

# PEMBELAJARAN MESIN DAN PEMBELAJARAN MENDALAM

# **Modul 6**

## **Convolutional Neural Network**

Yohanes Sigit Purnomo Wuryo Putro, ST., M.Kom., Ph.D.  
Aloysius Gonzaga Pradnya Sidhwara, S.T., M.Eng.

Atanasius Surya Gunadharma  
F Jonathan Putra Prasetyo

# Tugas Modul Convolutional Neural Network

MobileNet

## Import Library

- Tahap pertama adalah import seluruh library yang dibutuhkan

```
In [ ]: #Import library  
import os  
import numpy as np
```

```
#Import Library tensorflow dan modul keras yang diperlukan  
import tensorflow as tf  
from tensorflow.keras import layers  
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator  
from tensorflow.keras.models import Sequential, load_model  
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten  
  
#Penjelasan  
# Layers digunakan untuk menambahkan Lapisan ke dalam model  
# Load_img digunakan untuk memuat gambar  
# ImageDataGenerator digunakan untuk melakukan augmentasi pada gambar  
# Sequential digunakan untuk membuat model secara berurutan  
# Conv2D digunakan untuk membuat Lapisan konvolusi  
# MaxPooling2D digunakan untuk melakukan pooling pada Lapisan konvolusi  
# Dense digunakan untuk membuat Lapisan fully connected  
# Dropout digunakan untuk menghindari overfitting  
# Flatten digunakan untuk membuat Lapisan menjadi flat (rata) menjadi vektor 1 dimensi
```

## Load Data

- Load dataset berdasarkan path dimana dataset disimpan

```
In [ ]: count = 0 #digunakan untuk menghitung jumlah gambar  
dirs = os.listdir(r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\train_data')  
for dir in dirs:  
    files = list(os.listdir(r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\train_data/'+dir))  
    print(dir + ' Folder has ' + str(len(files)) + ' Images')  
    count = count + len(files)  
print('Images Folder has ' + str(count) + ' Images')
```

```
Matang Folder has 800 Images  
Mentah Folder has 800 Images  
Images Folder has 1600 Images
```

## Load Images into Arrays as Dataset

- Membuat dataset dari gambar yang ada di direktori

```
In [ ]: # Parameter  
base_dir = r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\train_data' #direktori folder dataset  
img_size = 180 #mengubah ukuran gambar menjadi 180  
batch = 32 #jumlah sample (gambar) yang akan diproses pada satu kali iterasi  
validation_split = 0.1 #data pelatihan yang akan digunakan sebagai data validasi
```

- Memasukkan parameter yang telah di definisikan tadi untuk membuat dataset dari gambar di direktori

```
In [ ]: dataset = tf.keras.utils.image_dataset_from_directory(  
    base_dir, #path direktori, subfolder dianggap sebagai label  
    seed=123, #untuk memastikan proses pemisahan data selalu konsisten (random_state)  
    image_size=(img_size, img_size), #ukuran gambar diubah (resize) menjadi 180x180 pixel  
    batch_size=batch, #jumlah gambar yang akan dikelompokkan  
)  
Found 1600 files belonging to 2 classes.
```

```
In [ ]: #mendapatkan nama kelas dari dataset  
class_names = dataset.class_names #dataset.class_names akan mengambil daftar nama kelas berdasarkan subfolder di dalam direktori  
print("Class Names:", class_names)
```

Class Names: ['Matang', 'Mentah']

### Train-Validation-Test Split

- Membagi dataset menjadi tiga subset yaitu train, validation, dan test
  - Train, digunakan untuk melatih model agar mengenali pola dalam data
  - Validation, digunakan untuk mengevaluasi performa model selama pelatihan
  - Test, digunakan untuk menguji model setelah pelatihan

```
In [ ]: total_count = len(dataset) #menghitung jumlah total gambar dalam dataset  
val_count = int(total_count * validation_split) #menghitung jumlah gambar untuk validasi  
train_count = total_count - val_count # menghitung jumlah gambar untuk train  
  
print("Total Images:", total_count)  
print("Train Images:", train_count)  
print("Validation Images:", val_count)
```

Total Images: 50  
Train Images: 45  
Validation Images: 5

```
In [ ]: train_ds = dataset.take(train_count) #digunakan untuk mengambil (take) sejumlah batch sebanyak 'train_count' yang pertama dari dataset  
val_ds = dataset.skip(train_count) #digunakan untuk melewati (skip) sejumlah batch sebanyak 'train_count' yang pertama dari dataset
```



Watermarkly

```
In [ ]: import matplotlib.pyplot as plt
```

```
i = 0
plt.figure(figsize=(10,10)) #membuat figure dengan ukuran 10x10 inch untuk menampilkan gambar

for images, labels in train_ds.take(1): #mengambil 1 batch pertama dari train_ds
    for i in range(9): #iterasi untuk menampilkan 9 gambar pertama dalam batch
        plt.subplot(3,3, i+1) #menyiapkan subplot dengan grid 3x3 dan menempatkan gambar pada posisi i+1
        plt.imshow(images[i].numpy().astype('uint8')) #menampilkan gambar dan mengonversi ke tipe uint8
        plt.title(class_names[labels[i]]) #menampilkan judul gambar sesuai dengan nama kelas
        plt.axis('off') #menonaktifkan sumbu pada gambar agar tidak terlihat
```

Mentah



Matang



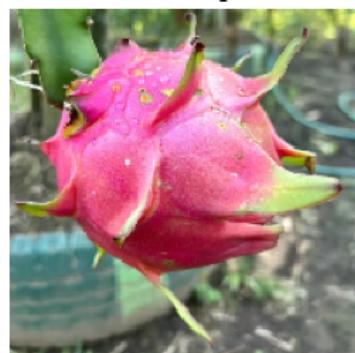
Mentah



Mentah



Matang



Matang



Matang



Mentah



Matang



Watermarkly

```
In [ ]: import numpy as np
```

```
# Tampilkan gambar dengan shape (32, 180, 180, 3)
for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape) # Output: (32, 180, 180, 3)
    #32: Jumlah gambar dalam batch.
    #180: Lebar gambar dalam piksel
    #180: Tinggi gambar dalam piksel
    #3: Jumlah channel gambar (RGB)
```

```
(32, 180, 180, 3)
```

```
In [ ]: #Mengatur AUTOTUNE untuk pemrosesan data otomatis oleh tensorflow
```

```
#AUTOTUNE digunakan untuk memungkinkan tensorflow mengoptimalkan jumlah thread secara otomatis saat memproses data
```

```
AUTOTUNE = tf.data.AUTOTUNE
```

```
In [ ]: #mengoptimalkan dataset pelatihan (train_ds)
```

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
#cache digunakan untuk menyimpan dataset di memori agar lebih cepat diakses
#shuffle mengacak data dalam batch agar model tidak terlalu terlatih pada urutan tertentu
#prefetch untuk menyiapkan data batch berikutnya secara otomatis
```

```
In [ ]: #mengoptimalkan dataset validasi (val_ds)
```

```
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
```

## Data Augmentation

- Digunakan untuk menambah variasi data pelatihan dengan membuat gambar baru dari yang sudah ada seperti dengan rotasi, flipping, zooming, dan sebagainya
- Untuk mengurangi overfitting dan memperbesar dataset tanpa mengumpulkan data baru

