

Import library

Bagan Baru

```
In [1]: from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

```
In [2]: import tensorflow as tf
import keras
import cv2
import os
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Conv2D
from tensorflow.keras.layers import Flatten, MaxPool2D, AvgPool2D
from tensorflow.keras.layers import Dropout, Concatenate, Dropout
from tensorflow.keras.callbacks import LoadModel
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
```

Load Data set

```
In [3]: data_tubes = '/content/drive/MyDrive/Colab Notebooks/Tubes alexnet/Train_Data_Tubes'
dataset = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    seed=123,
    image_size=(227,227),
    batch_size=32
)

print("Kelas yang terdeteksi:")
print(data.class_names)

class_names = data.class_names
img_size = 227
batch = 32

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)

train_count = int(total_count * 0.8)
val_count = int(total_count * 0.1)
test_count = total_count - train_count - val_count

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

Found 300 files belonging to 3 classes.

Kelas yang terdeteksi:

'Beras', 'Gandum', 'Sorgum'

Found 300 files belonging to 3 classes.

Pembagian Dataset:

Total Images (Batches): 10

Train Images (Batches): 8

Validation Images (Batches): 1

Test Images (Batches): 1

Augmentasi

```
In [4]: data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomHorizontalFlip(),
    tf.keras.layers.RandomVerticalFlip(),
    tf.keras.layers.RandomRotation(0.2),
    tf.keras.layers.RandomZoom(0.2),
    tf.keras.layers.RandomContrast(0.2)
])
```

i = 0

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

```
for images, labels in train_ds.take(i):
    for i in range(9):
        plt.imshow(images[3, i])
        print(np.array(images[1]).astype('uint8'))
        print(np.array(class_names[labels[1]]))
        plt.axis('off')

plt.tight_layout()
plt.show()
```

Output hidden; open in https://colab.research.google.com to view.

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

(32, 227, 227)

```
In [6]: from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
```

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)

img_size = 227

```
data_augmentation = Sequential([
    layers.RandomFlip("horizontal"),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
    layers.RandomContrast(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)
        plt.subplot(3, 3, i+1)
        plt.imshow(augmented_image[0].numpy().astype('uint8'))
        plt.axis('off')

plt.tight_layout()
plt.show()
```

Output hidden; open in https://colab.research.google.com to view.

AlexNet

```
In [7]: from tensorflow.keras.layers import Input, Conv2D, MaxPool2D, Flatten, Dense, Dropout
from tensorflow.keras.models import Model
import tensorflow.keras.backend as K
```

```
def alexnet(input_shape, n_classes):
    input = Input(input_shape)
```

```
    x = Conv2D(96, 11, strides=4, activation='relu', padding='valid')(input)
```

```
    x = MaxPool2D(3, 2, activation='relu', padding='same')(x)
```

```
    x = Conv2D(256, 5, activation='relu', padding='same')(x)
```

```
    x = MaxPool2D(3, 2, activation='relu', padding='same')(x)
```

```
    x = Conv2D(384, 3, activation='relu', padding='same')(x)
```

```
    x = Conv2D(384, 3, activation='relu', padding='same')(x)
```

```
    x = MaxPool2D(3, 2, activation='relu')(x)
```

```
    x = Flatten()(x)
```

```
    x = Dense(4096, activation='relu')(x)
```

```
    x = Dropout(0.5)(x)
```

```
    x = Dense(4096, activation='relu')(x)
```

```
    x = Dropout(0.5)(x)
```

```
    x = Dense(4096, activation='relu')(x)
```

```
    x = Dropout(0.5)(x)
```

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

```
    model = Model(input, output)
    return model
```

```
input_shape = (227, 227, 3)
```

n_classes = len(class_names)

K.set_session()

model = alexnet(input_shape, n_classes)

initial_learning_rate = 0.001

decay_steps = 1000

decay_rate = 0.9

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate,
    decay_steps=decay_steps,
    decay_rate=decay_rate,
    staircase=True
)
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

```
model.compile(
    optimizer=optimizer,
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

print("\nArsitektur Model: ")

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 227, 227, 3)	0
conv2d_0 (Conv2D)	(None, 96, 55, 96)	34,944
max_pooling2d_0 (MaxPooling2D)	(None, 27, 27, 96)	0
conv2d_1 (Conv2D)	(None, 27, 27, 256)	614,656
max_pooling2d_1 (MaxPooling2D)	(None, 13, 13, 256)	0
conv2d_2 (Conv2D)	(None, 13, 13, 384)	885,128
conv2d_3 (Conv2D)	(None, 13, 13, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 13, 13, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
flatten (Flatten)	(None, 9216)	0
dense (Dense)	(None, 4096)	37,752,832
dropout (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 3)	12,291

Total params: 58,933,638 (222.37 MB)

Trainable params: 58,293,635 (222.37 MB)

Non-trainable params: 0 (0.00 B)

training

```
In [8]: print("\nMulai proses training...")
```

```
model.compile(
    optimizer=Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

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

epochs = 50

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

Mulai proses training...

