CODE UAS (Scikit-Learn) PMDPM 2024

1. Notebook AlexNet (Arya)

```
import os
import numpy as np
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
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

count = 0
dirs = os.listdir(r'C:\Users\Lenovo\Downloads\uas\foto\train_data')
for dir in dirs:
    files = list(os.listdir(r'C:\Users\Lenovo\Downloads\uas\foto\train_data/'+dir))
    print(dir + ' Folder has ' + str(len(files)) + ' Images')
    count = count + len(files)
print('Images Folder has ' + str(count) + ' Images')

base_dir = r'C:\Users\Lenovo\Downloads\uas\foto\train_data'
img_size = 180
batch = 32
validation_split = 0.1
test_split = 0.1

dataset = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch,
)
class_names = dataset.class_names
print("Class Names:", class_names)

total_count = len(dataset)
test_count = int(total_count * test_split)
val_count = int(total_count * validation_split)
train_count = total_count - val_count - test_count

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

train_ds = dataset.take(train_count)
temp_ds = dataset.skip(train_count)
val_ds = temp_ds.take(val_count)
test_ds = temp_ds.skip(val_count)

import matplotlib.pyplot as plt

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

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')
for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape)
import tensorflow as tf
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# Assuming train_ds and val_ds are already defined and preprocessed

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)

# Define data augmentation layers
data_augmentation = Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1)
])

# Display some augmented images
i = 0
plt.figure(figsize=(10, 10))

# Show augmented images
for images, labels in train_ds.take(1):
    for i in range(9):
        augmented_image = data_augmentation(images[i:i+1])  # Augmenting a single image
        plt.subplot(3, 3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.axis('off')

plt.show()
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

def alexnet(input_image):
    # Step 1: Input layer (224x224x3 image)
    x = preprocess(input_image)  # Normalize and resize image to 224x224x3

    # Step 2: Convolutional Layers
    x = conv_layer(x, 96, kernel_size=11, stride=4, activation=ReLU)
    x = max_pool(x, size=3, stride=2)

    x = conv_layer(x, 256, kernel_size=5, stride=1, activation=ReLU)
    x = max_pool(x, size=3, stride=2)

    x = conv_layer(x, 384, kernel_size=3, stride=1, activation=ReLU)
    x = conv_layer(x, 384, kernel_size=3, stride=1, activation=ReLU)
    x = conv_layer(x, 256, kernel_size=3, stride=1, activation=ReLU)
    x = max_pool(x, size=3, stride=2)

    # Step 3: Flatten
    x = flatten(x)

    # Step 4: Fully Connected Layers
```

```
    x = fc_layer(x, 4096, activation=ReLU)
    x = dropout(x, rate=0.5)  # Apply dropout

    x = fc_layer(x, 4096, activation=ReLU)
    x = dropout(x, rate=0.5)

    x = fc_layer(x, 1000, activation=Softmax)  # Output layer

    # Step 5: Output the predicted probabilities
    return x  # 1000-dimensional vector for classification probabilities


input_shape = (180, 180, 3)
n_classes = 3
K.clear_session()
model = vgg16(input_shape, n_classes)
model.summary()

from tensorflow.keras.optimizers import Adam

# Compile the model
early_stoping = model.compile(
    optimizer=Adam(),  # You can use any optimizer (e.g., Adam, SGD, etc.)
    loss='sparse_categorical_crossentropy',  # For multi-class classification
    metrics=['accuracy']  # Metrics to track during training
)

# Fit the model with early stopping
history = model.fit(
    train_ds,
    epochs=30,
    validation_data=val_ds,
    callbacks=[early_stopping]
)
epochs_range = range(1, len(history.history['loss']) + 1)

plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(epochs_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(epochs_range, history.history['loss'], label='Training Loss')
plt.plot(epochs_range, history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

model.save('BestModel_AlexNet_Scikit-Learn.h5')

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

model    =    load_model(r'C:\Users\Lenovo\Downloads\uas\BestModel_AlexNet_Scikit-
Learn.h5')
class_names = ['Matang', 'Mentah', 'Setengah Matang']

def classify_images(image_path, save_path='predicted_image.jpg'):
    try:
```

```
        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)

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

        print(f"Prediksi: {class_names[class_idx]}")
        print(f"Confidence: {confidence:.2f}%")

        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}"

result                                                                          =
classify_images(r"C:\Users\Lenovo\Downloads\uas\foto\test_data\Mentah\hijau-test-
9.jpg")
print(result)
import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt

AlexNet_model                                                                   =
load_model(r'C:\Users\Lenovo\Downloads\uas\BestModel_AlexNet_Scikit-Learn.h5')

test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'C:\Users\Lenovo\Downloads\uas\foto\test_data',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(180, 180)
)

y_pred = AlexNet_model.predict(test_data)
y_pred_class = tf.argmax(y_pred, axis=1)

true_labels = []
for _, labels in test_data:
    true_labels.extend(tf.argmax(labels, axis=1).numpy())
true_labels = tf.convert_to_tensor(true_labels)

conf_mat = tf.math.confusion_matrix(true_labels, y_pred_class)

accuracy = tf.reduce_sum(tf.linalg.diag_part(conf_mat)) / tf.reduce_sum(conf_mat)

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)

f1_score = 2 * (precision * recall) / (precision + recall)

plt.figure(figsize=(6, 5))
```

```
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
            xticklabels=["Matang",       "Mentah",       "Setengah       Matang"],
yticklabels=["Matang", "Mentah", "Setengah 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())
```