```
In [ ]: data_augmentation = Sequential([
```

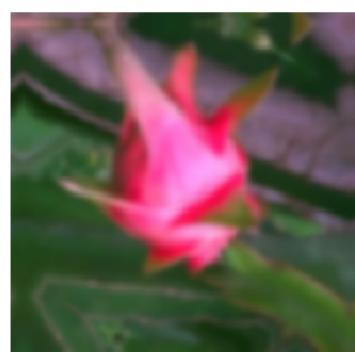
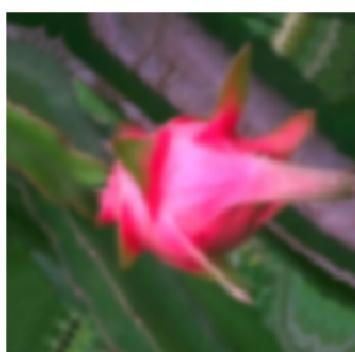
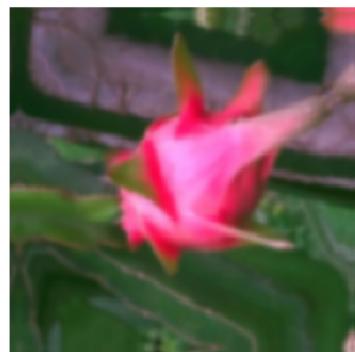
```
    layers.RandomFlip("horizontal", input_shape = (img_size,img_size,3)), #membalik gambar secara horizontal
    layers.RandomRotation(0.1), #merotasi gambar secara acak dalam kisaran 0°-36° (0.1 * 360)
    layers.RandomZoom(0.1) #melakukan zoom in/zoom out secara acak dengan rentang 10%
])
```

```
C:\Users\HP\AppData\Roaming\Python\Python312\site-packages\keras\src\layers\preprocessing\tf_data_layer.py:19: UserWarning: Do not pass an `input_shape` / `input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
super().__init__(**kwargs)
```



Watermarkly

```
In [ ]: #sama seperti sebelumnya, code ini digunakan untuk menampilkan gambar dari data_augmentation  
i = 0  
plt.figure(figsize=(10,10))  
  
for images, labels in train_ds.take(1):  
    for i in range(9):  
        images = data_augmentation(images)  
        plt.subplot(3,3, i+1)  
        plt.imshow(images[0].numpy().astype('uint8'))  
        plt.axis('off')
```



MobileNet

- Salah satu algoritma yang dirancang untuk perangkat dengan keterbatasan sumber daya seperti smartphone

 **Watermarkly**

```
In [ ]: #import Library yang dibutuhkan
from tensorflow.keras.applications import MobileNet #digunakan untuk memanfaatkan model yang sudah dilatih sebelumnya untuk pengenalan gambar
from tensorflow.keras.models import Model #digunakan untuk membuat dan mengonfigurasi arsitektur model

#membuat model dengan bobot yang telah dilatih sebelumnya
#include_top=False berarti tidak menggunakan lapisan klasifikasi dari mobilenet hanya bagian ekstraksi fitur
base_model = MobileNet(include_top=False, input_shape=(img_size, img_size, 3))

#membuka (unfreeze beberapa lapisan untuk proses fine tuning)
base_model.trainable = True #seluruh model bisa dilatih
fine_tune_at = len(base_model.layers) // 2 #menentukan bahwa setengah lapisan terakhir akan di unfreeze
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False #mengunci (freeze) lapisan pertama hingga setengah bagian pertama agar tidak dilatih kembali

#membuat model akhir dengan lapisan tambahan
model = Sequential([
    #sequential berarti lapisan diterapkan secara berurutan
    data_augmentation, #data augmentation untuk memperbanyak data Latih
    layers.Rescaling(1./255), #menormalisasi gambar, mengubah nilai pixel dari rentang [0, 255] menjadi [0, 1]
    base_model, #mobileNet sebagai dasar ekstraksi fitur
    layers.GlobalAveragePooling2D(), #lapisan pooling untuk meratakan hasil fitur dan mengurangi dimensi
    Dense(128, activation='relu'), #lapisan dense dengan 128 neuron dan aktivasi relu untuk pembelajaran non linear
    Dropout(0.3), #dropout dengan 30% neuron yang dihilangkan secara acak selama pelatihan
    Dense(len(class_names), activation='softmax') #lapisan output dengan jumlah neuron sesuai dengan jumlah kelas yang ingin diprediksi
        #aktivasi softmax untuk klasifikasi multi kelas
])

```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_22236\2739304275.py:5: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights
for input shape (224, 224) will be loaded as the default.
base_model = MobileNet(include_top=False, input_shape=(img_size, img_size, 3))
```

```
In [ ]: from tensorflow.keras.optimizers import Adam #untuk mengoptimalkan proses pelatihan model

#mengkompilasi model dengan optimizer, loss function, dan metrics
model.compile(
    optimizer=Adam(learning_rate=1e-4), #menggunakan optimizer Adam dengan Learning rate 0.0001
    loss='sparse_categorical_crossentropy', #untuk klasifikasi multi-kelas
    metrics=['accuracy']) #akurasi digunakan sebagai metrik evaluasi
)
```



Watermarkly

```
In [ ]: #menampilkan ringkasan dari model  
model.summary()  
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
sequential_2 (Sequential)	(None, 180, 180, 3)	0
rescaling_1 (Rescaling)	(None, 180, 180, 3)	0
mobilenet_1.00_224 (Functional)	(None, 5, 5, 1024)	3,228,864
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 1024)	0
dense_2 (Dense)	(None, 128)	131,200
dropout_1 (Dropout)	(None, 128)	0
dense_3 (Dense)	(None, 2)	258

Total params: 3,360,322 (12.82 MB)

Trainable params: 3,069,314 (11.71 MB)

Non-trainable params: 291,008 (1.11 MB)

```
In [ ]: #early stopping digunakan untuk menghentikan pelatihan lebih awal jika model tidak ada peningkatan
```

```
from tensorflow.keras.callbacks import EarlyStopping
```

```
early_stopping = EarlyStopping(monitor='val_accuracy', #memantau val_accuracy untuk menentukan kapan pelatihan dihentikan  
                                patience=3, #jumlah epoch berturut-turut tanpa peningkatan 'val_accuracy' sebelum dihentikan  
                                mode='max') #pelatihan berhenti ketika 'val_accuracy' tidak meningkat (nilai lebih besar diinginkan)
```

