan pada gambar 4.4 untuk sebelum perubahan dan 4.5 untuk sesudah perubahan.

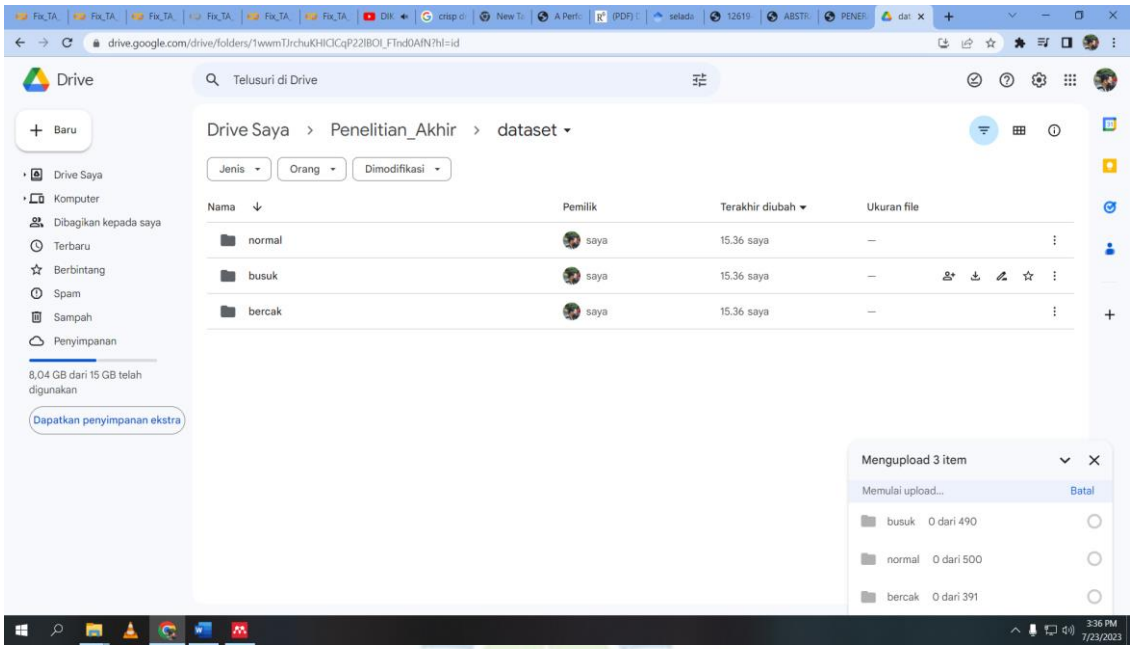
Gambar 4.4 Sebelum mengubah nama seluruh gambar sesuai kelas



Gambar 4.5 Sesudah mengubah nama seluruh gambar sesuai kelas

2. Import data

Selanjutnya data yang sudah terpilih dan disesuaikan namanya akan di-*upload* kedalam *cloud* agar dapat di-*import* kedalam *coding environment* kapanpun dan dimanapun. Terdapat dua tahapan yang dilakukan pada *import* data ini. Yang pertama adalah meng-*upload* data kedalam *cloud*. *Upload* data kedalam *cloud* bertujuan untuk memudahkan dan mempercepat proses *import* data kedalam *coding environment*. *Cloud* yang digunakan pada penelitian ini adalah google drive. Proses *upload* data kedalam google drive digambarkan pada gambar 4.6.



Gambar 4.6 *Upload* data ke google drive

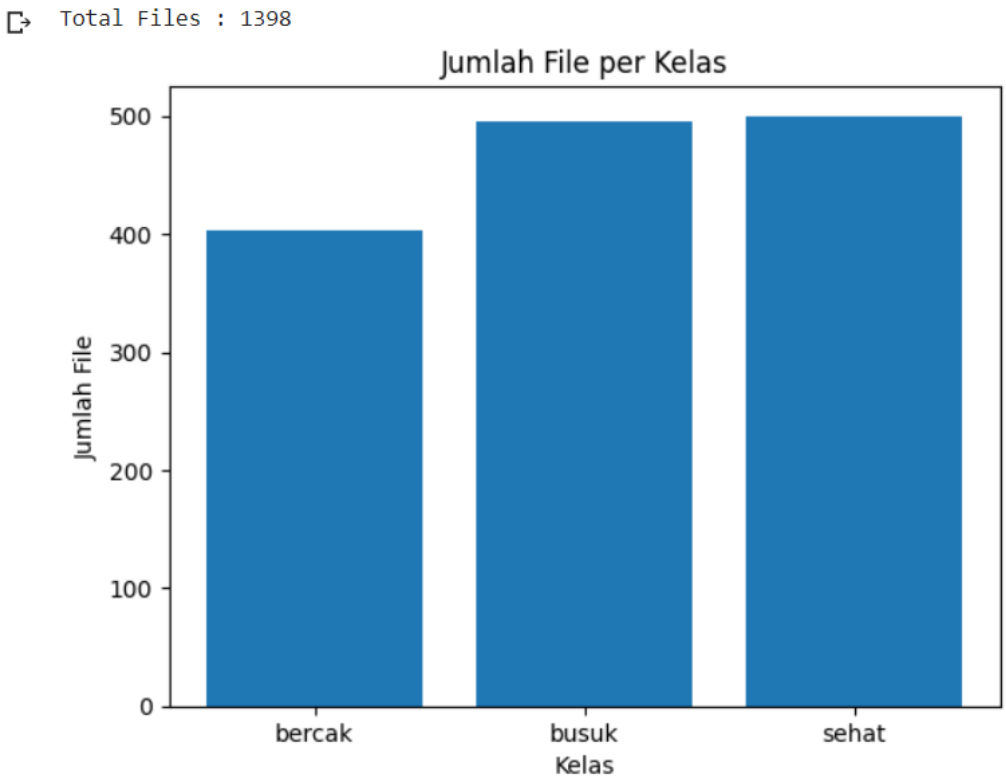
Setelah meng-*upload* data ke dalam google drive tahapan selanjutnya adalah melakukan *import data* kedalam *google collab* sebagai *coding environment* yang digunakan. Proses *import* dilakukan dengan menggunakan *library* google colab. Proses *import data* digambarkan pada gambar 4.7.

```
from google.colab import drive
drive.mount("/content/drive/")

dir = "/content/drive/MyDrive/Tugas_Akhir/dataset"
```

```
Mounted at /content/drive/
```

Gambar 4.7 *Import Data*



Gambar 4.8 Visualisasi gambar yang sudah di-*import*

Pada gambar 4.8 menunjukan informasi data gambar yang berhasil di-import menggunakan *library matplotlib pyplot*. dapat dilihat jumlah gambar yang berhasil di-*import* berjumlah 1.381 data dengan penyebaran 500 kelas sehat, 490 kelas busuk, dan 391 kelas hawar atau bercak.

3. Melakukan *Split Data*

Pada tahapan *split data*, data akan dibagi menjadi 2 yaitu *data train* dan *data valid*. *Data train* digunakan untuk *training model*, dan *data valid* digunakan sebagai *validation* yang nantinya menjadi pembanding model dan informasi hasil sebagai bahan evaluasi. *Split data* dilakukan dengan menggunakan salah satu *class* dari *library* tensorflow yaitu *image generator*. Data dibagi menjadi dua dengan perbandingan 7:3. 7 untuk *data train* dan 3 untuk *data validation*. *Source code* dari proses *split data* digambarkan pada gambar 4.9.

```
import tensorflow as tf

data = tf.keras.preprocessing.image.ImageDataGenerator(
    validation_split=0.3
)
```

Gambar 4.9 *Source code split data*

4. Melakukan Konfigurasi Data

Tahapan selanjutnya adalah konfigurasi data. data akan disesuakan dengan apa yang dibutuhkan model konfigurasi dilakukan dengan menggunakan *class flow_from_directory* dari *tensorflow*. Konfigurasi yang dilakukan diantaranya melakukan *resize* yaitu menyamakan semua gambar menjadi ukuran 200x200, lalu mengatur *class mode* menjadi categorical, mengatur *subset* menjadi *training* dan *validation* sebagai lanjutan dari tahapan sebelumnya, dan mengatur *batch size* menjadi 32. Proses konfigurasi dan hasilnya digambarkan oleh gambar 4.10.

```
image_size = (200,200)
batch_size = 32
seed = 999

train_data = data.flow_from_directory(
    dir,
    class_mode='categorical',
    subset='training',
    target_size=image_size,
    batch_size=batch_size,
    seed=seed
)

valid_data = data.flow_from_directory(
    dir,
    class_mode='categorical',
    subset='validation',
    target_size=image_size,
    batch_size=batch_size,
    seed=seed
)
```

```
Found 986 images belonging to 3 classes.
Found 420 images belonging to 3 classes.
```

Gambar 4.10 Proses konfigurasi data

4.2.3 *Data Augmentation*

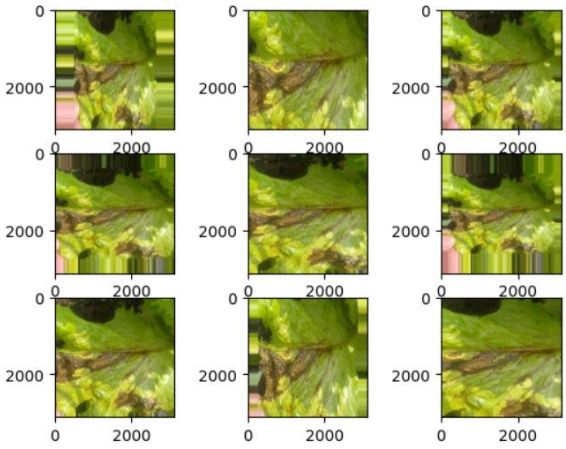
Tahapan selanjutnya adalah tahapan terakhir dari *data preparation* yaitu data augmentasi. Augmentasi data berfungsi untuk menambah jumlah data yang akan digunakan pada model. Augmentasi data dilakukan karena data jumlah data yang terbatas. Semakin banyak gambar maka sebuah model akan semakin baik dalam mengidentifikasi. Untuk augmentasi sendiri hanya dilakukan pada *data*

training. Tahapan augmentasi dilakukan menggunakan bantuan *library tensorflow.* Ada 3 jenis augmentasi yang dilakukan yaitu *random flip*, *random rotation*, dan *random zoom.* Kemudian semua gambar termasuk hasil augmentasi akan di-*rescale* menjadi 1./225. *Source code* dari proses augmentasi digambarkan pada gambar 4.11.

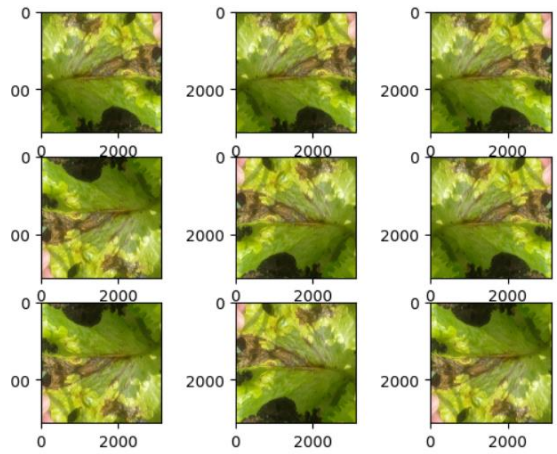
```
[ ]  data_augmentation = tf.keras.Sequential(
       [
         tf.keras.layers.RandomZoom(0.5),
         tf.keras.layers.RandomFlip("horizontal_and_vertical"),
         tf.keras.layers.RandomRotation(0.5),
         tf.keras.layers.Rescaling(1./255)
       ]
     )
```

Gambar 4.11 *Source code* augmentasi data

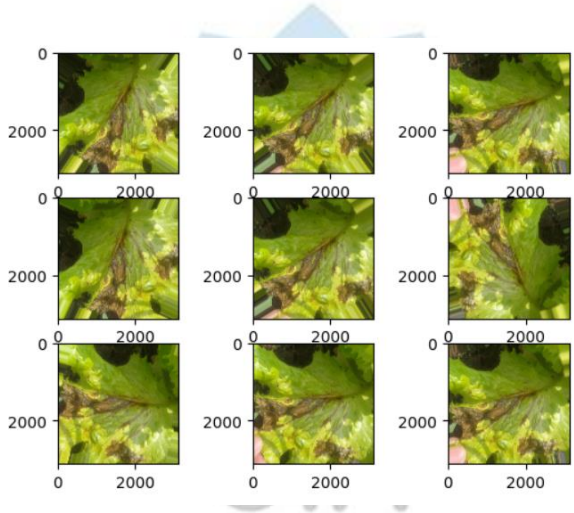
Karena proses augmentasi dilakukan menggunakan *keras sequential* maka proses augmentasi dilakukan ketika proses *training*. Oleh karana itu ilustrasi hasil augmentasi diperoleh dengan melakukan augmentasi terpisah menggunakan *image data generator*. Ilustrasi hasil dari agumentasi digambarkan pada gambar 4.12 hingga 4.14.