Epoch 1/50

59s 7s/step - accuracy: 0.3279 - loss: 490.1442 - val_accuracy: 0.2727 - val_loss: 1.1876

Epoch 2/50

51s 7s/step - accuracy: 0.3304 - loss: 1.1301 - val_accuracy: 0.6591 - val_loss: 1.0747

Epoch 3/50

76s 6s/step - accuracy: 0.4017 - loss: 1.1017 - val_accuracy: 0.5227 - val_loss: 1.0043

Epoch 4/50

47s 6s/step - accuracy: 0.4068 - loss: 1.0073 - val_accuracy: 0.4545 - val_loss: 0.9354

Epoch 5/50

78s 5s/step - accuracy: 0.3535 - loss: 1.1772 - val_accuracy: 0.5455 - val_loss: 0.10635

Epoch 6/50

45s 5s/step - accuracy: 0.4563 - loss: 1.0513 - val_accuracy: 0.3864 - val_loss: 0.9824

Epoch 7/50

45s 6s/step - accuracy: 0.5202 - loss: 0.9525 - val_accuracy: 0.7955 - val_loss: 0.7938

Epoch 8/50

44s 6s/step - accuracy: 0.6452 - loss: 0.7993 - val_accuracy: 0.7500 - val_loss: 0.63636

Epoch 9/50

83s 6s/step - accuracy: 0.6910 - loss: 0.7146 - val_accuracy: 0.6364 - val_loss: 0.5910

Epoch 10/50

81s 6s/step - accuracy: 0.6754 - loss: 0.6184 - val_accuracy: 0.7045 - val_loss: 0.7709

Epoch 11/50

83s 6s/step - accuracy: 0.6724 - loss: 0.6189 - val_accuracy: 0.6818 - val_loss: 0.6985

Epoch 12/50

83s 6s/step - accuracy: 0.6655 - loss: 0.6861 - val_accuracy: 0.7500 - val_loss: 0.4647

graph accuracy

```
In [9]: epoch_range = range(1, len(history.history['loss']) + 1)
```

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

```
plt.subplot(2, 1)
plt.plot(epoch_range, history.history['accuracy'], label='Training Accuracy')
```

```
plt.plot(epoch_range, history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.title('Training and Validation Accuracy')
```

```
plt.subplot(2, 2)
plt.plot(epoch_range, history.history['loss'], label='Training Loss')
```

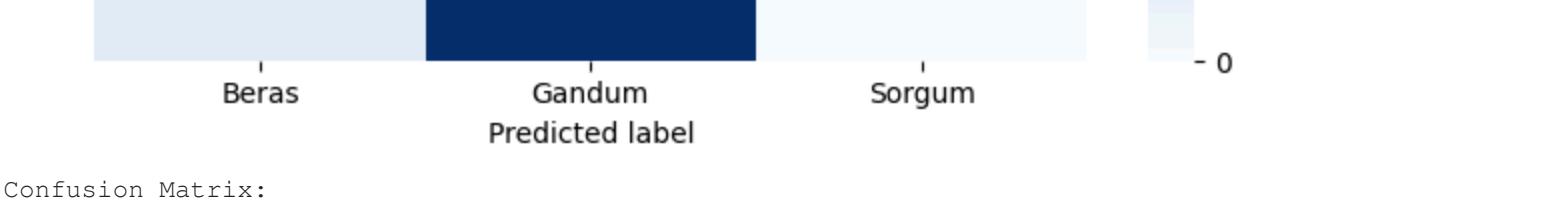
```
plt.plot(epoch_range, history.history['val_loss'], label='Validation Loss')
```

```
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
```

```
plt.show()
```

Training and Validation Accuracy

Training and Validation Loss



```
In [10]: print("\nMulai proses training...")
```

```
model.compile(
    optimizer=Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

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

epochs = 50

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

Mulai proses training...

Epoch 1/50

59s 7s/step - accuracy: 0.3279 - loss: 490.1442 - val_accuracy: 0.2727 - val_loss: 1.1876

Epoch 2/50

51s 7s/step - accuracy: 0.3304 - loss: 1.1301 - val_accuracy: 0.6591 - val_loss: 1.0747

Epoch 3/50

76s 6s/step - accuracy: 0.4017 - loss: 1.1017 - val_accuracy: 0.5227 - val_loss: 1.0043

Epoch 4/50

47s 6s/step - accuracy: 0.4068 - loss: 1.0073 - val_accuracy: 0.4545 - val_loss: 0.9354

Epoch 5/50

78s 5s/step - accuracy: 0.3535 - loss: 1.1772 - val_accuracy: 0.5455 - val_loss: 0.10635

Epoch 6/50

45s 5s/step -

Import dan Load Data