```
#melatih model menggunakan data Latih dan validasi dengan early stopping
```

```
history= model.fit(train_ds, #data pelatihan yang telah disiapkan  
                    epochs=30, # jumlah maksimal epoch  
                    validation_data=val_ds, #data validasi untuk mengevaluasi model pada setiap epoch  
                    callbacks=[early_stopping]) #menambahkan early stopping ke dalam callback untuk pelatihan
```

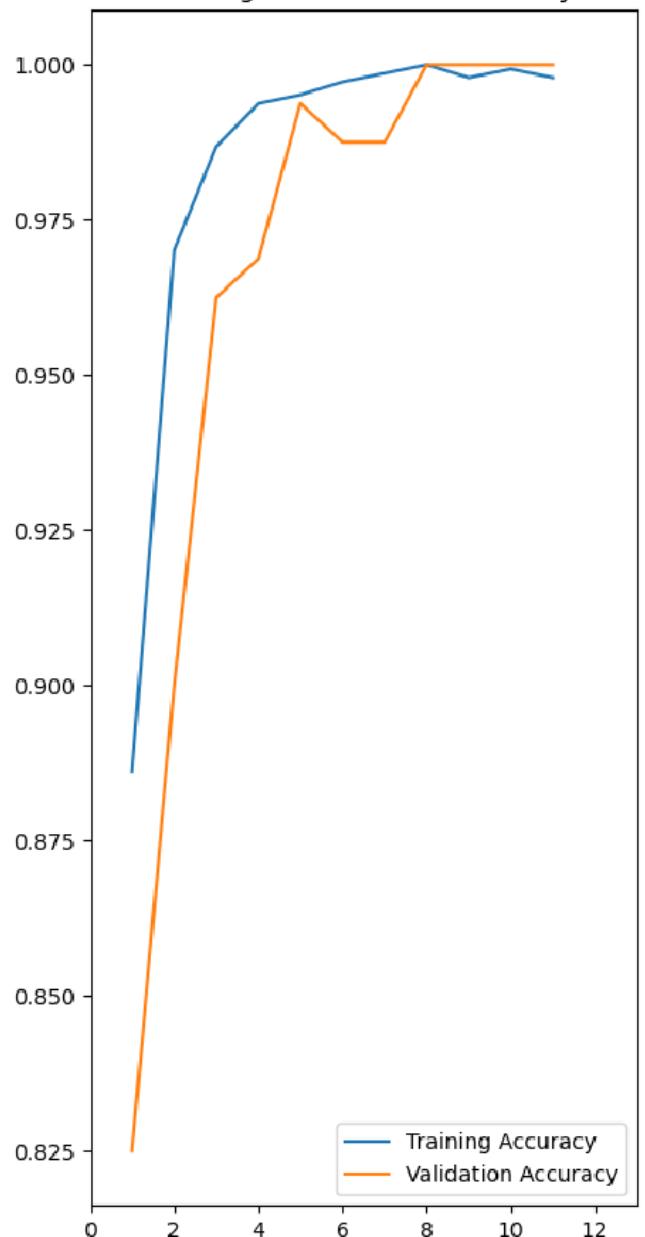
```
Epoch 1/30  
45/45 31s 463ms/step - accuracy: 0.8128 - loss: 0.4182 - val_accuracy: 0.8250 - val_loss: 0.4521  
Epoch 2/30  
45/45 19s 426ms/step - accuracy: 0.9647 - loss: 0.0970 - val_accuracy: 0.9000 - val_loss: 0.2666  
Epoch 3/30  
45/45 18s 410ms/step - accuracy: 0.9858 - loss: 0.0462 - val_accuracy: 0.9625 - val_loss: 0.1162  
Epoch 4/30  
45/45 18s 410ms/step - accuracy: 0.9922 - loss: 0.0258 - val_accuracy: 0.9688 - val_loss: 0.0412  
Epoch 5/30  
45/45 18s 407ms/step - accuracy: 0.9928 - loss: 0.0166 - val_accuracy: 0.9937 - val_loss: 0.0203  
Epoch 6/30  
45/45 19s 412ms/step - accuracy: 0.9983 - loss: 0.0110 - val_accuracy: 0.9875 - val_loss: 0.0312  
Epoch 7/30  
45/45 18s 411ms/step - accuracy: 0.9987 - loss: 0.0060 - val_accuracy: 0.9875 - val_loss: 0.0206  
Epoch 8/30  
45/45 18s 407ms/step - accuracy: 1.0000 - loss: 0.0038 - val_accuracy: 1.0000 - val_loss: 0.0041  
Epoch 9/30  
45/45 18s 405ms/step - accuracy: 0.9991 - loss: 0.0046 - val_accuracy: 1.0000 - val_loss: 0.0024  
Epoch 10/30  
45/45 18s 408ms/step - accuracy: 0.9999 - loss: 0.0033 - val_accuracy: 1.0000 - val_loss: 0.0032  
Epoch 11/30  
45/45 18s 407ms/step - accuracy: 0.9979 - loss: 0.0037 - val_accuracy: 1.0000 - val_loss: 0.0026
```



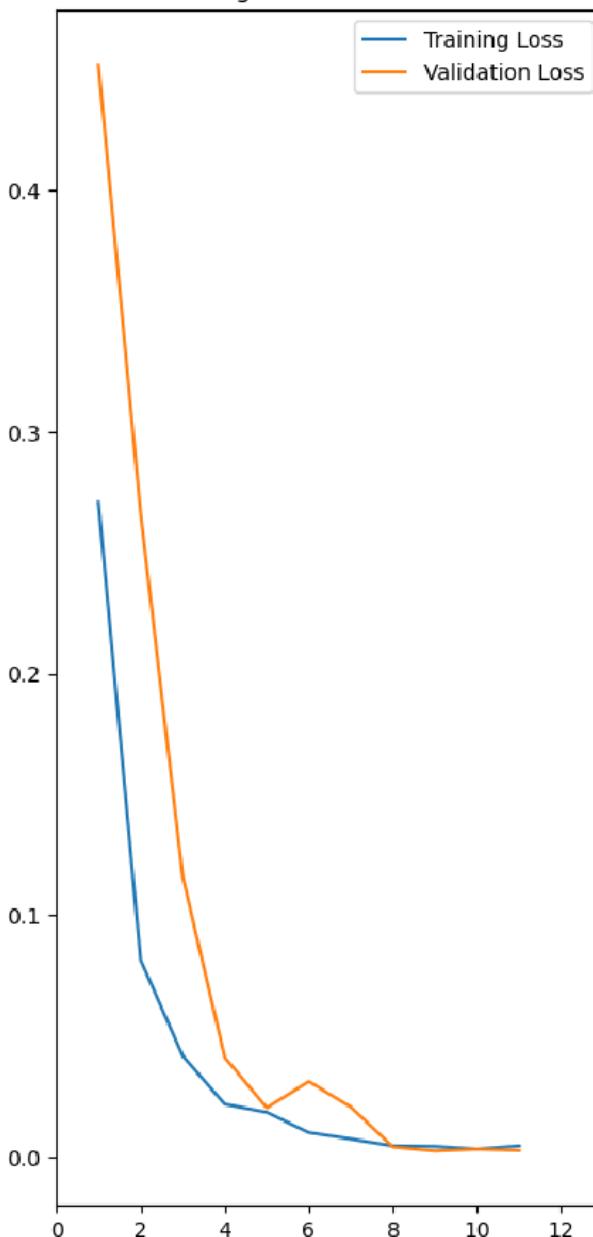
Watermarkly

```
In [ ]: #membuat range untuk epoch berdasarkan panjang data Loss dari pelatihan  
ephocs_range = range(1, len(history.history['loss']) + 1)  
  
plt.figure(figsize=(10, 10)) #membuat figure dengan ukuran 10x10 untuk menampilkan 2 grafik (Training and Validation Accuracy dan Loss)  
  