Gambar 4.12 Ilustrasi augmentasi *random zoom*



Gambar 4.13 Ilustrasi augmentasi *ramdom flip*



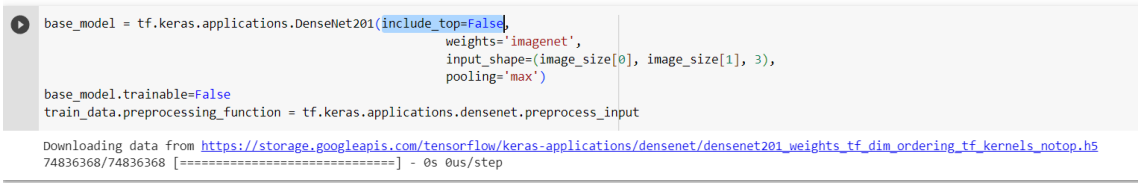
Gambar 4.14 Ilustrasi augmentasi *ramdom rotation*

4.4 Hasil *Modeling*

4.4.1 *Select Model Technique*

Model yang digunakan pada penelitian ini adalah salah satu arsitektur dari model pembelajaran mesin *Convolutional Neural Network* yaitu *DenseNet201*. Proses *modeling* dilakukan menggunakan bahasa pemograman *python* dan dibuat menggunakan google colabs sebagai *coding environment*. model *machine learning* yang digunakan pada penelitian ini adalah arsitektur *DenseNet201* yang dikembangkan oleh *tensorflow keras* dengan *configurasi include_top = False, weight = imagetnet, input_shape* 200,200,3 dan *pooling = max*. kemudian peneliti tidak mengatur *classifier activation* dan *classes configurasi* pada *DenseNet* namun menambahkan *Dense Layer* dengan *softmax activation* sebagai *output layer*. Proses

load model arsitektur DenseNet digambarkan pada gambar 4.15. kemudia karena proses augmentasi dilakukan ketika proses *training* maka model yang digunakan pada akan seperti yang digambarkan pada gambar 4.16.



Gambar 4.15 Proses *load model DenseNet201*

```
densenet_model = tf.keras.models.Sequential([
  data_augmentation,
  base_model,
  tf.keras.layers.Dense(3, activation='softmax')
])
```

Gambar 4.16 *Source code* pembuatan model

4.4.2 *Generate Test Design*

Test design pada penelitian ini dilakukan berdasarkan penelitian yang sudah dilakukan pada penelitian yang sudah dilakukan peneliti lain sebelumnya diantaranya penelitian yang dilakukan oleh Primatua Sitompul pada tahun 2022, penelitian ini menggunakan arsitektur *DenseNet201* dengan configurasi *optimizer Adam* dengan *learning rate* sebesar 0,0001 [53], kemudian penelitian lain yang dilakukan oleh Jasman Pardede pada tahun 2020, penelitian ini menggunakan *optimizer SGD* dengan *learning rate 0,01* [16], penelitian lain yang dilakukan oleh Diffran Nur Cahyo dkk mengenai perbandingan *optimizer* menunjukan bahwa *RMSprop* dengan *learning rate* 0,0001 menunjukan hasil yang terbaik pada arsitektur lain [58]. Terakhir Aditya Fahreza dkk melakukan penelitian yang menggunakan berbagai *optimizer* dan *learning rate* pada arsitektur *DenseNet*. Pada penelitian yang dilakukan Fahreza dkk *optimizer Adam* dan *RMSprop* memperoleh nilai akurasi yang baik yaitu sebesar 98% dan 99% [55]. Berdasarkan referensi yang dijelaskan sebelumnya maka pada penelitian ini akan melakukan 18 skenario percobaan berbeda dengan kombinasi *optimizer, learning rate, dan epoch* Jumlah *epoch* pada penelitian ini hanya 5 dan 10 karena pada penelitian ini ingin mencari

akurasi terbaik dengan waktu *training* paling sedikit sehingga jumlah *epoch* yang dilakukan terbilang sedikit dibanding penelitian lainya. Oleh karena itu, berdasarkan informasi yang sudah dikumpulkan sebelumnya diperolehlah *test skenario* seperti yang disajikan pada tabel 4.1 berikut.

Tabel 4.1 Skenario Pengujian

skenario	*Optimizer*	*Learning Rate*	*epoch*
1	*Adam*	0,001	5
2	*RMSprop*	0,001	5
3	*SGD*	0,001	5
4	*Adam*	0,001	10
5	*RMSprop*	0,001	10
6	*SGD*	0,001	10
7	*Adam*	0,01	5
8	*RMSprop*	0,01	5
9	*SGD*	0,01	5
10	*Adam*	0,01	10
11	*RMSprop*	0,01	10
12	*SGD*	0,01	10
13	*Adam*	0,0001	5
14	*RMSprop*	0,0001	5
15	*SGD*	0,0001	5
16	*Adam*	0,0001	10
17	*RMSprop*	0,0001	10
18	*SGD*	0,0001	10

4.4.3 *Build The Model*

1. Skenario 1

```
[ ]  densenet_model_1 = densenet_model
     densenet_model_1.compile(
         loss='categorical_crossentropy',
         optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
         metrics=['accuracy']
       )
```

```
     densenet_hist = densenet_model_1.fit(
         train_data,
         epochs=5,
         validation_data = valid_data
     )
```

Gambar 4.17 *Source Code* Model Skenario 1

Gambar 4.17 mengambarkan *source code* untuk model 1. Pada model skenario 1 *optimizer* yang digunakan adalah *categorical_crossentropy* sebagai *loss, Adam* dengan *learning rate* sebesar 0,001, *matrics accuracy*, dan jumlah *epoch* 5.

```
1/5
[==============================] - 667s 21s/step - loss: 0.9414 - accuracy: 0.6327 - val_loss: 0.4863 - val_accuracy: 0.8086
2/5
[==============================] - 345s 11s/step - loss: 0.3610 - accuracy: 0.8500 - val_loss: 0.4091 - val_accuracy: 0.8541
3/5
[==============================] - 269s 9s/step - loss: 0.3615 - accuracy: 0.8653 - val_loss: 0.3934 - val_accuracy: 0.8636
4/5
[==============================] - 273s 9s/step - loss: 0.2371 - accuracy: 0.9020 - val_loss: 0.3542 - val_accuracy: 0.8660
5/5
[==============================] - 267s 9s/step - loss: 0.2447 - accuracy: 0.9071 - val_loss: 0.2846 - val_accuracy: 0.9067
```

Gambar 4.18 Hasil Pengujian Skenario 1

Hasil pengujian model skenario 1 ditampilkan pada gambar 4.18. Skenario 1 membutuhkan waktu *training* sebesar 1821 detik atau 30 menit 21 detik. Hasil dari proses *training* model skenario 1 akurasi yang diperoleh adalah 90% dan akurasi validasi adalah 90%.

2. Skenario 2

```
model_skenario_2 = densenet_model
model_skenario_2.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy']
  )
```

```
model_hist_2 = model_skenario_2.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.19 *Source Code* Model Skenario 2

Gambar 4.19 menunjukan *source code* model skneario ke 2. Pada model skenario 2 configurasi yang digunakan adalah *categorical_crossentropy* sebagai *loss*, *optimizer RMSprop* dengan *learning rate* sebesar 0,001, *matrics accuracy*, dan jumlah epoch 5.

```
1/5
[==============================] - 286s 9s/step - loss: 0.9389 - accuracy: 0.6847 - val_loss: 0.5008 - val_accuracy: 0.8110
2/5
[==============================] - 262s 8s/step - loss: 0.5479 - accuracy: 0.7857 - val_loss: 0.4439 - val_accuracy: 0.8660
3/5
[==============================] - 266s 9s/step - loss: 0.3492 - accuracy: 0.8612 - val_loss: 0.3852 - val_accuracy: 0.8589
4/5
[==============================] - 265s 9s/step - loss: 0.3216 - accuracy: 0.8735 - val_loss: 0.2572 - val_accuracy: 0.8923
5/5
[==============================] - 265s 9s/step - loss: 0.2993 - accuracy: 0.8847 - val_loss: 0.2838 - val_accuracy: 0.9067
```

Gambar 4.20 Hasil Pengujian Skenario 2

Skenario 2 membutuhkan waktu *training* sebesar 1344 detik atau 22 menit 24 detik. Hasil dari proses *training* skenario 2 digambarkan pada gambar 4.20. seperti yang dapat kita lihat akurasi *training* dan akurasi validasi yang diperoleh adalah 88% dan 90%.

3. Skenario 3

```
model_skenario_3 = densenet_model
model_skenario_3.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
    metrics=['accuracy']
  )

model_hist_3 = model_skenario_3.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.21 *Source Code* Model Skenario 3

Gambar 4.21 menampilkan *source code* dari model skenario 3. Skenario ke memiliki konfigurasi *loss categorical_crossentropy*, *optimizer SGD* dengan *learning rate 0,001*, *metrics accuracy* dan *epoch 10.*

```
1/5
[==============================] - 281s 9s/step - loss: 3.7417 - accuracy: 0.6276 - val_loss: 1.1808 - val_accuracy: 0.7871
2/5
[==============================] - 264s 9s/step - loss: 0.7748 - accuracy: 0.8490 - val_loss: 0.3738 - val_accuracy: 0.9211
3/5
[==============================] - 261s 9s/step - loss: 0.5807 - accuracy: 0.8776 - val_loss: 0.9554 - val_accuracy: 0.8158
4/5
[==============================] - 264s 9s/step - loss: 0.6935 - accuracy: 0.8582 - val_loss: 0.3446 - val_accuracy: 0.9163
5/5
[==============================] - 264s 9s/step - loss: 0.3165 - accuracy: 0.9102 - val_loss: 0.3768 - val_accuracy: 0.9211
```

Gambar 4.22 Hasil Pengujian Skenario 3

Skenario 3 memakan waktu sebanyak 1334 detik atau 22 menit 14 detik. Seperti yang ditampilkan pada gambar 4.22 Akurasi yang diperoleh adalah 91% dan *validation* akurasi yang diperoleh adalah 92%.

4. Skenario 4

```
model_skenario_4 = densenet_model
model_skenario_4.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy']
  )
```

```
model_hist_4 = model_skenario_4.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.23 *Source Code* Model Skenario 4

Gambar 4.23 merupakan *source code* dari model skenario 4. Skenario ke memiliki konfigurasi *loss categorical_crossentropy, optimizer Adam* dengan *learning rate 0,001, metrics accuracy* dan *epoch* 10.

```
1/10
[==============================] - 489s 15s/step - loss: 0.8309 - accuracy: 0.6694 - val_loss: 0.4076 - val_accuracy: 0.8397
2/10
[==============================] - 265s 9s/step - loss: 0.3337 - accuracy: 0.8673 - val_loss: 0.3425 - val_accuracy: 0.8900
3/10
[==============================] - 269s 9s/step - loss: 0.3358 - accuracy: 0.8796 - val_loss: 0.3010 - val_accuracy: 0.8995
4/10
[==============================] - 260s 8s/step - loss: 0.2635 - accuracy: 0.9020 - val_loss: 0.2734 - val_accuracy: 0.9139
5/10
[==============================] - 269s 9s/step - loss: 0.2354 - accuracy: 0.9102 - val_loss: 0.2681 - val_accuracy: 0.9115
6/10
[==============================] - 267s 9s/step - loss: 0.2165 - accuracy: 0.9255 - val_loss: 0.2646 - val_accuracy: 0.8995
7/10
[==============================] - 261s 8s/step - loss: 0.2170 - accuracy: 0.9194 - val_loss: 0.2935 - val_accuracy: 0.9043
8/10
[==============================] - 265s 9s/step - loss: 0.1898 - accuracy: 0.9214 - val_loss: 0.2280 - val_accuracy: 0.9234
9/10
[==============================] - 265s 9s/step - loss: 0.1862 - accuracy: 0.9286 - val_loss: 0.2554 - val_accuracy: 0.9234
10/10
[==============================] - 263s 8s/step - loss: 0.1614 - accuracy: 0.9439 - val_loss: 0.2270 - val_accuracy: 0.9234
```

Gambar 4.24 Hasil Pengujian Skenario 4

Waktu *training* yang dibutuhkan adalah 2873 detik atau 47 menit 53 detik. Pada gambar 4.24 menunjukan hasil dari proses *training* skenario ke 4. Akurasi yang diperoleh adalah 94% untuk *train* dan 92% untuk *validation*.

5. Skenario 5.