2. VGG-16 (Chris)

```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt

data_dir = r"C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\foto\train_data"

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
test_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)
test_count = int(total_count * test_split)
val_count = int(total_count * validation_split)
train_count = total_count - val_count - test_count

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

train_ds = dataset.take(train_count)
temp_ds = dataset.skip(train_count)
val_ds = temp_ds.take(val_count)
test_ds = temp_ds.skip(val_count)

import matplotlib.pyplot as plt

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


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')

for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape)

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)

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))

#tampilkan untuk memastikan data sudah di load
for images, labels in train_ds.take(1):
    for i in range(9):
        augmented_image = data_augmentation(images[i:i+1])
        plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.axis('off')
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 vgg16(input_shape, n_classes):
    model = Sequential()

    # Block 1
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same', input_shape=input_shape))
    model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(MaxPool2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(MaxPool2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(MaxPool2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
```

```
    model.add(MaxPool2D((2, 2), strides=(2, 2)))

    # Block 5
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(MaxPool2D((2, 2), strides=(2, 2)))

    # Fully connected layers
    model.add(Flatten())
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(n_classes, activation='softmax'))

    return model

input_shape = (180, 180, 3)
n_classes = 3

K.clear_session()

model = vgg16(input_shape, n_classes)

model.summary()
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=5,
    mode='max'
)

history = model.fit(
    train_ds,
    epochs=30,
    validation_data=val_ds,
    callbacks=[early_stopping]
)

epochs_range = range(1, len(history.history['loss']) + 1)

plt.figure(figsize=(10, 10))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(epochs_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(epochs_range, history.history['loss'], label='Training Loss')
plt.plot(epochs_range, history.history['val_loss'], label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()

model.save('BestModel_VGG-16_Scikit-Learn.h5')
```

```
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:\Users\Lenovo\OneDrive\Documents\K\S      5\PMdPM\Projek-UAS-
PMDPM_B_Scikit-Learn\BestModel_VGG-16_Scikit-Learn.h5')
class_names = ['Matang', 'Mentah', 'Setengah Matang']

# 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"C:\Users\Lenovo\OneDrive\Documents\K\S
5\PMdPM\foto\test_data\Setengah Matang\kuning-test-4.jpg")
print(result)
import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt


test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\foto\test_data',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(180, 180)
)

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

true_labels = []
for _, labels in test_data:
    true_labels.extend(tf.argmax(labels, axis=1).numpy())
true_labels = tf.convert_to_tensor(true_labels)

conf_mat = tf.math.confusion_matrix(true_labels, y_pred_class, num_classes=3)
```

```
accuracy = tf.reduce_sum(tf.linalg.diag_part(conf_mat)) / tf.reduce_sum(conf_mat)

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)

f1_score = 2 * (precision * recall) / (precision + recall)

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

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())
```