#grafik pertama (Training and Validation Accuracy)  
plt.subplot(1, 2, 1) #membuat subplot pertama dalam Layout 1 baris dan 2 kolom  
plt.plot(ephocs_range, history.history['accuracy'], label='Training Accuracy') #plot akurasi pelatihan  
plt.plot(ephocs_range, history.history['val_accuracy'], label='Validation Accuracy') #plot akurasi validasi  
plt.legend(loc='lower right') #membuat Legenda (informasi elemen visual) di sudut kanan bawah  
plt.xlim(0, 13) #mengatur batas nilai pada sumbu x dari epoch 1 sampai 13  
plt.title('Training and Validation Accuracy') #memberi judul grafik  
  
#grafik kedua (Training and Validation Loss)  
plt.subplot(1, 2, 2)  
plt.plot(ephocs_range, history.history['loss'], label='Training Loss')  
plt.plot(ephocs_range, history.history['val_loss'], label='Validation Loss')  
plt.legend(loc='upper right')  
plt.xlim(0, 13)  
plt.title('Training and Validation Loss')  
plt.show()
```

Training and Validation Accuracy



Training and Validation Loss



No Copy

No Copy

N

No Copy

py

No Cop

No Copy

No Copy

N

No Copy

py

No Cop

```
In [ ]: #menyimpan model yang telah dilatih  
model.save('model_mobilenet.h5')
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

No Copy

No Copy

N



Watermarkly

```
In [ ]: import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
from tensorflow.keras.models import load_model  
from PIL import Image
```

```
#memuat model yang sudah dilatih  
model = load_model(r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\model_mobilenet.h5') # Ganti dengan path model Anda  
class_names = ['Matang', 'Mentah'] #kelas yang ada pada model  
  
#fungsi untuk mengklasifikasikan gambar dan menyimpan gambar asli  
def classify_images(image_path, save_path='predicted_image.jpg'): try:  
    #memuat dan mempersiapkan gambar untuk prediksi  
    input_image = tf.keras.utils.load_img(image_path, target_size=(180, 180)) #membuat gambar dari path dan mnegubah ukurannya menjadi 180x180 pixel  
    input_image_array = tf.keras.utils.img_to_array(input_image) #mengubah gambar jadi array numpy agar bisa di proses model  
    input_image_exp_dim = tf.expand_dims(input_image_array, 0) #menambahkan dimensi batch agar sesuai dengan input model  
    #dimensi menjadi (1, 180, 180, 3)  
  
    #melakukan prediksi  
    predictions = model.predict(input_image_exp_dim) #melakukan prediksi pada gambar yang telah diproses  
    result = tf.nn.softmax(predictions[0]) #menghitung hasil prediksi menggunakan softmax untuk mendapatkan probabilitas tiap kelas  
    class_idx = np.argmax(result) #menemukan indeks kelas dengan probabilitas tertinggi  
    confidence = np.max(result) * 100 #menghitung confidence dalam persentase  
  
    #menampilkan hasil prediksi dan confidence  
    print(f"Prediksi: {class_names[class_idx]}") #menampilkan nama kelas yang diprediksi  
    print(f"Confidence: {confidence:.2f}%") #menampilkan nilai confidence  
  
    #menyimpan gambar asli tanpa teks  
    input_image = Image.open(image_path) #membuka gambar yang ada di path  
    input_image.save(save_path) #menyimpan gambar asli ke dalam path yang telah ditentukan  
  
    return f"Prediksi: {class_names[class_idx]} dengan confidence {confidence:.2f}%. Gambar asli disimpan di {save_path}."  
except Exception as e:  
    return f"Terjadi kesalahan: {e}"  
  
#contoh penggunaan fungsi
```

```
result = classify_images(r'test_data/Matang/Mature_Dragon_Original_Data0007.jpg', save_path='matang.jpg')  
print(result)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
1/1 _____ 1s 988ms/step
```

```
Prediksi: Matang  
Confidence: 73.11%  
Prediksi: Matang dengan confidence 73.11%. Gambar asli disimpan di matang.jpg.
```



Watermarkly

```
In [ ]: import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt

#memuat model yang telah dilatih sebelumnya
mobileNet_model = load_model(r'D:\SURYA\UAYJ\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\model_mobilenet.h5')

#memuat data test yang sebenarnya
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'test_data', #direktori data uji
    labels='inferred', #label otomatis dari subfolder yang ada
    label_mode='categorical', #menghasilkan Label dalam bentuk one-hot encoding
    batch_size=32, #ukuran batch untuk pemrosesan
    image_size=(180, 180) #ukuran gambar yang akan diproses
)
#prediksi model
y_pred = mobileNet_model.predict(test_data)
y_pred_class = tf.argmax(y_pred, axis=1) #konversi ke kelas prediksi

#ekstrak label sebenarnya dari test_data dan konversi ke bentuk indeks kelas
true_labels = [] #menyimpan label asli dalam bentuk indeks
for _, labels in test_data:
    true_labels.extend(tf.argmax(labels, axis=1).numpy()) #konversi one-hot ke indeks kelas
true_labels = tf.convert_to_tensor(true_labels) #mengkonversi list ke tensor untuk perhitungan

#membuat confusion matrix untuk evaluasi
conf_mat = tf.math.confusion_matrix(true_labels, y_pred_class)

#menghitung akurasi berdasarkan confusion matrix
accuracy = tf.reduce_sum(tf.linalg.diag_part(conf_mat)) / tf.reduce_sum(conf_mat)

#menghitung presisi dan recall dari confusion matrix
precision = tf.linalg.diag_part(conf_mat) / tf.reduce_sum(conf_mat, axis=0)
recall = tf.linalg.diag_part(conf_mat) / tf.reduce_sum(conf_mat, axis=1)

#menghitung F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

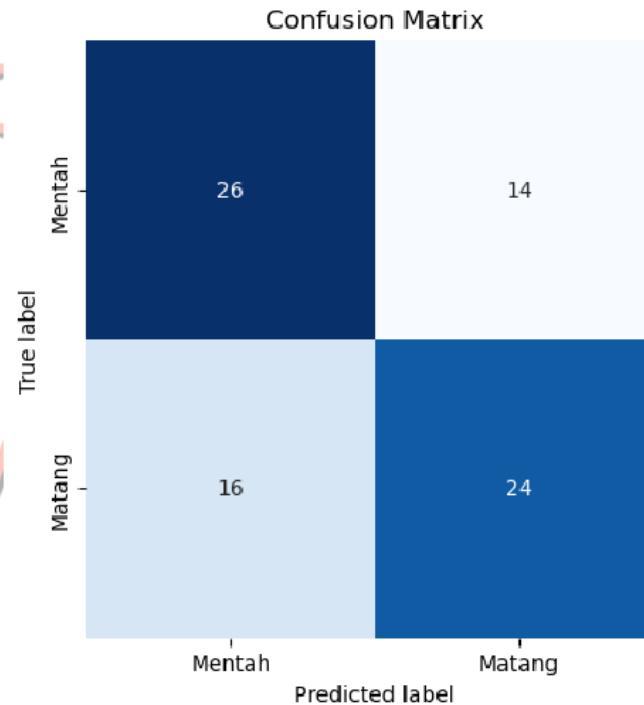
#visualisasi Confusion Matrix
plt.figure(figsize=(6, 5)) #mengatur ukuran gambar
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues', #annot=True untuk menampilkan angka di dalam setiap sel matriks
            #fmt='d' untuk menampilkan bilangan bulat tanpa desimal
            xticklabels=["Mentah", "Matang"], yticklabels=["Mentah", "Matang"])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# Menampilkan hasil
print("Confusion Matrix:\n", conf_mat.numpy())
print("Akurasi:", accuracy.numpy())
print("Presisi:", precision.numpy())
print("Recall:", recall.numpy())
print("F1 Score:", f1_score.numpy())
```

WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile\_metrics` will be empty until you train or evaluate the model.