```
model_skenario_5 = densenet_model
model_skenario_5.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001),
    metrics=['accuracy']
  )
```

```
model_hist_5 = model_skenario_5.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.25 *Source Code* Model Skenario 5

Gambar 4.25 merupakan *source code* dari model skenario 5. Konfigurasi pada skenario ke 5 adalah loss *categorical_crossentropy*, *optimizer RMSprop* dengan *learning rate* 0,001, *metrics accuracy* dan *epoch* 10.

```
1/10
[==============================] - 278s 9s/step - loss: 0.9649 - accuracy: 0.6541 - val_loss: 0.4613 - val_accuracy: 0.8301
2/10
[==============================] - 263s 8s/step - loss: 0.4709 - accuracy: 0.8235 - val_loss: 0.4044 - val_accuracy: 0.8445
3/10
[==============================] - 262s 8s/step - loss: 0.3589 - accuracy: 0.8551 - val_loss: 0.2862 - val_accuracy: 0.8900
4/10
[==============================] - 264s 8s/step - loss: 0.3329 - accuracy: 0.8898 - val_loss: 0.3017 - val_accuracy: 0.8876
5/10
[==============================] - 257s 8s/step - loss: 0.3100 - accuracy: 0.8827 - val_loss: 0.2318 - val_accuracy: 0.9282
6/10
[==============================] - 266s 9s/step - loss: 0.2429 - accuracy: 0.9112 - val_loss: 0.2136 - val_accuracy: 0.9187
7/10
[==============================] - 260s 8s/step - loss: 0.2368 - accuracy: 0.9041 - val_loss: 0.2040 - val_accuracy: 0.9306
8/10
[==============================] - 262s 8s/step - loss: 0.2291 - accuracy: 0.9112 - val_loss: 0.3610 - val_accuracy: 0.8589
9/10
[==============================] - 264s 8s/step - loss: 0.2138 - accuracy: 0.9265 - val_loss: 0.2011 - val_accuracy: 0.9330
10/10
[==============================] - 263s 8s/step - loss: 0.1815 - accuracy: 0.9276 - val_loss: 0.2266 - val_accuracy: 0.9282
```


Gambar 4.26 Hasil Pengujian Skenario 5

Waktu *training* yang dibutuhkan adalah 2639 detik atau 43 menit 59 detik. Pada gambar 4.26 menunjukan hasil dari proses *training* skenario ke 4. Akurasi yang diperoleh adalah 92% untuk train dan 93% untuk *validation*. pada skenario ke 3 *validation* akurasi pada *epoch* ke 7 menyentuh angka 93% namun berkurang pada *epoch* selanjutnya.

6. Skenario 6

```
model_skenario_6 = densenet_model
model_skenario_6.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
    metrics=['accuracy']
  )
```

```
model_hist_6 = model_skenario_6.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.27 *Source Code* Model Skenario 6

Gambar 4.27 merupakan *source code* dari model skenario 6. Konfigurasi pada skenario ke 6 adalah *loss categorical_crossentropy*, *optimizer SGD dengan learning rate* 0,001, *metrics accuracy* dan *epoch* 10.

```
1/10
[==============================] - 272s 9s/step - loss: 3.9270 - accuracy: 0.6388 - val_loss: 0.4488 - val_accuracy: 0.8995
2/10
[==============================] - 257s 8s/step - loss: 0.4527 - accuracy: 0.8857 - val_loss: 0.9150 - val_accuracy: 0.8134
3/10
[==============================] - 263s 8s/step - loss: 0.4843 - accuracy: 0.8796 - val_loss: 0.5965 - val_accuracy: 0.8780
4/10
[==============================] - 256s 8s/step - loss: 0.4122 - accuracy: 0.8898 - val_loss: 1.3043 - val_accuracy: 0.7344
5/10
[==============================] - 261s 8s/step - loss: 0.7173 - accuracy: 0.8796 - val_loss: 0.4051 - val_accuracy: 0.9187
6/10
[==============================] - 263s 8s/step - loss: 0.2895 - accuracy: 0.9214 - val_loss: 0.2847 - val_accuracy: 0.9402
7/10
[==============================] - 263s 8s/step - loss: 0.4224 - accuracy: 0.9000 - val_loss: 0.3804 - val_accuracy: 0.9211
8/10
[==============================] - 262s 8s/step - loss: 0.3271 - accuracy: 0.9092 - val_loss: 0.3061 - val_accuracy: 0.9282
9/10
[==============================] - 266s 9s/step - loss: 0.3638 - accuracy: 0.9163 - val_loss: 0.2617 - val_accuracy: 0.9354
10/10
[==============================] - 262s 8s/step - loss: 0.2904 - accuracy: 0.9184 - val_loss: 0.2493 - val_accuracy: 0.9258
```

Gambar 4.28 Hasil Pengujian Skenario 6

Menurut hasil yang digambarkan pada gambar 4.28. Total waktu dalam proses *training* skenario ke 6 adalah 2625 detik atau 43 menit 45 detik. Akurasi yang diperoleh adalah 91% untuk *training* dan 92% untuk *validation*.

7. Skenario 7

```
densenet_model_7.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    metrics=['accuracy']
  )
```

```
densenet_hist_7 = densenet_model_7.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.29 *Source Code* Model Skenario 7

Gambar 4.29 menampilan *source code* dari model skenario 7. Konfigurasi pada skenario ke 7 adalah *loss categorical_crossentropy*, *optimizer adam dengan learning rate* 0,01, *metrics accuracy* dan *epoch* 5.

```
1/5
[==============================] - 573s 18s/step - loss: 2.8788 - accuracy: 0.6418 - val_loss: 1.0206 - val_accuracy: 0.8684
2/5
[==============================] - 467s 15s/step - loss: 0.5444 - accuracy: 0.8990 - val_loss: 0.4826 - val_accuracy: 0.9187
3/5
[==============================] - 472s 15s/step - loss: 0.3891 - accuracy: 0.9061 - val_loss: 0.6091 - val_accuracy: 0.8780
4/5
[==============================] - 474s 15s/step - loss: 0.5430 - accuracy: 0.8837 - val_loss: 0.4346 - val_accuracy: 0.9330
5/5
[==============================] - 461s 15s/step - loss: 0.3931 - accuracy: 0.9163 - val_loss: 0.3891 - val_accuracy: 0.9306
```

Gambar 4.30 Hasil Pengujian Skenario 7

Waktu *training* yang dihabiskan adalah 2447 detik. Gambar 4.30 menunjukan hasil dari *training* model skenario ke 7. Akurasi yang diperoleh adalah 91% dan 93% untuk *validation accuracy*.

8. Skenario 8

```
model_skenario_8 = densenet_model
model_skenario_8.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.01),
    metrics=['accuracy']
  )
```

```
model_hist_8 = model_skenario_8.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.31 *Source Code* Model Skenario 8

Selanjutnya gambar 4.31 menampilkan *source code* untuk model skenario 8. Konfigurasi pada skenario ke 8 adalah loss *categorical_crossentropy*, *optimizer RMSprop* dengan *learning rate* 0,01, *metrics accuracy* dan *epoch* 5.

```
1/5
[==============================] - 488s 15s/step - loss: 5.9196 - accuracy: 0.6388 - val_loss: 0.5750 - val_accuracy: 0.8780
2/5
[==============================] - 454s 15s/step - loss: 2.5638 - accuracy: 0.7592 - val_loss: 0.6176 - val_accuracy: 0.8971
3/5
[==============================] - 462s 15s/step - loss: 1.6550 - accuracy: 0.8255 - val_loss: 3.0329 - val_accuracy: 0.7536
4/5
[==============================] - 465s 15s/step - loss: 1.8631 - accuracy: 0.8429 - val_loss: 5.6151 - val_accuracy: 0.6746
5/5
[==============================] - 462s 15s/step - loss: 1.4974 - accuracy: 0.8765 - val_loss: 0.8783 - val_accuracy: 0.9067
```

Gambar 4.32 Hasil Pengujian Skenario 8

Proses *training* skenario 8 membutuhkan waktu sebanyak 2331 detik atau 38 menit 51 detik. Hasil dari proses *training* skenario 8 ditampilkan pada gambar 4.32. Nilai akurasi dari skenario 8 adalah 87% dan nilai *validation* akurasi yang diperoleh adalah 90%.

9. Skenario 9

```
model_skenario_9 = densenet_model
model_skenario_9.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
    metrics=['accuracy']
  )
```

```
model_hist_9 = model_skenario_9.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.33 *Source Code* Model Skenario 9

Gambar 4.33 menampilkan *source code* dari model skenario 9. Pada skenario 9 ini konfigurasi yang digunakan adalah *loss categorical_crossentropy*, *optimizer RMSprop* dengan *learning rate* 0,01, *metrics accuracy* dan *epoch* 5.

```
Epoch 1/5
31/31 [==============================] - 489s 15s/step - loss: 3.7650 - accuracy: 0.6469 - val_loss: 0.4557 - val_accuracy: 0.8971
Epoch 2/5
31/31 [==============================] - 463s 15s/step - loss: 0.7987 - accuracy: 0.8408 - val_loss: 2.8235 - val_accuracy: 0.6986
Epoch 3/5
31/31 [==============================] - 456s 15s/step - loss: 0.9381 - accuracy: 0.8306 - val_loss: 0.5172 - val_accuracy: 0.8947
Epoch 4/5
31/31 [==============================] - 461s 15s/step - loss: 0.4986 - accuracy: 0.8806 - val_loss: 0.4564 - val_accuracy: 0.9067
Epoch 5/5
31/31 [==============================] - 466s 15s/step - loss: 0.2737 - accuracy: 0.9133 - val_loss: 0.3398 - val_accuracy: 0.9354
```

Gambar 4.34 Hasil Pengujian Skenario 9

Kemudian proses *training* ini membutuhkan waktu sebanyak 2335 detik atau 38 menit 55 detik. Akurasi yang diperoleh seperti yang ditampilkan pada gambar 4.34 yaitu sebesar 93% dan *validation accuracy* 93%.

10. Skenario 10

```
densenet_model_10 = tf.keras.models.Sequential([
  data_augmentation,
  base_model,
  tf.keras.layers.Dense(3, activation='softmax')
])
```

```
model_skenario_10 = densenet_model
model_skenario_10.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.01),
    metrics=['accuracy']
  )
```

Gambar 4.35 *Source Code* Model Skenario 10

Gambar 4.35 merupakan *source code* dari model skenario 10. Pada model skenario 10 konfigurasi yang digunakan adalah *categorical_crossentropy* sebagai *loss*, *Adam* dengan *learning rate* sebesar 0,01 sebagai *optimizer*, *accuracy* sebagai metrics, dan jumlah *epoch* 10.

```
1/10
[==============================] - 553s 17s/step - loss: 2.9896 - accuracy: 0.6510 - val_loss: 0.5700 - val_accuracy: 0.8971
2/10
[==============================] - 463s 15s/step - loss: 0.5663 - accuracy: 0.8867 - val_loss: 0.4141 - val_accuracy: 0.9234
3/10
[==============================] - 469s 15s/step - loss: 0.3526 - accuracy: 0.9204 - val_loss: 0.3546 - val_accuracy: 0.9282
4/10
[==============================] - 458s 15s/step - loss: 0.2779 - accuracy: 0.9224 - val_loss: 0.3845 - val_accuracy: 0.9234
5/10
[==============================] - 460s 15s/step - loss: 0.2880 - accuracy: 0.9224 - val_loss: 0.2977 - val_accuracy: 0.9402
6/10
[==============================] - 460s 15s/step - loss: 0.2738 - accuracy: 0.9286 - val_loss: 0.4399 - val_accuracy: 0.8971
7/10
[==============================] - 463s 15s/step - loss: 0.3520 - accuracy: 0.9163 - val_loss: 0.4598 - val_accuracy: 0.8971
8/10
[==============================] - 460s 15s/step - loss: 0.2724 - accuracy: 0.9306 - val_loss: 0.2912 - val_accuracy: 0.9426
9/10
[==============================] - 468s 15s/step - loss: 0.2877 - accuracy: 0.9316 - val_loss: 0.3574 - val_accuracy: 0.9402
10/10
[==============================] - 467s 15s/step - loss: 0.5314 - accuracy: 0.9041 - val_loss: 0.9646 - val_accuracy: 0.8589
```

Gambar 4.36 Hasil Pengujian Skenario 10

Hasil *training* digambarkan pada gambar 4.36. Skenario 10 membutuhkan waktu *training* sebesar 4721 detik atau 1 jam 18 menit 41 detik. Akurasi yang diperoleh adalah 90% dan akurasi validasi adalah 85%.