```
In [ ]: import tensorflow as tf
import cv2
import numpy as np
from matplotlib import pyplot as plt

data_dir = r"D:\Logo GK dan UAJY\Tubes vgg16\Train_Data_Tubes"

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
```

Found 300 files belonging to 3 classes.
['Beras', 'Gandum', 'Sorgum']

Load Dataset dari direktori

```
In [14]: 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,
)
```

Found 300 files belonging to 3 classes.

Datset split

```
In [15]: 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)
```

Total Images: 10

Train Images: 9

Validation Images: 1

Visualisasi Gambar

```
In [17]: 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.axis('off')
```



Melakukan augmentasi data

```
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)

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(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')
```



Buat Model VGG-16 Manual

```
In [19]: import tensorflow as tf
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPool2D, Flatten, Dense, Dropout

def vgg16(input_shape, n_classes):
    input = Input(input_shape)

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

    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = MaxPool2D((2, 2), strides=2)(x)

    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = MaxPool2D((2, 2), strides=2)(x)

    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = MaxPool2D((2, 2), strides=2)(x)

    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same')(x)
```

```

x = MaxPool2D((2, 2), strides=2)(x)

x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(n_classes, activation='softmax')(x)

model = Model(input, output)
return model

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

model = vgg16(input_shape, n_classes)

model.summary()

```

Model: "functional_4"

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 180, 180, 3)	0
conv2d_26 (Conv2D)	(None, 180, 180, 64)	1,792
conv2d_27 (Conv2D)	(None, 180, 180, 64)	36,928
max_pooling2d_10 (MaxPooling2D)	(None, 90, 90, 64)	0
conv2d_28 (Conv2D)	(None, 90, 90, 128)	73,856
conv2d_29 (Conv2D)	(None, 90, 90, 128)	147,584
max_pooling2d_11 (MaxPooling2D)	(None, 45, 45, 128)	0
conv2d_30 (Conv2D)	(None, 45, 45, 256)	295,168
conv2d_31 (Conv2D)	(None, 45, 45, 256)	590,080
conv2d_32 (Conv2D)	(None, 45, 45, 256)	590,080
max_pooling2d_12 (MaxPooling2D)	(None, 22, 22, 256)	0
conv2d_33 (Conv2D)	(None, 22, 22, 512)	1,180,160
conv2d_34 (Conv2D)	(None, 22, 22, 512)	2,359,808
conv2d_35 (Conv2D)	(None, 22, 22, 512)	2,359,808
max_pooling2d_13 (MaxPooling2D)	(None, 11, 11, 512)	0
conv2d_36 (Conv2D)	(None, 11, 11, 512)	2,359,808
conv2d_37 (Conv2D)	(None, 11, 11, 512)	2,359,808
conv2d_38 (Conv2D)	(None, 11, 11, 512)	2,359,808
max_pooling2d_14 (MaxPooling2D)	(None, 5, 5, 512)	0
flatten_2 (Flatten)	(None, 12800)	0
dense_6 (Dense)	(None, 512)	6,554,112
dropout_4 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 256)	131,328
dropout_5 (Dropout)	(None, 256)	0
dense_8 (Dense)	(None, 3)	771

Total params: 21,400,899 (81.64 MB)

Trainable params: 21,400,899 (81.64 MB)

Non-trainable params: 0 (0.00 B)

```
In [ ]: from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(),
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

early_stopping = EarlyStopping(monitor='val_accuracy',
                               patience=10,
                               mode='max')

history= model.fit(train_ds,
                    epochs=50,
                    validation_data=val_ds,
                    callbacks=[early_stopping])