Found 80 files belonging to 2 classes.  
3/3 2s 448ms/step





```
Confusion Matrix:  
[[26 14]  
 [16 24]]  
Akurasi: 0.625  
Presisi: [0.61904762 0.63157895]  
Recall: [0.65 0.6 ]  
F1 Score: [0.63414634 0.61538462]
```

## GoogleNet

```
In [ ]: import tensorflow as tf
import cv2
import numpy as np
from matplotlib import pyplot as plt
#Load data
data_dir = r"C:\Kuliah\ASDOS ML\CNN Part 2\train_data"
#Randomize data yang telah di Load sekaligus resize menjadi 180 x 180
data = tf.keras.utils.image_dataset_from_directory(data_dir, seed = 123, image_size=(180, 180), batch_size=16)
print(data.class_names)

class_names = data.class_names

img_size = 180
batch = 32
validation_split = 0.1
dataset = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch,
)
total_count = len(dataset)
val_count = int(total_count * validation_split)
train_count = total_count - val_count

print("Total Images:", total_count)
print("Train Images:", train_count)
print("Validation Images:", val_count)

train_ds = dataset.take(train_count)
val_ds = dataset.skip(train_count)
```

Found 1600 files belonging to 2 classes.  
['Matang', 'Mentah']



Watermarkly

In [ ]: import matplotlib.pyplot as plt

```
i = 0
plt.figure(figsize=(10,10))

#tampilkan untuk memastikan data sudah di Load
for images, labels in train_ds.take(1):
    for i in range(9):
        plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(class_names[labels[i]])
        plt.axis('off')
```

Mentah



Matang



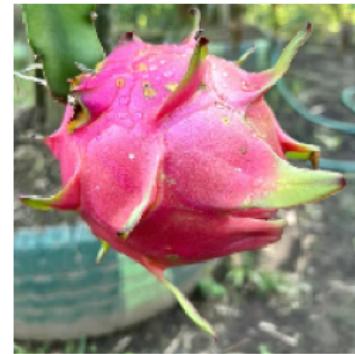
Mentah



Mentah



Matang



Matang



Matang



Mentah



Matang



Watermarkly

```
In [5]: for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape)

#Loop untuk mengecek atribut gambar(jumlah, tinggi, Lebar, dan channel(RGB))
```

(32, 180, 180, 3)

No Copy No Copy No Copy N  
opy No Copy No Copy No Cop  
No Copy No Copy No Copy N  
opy No Copy No Copy No Cop  
No Copy No Copy No Copy N

```
In [ ]: from tensorflow.keras import layers
```

```
from tensorflow.keras.models import Sequential, load_model
```

```
Tuner = tf.data.AUTOTUNE  
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = Tuner)  
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = Tuner)
```

```
#Augmentasi data dengan menggunakan Sequential
```

```
data_augmentation = Sequential([  
    layers.RandomFlip("horizontal", input_shape = (img_size,img_size,3)),  
    layers.RandomRotation(0.1),  
    layers.RandomZoom(0.1)
```

```
])
```

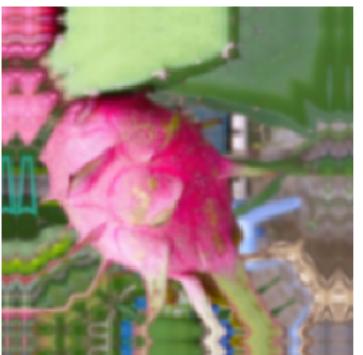
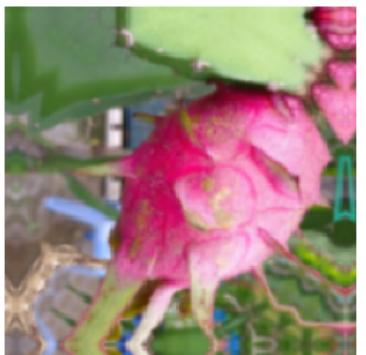
```
i = 0  
plt.figure(figsize=(10,10))  
#Lihat data setelah di augmentasi  
for images, labels in train_ds.take(69):  
    for i in range(9):  
        images = data_augmentation(images)  
        plt.subplot(3,3, i+1)  
        plt.imshow(images[0].numpy().astype('uint8'))  
        plt.axis('off')
```

```
c:\Users\jppjo\anaconda3\Lib\site-packages\keras\src\layers\preprocessing\tf_data_layer.py:18: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer.  
When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.  
super().__init__(**kwargs)
```

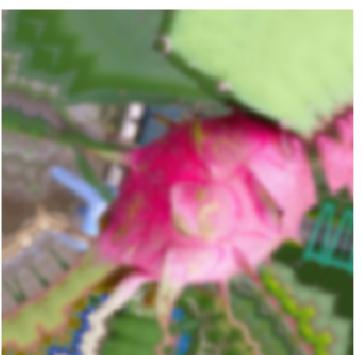
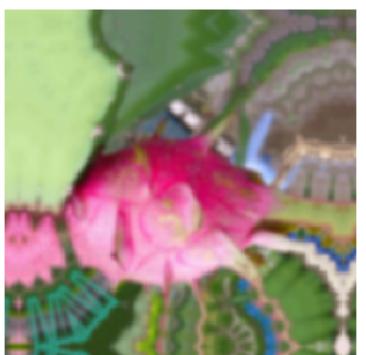
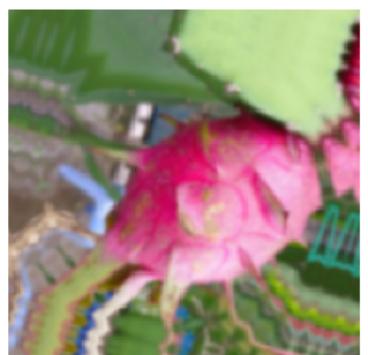
No Copy