11. Skenario 11

```
model_skenario_11 = densenet_mode
model_skenario_11.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.01),
    metrics=['accuracy']
  )
```

```
model_hist_11 = model_skenario_11.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.37 *Source Code* Model Skenario 11

Gambar 4.37 merupakan *source code* dari model skenario 11. Pada model skenario 11 konfigurasi yang digunakan adalah *categorical_crossentropy* sebagai *loss, RMSdrop* dengan *learning rate* sebesar 0,01 sebagai *optimizer*, *accuracy* sebagai *metrics*, dan jumlah *epoch* 10.

```
1/10
[==============================] - 477s 15s/step - loss: 6.1370 - accuracy: 0.6378 - val_loss: 1.3745 - val_accuracy: 0.8038
2/10
[==============================] - 462s 15s/step - loss: 2.3447 - accuracy: 0.7857 - val_loss: 0.6264 - val_accuracy: 0.9091
3/10
[==============================] - 464s 15s/step - loss: 1.4485 - accuracy: 0.8449 - val_loss: 1.9839 - val_accuracy: 0.8134
4/10
[==============================] - 463s 15s/step - loss: 1.4942 - accuracy: 0.8510 - val_loss: 0.7042 - val_accuracy: 0.9234
5/10
[==============================] - 460s 15s/step - loss: 1.5871 - accuracy: 0.8622 - val_loss: 0.7450 - val_accuracy: 0.9115
6/10
[==============================] - 465s 15s/step - loss: 1.4373 - accuracy: 0.8714 - val_loss: 1.6932 - val_accuracy: 0.8493
7/10
[==============================] - 464s 15s/step - loss: 1.0731 - accuracy: 0.9041 - val_loss: 0.6583 - val_accuracy: 0.9354
8/10
[==============================] - 453s 15s/step - loss: 1.1351 - accuracy: 0.8949 - val_loss: 0.6528 - val_accuracy: 0.9306
9/10
[==============================] - 467s 15s/step - loss: 1.3210 - accuracy: 0.8959 - val_loss: 1.0608 - val_accuracy: 0.9019
10/10
[==============================] - 462s 15s/step - loss: 1.2873 - accuracy: 0.8908 - val_loss: 1.0943 - val_accuracy: 0.9115
```

Gambar 4.38 Hasil Pengujian Skenario 11

Berdasarkan hasil yang digambarkan gambar 4.38. Skenario 11 membutuhkan waktu *training* sebesar 4637 detik atau 1 jam 17 menit 17 detik. Akurasi yang diperoleh adalah 89% dan akurasi validasi adalah 91%.

12. Skenario 12

```
model_skenario_12 = densenet_mode
model_skenario_12.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.01),
    metrics=['accuracy']
  )
```

```
model_hist_12 = model_skenario_12.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.39 *Source Code* Model Skenario 12

Gambar 4.39 merupakan *source code* dari model skenario 12. Pada model skenario 12 konfigurasi yang digunakan adalah *categorical_crossentropy* sebagai *loss*, *SGD* dengan *learning rate* sebesar 0,01 sebagai *optimizer*, *accuracy* sebagai *metrics*, dan jumlah *epoch* 10.

```
1/10
[==============================] - 606s 19s/step - loss: 3.2746 - accuracy: 0.6582 - val_loss: 0.4432 - val_accuracy: 0.8876
2/10
[==============================] - 461s 15s/step - loss: 0.6815 - accuracy: 0.8469 - val_loss: 0.4293 - val_accuracy: 0.9139
3/10
[==============================] - 465s 15s/step - loss: 0.5634 - accuracy: 0.8765 - val_loss: 0.4548 - val_accuracy: 0.8876
4/10
[==============================] - 464s 15s/step - loss: 0.5904 - accuracy: 0.8622 - val_loss: 0.3619 - val_accuracy: 0.9163
5/10
[==============================] - 464s 15s/step - loss: 0.3534 - accuracy: 0.9102 - val_loss: 0.2473 - val_accuracy: 0.9378
6/10
[==============================] - 463s 15s/step - loss: 0.3336 - accuracy: 0.9082 - val_loss: 0.2542 - val_accuracy: 0.9306
7/10
[==============================] - 462s 15s/step - loss: 0.3454 - accuracy: 0.9143 - val_loss: 0.2655 - val_accuracy: 0.9258
8/10
[==============================] - 462s 15s/step - loss: 0.4433 - accuracy: 0.9051 - val_loss: 0.2622 - val_accuracy: 0.9402
9/10
[==============================] - 465s 15s/step - loss: 0.5568 - accuracy: 0.8939 - val_loss: 0.5723 - val_accuracy: 0.8780
10/10
[==============================] - 463s 15s/step - loss: 0.2830 - accuracy: 0.9276 - val_loss: 0.3307 - val_accuracy: 0.9163
```

Gambar 4.40 Hasil Pengujian Skenario 12

Berdasarkan hasil yang digambarkan pada gambar 4.40. Skenario 11 membutuhkan waktu *training* sebesar 4775 detik atau 1 jam 19 menit 35 detik. Akurasi yang diperoleh adalah 92% dan akurasi validasi adalah 91%.

13. Skenario 13

```
model_skenario_13 = densenet_mode
densenet_model_13.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    metrics=['accuracy']
  )
```

```
densenet_hist_13 = densenet_model_13.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.41 *Source Code* Model Skenario 13

Gambar 4.41 merupakan *source code* dari model skenario 13. Pada model skenario 13 konfigurasi yang digunakan adalah *categorical_crossentropy* sebagai *loss*, *Adam* dengan *learning rate* sebesar 0,0001 sebagai *optimizer*, *accuracy* sebagai *metrics*, dan jumlah *epoch* 5.

```
1/5
[==============================] - 478s 15s/step - loss: 1.4553 - accuracy: 0.4031 - val_loss: 1.2231 - val_accuracy: 0.4665
2/5
[==============================] - 275s 9s/step - loss: 1.0379 - accuracy: 0.5551 - val_loss: 0.9648 - val_accuracy: 0.6053
3/5
[==============================] - 271s 9s/step - loss: 0.8376 - accuracy: 0.6459 - val_loss: 0.7477 - val_accuracy: 0.6770
4/5
[==============================] - 275s 9s/step - loss: 0.7209 - accuracy: 0.7214 - val_loss: 0.6480 - val_accuracy: 0.7201
5/5
[==============================] - 271s 9s/step - loss: 0.6361 - accuracy: 0.7296 - val_loss: 0.6262 - val_accuracy: 0.7297
```

Gambar 4.42 Hasil Pengujian Skenario 13

Seperti yang terlihat pada gambar 4.42**.** Skenario 13 membutuhkan waktu *training* sebesar 1570 detik atau 26 menit 10 detik. Akurasi yang diperoleh adalah 72% dan akurasi validasi adalah 72%. Pada skenario ke 13 untuk *learning rate* 0,0001 menunjukan akurasi yang kurang bagus namun memiliki peluang peningkatan akurasi Ketika jumlah epoch ditambah. Hal ini dikarenakan kenaikan akurasi pada setiap *epoch* yang stabil.

14. Skenario 14

```
model_skenario_14 = densenet_mode
model_skenario_14.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001),
    metrics=['accuracy']
  )
```

```
model_hist_14 = model_skenario_14.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.43 *Source Code* Model Skenario 14

Gambar 4.43 merupakan *source code* dari model 59cenario 14. Skenario 14 memiliki konfigurasi *loss categorical_crossentropy*, optimizer *RMSprop* dengan *learning rate 0,0001*, *metrics accuracy* dan jumlah *epoch* 5.

```
1/5
[==============================] - 293s 9s/step - loss: 1.3659 - accuracy: 0.4765 - val_loss: 1.2157 - val_accuracy: 0.5335
2/5
[==============================] - 269s 9s/step - loss: 1.0819 - accuracy: 0.5694 - val_loss: 0.9686 - val_accuracy: 0.6053
3/5
[==============================] - 273s 9s/step - loss: 0.8693 - accuracy: 0.6582 - val_loss: 0.7938 - val_accuracy: 0.6818
4/5
[==============================] - 273s 9s/step - loss: 0.7408 - accuracy: 0.7010 - val_loss: 0.6965 - val_accuracy: 0.7081
5/5
[==============================] - 271s 9s/step - loss: 0.6499 - accuracy: 0.7357 - val_loss: 0.6388 - val_accuracy: 0.7321
```

Gambar 4.44 Hasil Pengujian Skenario 14

Waktu *training* yang dibutuhkan adalah 1379 detik atau 22 menit 59 detik. Pada gambar 4.44 menunjukan hasil dari proses *training* 59cenario ke 14. Akurasi yang diperoleh adalah 73% untuk *train* dan 73% untuk validasi.

15. Skenario 15

```
model_skenario_15 = densenet_mode
model_skenario_15.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.0001),
    metrics=['accuracy']
  )
```

```
model_hist_15 = model_skenario_16.fit(
    train_data,
    epochs=5,
    validation_data = valid_data
)
```

Gambar 4.45 *Source Code* Model Skenario 15

Gambar 4.45 menampilkan *source code* dari model 60cenario 15. Skenario 15 memiliki konfigurasi *loss categorical_crossentropy*, *optimizer SGD* dengan *learning rate 0,0001*, *metrics accuracy* dan jumlah *epoch* 5.

```
1/5
[==============================] - 285s 9s/step - loss: 1.9172 - accuracy: 0.3755 - val_loss: 1.3035 - val_accuracy: 0.4976
2/5
[==============================] - 269s 9s/step - loss: 1.3308 - accuracy: 0.4347 - val_loss: 1.1499 - val_accuracy: 0.5718
3/5
[==============================] - 270s 9s/step - loss: 1.1942 - accuracy: 0.4837 - val_loss: 1.0325 - val_accuracy: 0.6148
4/5
[==============================] - 267s 9s/step - loss: 1.1232 - accuracy: 0.5429 - val_loss: 0.9446 - val_accuracy: 0.6459
5/5
[==============================] - 270s 9s/step - loss: 1.0083 - accuracy: 0.5694 - val_loss: 0.8800 - val_accuracy: 0.6675
```

Gambar 4.46 Hasil Pengujian Skenario 15

Waktu *training* yang dibutuhkan adalah 1361 detik atau 22 menit 41 detik. Pada gambar 4.46 menunjukan hasil dari proses *training* 60cenario ke 15. Akurasi yang diperoleh adalah 56% untuk *train* dan 66% untuk validasi.

16. Skenario 16

```
model_skenario_16 = densenet_mode
densenet_model_16.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    metrics=['accuracy']
  )
```

```
model_hist_16 = densenet_model_16.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.47 *Source Code* Model Skenario 16

Gambar 4.47 menunjukan *source code* dari model 61cenario 16. Skenario 16 memiliki *konfigurasi loss categorical_crossentropy*, *optimizer adam* dengan *learning rate 0,0001, metrics accuracy* dan jumlah *epoch 10*.