Epoch 1/50
9/9 ━━━━━━━━━━ 103s 9s/step - accuracy: 0.3482 - loss: 1.0977 - val_accuracy: 0.1667 - val_loss: 1.103
6
Epoch 2/50
9/9 ━━━━━━━━ 76s 8s/step - accuracy: 0.3196 - loss: 1.0987 - val_accuracy: 0.4167 - val_loss: 1.1007
Epoch 3/50
9/9 ━━━━━━ 72s 8s/step - accuracy: 0.3145 - loss: 1.0983 - val_accuracy: 0.4167 - val_loss: 1.1008
Epoch 4/50
9/9 ━━━━ 74s 8s/step - accuracy: 0.3433 - loss: 1.0981 - val_accuracy: 0.4167 - val_loss: 1.1011
Epoch 5/50
9/9 ━━━━ 84s 9s/step - accuracy: 0.3290 - loss: 1.0989 - val_accuracy: 0.1667 - val_loss: 1.1010
Epoch 6/50
9/9 ━━━━ 87s 10s/step - accuracy: 0.3807 - loss: 1.0986 - val_accuracy: 0.4167 - val_loss: 1.100
0
Epoch 7/50
9/9 ━━━━ 93s 10s/step - accuracy: 0.3475 - loss: 1.1625 - val_accuracy: 0.1667 - val_loss: 1.100
1
Epoch 8/50
9/9 ━━━━ 90s 10s/step - accuracy: 0.2998 - loss: 1.0996 - val_accuracy: 0.1667 - val_loss: 1.100
7
Epoch 9/50
9/9 ━━━━ 86s 9s/step - accuracy: 0.3628 - loss: 1.1000 - val_accuracy: 0.1667 - val_loss: 1.1020
Epoch 10/50
9/9 ━━━━ 85s 9s/step - accuracy: 0.3102 - loss: 1.1004 - val_accuracy: 0.1667 - val_loss: 1.0999
Epoch 11/50
9/9 ━━━━ 77s 8s/step - accuracy: 0.3445 - loss: 1.0996 - val_accuracy: 0.1667 - val_loss: 1.0991
Epoch 12/50
9/9 ━━━━ 73s 8s/step - accuracy: 0.3234 - loss: 1.1003 - val_accuracy: 0.1667 - val_loss: 1.0998

Memvisualisasikan Hasil Pelatihan Data
```

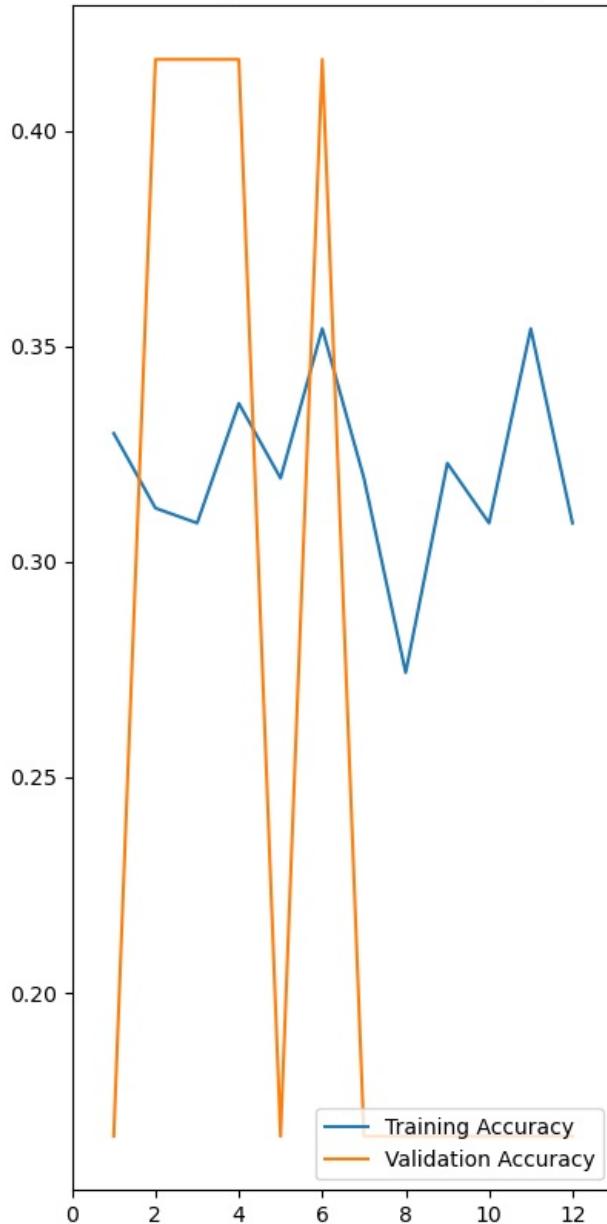
```
In [30]: ephocs_range = range(1, len(history.history['loss']) + 1)