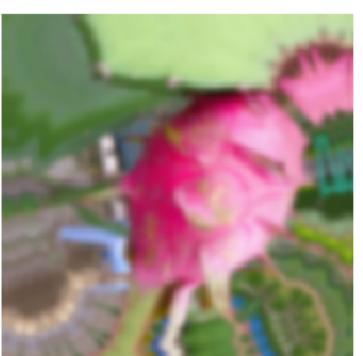
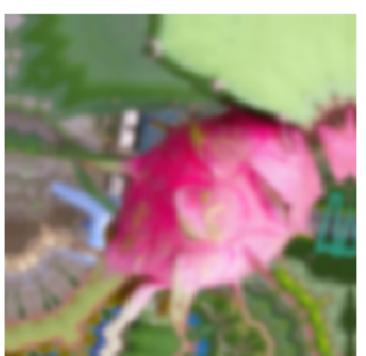
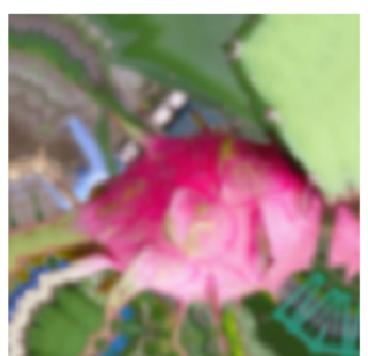
Watermarkly



No Copy



opy No Cop



No Copy

No Copy

No Copy

No Copy



Watermarkly

```
In [ ]: import tensorflow as tf
```

```
import keras
```

```
import keras._tf_keras.keras.backend as K
from keras._tf_keras.keras.models import Model
from keras._tf_keras.keras.layers import Input, Dense, Conv2D
from keras._tf_keras.keras.layers import Flatten, MaxPool2D, AvgPool2D
from keras._tf_keras.keras.layers import Concatenate, Dropout

from keras._tf_keras.keras.models import load_model

#membuat model from scratch
def googlenet(input_shape, n_classes):

    def inception_block(x, f):
        t1 = Conv2D(f[0], 1, activation='relu')(x)

        t2 = Conv2D(f[1], 1, activation='relu')(x)
        t2 = Conv2D(f[2], 3, padding='same', activation='relu')(t2)

        t3 = Conv2D(f[3], 1, activation='relu')(x)
        t3 = Conv2D(f[4], 5, padding='same', activation='relu')(t3)

        t4 = MaxPool2D(3, 1, padding='same')(x)
        t4 = Conv2D(f[5], 1, activation='relu')(t4)

        output = Concatenate()([t1, t2, t3, t4])
        return output

    input = Input(input_shape)

    x = Conv2D(64, 7, strides=2, padding='same', activation='relu')(input)
    x = MaxPool2D(3, strides=2, padding='same')(x)

    x = Conv2D(64, 1, activation='relu')(x)
    x = Conv2D(192, 3, padding='same', activation='relu')(x)
    x = MaxPool2D(3, strides=2)(x)

    x = inception_block(x, [64, 96, 128, 16, 32, 32])
    x = inception_block(x, [128, 128, 192, 32, 96, 64])
    x = MaxPool2D(3, strides=2, padding='same')(x)

    x = inception_block(x, [192, 96, 208, 16, 48, 64])
    x = inception_block(x, [160, 112, 224, 24, 64, 64])
    x = inception_block(x, [128, 128, 256, 24, 64, 64])
    x = inception_block(x, [112, 144, 288, 32, 64, 64])
    x = inception_block(x, [256, 160, 320, 32, 128, 128])
    x = MaxPool2D(3, strides=2, padding='same')(x)

    x = inception_block(x, [256, 160, 320, 32, 128, 128])
    x = inception_block(x, [384, 192, 384, 48, 128, 128])

    x = AvgPool2D(3, strides=1)(x)
    x = Dropout(0.4)(x)

    x = Flatten()(x)
    output = Dense(n_classes, activation='softmax')(x)

    model = Model(input, output)
    return model
#Pastikan input shape dan jumlah kelas sesuai
input_shape = 180, 180, 3
n_classes = 2
```



Watermarkly

```
#Clear Cache Keras menggunakan clear session
```

```
K.clear_session()  
#buat model dengan
```

```
model = googlenet(input_shape, n_classes)  
model.summary()
```

```
WARNING:tensorflow:From c:\Users\jppjo\anaconda3\lib\site-packages\keras\src\backend\common\global_state.py:82: The name tf.reset_default_graph is deprecated. Please use tf.compat.v1.reset_default_graph instead.
```

```
Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 180, 180, 3)	0	-
conv2d (Conv2D)	(None, 90, 90, 64)	9,472	input_layer[0][0]
max_pooling2d (MaxPooling2D)	(None, 45, 45, 64)	0	conv2d[0][0]
conv2d_1 (Conv2D)	(None, 45, 45, 64)	4,160	max_pooling2d[0]...

```
In [ ]: from tensorflow.keras.callbacks import EarlyStopping  
from tensorflow.keras.optimizers import Adam  
#Coimpile dengan optimizer adam  
model.compile(  
    optimizer=Adam(),  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy'])  
  
#buat early stopping  
early_stopping = EarlyStopping(monitor='val_accuracy',  
                               patience=5,  
                               mode='max')  
  