```
1/10
[==============================] - 674s 21s/step - loss: 1.3889 - accuracy: 0.4510 - val_loss: 1.0357 - val_accuracy: 0.6124
2/10
[==============================] - 437s 14s/step - loss: 1.0620 - accuracy: 0.5694 - val_loss: 0.8361 - val_accuracy: 0.6411
3/10
[==============================] - 442s 14s/step - loss: 0.8977 - accuracy: 0.6429 - val_loss: 0.7161 - val_accuracy: 0.6914
4/10
[==============================] - 439s 14s/step - loss: 0.7803 - accuracy: 0.6837 - val_loss: 0.6242 - val_accuracy: 0.7368
5/10
[==============================] - 417s 13s/step - loss: 0.6798 - accuracy: 0.7010 - val_loss: 0.5561 - val_accuracy: 0.7584
6/10
[==============================] - 419s 13s/step - loss: 0.6077 - accuracy: 0.7439 - val_loss: 0.4971 - val_accuracy: 0.7823
7/10
[==============================] - 427s 14s/step - loss: 0.5720 - accuracy: 0.7612 - val_loss: 0.4621 - val_accuracy: 0.7967
8/10
[==============================] - 403s 13s/step - loss: 0.5087 - accuracy: 0.7949 - val_loss: 0.4274 - val_accuracy: 0.8182
9/10
[==============================] - 427s 14s/step - loss: 0.4604 - accuracy: 0.8112 - val_loss: 0.4046 - val_accuracy: 0.8325
10/10
[==============================] - 426s 14s/step - loss: 0.4204 - accuracy: 0.8306 - val_loss: 0.3839 - val_accuracy: 0.8469
```

Gambar 4.48 Hasil Pengujian Skenario 16

Waktu *training* yang dibutuhkan adalah 4511 detik atau 1 jam 15 menit 11 detik. Pada gambar 4.48 menunjukan hasil dari proses *training* 61cenario ke 16. Akurasi yang diperoleh adalah 83% untuk *train* dan 84% untuk validasi.

17. Skenario 17

```
[ ]  model_skenario_17 = densenet_mode
     model_skenario_17.compile(
         loss='categorical_crossentropy',
         optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0001),
         metrics=['accuracy']
       )
```

```
[14] model_hist_17 = model_skenario_17.fit(
         train_data,
         epochs=10,
         validation_data = valid_data
     )
```

Gambar 4.49 *Source Code* Model Skenario 17

Gambar 4.49 merupakan *source code* dari model 62cenario 17. Skenario 17 memiliki konfigurasi *loss categorical_crossentropy*, *optimizer RMSdrop* dengan *learning rate* 0,0001, metrics *accuracy* dan jumlah *epoch* 10.

```
1/10
[==============================] - 455s 14s/step - loss: 1.4381 - accuracy: 0.4051 - val_loss: 1.1278 - val_accuracy: 0.5407
2/10
[==============================] - 431s 14s/step - loss: 1.1070 - accuracy: 0.5378 - val_loss: 0.8483 - val_accuracy: 0.6483
3/10
[==============================] - 411s 13s/step - loss: 0.9324 - accuracy: 0.6010 - val_loss: 0.6701 - val_accuracy: 0.7129
4/10
[==============================] - 438s 14s/step - loss: 0.7639 - accuracy: 0.6776 - val_loss: 0.5781 - val_accuracy: 0.7608
5/10
[==============================] - 431s 14s/step - loss: 0.6283 - accuracy: 0.7398 - val_loss: 0.5057 - val_accuracy: 0.7967
6/10
[==============================] - 407s 13s/step - loss: 0.5640 - accuracy: 0.7724 - val_loss: 0.4543 - val_accuracy: 0.8134
7/10
[==============================] - 432s 14s/step - loss: 0.5384 - accuracy: 0.7969 - val_loss: 0.4175 - val_accuracy: 0.8301
8/10
[==============================] - 432s 14s/step - loss: 0.4768 - accuracy: 0.8041 - val_loss: 0.3825 - val_accuracy: 0.8493
9/10
[==============================] - 430s 14s/step - loss: 0.4528 - accuracy: 0.8204 - val_loss: 0.3740 - val_accuracy: 0.8565
10/10
[==============================] - 428s 14s/step - loss: 0.4480 - accuracy: 0.8224 - val_loss: 0.3672 - val_accuracy: 0.8541
```

Gambar 4.50 Hasil Pengujian Skenario 17

Waktu *training* yang dibutuhkan adalah 4295 detik atau 1 jam 11 menit 35 detik. Pada gambar 4.50 menunjukan hasil dari proses *training* 62cenario ke 17. Akurasi yang diperoleh adalah 82% untuk *train* dan 85% untuk validasi.

18. Skenario 18

```
model_skenario_18.compile(
    loss='categorical_crossentropy',
    optimizer=tf.keras.optimizers.SGD(learning_rate=0.0001),
    metrics=['accuracy']
  )
```

```
model_hist_18 = model_skenario_18.fit(
    train_data,
    epochs=10,
    validation_data = valid_data
)
```

Gambar 4.51 *Source Code* Model Skenario 18

Gambar 4.51 merupakan *source code* dari model 63cenario 18. Skenario 18 memiliki konfigurasi *loss categorical_crossentropy*, *optimizer SGD* dengan *learning rate* 0,0001, *metrics accuracy* dan jumlah *epoch* 10.