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

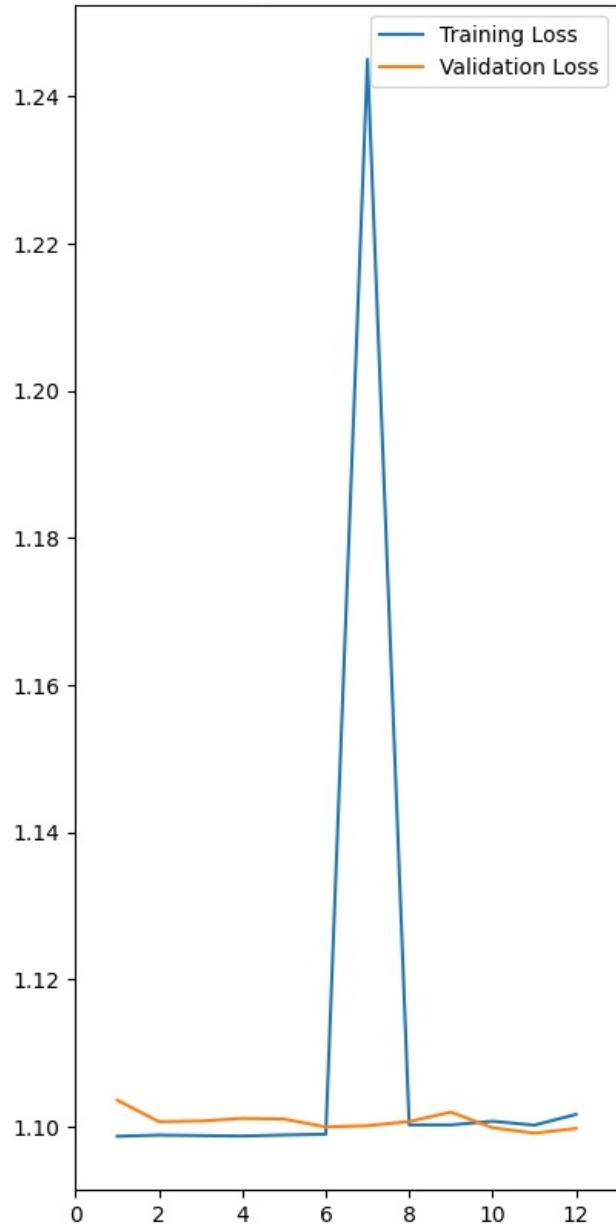
plt.subplot(1, 2, 1)
plt.plot(ephocs_range, history.history['accuracy'], label='Training Accuracy')
plt.plot(ephocs_range, history.history['val_accuracy'], label='Validation Accuracy')
plt.legend(loc='lower right')
plt.xlim(0, 13)
plt.title('Training and Validation Accuracy')

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()
```

Training and Validation Accuracy



Training and Validation Loss



Save Model

In [31]: `model.save('vgg-16.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')` .

Test Klasifikasi Gambar

```
In [32]: 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_path = r'D:\Logo GK dan UAJY\Tubes vgg16\vgg-16.h5'
model = load_model(model_path)
print("Model loaded successfully.")

class_names = ['Beras', 'Gandum', 'Sorgum']
print("Class names:", class_names)

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_array = input_image_array / 255.0
        input_image_exp_dim = tf.expand_dims(input_image_array, 0)
```

```

print(f"Input image shape: {input_image_array.shape}")
print(f"Image normalized: Min={np.min(input_image_array)}, Max={np.max(input_image_array)}")

predictions = model.predict(input_image_exp_dim)
print(f"Raw predictions: {predictions}")

class_idx = np.argmax(predictions[0])
confidence = np.max(predictions[0]) * 100

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

input_image = Image.open(image_path)
input_image.save(save_path)
print(f"Original image saved to {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}"

image_path = r'D:\Logo GK dan UAJY\Tubes vgg16\Train_Data_Tubes\Beras\Beras 2.jpg'
save_path = 'Beras_2_predicted.jpg'
result = classify_images(image_path, save_path=save_path)
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.

Model loaded successfully.

Class names: ['Beras', 'Gandum', 'Sorgum']

Input image shape: (180, 180, 3)

Image normalized: Min=0.0, Max=0.9882352948188782

1/1 2s 2s/step

Raw predictions: [[0.3329767 0.33194458 0.33507863]]

Prediksi: Sorgum

Confidence: 33.51%

Original image saved to Beras_2_predicted.jpg

Prediksi: Sorgum dengan confidence 33.51%. Gambar asli disimpan di Beras_2_predicted.jpg.