3. MobileNet (Alexa)

```
import os
import numpy as np
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

count = 0
dirs = os.listdir(r'C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\foto\train_data')
for dir in dirs:
    files             =             list(os.listdir(r'C:\Users\Lenovo\OneDrive\Documents\K\S
5\PMdPM\foto\train_data/'+dir))
    print(dir + ' Folder has ' + str(len(files)) + ' Images')
    count = count + len(files)
print('Images Folder has ' + str(count) + ' Images')

base_dir = r'C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\foto\train_data'
img_size = 180
batch = 32
validation_split = 0.1
test_split = 0.1

dataset = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch,
)
class_names = dataset.class_names
print("Class Names:", class_names)
total_count = len(dataset)
test_count = int(total_count * test_split)
val_count = int(total_count * validation_split)
train_count = total_count - val_count - test_count

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

```
train_ds = dataset.take(train_count)
temp_ds = dataset.skip(train_count)
val_ds = temp_ds.take(val_count)
test_ds = temp_ds.skip(val_count)

import matplotlib.pyplot as plt

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

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')

import numpy as np


for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape)
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)

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))

for images, labels in train_ds.take(1):
    for i in range(9):
        augmented_image = data_augmentation(images[i:i+1])
        plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.axis('off')
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Model

base_model = MobileNet(include_top=False, input_shape=(img_size, img_size, 3))

base_model.trainable = True
fine_tune_at = len(base_model.layers) // 2
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    base_model,
    layers.GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(len(class_names), activation='softmax')
])
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
```

```
)
model.summary()
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(
    monitor='val_accuracy',
    patience=3,
    mode='max'
)

history= model.fit(
    train_ds,
    epochs=30,
    validation_data=val_ds,
    callbacks=[early_stopping]
)
ephocs_range = range(1, len(history.history['loss']) + 1)

plt.figure(figsize=(10, 10))

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.xlim(0, 13)
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.xlim(0, 13)
plt.title('Training and Validation Loss')
plt.show()

model.save('BestModel_MobileNet_Scikit-Learn.h5')

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

model      =      load_model(r'C:\Users\Lenovo\OneDrive\Documents\K\S      5\PMdPM\Projek-UAS-
PMDPM_B_Scikit-Learn\BestModel_MobileNet_Scikit-Learn.h5')
class_names = ['Matang', 'Mentah', 'Setengah Matang']

def classify_images(image_path, save_path='predicted_image.jpg'):
    try:
        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)

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

        print(f"Prediksi: {class_names[class_idx]}")
        print(f"Confidence: {confidence:.2f}%")

        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}"

result               =               classify_images(r"C:\Users\Lenovo\OneDrive\Documents\K\S
5\PMdPM\foto\test_data\Setengah Matang\kuning-test-4.jpg")
print(result)
import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt

mobileNet_model  =  load_model(r'C:\Users\Lenovo\OneDrive\Documents\K\S  5\PMdPM\Projek-UAS-
PMDPM_B_Scikit-Learn\BestModel_MobileNet_Scikit-Learn.h5')

test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'C:\Users\Lenovo\OneDrive\Documents\K\S 5\PMdPM\foto\test_data',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(180, 180)
)

y_pred = mobileNet_model.predict(test_data)
y_pred_class = tf.argmax(y_pred, axis=1)

true_labels = []
for _, labels in test_data:
    true_labels.extend(tf.argmax(labels, axis=1).numpy())
true_labels = tf.convert_to_tensor(true_labels)

conf_mat = tf.math.confusion_matrix(true_labels, y_pred_class)

accuracy = tf.reduce_sum(tf.linalg.diag_part(conf_mat)) / tf.reduce_sum(conf_mat)