```
1/10
[==============================] - 451s 14s/step - loss: 1.6388 - accuracy: 0.3806 - val_loss: 1.5166 - val_accuracy: 0.3708
2/10
[==============================] - 423s 14s/step - loss: 1.3979 - accuracy: 0.4429 - val_loss: 1.3574 - val_accuracy: 0.4450
3/10
[==============================] - 428s 14s/step - loss: 1.2700 - accuracy: 0.4816 - val_loss: 1.1985 - val_accuracy: 0.5167
4/10
[==============================] - 427s 14s/step - loss: 1.1598 - accuracy: 0.5510 - val_loss: 1.0889 - val_accuracy: 0.5359
5/10
[==============================] - 428s 14s/step - loss: 1.0650 - accuracy: 0.5786 - val_loss: 1.0302 - val_accuracy: 0.5622
6/10
[==============================] - 427s 14s/step - loss: 0.9471 - accuracy: 0.6092 - val_loss: 0.9468 - val_accuracy: 0.5909
7/10
[==============================] - 427s 14s/step - loss: 0.9074 - accuracy: 0.6418 - val_loss: 0.8878 - val_accuracy: 0.6100
8/10
[==============================] - 427s 14s/step - loss: 0.8428 - accuracy: 0.6582 - val_loss: 0.8390 - val_accuracy: 0.6244
9/10
[==============================] - 429s 14s/step - loss: 0.8327 - accuracy: 0.6643 - val_loss: 0.7951 - val_accuracy: 0.6388
10/10
[==============================] - 428s 14s/step - loss: 0.7742 - accuracy: 0.6827 - val_loss: 0.7602 - val_accuracy: 0.6627
```

Gambar 4.52 Hasil Pengujian Skenario 18

Waktu *training* yang dibutuhkan adalah 4295 detik atau 1 jam 11 menit 35 detik. Pada gambar 4.52 menunjukan hasil dari proses *training* 63cenario ke 18. Akurasi yang diperoleh adalah 68% untuk *train* dan 66% untuk validasi.

4.5 Hasil *Evaluation*

4.5.1 *Evaluate Result*

Setelah melakukan *training* pada semua skenario yang ada, langkah berikutnya adalah mengevaluasi semua hasil yang diperoleh dengan membandingkan model satu sama lain. Pada *evaluate result* dilakukan 2 jenis evaluasi. Yang pertama adalah evaluasi waktu pelatihan model, yang kedua adalah evaluasi hasil pelatihan model. Pada evaluasi hasil pelatihan model ada 4 variabel yang akan dianalisi yaitu akurasi, val akurasi, *loss*, dan *val loss*. Akurasi adalah nilai rasio prediksi benar dibagi total percobaan pada data *training*, akurasi validasi adalah nilai rasio prediksi benar dibagi total percobaan pada data validasi, loss adalah ukuran perbedaan antara *output* prediksi dengan data *input* pada data *training* dan validasi loss adalah ukuran perbedaan antara *output* prediksi dengan data *input* pada data *validation* [59]. Kemudian evaluasi akan dibagi menjadi 3 bagian berdasarkan *learning rate*. Sehingga diperoleh hasil evaluasi sebagai berikut:

1. Evaluasi Durasi Pelatihan

Tabel 4.2 Durasi Pelatihan Model

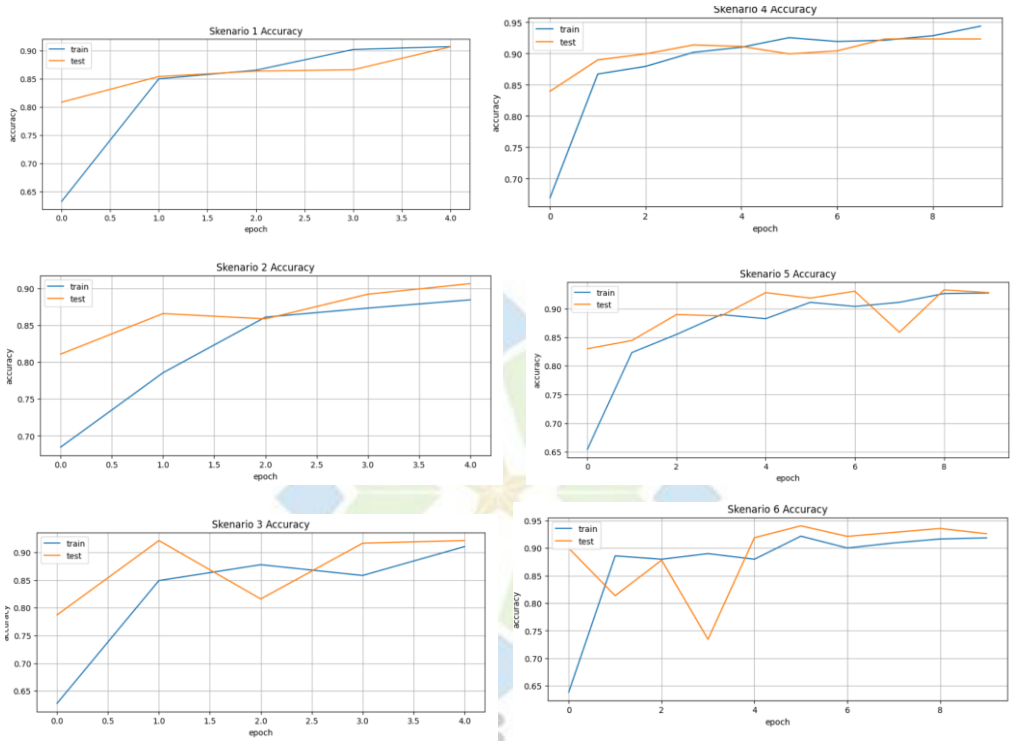
skenario	*Optimizer*	*Learning Rate*	*epoch*	waktu (satuan detik)	
				Rata rata /step	Total
1	*Adam*	0,001	5	11,3	1821
2	*RMSprop*	0,001	5	8,4	1344
3	*SGD*	0,001	5	8,3	1334
4	*Adam*	0,001	10	8,9	2873
5	*RMSprop*	0,001	10	8,2	2639
6	*SGD*	0,001	10	8,2	2625
7	*Adam*	0,01	5	15,2	2447
8	*RMSprop*	0,01	5	14,5	2331
9	*SGD*	0,01	5	14,5	2335
10	*Adam*	0,01	10	14,7	4721
11	*RMSprop*	0,01	10	14,5	4637
12	*SGD*	0,01	10	14,9	4775

skenario	*Optimizer*	*Learning Rate*	*epoch*	waktu (satuan detik)	
				Rata rata	Total
13	*Adam*	0,0001	5	9,8	1570
14	*RMSprop*	0,0001	5	8,6	1379
15	*SGD*	0,0001	5	8,5	1361
16	*Adam*	0,0001	10	14,1	4511
17	*RMSprop*	0,0001	10	13,4	4295
18	*SGD*	0,0001	10	13,4	4295

Durasi pelatihan model dari proses *training* keseluruhan model disajikan pada tabel 4.2. Setiap model membutuhkan durasi pelatihan yang berbeda beda. Berdasarkan tabel yang tersaji, model yang menggunakan *optimizer Adam* membutuhkan waktu pelatihan lebih banyak dibanding 2 optimizer lainya. Sedangkan *optimizer RMSprop* dan *SGD* membutuhkan durasi pelatihan yang hampir sama. Hal ini terjadi hampir disetiap skenario yang menggunakan *learning rate* dan *epoch* yang sama. Model dengan durasi pelatihan terlama diperoleh model skenario 10 yaitu 4.721 detik. Kemudian model dengan durasi pelatihan terkecil diperoleh model skenario 6 yaitu 2.625 detik. Pada *learning rate* 0,001 jumlah waktu yang dibutuhkan hampir sama disetiap stepnya satu step yang waktunya berbeda adalah skenario ke 1. Pada skenario 1 waktu yang dibutuhkan perstep sebesar 11,3 detik sedangkan skenario 2-6 menghabiskan 8-9 detik perstep. Kemudian pada learning rate *0,001* waktu yang dihabiskan perstepnya semua diantara 14-15.5 detik perstepnya. Terakhir pada *learning rate* 0,0001 waktu yang dibutuhkan untuk *training* ada diangka 8-9 detik untuk epoch 5 dan 13-14 detik untuk epoch 10. Berdasarkan tabel diatas dapat terlihat bahwa *optimizer*, *learning rate*, dan *epoch* mempengaruhi durasi pelatihan. *Optimizer* adam memakan waktu lebih lama dibanding dua *optimizer* lain. Lalu pada *learning rate* terlihat setiap *learning rate* memiliki hasil yang berbeda. Pada *learning rate* 0,001 rata rata waktu setiap langkah memperoleh nilai yang sama yaitu 8-9 detik kecuali pada *learning rate* dengan *optimizer Adam*. Kemudian pada *learning rate* 0,001 rata rata waktu yang dibutuhkan untuk setiap lankahnya adalah 14-15 detik. Terakhir pada *learning rate* 0,0001 dapat terlihat semakin banyak *epoch* yang digunakan maka rata rata

waktu per-*step*nya semakin besar juga. Kemudian *epoch* sendiri mempengaruhi total waktu karena semakin banyak *epoch* yang digunakan maka semakin lama juga waktu total yang dibutuhkan.

2. Evaluasi Model Skenario dengan *learning rate* 0,001



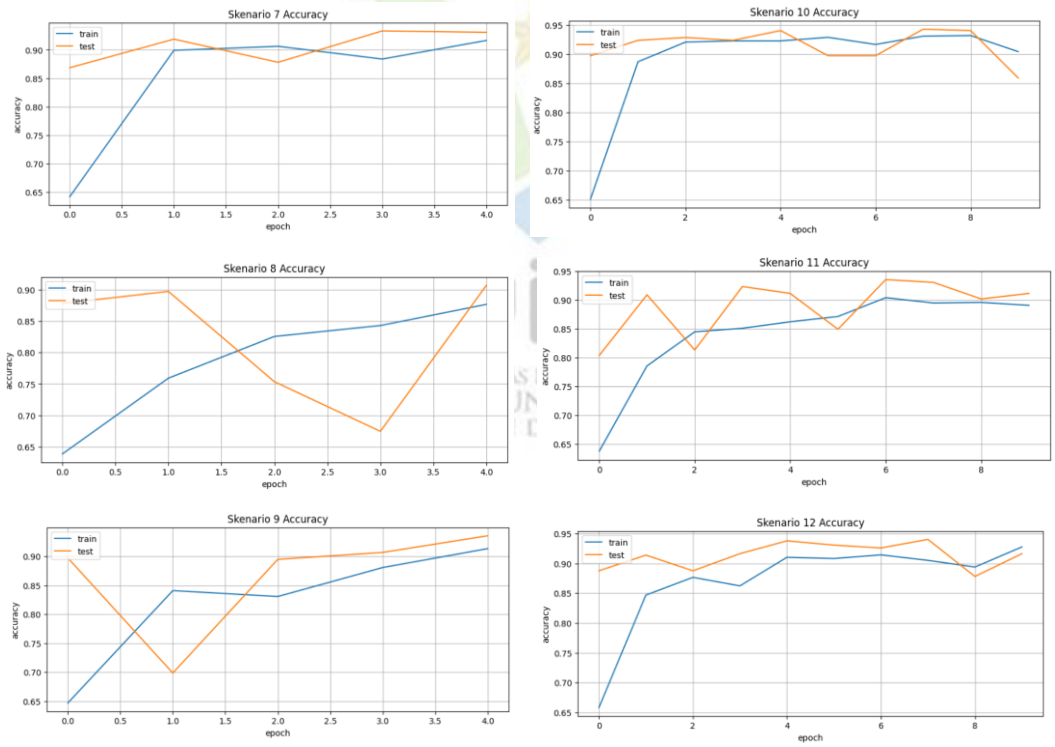
Gambar 4.53 Grafik hasil akurasi skenario menggunakan *learning rate* 0,001

Tabel 4.3 Hasil Model Skenario *Learning Rate* 0,001

skenario	*Optimizer*	*epoch*	Hasil *Training*			
			Akurasi	*Val_accuracy*	*Loss*	*Val_Loss*
1	*Adam*	5	90,71%	90,67%	0,24	0,28
2	*RMSprop*	5	88,47%	90,67%	0,29	0,28
3	*SGD*	5	91,02%	92,11%	0,31	0,37
4	*Adam*	10	94,39%	93,34%	0,16	0,22
5	*RMSprop*	10	92,76%	92,82%	0,18	0,22
6	*SGD*	10	91,84%	92,58%	0,29	0,24

Evaluasi hasil loss dan akurasi yang pertama adalah skenario yang menggunakan *learning rate* 0,001. pada tabel 4.3 tersaji hasil *training model* skenario 1-6. Seperti yang tersaji pada tabel. *Learning rate* 0,001 memperoleh nilai yang baik pada *epoch* 5 maupun 10. Setiap *optimizer* mengalami peningkatan akurasi sebesar 1-4%. Model skenario 0,001 ini memperoleh nilai loss dibawah 0.40 yang mana menunjukan bahwa seluruh model memiliki performa pelatihan yang baik karena semakin kecil nilai loss maka semakin baik hasil pengklasifikasian model. Model terbaik diperoleh skenario 4 dengan hasil akruasi 94%, akurasi validasi 93%, *loss* 0,16, dan *loss* validasi 0,22. Rata-rata akurasi pada model yang menggunakan *learning rate* 0,001 adalah 91,53% *train* dan 92,03 untuk *val_accuracy*.

3. Evaluasi Model Skenario dengan *learning rate* 0,01



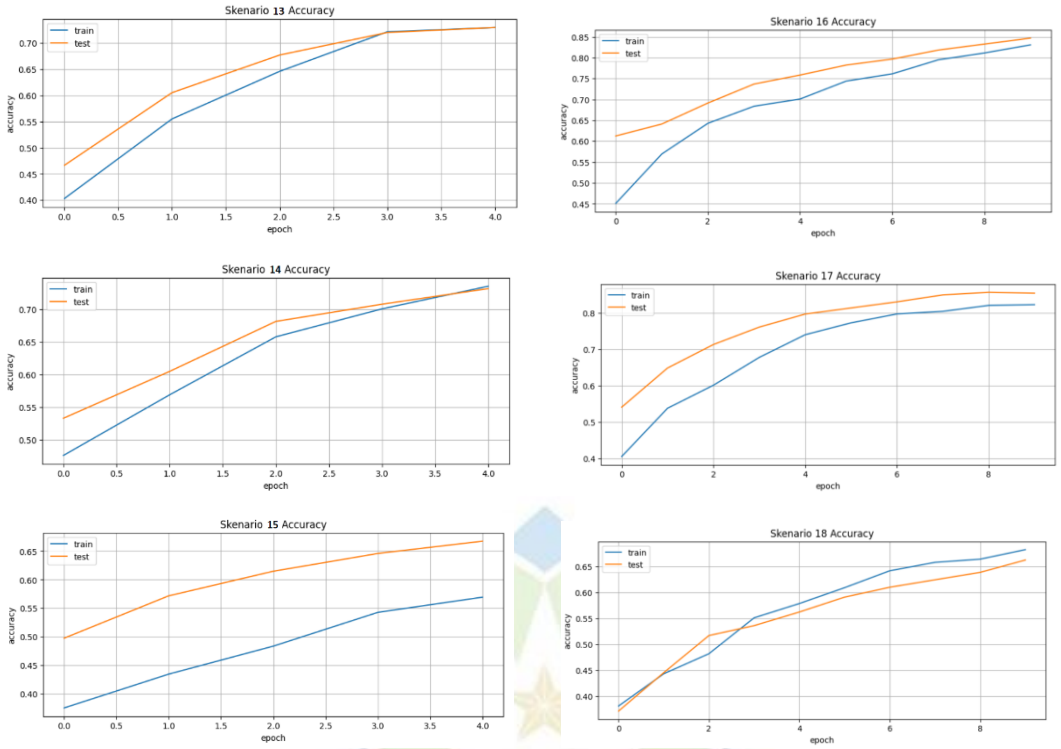
Gambar 4.54 Grafik hasil akurasi skenario menggunakan *learning rate* 0,01

Tabel 4.4 Hasil Model Skenario *Learning Rate* 0,01

skenario	*Optimizer*	*epoch*	Hasil *Training*			
			Akurasi	*Val_accuracy*	*Loss*	*Val_Loss*
7	*Adam*	5	91,63%	93,06%	0,39	0,38
8	*RMSprop*	5	87,65%	90,67%	1,49	0,87
9	*SGD*	5	91,33%	93,54%	0,27	0,33
10	*Adam*	10	90,41%	85,89%	0,53	0,96
11	*RMSprop*	10	89,08%	91,15%	1,28	1,09
12	*SGD*	10	92,79%	91,63%	0,28	0,33

Tabel 4.4 menyajikan hasil dari evaluasi yang menggunakan *learning rate* 0,01. berdasarkan tabel 4.4 keenam model skenario yang menggunakan *learning rate* 0,01 memperoleh hasil 89% hingga 92%. Nilai yang diperoleh hampir sama dengan model yang menggunakan *learning rate* 0,01. namun karena jumlah *learning rate* yang digunakan hanya 0,01, seperti yang ditampilkan pada gambar 4.64 seluruh model mengalami kenaikan dan penurunan akurasi disetiap *epochnya*. Salah satunya adalah skenario ke 10 yang menggunakan *optimizer* Adam jika nilai *epoch* ditambah terdapat kemungkinan akurasi yang diperoleh akan menurun. Hal ini terlihat dari penurunan akurasi yang diperoleh pada *epoch* ke 8 sampai 10. Kemudian pada *optimizer RMSprop* walaupun memperoleh nilai akurasi yang baik namun *optimizer* ini memperoleh nilai *loss* yang tinggi yaitu diangka 1,49 hal ini menunjukan bahwa model berkemungkinan besar akan kesulitan ketika memprediksi data yang baru. Nilai terbaik yang diperoleh pada *skenario* dengan *learning rate* 0,01 adalah skenario ke 12 dengan perolehan akurasi 92%, akurasi validasi 91%, *loss* 0,28, dan *loss* validasi 0,33. Rata-rata akurasi pada model yang menggunakan *learning rate* 0,01 adalah 90,48% untuk akurasi dan 90,99% untuk *val_accuracy*.

4. Evaluasi Model Skenario dengan *Learning Rate* 0,0001



Gambar 4.55 Grafik hasil akurasi skenario menggunakan *learning rate* 0,0001

Tabel 4. 5 Hasil model skenario *learning rate* 0,0001

skenario	*Optimizer*	*epoch*	Hasil *Training*			
			Akurasi	*Val_accuracy*	Loss	*Val_Loss*
13	*Adam*	5	72,96%	62,62%	0,63	0,62
14	*RMSprop*	5	73,57%	73,21%	0,64	0,63
15	*SGD*	5	56,93%	66,75%	0,57	0,88
16	*Adam*	10	83,06%	84,69%	0,42	0,38
17	*RMSprop*	10	82,24%	85,41%	0,44	0,36
18	*SGD*	10	68,27%	66,27%	0,77	0,76

Tabel 4.5 menyajikan hasil dari proses *training* skenario 13-18. Tabel menunjukan bahwa *learning rate* 0,0001 tidak bekerja dengan baik pada keseluruhan *skenario*. Terlihat nilai akurasi yang diperoleh berada dibawah nilai akurasi yang menggunakan *learning rate* 0,001 dan 0,01. Namun terlihat dari grafik yang digambarkan pada gambar 4.55 nilai akurasi memiliki kemungkinan untuk

terus meningkat. Terlihat dari akurasi yang terus meningkat disetiap *epoch*-nya. kemudian nilai *loss* yang masih dibawah 1 menyatakan bahwa model memiliki peforma yang baik hanya saja nilai *learning rate* yang kecil mengakibatkan model memerlukan pelatihan lebih lama agar menghasilkan nilai yang lebih baik. Nilai terbaik diperoleh skenario 16 dengan hasil akurasi 83%, akurasi validasi 84%, *loss* 0,42, dan *loss* validasi 0,38. Rata-rata akurasi *learning rate* 0,0001 adalah 72,99% akurasi *training* dan 73,15% untuk *val_accuracy*.

4.5.2 *Determine Next Step*

Berdasarkan hasil evaluasi yang dilakukan sebelumnya, model dengan akurasi terbaik diperoleh pada model skenario ke 4 dengan akurasi sebesar 94,39%. dengan akurasi yang tinggi maka model pada skenario 4 akan digunakan untuk tahapan *deployment*. Tahapan *deployment* akan dilakukan dengan mengubah model yang sudah dilatih kedalam bentuk *pretrained* model sehingga pada proses *deployment* tidak perlu melakukan *training* kembali, model skenario 4 akan dikonversi kedalam bentuk h5. Proses menyimpan model kedalam bentuk h5 digambarkan pada gambar 4.56.