Confusion Matrix

```

In [ ]: 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'D:\Logo GK dan UAJY\Tubes vgg16\Test_Data_Tubes',
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(180, 180)
)

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)

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=(8, 6))

```

```

sns.heatmap(conf_mat.numpy(),
            annot=True,
            fmt='d',
            cmap='Blues',
            xticklabels=["Beras", "Gandum", "Sorgum"],
            yticklabels=["Beras", "Gandum", "Sorgum"])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

print("\nEvaluation Metrics:")
print("====")
print("Confusion Matrix:\n", conf_mat.numpy())
print("\nAccuracy:", accuracy.numpy())
print("\nPrecision for each class:", precision.numpy())
print("Recall for each class:", recall.numpy())
print("F1 Score for each class:", f1_score.numpy())

class_accuracies = tf.linalg.diag_part(conf_mat) / tf.reduce_sum(conf_mat, axis=1)
print("\nClass-specific accuracies:")
for i, class_name in enumerate(["Beras", "Gandum", "Sorgum"]):
    print(f"[{class_name}]: {class_accuracies[i].numpy():.4f}")

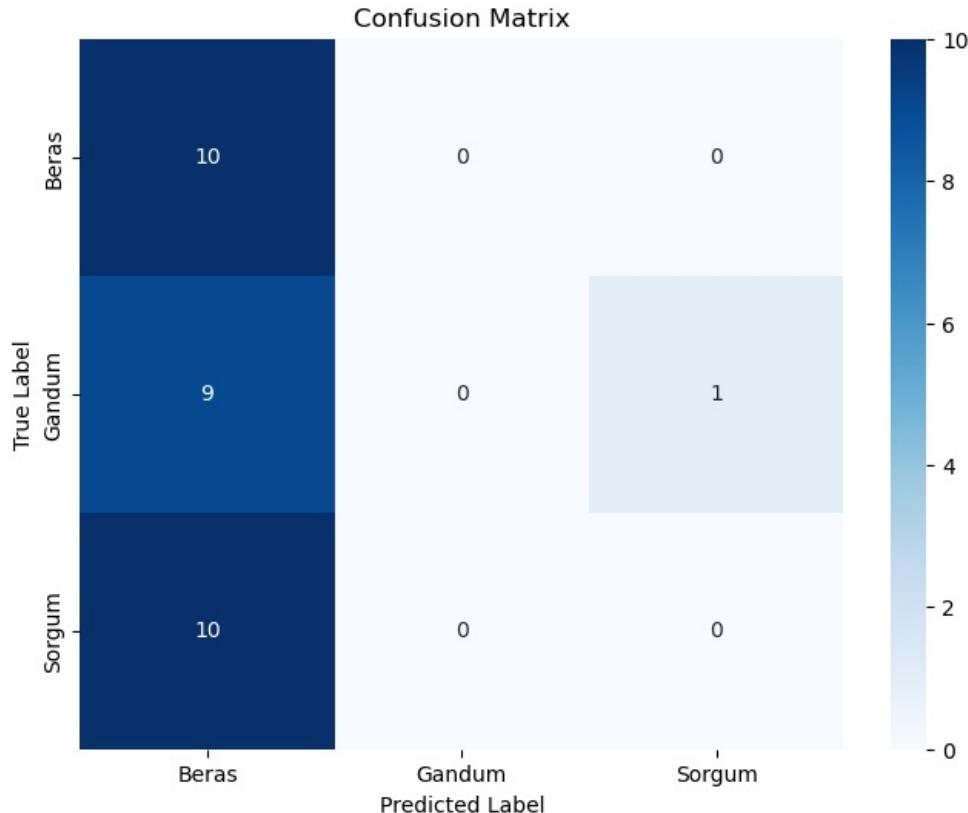
```

Found 30 files belonging to 3 classes.

WARNING:tensorflow:5 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0000022E68F12200> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

WARNING:tensorflow:5 out of the last 9 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x0000022E68F12200> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.

1/1 ————— 5s 5s/step



Evaluation Metrics:

=====

Confusion Matrix:

```
[[10  0  0]
 [ 9  0  1]
 [10  0  0]]
```

Accuracy: 0.3333333333333333

Precision for each class: [0.34482759 nan 0.]

Recall for each class: [1. 0. 0.]

F1 Score for each class: [0.51282051 nan nan]

Class-specific accuracies:

Beras: 1.0000

Gandum: 0.0000

Sorgum: 0.0000

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
In [25]: 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
```

```
In [27]: count = 0
dirs = os.listdir(r'Train_Data_Tubes')
for dir in dirs:
    files = list(os.listdir(r'Train_Data_Tubes/'+dir))
    print(dir + ' Folder has ' + str(len(files)) + ' Images')
    count = count + len(files)
print('Images Folder has ' + str(count) + ' Images')

Gandum Folder has 101 Images
Sorgum Folder has 101 Images
Beras Folder has 101 Images
Images Folder has 303 Images
```

```
In [28]: # Parameter
base_dir = r'train_Data_Tubes'
img_size = 180
batch = 32
validation_split = 0.1
```

```
In [29]: dataset = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch,
)
```

Found 310 files belonging to 3 classes.