#fit validation data ke dalam model  
history= model.fit(train_ds,  
                    epochs=30,  
                    validation_data=val_ds,  
                    callbacks=[early_stopping])
```

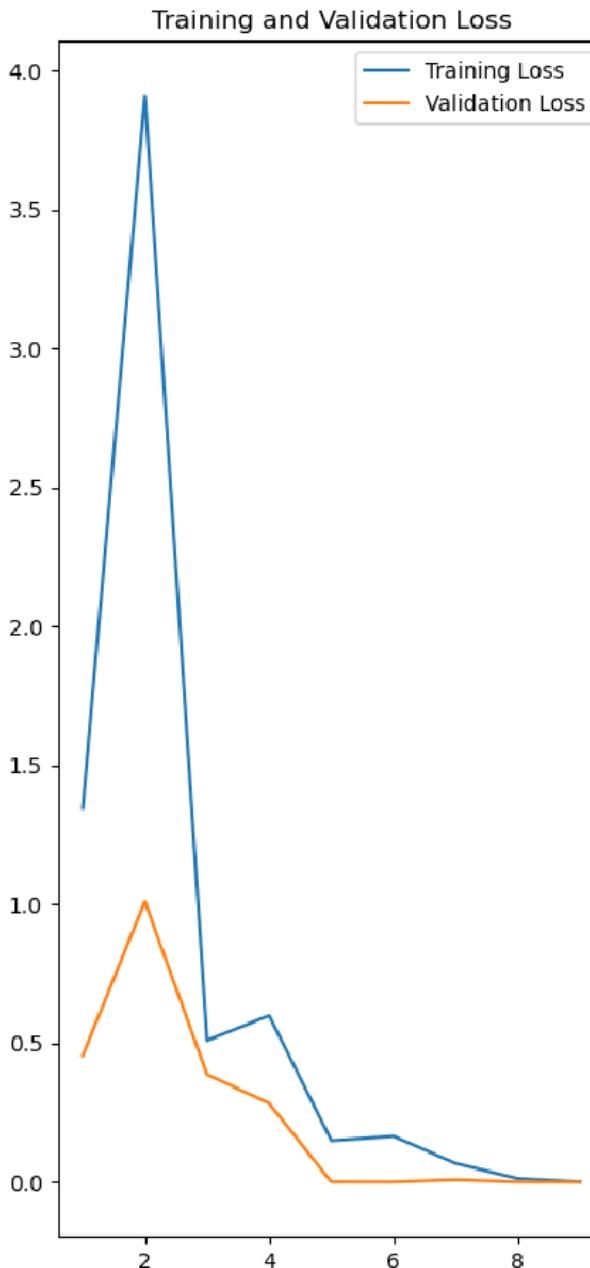
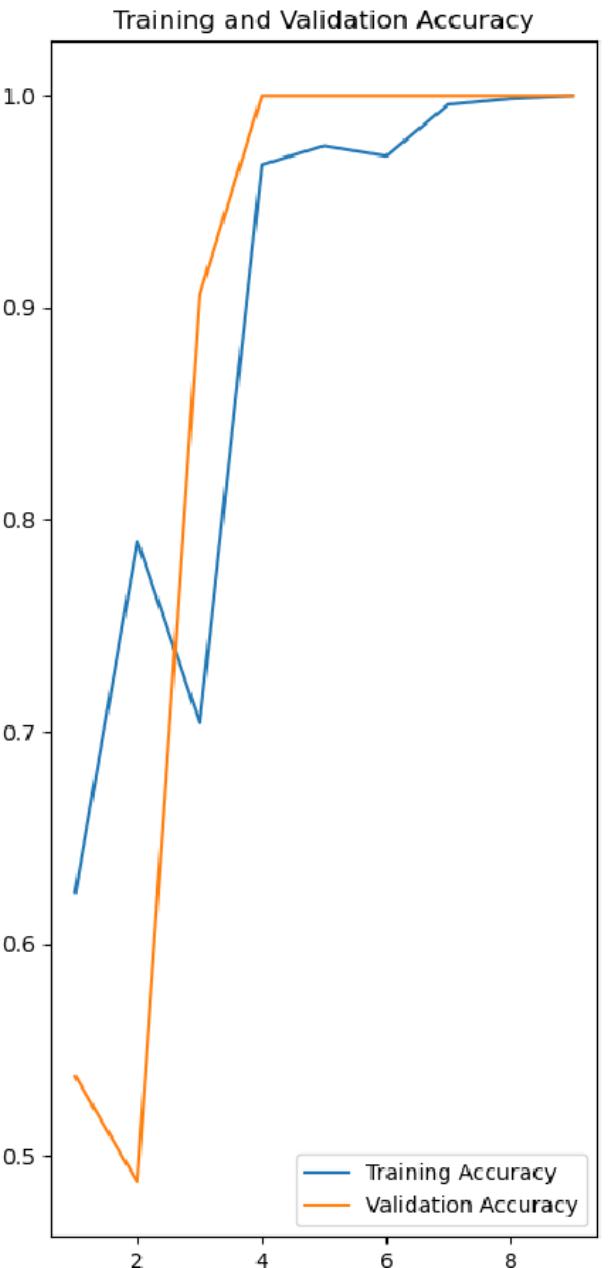
```
Epoch 1/30  
45/45 45s 722ms/step - accuracy: 0.5372 - loss: 2.7571 - val_accuracy: 0.5375 - val_loss: 0.4554  
Epoch 2/30  
45/45 31s 685ms/step - accuracy: 0.8222 - loss: 1.7091 - val_accuracy: 0.4875 - val_loss: 1.0099  
Epoch 3/30  
45/45 31s 686ms/step - accuracy: 0.5750 - loss: 0.6798 - val_accuracy: 0.9062 - val_loss: 0.3879  
Epoch 4/30  
45/45 31s 690ms/step - accuracy: 0.9669 - loss: 0.2555 - val_accuracy: 1.0000 - val_loss: 0.2833  
Epoch 5/30  
45/45 34s 756ms/step - accuracy: 0.9597 - loss: 0.2722 - val_accuracy: 1.0000 - val_loss: 4.5508e-04  
Epoch 6/30  
45/45 34s 745ms/step - accuracy: 0.9780 - loss: 0.1443 - val_accuracy: 1.0000 - val_loss: 2.4184e-06  
Epoch 7/30  
45/45 33s 728ms/step - accuracy: 0.9952 - loss: 0.0831 - val_accuracy: 1.0000 - val_loss: 0.0083  
Epoch 8/30  
45/45 33s 730ms/step - accuracy: 0.9976 - loss: 0.0321 - val_accuracy: 1.0000 - val_loss: 2.2500e-05  
Epoch 9/30  
45/45 33s 726ms/step - accuracy: 1.0000 - loss: 9.5568e-04 - val_accuracy: 1.0000 - val_loss: 1.0728e-05
```



Watermarkly

```
In [ ]: #buat plot dengan menggunakan history supaya jumlahnya sesuai epoch yang dilakukan
ephocs_range = range(1, len(history.history['loss']) + 1)
plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.plot(ephocs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(ephocs_range, history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(ephocs_range, history.history['loss'], label='Training Loss')
plt.plot(ephocs_range, history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



```
In [10]: model.save('gugelnet.h5')
```

WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save\_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my\_model.keras')` or `keras.saving.save\_model(model, 'my\_model.keras')`.

```
In [11]:
```

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from PIL import Image

# Load the trained model
model = load_model(r'C:\Kuliah\ASDOS ML\CNN Part 2\gugelnet.h5') # Ganti dengan path model Anda
class_names = ['Matang', 'Mentah']

# Function to classify images and save the original image
def classify_images(image_path, save_path='predicted_image.jpg'):
    try:
        # Load and preprocess the image
        input_image = tf.keras.utils.load_img(image_path, target_size=(180, 180))
        input_image_array = tf.keras.utils.img_to_array(input_image)
        input_image_exp_dim = tf.expand_dims(input_image_array, 0) # Add batch dimension

        # Predict
        predictions = model.predict(input_image_exp_dim)
        result = tf.nn.softmax(predictions[0])
        class_idx = np.argmax(result)
        confidence = np.max(result) * 100

        # Display prediction and confidence in notebook
        print(f"Prediksi: {class_names[class_idx]}")
        print(f"Confidence: {confidence:.2f}%")

        # Save the original image (without text)
        input_image = Image.open(image_path)
        input_image.save(save_path)

        return f"Prediksi: {class_names[class_idx]} dengan confidence {confidence:.2f}%. Gambar asli disimpan di {save_path}."
    except Exception as e:
        return f"Terjadi kesalahan: {e}"

# Contoh penggunaan fungsi
result = classify_images(r'test_data\Mentah\Immature_Dragon_Original_Data0012.jpg', save_path='mentah2.jpg')
print(result)
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the model.
```

```
1/1 _____ 1s 511ms/step
```

```
Prediksi: Mentah
```

```
Confidence: 73.11%
```

```
Prediksi: Mentah dengan confidence 73.11%. Gambar asli disimpan di mentah2.jpg.
```



Watermarkly

```
In [17]: import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt

# Muat data test yang sebenarnya
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'test_data',
    labels='inferred',
    label_mode='categorical', # Menghasilkan Label dalam bentuk one-hot encoding
    batch_size=32,
    image_size=(180, 180)
)