```
model_skenario_4.save('model.h5')
```

Gambar 4.56 Proses menyimpan model kedalam bentuk H5

4.6 Hasil *Deployment*

4.6.1 Deployment Plan

Pada tahapan *deployment* plan akan disusun proses perencanaan pembuatan *Application Programming Interface* (*API*).

1. *API Design*

Hal pertama yang dibuat adalah desain *API*. Pada desain *API* akan ditentukan fungsi fungsi yang akan dibuat. Hal ini bertujuan agar pengembangan system tidak keluar dari pengembangan *API* yang sudah ditentukan. Pada penelitian ini fungsi *API* yang dibuat hanyalah fungsi *predict* saja sehingga diperoleh *API design* seperti pada tabel 4.6.

Tabel 4.6 *API Design*

NO	Proses	*API*	
		Method	*Path*
1	*Predict*	*POST*	*/predict*
2	*Get_info*	*GET*	*/get_info*

2. End Point API Design

Setelah membuat fungsi apa saja yang akan dibuat pada *API* proses selanjutnya adalah membuat end point. Proses ini memuat aturan permintaan dari sumber data yang dikirim dan respon yang akan dikirim pada sebuah *API*.

1. *Predict*

Predict digunakan untuk mengupload dan memprediksi gambar yang dimasukan Adapun rancangan *request and respon* atau rancangan *end point API* yang dibuat sebagai berikut:

Tabel 4.7 *End Point API Design /predict*

Variabel	Keterangan
Method	*POST*
URL/PATH	*./predict*
Description	Menerima gambar sebagai imput kemudian memberikan prediksi gambar.
Request	*Image file (format file jpg, jpeg, png)*

Response	{
	"prediction": prediction,
	"image_name": file.filename,
	"image_path": file_location
	}

2. *Get_info*

Get_info digunakan untuk memperoleh informasi dari model yang digunakan.

Tabel 4.8 *End Point API Design /get_info*

Variabel	Keterangan
Method	*GET*
URL/PATH	*./get_info*
Description	Menerima informasi seputar model yang buat
Request	*Info_type (optional)*

Response	{
	"model":"CNN Arsitektur DenseNet201",
	"fungsi_model":"Memprediksi Penyakit pada Tanaman Selada",
	"nama_pengguna":"Harry Akbar Fauzan",
	"Email":"harryakbar470@gmail.com"
	}

4.6.2 *Produce Final Report*

Pada tahapan *produce final report*, semua yang sudah direncanakan pada tahapan sebelumnya akan dieksekusi. *Application Programming Interface (API)* dibuat menggunakan *Fast API*. *Fast API* merupakan *API* yang menggunakan bahasa pemorgraman *python*. *Source Code* dari pembuatan *API* adalah sebagai berikut:


```
from fastapi import FastAPI
import uvicorn
from keras.models import load_model
import numpy as np
import tensorflow as tf
from fastapi import FastAPI, File, UploadFile, Form
import os

app = FastAPI()
model = load_model("model.h5")
```

Gambar 4.57 *Source code import library* dan *load model*

Langkah pertama adalah meng-*import library* yang dibutuhkan, seperti yang ditampilkan pada gambar 4.57. *Library* yang digunakan adalah *fastapi*, *uvicorn*, *keras*, *numpy*, *tensorflow* dan *os*. Pada gambar 4.57 juga menunjukan bagaimana proses *load model CCN* yang sudah dibuat sebelumya kedalam *API*.

```
def predict(filename):
    BATCH_SIZE = 32
    IMAGE_SIZE = (200,200)
    image_path = filename
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=IMAGE_SIZE)
    x = tf.keras.utils.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    image = np.vstack([x])
    classes = model.predict(image, batch_size=BATCH_SIZE)
    classes = np.argmax(classes)

    if classes==0:
        kelas = 'Hawar Daun Alternaria SP'
    elif classes==1:
        kelas = 'Busuk Daun Bremia lactuae '
    elif classes==2:
        kelas = 'Sehat'

    return kelas
```

Gambar 4.58 *Source code fungsi predict*

Pada gambar 4.58 menunjukan *source code* dari fungsi *predict*. Pada fungsi *predict* inilah gambar akan diproses dan diidentifikasi. Ada beberapa tahapan pada proses identifikasi gambar ini diataranya: mengambil gambar, merubah gambar menjadi ukuran 200.200, merubah gambar kedalam *array*, prediksi gambar, dan terakhir menentukan kelas berdasarkan hasil prediksi.

```
@app.post("/predict/")
async def predict_api(file: UploadFile = File(...)):
    extension = file.filename.split(".")[-1] in ("jpg", "jpeg", "png")
    if not extension:
        return "Image must be jpg or png format!"

    folder = "image"
    file_location = os.path.join(folder, file.filename)
    image = file.file.read()
    with open(file_location, 'wb') as f:
        f.write(image)

    prediction = predict(file_location)
    result = {
        "prediction": prediction,
        "image_name": file.filename,
        "image_path": file_location,
        "message":"prediksi berhasil"
        }
    return result
```

Gambar 4.59 *Source code end point predict*

Kemudian pada gambar 4.59 adalah *source code* fungsi dari *endpoint predict*. Fungsi ini berfungsi untuk menanggil fungsi *predict_api* kedalam *FastAPI.* Fungsi *predict_api* dipanggil menggunakan *@app.post*. seperti yang terlihat pada gambar *output* dari proses ini adalah *prediction, image_name, file_location*, dan pesan prediksi berhasil.

```python
@app.get("/get_info/")
async def get_info(info_type: str = None):

    informasi = {
        "model":"CNN Arsitektur DenseNet201",
        "fungsi_model":"Memprediksi Penyakit pada Tanaman Selada",
        "nama_pengguna":"Harry Akbar Fauzan",
        "Email":"harryakbar470@gmail.com"
    }
    return informasi
```

Gambar 4.60 *Source code end point get_info*

pada gambar 4.60. menunjukan *source code* fungsi *end point get_info.* Fungsi ini berfungsi untuk menampilkan informasi dari model yang digunakan seperti nama model, fungsi model, pengguna model, dan email.

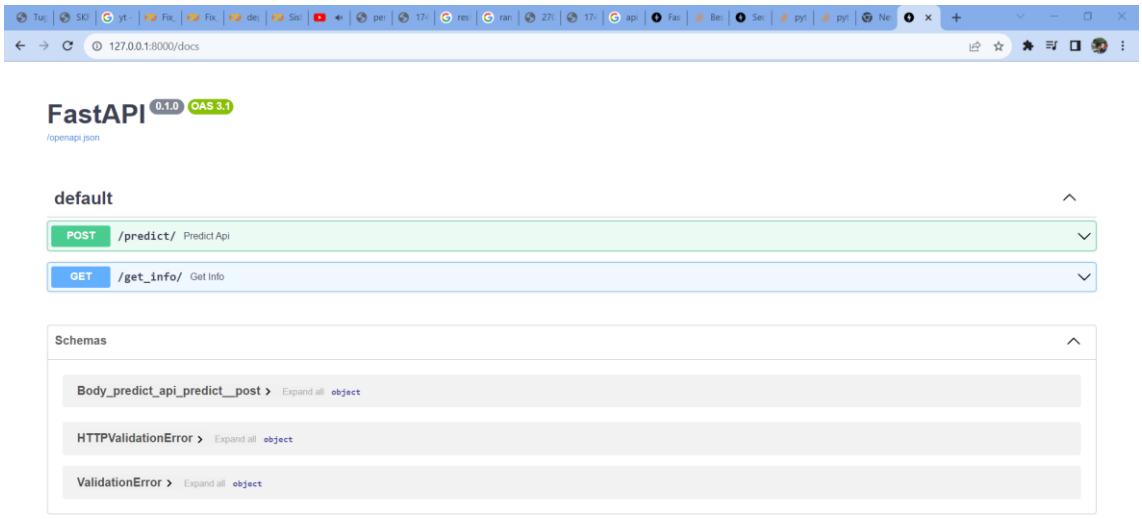
```python
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

Gambar 4.61 *Source code lokal server fast api*

Terakhir pada gambar 4.61. menunjukan konfigurasi *local server* yang digunakan pada *FastAPI*. *FastAPI* di *run* menggunakan *host* 127.0,0.1 dengan *port* 8000.

Setelah berhasil membuat *FastAPI.* Langkah selanjutnya adalah mencoba menjalankan *end point* yang sudah dibuat. Percobaan dilakukan dengan menggunakan halaman *docs* yang sudah disediakan oleh *FastAPI*. Halaman *docs* dapat diakses dengan membuka *Server/docs*. Karena menggunakan host 127.0,0.1 port 8000, maka halaman *docs* dapat dibuka dengan membuka

http://127.0,0.1:8000/docs. Tampilan dari halaman tersebut digambarkan pada gambar 4.62.



Gambar 4.62 tampilan halaman *docs*

Setelah membuka halaman docs maka selanjutnya adalah mencoba setiap *end point* yang ada. Gambar 4.63 menunjukan hasil *end point predict*, dan gambar 4.64 menunjukan hasil *end point* get_info.

Gambar 4.63 Tampilan hasil *end point predict*

Gambar 4.64 Tampilan hasil *end point get_info*

Proses terakhir yang dilakukan adalah melakukan percobaan *predict* dengan menggunakan 10 data untuk setiap kelasnya sehingga total dilakukan 30 kali percobaan. Adapun hasil percobaan disajikan pada tabel 4.9.

Tabel 4.9 Hasil Percobaan

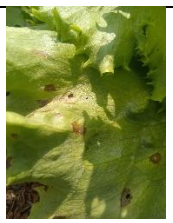
Kelas Gambar	Benar	Salah	Persentase benar
Hawar Daun	9	1	90%
Busuk Daun	7	3	70%
Sehat	9	1	90%

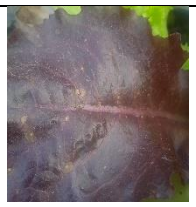
Hasil dari 15 kali percobaan adalah untuk kelas hawar daun memperoleh akurasi 90% dengan prediksi benar sebanyak 9 dan salah sebanyak 1. Pada percobaan ke dua data yang harusnya terprediksi hawar daun malah terprediksi sehat. Kemudian pada kelas busuk daun akurasi yang diperoleh setelah melakukan 10 kali percobaan adalah 70%. Pada percobaan ke 2 hasil prediksi yang diperoleh adalah bercak daun, kemudian 2 percobaan lainya pun sama hasil prediksi yang diperoleh adalah hawar daun bukan busuk daun. Pada kelas sehat akurasi yang diperoleh adalah 90%. Satu percobaan yang menggunakan foto yang didalamnya terdapat tanah mengakibatkan teridentifikasi hawar daun. Akurasi keseluruhan pengujian diperoleh sebesar 83.3%.

Kemudian dilakukan juga percobaan menggunakan jenis selada yang berbeda. Pada percobaan yang pertama dilakukan menggunakan selada bulat (*Head Leattuce* atau *Iceberg*). Perbedaan yang ada pada head lettuce dengan selada hijau

(*Lollo Verde*) atau selada yang digunakan pada penelitian adalah bentuk tanaman yang bulat dan daun yang lebih rata. kemudian jenis selada lain yang dicoba adalah selada daun merah (*Lollo Roso)*. Selada merah memiliki perbedaan diwarna saja yaitu warna merah. Sedangkan bentuk daun hampir sama dengan selada hijau (*Lollo Verde*). Hasil percobaan adalah sebagai berikut:

Tabel 4.10 Hasil Percobaan Jenis Selada Lain

no	Gambar	Jenis selada	Kelas	Hasil prediksi
1		Selada bulat (*head lettuce*)	Busuk Daun	Busuk Daun
2		Selada bulat (*head lettuce*)	Hawar Daun	Hawar Daun
3		Selada bulat *(head lettuce*)	Hawar Daun	Busuk Daun
4		Selada bulat *(head lettuce)*	Sehat	Sehat
5		Selada Merah *(Lollo Roso)*	Busuk Daun	Busuk Daun

no	Gambar	Jenis selada	Kelas	Hasil prediksi
6		Selada Merah *(Lollo Roso*)	Hawar Daun	Busuk Daun
7		Selada Merah (*Lollo Roso)*	Sehat	Busuk Daun

Pada tabel 4.10 menyajikan hasil dari pengujian model pada jenis tanaman lain. Berdasarkan tabel 4.10, model menghasilkan prediksi benar pada 3 kelas yang diujikan. Hanya pada kelas hawar daun saja model mengalami kesalahan prediksi. Sedangkan pada kelas busuk daun dan sehat, model dapat memprediksi dengan benar. Pada jenis selada merah model tidak dapat mengidentifikasi penyakit dengan benar. Pengujian pada ketiga gambar dengan 3 kelas yang berbeda menghasilkan prediksi busuk daun pada ketiga percobaan. Hal ini menunjukan model kesulitan dalam memprediksi gambar yang memiliki jenis warna yang berbeda.