```
In [30]: class_names = dataset.class_names
print("Class Names:", class_names)

Class Names: ['Beras', 'Gandum', 'Sorgum']
```

```
In [31]: 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)
```

Total Images: 10

Train Images: 9

Validation Images: 1

```
In [32]: train_ds = dataset.take(train_count)
val_ds = dataset.skip(train_count)
```

```
In [33]: 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')
```



```
In [34]: import numpy as np

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

(32, 180, 180, 3)

```
In [35]: AUTOTUNE = tf.data.AUTOTUNE
```

```
In [36]: train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
```

```
In [37]: val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
```

```
In [38]: data_augmentation = Sequential([
    layers.RandomFlip("horizontal", input_shape = (img_size, img_size, 3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1)
])

/opt/tljh/user/envs/dlff/lib/python3.10/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)
```

i = 0

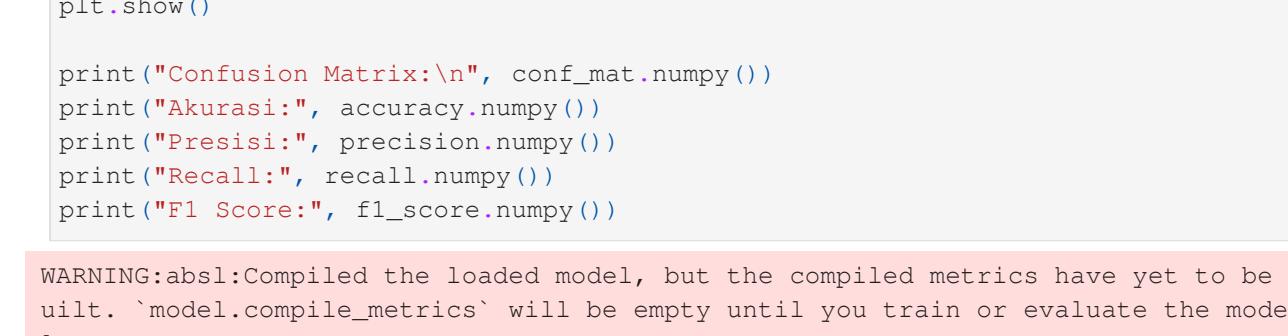
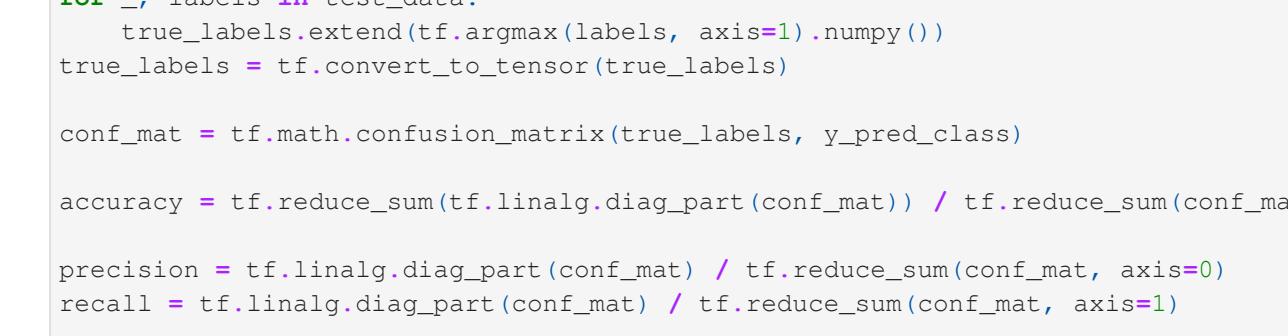
plt.figure(figsize=(10, 10))

```
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')



```
In [39]: from tensorflow.keras.optimizers import Adam

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

```
In [40]: 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, 3)	387

Total params: 3,360,451 (12.82 MB)

Trainable params: 3,069,443 (11.71 MB)

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