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)

f1_score = 2 * (precision * recall) / (precision + recall)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
            xticklabels=["Matang",  "Mentah",  "Setengah  Matang"],  yticklabels=["Matang",
"Mentah", "Setengah 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())
```

4. GoogleNet (Oktavio)

```
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
#load data
data_dir = r"C:\Kuliah\ML\foto\foto\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)

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')
for images, labels in train_ds.take(1):
    images_array = np.array(images)
    print(images_array.shape)

from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, load_model
import matplotlib.pyplot as plt

img_size = 180
batch_size = 32


Tuner = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = Tuner)
val_ds = val_ds.cache().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(1):
    for i in range(9):
        single_image = images[i]
        augmented_image = data_augmentation(tf.expand_dims(single_image, 0))
        plt.subplot(3, 3, i + 1)
```

```
        plt.imshow(augmented_image[0].numpy().astype("uint8"))
        plt.axis("off")
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 shae dan jumlah kelas sesuai
input_shape = 180, 180, 3
```

```
n_classes = 3

#Clear Cache Keras menggunakan clear session
K.clear_session()
#buat model dengan
model = googlenet(input_shape, n_classes)
model.summary()

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])
#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()

model.save('BestModel_GoogleNet_Scikit-Learn.h5')

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\ML\Tugas6_B_12068\Tugas6_B_12068\BestModel_GoogleNet_Scikit-
Learn.h5')  # Ganti dengan path model Anda
class_names = ['Matang', 'Setengah 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'C:\Kuliah\ML\foto\foto\test_data\Mentah\hijau-test-1.jpg',
save_path='mentah1.jpg')
print(result)

import tensorflow as tf
from tensorflow.keras.models import load_model
import seaborn as sns
import matplotlib.pyplot as plt

model = load_model(r'C:\Kuliah\ML\Tugas6_B_12068\Tugas6_B_12068\gugelnet.h5')

# Muat data test yang sebenarnya
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    r'C:\Kuliah\ML\foto\foto\test_data',
    labels='inferred',
    label_mode='categorical',  # Pastikan label berupa 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,
num_classes=len(test_data.class_names))

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


# Menghitung presisi dan recall
precision = tf.math.divide_no_nan(tf.linalg.diag_part(conf_mat), tf.reduce_sum(conf_mat,
axis=0))
recall = tf.math.divide_no_nan(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=test_data.class_names, yticklabels=test_data.class_names)
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:\n", accuracy.numpy())
print("Presisi:\n", precision.numpy())
print("Recall:\n", recall.numpy())
print("F1 Score:\n", f1_score.numpy())
```

5. Streamlit Python

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

model = load_model('BestModel_VGG-16_Scikit-Learn.h5')
class_names = ['Matang', 'Setengah 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, color3):
    percentage1 = confidence[0] * 100 # Confidence for class 0 (Matang)
    percentage2 = confidence[1] * 100 # Confidence for class 1 (Setengah Matang)
    percentage3 = confidence[2] * 100 # Confidence for class 2 (Mentah)
    progress_html = f"""
    <div style="border: 1px solid #ddd; border-radius: 5px; overflow: hidden; width: 100%;
font-size: 14px;">
        <div style="width: {percentage1:.2f}%; background: {color1}; color: white; text-align:
center; height: 24px; float: left;">
            {percentage1:.2f}%
        </div>
        <div style="width: {percentage2:.2f}%; background: {color2}; color: white; text-align:
center; height: 24px; float: left;">
            {percentage2:.2f}%
        </div>
        <div style="width: {percentage3:.2f}%; background: {color3}; color: white; text-align:
center; height: 24px; float: left;">
            {percentage3:.2f}%
        </div>
    </div>
    """
```

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

# StreamLit UI
st.title("Prediksi Kematangan Tomat")

# 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
                first_color="#FF4136" # Red for "Matang"
                third_color="#58ff36" # Green for "Mentah"
                second_color="#fff236" # Yellow for "Setengah Matang"
                label_color = first_color if label == "Matang" else second_color if label ==
"Setengah Matang" else third_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, first_color, second_color, third_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)
```