4.7 Pembahasan Implementasi Algoritma *Convolutional Neural Network* Arsitektur *DenseNet201* Dalam Mengidentifikasi Penyakit pada Tanaman Selada

Pembuatan model identifikasi penyakit pada tanaman selada dilakukan dengan menggunakan algoritma *Convolutional Neural Networ*k arsitektur *DenseNet201*. Metode pengembangan yang digunakan dalam proses pembuatan model indentifikasi penyakit dan hama adalah metode pengembangan *CRISP-DM* (*Cross Industry Standard Process for Data Mining*). Metode pengembangan *CRISP-DM* terdiri dari 6 tahapan yaitu *Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation,* dan *Deployment*.

Tahapan *bussines understanding* berfokus pada pengumpulan informasi dan analisis kebutuhan. Dalam proses *bussines understanding* terdapat 4 tahapan yang

dilakukan. Tahapan pertama adalah *Determine Business Objectives*. Tahapan ini bertujuan untuk menganalisi informasi yang diperoleh kemudian menentukan tujuan penelitian. Hasil dari tahapan *determite business objectives* adalah tujuan penelitian yaitu model yang mampu membantu dalam mengidentifikasi penyakit. Tahapan kedua dari *bussines understanding* adalah *assest situation*. Pada tahapan *assest situation* dilakukan analisis pada penelitian yang sudah dilakukan dan informasi situasi yang berhubunngan dengan penelitian. Tahapan ini menghasilkan informasi tambahan dalam menentukan beberapa keputusan seperti jumlah kelas yang digunakan dan *deployment* yang akan dilakukan. Tahapan selanjutnya adalah *determine* data *mining goals*. Pada tahapan ini ditentukanlah tujuan akhir dari penelitian yang akan dilakukan. Hasil dari tahapan ini adalah membuat model yang mampu mengidentifikasi penyakit pada tanaman selada menggunakan algoritma *CNN* arsitektur *DenseNet201* yang kemudian *dideploy* kedalam sebuah *API*. Tahapan terakhir pada proses *bussines understanding* adalah *plan activities*.

Setelah melakukan tahapan *bussines understanding* tahapan berikutnya adalah *data understanding*. Tahapan ini bertujuan untuk mengumpulkan dan memahami data yang digunakan. Pada tahapan ini terdapat 3 tahapan yang dilakukan yaitu mengumpulkan data, pendesrkipsian data, dan eksplorasi data. setelah melakukan *data understanding* tahapan berikutnya adalah *data prepraration*. Pada tahapan ini data yang sudah dikumpulkan akan diolah sehingga dapat digunakan pada tahapan berikunya. Pada tahapan data *preparation* dilakukan beberapa pengolahan seperti data *selection*, mengubah nama, *import* data kedalam *cloud*, melakukan data *split*, melakulan konfigurasi tambahan dan data augmentasi. Setelah data dianggap siap langkah berikutnya adalah *modeling*. Pada tahapan *modeling* proses pertama yaitu membuat model utama yang mana pada penelitian ini menggunakan algoritma *CNN* arsitektur *DenseNet201*. Setelah membuat model utama kemudian membuat *test design*. Pada penelitian ini dibuat 18 model dengan variasi *optimizer, learning rate,* dan *epoch* yang berbeda.

Setelah melakukan *modeling* tahapan berikutnya adalah *evaluation*. Pada proses *evaluation* semua hasil dari proses *training* dibandingkan dan dianalisis. Berdasarkan hasil evaluasi diperoleh model terbaik yaitu model skenario ke 4 dengan akurasi 94.39% dan *validation* akurasi 93,34%. Model terbaik yang

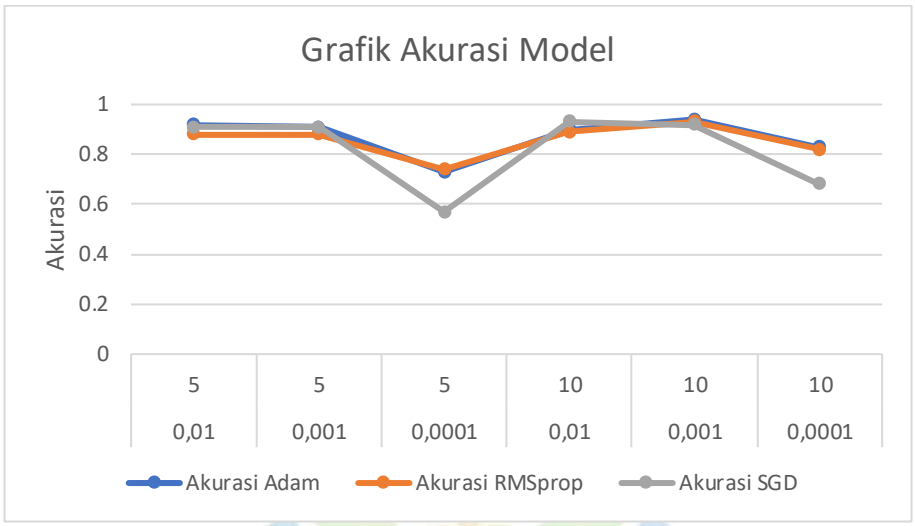
diperoleh kemudian akan digunakan pada tahapan terakhir atau *deployment*. Model terbaik disimpan kedalam bentuk model.h5 kemudian *dideploy* kedalam *API*. Dibuatlah 2 *end point* pada *API* yaitu /*predict* dan /*get_info*. Setelah *deployment* berhasil dibuat kemudian dilakukan pengujian dengan menggunakan 5 data perkelas. Hasil dari pengujian tersebut adalah 9 benar 1 salah untuk kelas hawar daun, 7 benar 3 salah untuk kelas busuk daun dan 9 benar 1 salah untuk kelas sehat (akurasi pengujan 83.3%).

4.8 Pembahasan Tingkat Akurasi Algoritma *Convolutional Neural Network* Arsitektur *DenseNet201* Dalam Mengidentifikasi Penyakit pada Tanaman Selada

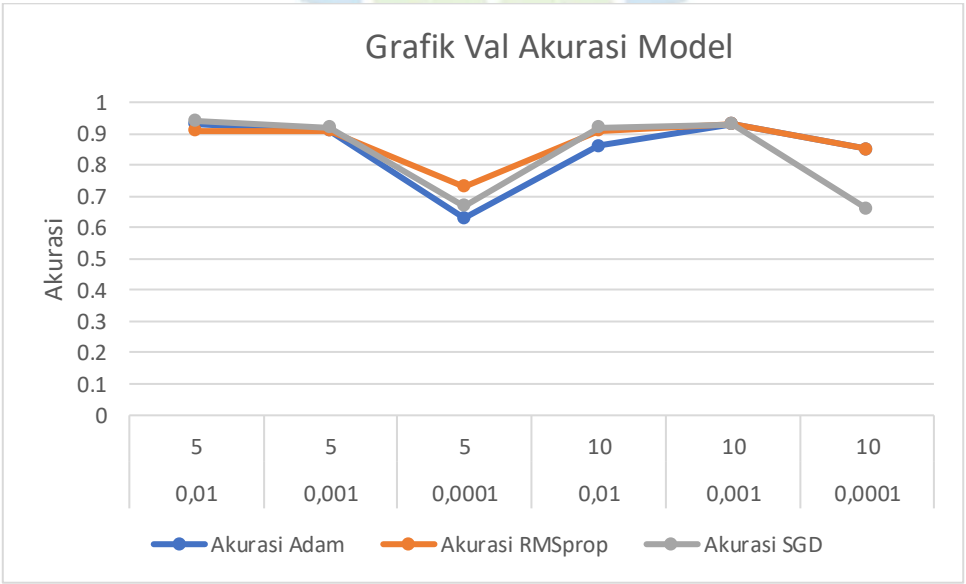
Tabel 4.11 Akurasi model

skenario	*Optimizer*	*Learning Rate*	*epoch*	*Accuracy*	*Val_Acc*
1	*Adam*	0,001	5	90,71%	90,67%
2	*RMSprop*	0,001	5	88,47%	90,67%
3	*SGD*	0,001	5	91,02%	92,11%
4	*Adam*	0,001	10	94,39%	93,34%
5	*RMSprop*	0,001	10	92,76%	92,82%
6	*SGD*	0,001	10	91,84%	92,58%
7	*Adam*	0,01	5	91,63%	93,06%
8	*RMSprop*	0,01	5	87,65%	90,67%
9	*SGD*	0,01	5	91,33%	93,54%
10	*Adam*	0,01	10	90,41%	85,89%
11	*RMSprop*	0,01	10	89,08%	91,15%
12	*SGD*	0,01	10	92,79%	91,63%
13	*Adam*	0,0001	5	72,96%	62,62%
14	*RMSprop*	0,0001	5	73,57%	73,21%
skenario	*Optimizer*	*Learning Rate*	*epoch*	*Accuracy*	*Val_Acc*
15	*SGD*	0,0001	5	56,93%	66,75%
16	*Adam*	0,0001	10	83,06%	84,69%

17	*RMSprop*	0,0001	10	82,24%	85,41%
18	*SGD*	0,0001	10	68,27%	66,27%



Gambar 4. 65 Grafik akurasi model



Gambar 4. 66 Grafik akurasi validasi model

Tabel 4.10 menyajikan hasil akurasi dari setiap model yang sudah dilatih kemudian gambar 4.64 dan gamabr 4.66 menggambarkan grafik akurasi latih dan akurasi validasi. seperti yang tersaji pada tabel. *optimizer, learning rate,* dan *epoch* mempengaruhi nilai akurasi pada model yang dibuat. Hampir semua *optimizer*

memperoleh nilai yang baik. Nilai akurasi terbaik diperoleh skenario 4 yang menggunakan *optimizer adam, learning rate* 0,001 dan *epoch* 10. Skenario 4 memperoleh akurasi sebesar 94.39% dan akurasi validasi sebesar 93,34%. kemudian pada skenario yang menggunakan *learning rate 0,01,* optimizer adam dan SGD memperoleh nilai akurasi yang hampir sama yaitu sebesar 91% untuk akurasi dan 93% untuk akurasi validasi pada epoch ke 5. Seperti yang terlihat pada gambar 4.64. Akurasi lebih kecil diperoleh pada skenario yang menggunakan *learning rate* 0,0001. Pada salah satu scenario ini memperoleh akurasi sebesar 56,93% dan 66,75%. Hal ini sesuai denga napa yang ada pada penelitian terdahulu. Pada penelitian yang dilakukan oleh Difran Nur Cahyo [58]. Menunjukan bahwa *learning rate* 0,0001 memperoleh nilai akurasi yang baik pada *epoch* ke 20 hingga ke 40. Hal ini menunjukan bahwa *learning rate* 0,0001 membutuhkan jumlah *epoch* yang lebih banyak dibanding learning rate lain dan tidak cocok digunakan pada proses *training* dengan *learning rate* yang menggunakan *epoch* yang terbatas. Berdasarkan penelitian yang sudah dilakukan, *learning rate* merupakan faktor yang paling mempengaruhi tingkat akurasi. *learning rate* yang memperoleh nilai akurasi terbaik adalah 0,001 dengan rata rata akurasi 91,53%. Akurasi terbaik kedua diperoleh *learning rate* 0,01 dengan akurasi rata rata 90,48%. Terakhir, *learning rate* terkecil diperoleh *learning rate* 0,0001 dengan akurasi 72,99%.