```
In [43]: from tensorflow.keras.callbacks import EarlyStopping

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

history = model.fit(train_ds,
                     epochs=50,
                     validation_data=val_ds,
                     callbacks=[early_stopping])
```

Epoch 1/50

9/9 8s 223ms/step - accuracy: 0.5556 - loss: 0.9623 - val_accuracy: 0.6364 - val_loss: 0.7106

Epoch 2/50

9/9 1s 127ms/step - accuracy: 0.8904 - loss: 0.3034 - val_accuracy: 0.8182 - val_loss: 0.4587

Epoch 3/50

9/9 1s 128ms/step - accuracy: 0.9603 - loss: 0.1270 - val_accuracy: 0.8636 - val_loss: 0.2982

Epoch 4/50

9/9 1s 125ms/step - accuracy: 0.9727 - loss: 0.0760 - val_accuracy: 0.9091 - val_loss: 0.2101

Epoch 5/50

9/9 1s 125ms/step - accuracy: 0.9752 - loss: 0.0795 - val_accuracy: 0.9545 - val_loss: 0.1805

Epoch 6/50

9/9 1s 126ms/step - accuracy: 0.9893 - loss: 0.0465 - val_accuracy: 0.9545 - val_loss: 0.1653

Epoch 7/50

9/9 1s 126ms/step - accuracy: 0.9894 - loss: 0.0377 - val_accuracy: 0.9545 - val_loss: 0.1091

Epoch 8/50

9/9 1s 126ms/step - accuracy: 0.9953 - loss: 0.0253 - val_accuracy: 0.9545 - val_loss: 0.0753

```
In [44]: 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()
```



```
In [45]: 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')`.

```
In [46]: 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'Model_MobileNet.h5')
class_names = ['Beras', 'Gandum', 'Sorgum']

```
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]} dengan confidence {confidence:.2f}%')
        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}%'
    except Exception as e:
        return f'Terjadi kesalahan: {e}'
```

result = classify_images(r'Train_Data_Tubes/Beras/Beras_99.jpg', save_path='Beras2.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 568ms/step

Prediksi: Beras

Confidence: 57.61%

Prediksi: Beras dengan confidence 57.61%. Gambar asli disimpan di Beras2.jpg.

```
In [47]: import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping

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

history = model.fit(train_ds,
                     epochs=50,
                     validation_data=val_ds,
                     callbacks=[early_stopping])
```

Epoch 1/50

9/9 8s 223ms/step - accuracy: 0.5556 - loss: 0.9623 - val_accuracy: 0.6364 - val_loss: 0.7106

Epoch 2/50

9/9 1s 127ms/step - accuracy: 0.8904 - loss: 0.3034 - val_accuracy: 0.8182 - val_loss: 0.4587

Epoch 3/50

9/9 1s 128ms/step - accuracy: 0.9603 - loss: 0.1270 - val_accuracy: 0.8636 - val_loss: 0.2982

Epoch 4/50

9/9 1s 125ms/step - accuracy: 0.9727 - loss: 0.0760 - val_accuracy: 0.9091 - val_loss: 0.2101

Epoch 5/50

9/9 1s 125ms/step - accuracy: 0.9752 - loss: 0.0795 - val_accuracy: 0.9545 - val_loss: 0.1805

Epoch 6/50

9/9 1s 126ms/step - accuracy: 0.9893 - loss: 0.0465 - val_accuracy: 0.9545 - val_loss: 0.1653

Epoch 7/50

9/9 1s 126ms/step - accuracy: 0.9894 - loss: 0.0377 - val_accuracy: 0.9545 - val_loss: 0.1091

Epoch 8/50

9/9 1s 126ms/step - accuracy: 0.9953 - loss: 0.0253 - val_accuracy: 0.9545 - val_loss: 0.0753

```
In [48]: 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()
```



```
In [49]: from tensorflow.keras.callbacks import EarlyStopping

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

history = model.fit(train_ds,
                     epochs=50,
                     validation_data=val_ds,
                     callbacks=[early_stopping])
```

Epoch 1/50

9/9 8s 223ms/step - accuracy: 0.5556 - loss: 0.9623 - val_accuracy: 0.6364 - val_loss: 0.7106

Epoch 2/50

9/9 1s 127ms/step - accuracy: 0.8904 - loss: 0.3034 - val_accuracy: 0.8182 - val_loss: 0.4587

Epoch 3/50

9/9 1s 128ms/step - accuracy: 0.9603 - loss: 0.1270 - val_accuracy: 0.8636 - val_loss: 0.2982

Epoch 4/50

9/9 1s 125ms/step - accuracy: 0.9727 - loss: 0.0760 - val_accuracy: 0.9091 - val_loss: 0.2101

Epoch 5/50

9/9 1s 125ms/step - accuracy: 0.9752 - loss: 0.0795 - val_accuracy: 0.9545 - val_loss: 0.1805

Epoch 6/50

9/9 1s 126ms/step - accuracy: 0.9893 - loss: 0.0465 - val_accuracy: 0.9545 - val_loss: 0.1653

Epoch 7/50

9/9 1s 126ms/step - accuracy: 0.9894 - loss: 0.0377 - val_accuracy: 0.9545 - val_loss: 0.1091

Epoch 8/50

9/9 1s 126ms/step - accuracy: 0.9953 - loss: 0.0253 - val_accuracy: 0.9545 - val_loss: 0.0753

```
In [50]: 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
```