# Prediksi model
y_pred = model.predict(test_data)
y_pred_class = tf.argmax(y_pred, axis=1) # Konversi ke kelas prediksi

# Ekstrak Label sebenarnya dari test_data dan konversi ke bentuk indeks kelas
true_labels = []
for _, labels in test_data:
    true_labels.extend(tf.argmax(labels, axis=1).numpy()) # Konversi one-hot ke indeks kelas
true_labels = tf.convert_to_tensor(true_labels)

# Membuat matriks kebingungan
conf_mat = tf.math.confusion_matrix(true_labels, y_pred_class)

# Menghitung akurasi
accuracy = tf.reduce_sum(tf.linalg.diag_part(conf_mat)) / tf.reduce_sum(conf_mat)

# Menghitung presisi dan recall
precision = tf.linalg.diag_part(conf_mat) / tf.reduce_sum(conf_mat, axis=0)
recall = tf.linalg.diag_part(conf_mat) / tf.reduce_sum(conf_mat, axis=1)

# Menghitung F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

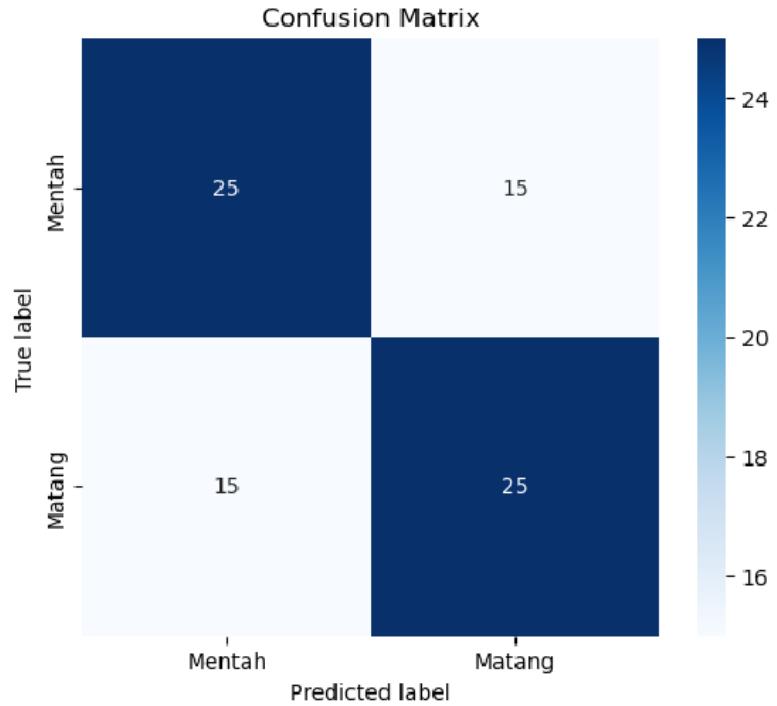
# Visualisasi Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
            xticklabels=["Mentah", "Matang"], yticklabels=["Mentah", "Matang"])
plt.title('Confusion Matrix')
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()

# Menampilkan hasil
print("Confusion Matrix:\n", conf_mat.numpy())
print("Akurasi:", accuracy.numpy())
print("Presisi:", precision.numpy())
print("Recall:", recall.numpy())
print("F1 Score:", f1_score.numpy())
```

Found 80 files belonging to 2 classes.  
3/3 1s 192ms/step



Watermarkly



Confusion Matrix:  
[[25 15]  
[15 25]]  
Akurasi: 0.625  
Presisi: [0.625 0.625]  
Recall: [0.625 0.625]  
F1 Score: [0.625 0.625]

### Streamlit

- Buat file baru yang berekstensi Python (.py)
- Salin code dibawah ini untuk membuat interface



Watermarkly

```
In [ ]: import streamlit as st
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import load_model
from PIL import Image

# Load the pre-trained model
# Sesuaikan dengan path model Anda (model terbaik yang di dump dalam format .h5)
model = load_model(r'D:\SURYA\UAJY\Semester 5\Asdos Machine Learning\Pemegang Modul\Modul CNN>Notebook2\model_mobilenet.h5')
class_names = ['Matang', 'Mentah']

# Function to preprocess and classify image
def classify_image(image_path):
    try:
        # Load and preprocess the image
        input_image = tf.keras.utils.load_img(image_path, target_size=(180, 180))
        input_image_array = tf.keras.utils.img_to_array(input_image)
        input_image_exp_dim = tf.expand_dims(input_image_array, 0)

        # Predict using the model
        predictions = model.predict(input_image_exp_dim)
        result = tf.nn.softmax(predictions[0]) # Apply softmax for probability

        # Get class with highest confidence
        class_idx = np.argmax(result)
        confidence_scores = result.numpy()
        return class_names[class_idx], confidence_scores
    except Exception as e:
        return "Error", str(e)

# Function to create a custom progress bar
def custom_progress_bar(confidence, color1, color2):
    percentage1 = confidence[0] * 100 # Confidence for class 0 (Matang)
    percentage2 = confidence[1] * 100 # Confidence for class 1 (Mentah)
    progress_html = f"""


{percentage1:.2f}%



{percentage2:.2f}%


"""

    st.sidebar.markdown(progress_html, unsafe_allow_html=True)

# Streamlit UI
st.title("Prediksi Kematangan Buah Naga - XXXX") # 4 digit npm terakhir

# Upload multiple files in the main page
uploaded_files = st.file_uploader("Unggah Gambar (Beberapa diperbolehkan)", type=["jpg", "png", "jpeg"], accept_multiple_files=True)

# Sidebar for prediction button and results
if st.sidebar.button("Prediksi"):
    if uploaded_files:
        st.sidebar.write("### Hasil Prediksi")
        for uploaded_file in uploaded_files:
            with open(uploaded_file.name, "wb") as f:
                f.write(uploaded_file.getbuffer())

        # Perform prediction
        label, confidence = classify_image(uploaded_file.name)

        if label != "Error":
            # Define colors for the bar and Label
            if label == 'Matang':
                color1, color2 = "#4CAF50", "#FF9800"
            else:
                color1, color2 = "#FF9800", "#4CAF50"

            custom_progress_bar(confidence, color1, color2)
```

```
primary_color = "#007BFF" # Blue for "Matang"
secondary_color = "#FF4136" # Red for "Mentah"
label_color = primary_color if label == "Matang" else secondary_color

# Display prediction results
st.sidebar.write(f"**Nama File:** {uploaded_file.name}")
st.sidebar.markdown(f"<h4 style='color: {label_color};>Prediksi: {label}</h4>", unsafe_allow_html=True)

# Display confidence scores
st.sidebar.write("**Confidence:**")
for i, class_name in enumerate(class_names):
    st.sidebar.write(f"- {class_name}: {confidence[i] * 100:.2f}%")

# Display custom progress bar
custom_progress_bar(confidence, primary_color, secondary_color)

st.sidebar.write("---")
else:
    st.sidebar.error(f"Kesalahan saat memproses gambar {uploaded_file.name}: {confidence}")
else:
    st.sidebar.error("Silakan unggah setidaknya satu gambar untuk diprediksi.")

# Preview images in the main page
if uploaded_files:
    st.write("## Preview Gambar")
    for uploaded_file in uploaded_files:
        image = Image.open(uploaded_file)
        st.image(image, caption=f"{uploaded_file.name}", use_column_width=True)
```

