D:\Semester 5\Pembelajaran Mesin dan Pembelajaran Mendalam\UAS\MainStreamlit_A_Bokeh.py

```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('model_mobilenet.h5')
class_names = ['Beras Hitam', 'Beras Merah', 'Beras Putih']

def classify_image(image_path):
    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_scores = result.numpy()
        return class_names[class_idx], confidence_scores
    except Exception as e:
        return "Error", str(e)

def custom_progress_bar(confidence, class_colors):
    progress_html = "<div style='border: 1px solid #ddd; border-radius: 5px; overflow: hidden; width: 100%; font-size: 14px;'>"
    for i, (conf, color) in enumerate(zip(confidence, class_colors)):
        percentage = conf * 100
        progress_html += f"""
        <div style="width: {percentage:.2f}%; background: {color}; color: white; text-align: center; height: 24px; float: left;">
            {class_names[i]}: {percentage:.2f}%
        </div>
        """
    progress_html += "</div>"
    st.sidebar.markdown(progress_html, unsafe_allow_html=True)

st.title("Klasifikasi Jenis Beras")

uploaded_files = st.file_uploader("Unggah Gambar Beras (Beberapa diperbolehkan)", type=["jpg", "png", "jpeg"], accept_multiple_files=True)

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())

            label, confidence = classify_image(uploaded_file.name)

            if label != "Error":
                class_colors = ["#007BFF", "#FF4136", "#28A745"]  # Warna untuk masing-masing kelas

                st.sidebar.write(f"Nama File: {uploaded_file.name}")
                st.sidebar.markdown(f"<h4 style='color: #007BFF;'>Prediksi: {label}</h4>", unsafe_allow_html=True)

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

                custom_progress_bar(confidence, class_colors)

                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.")

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)
```

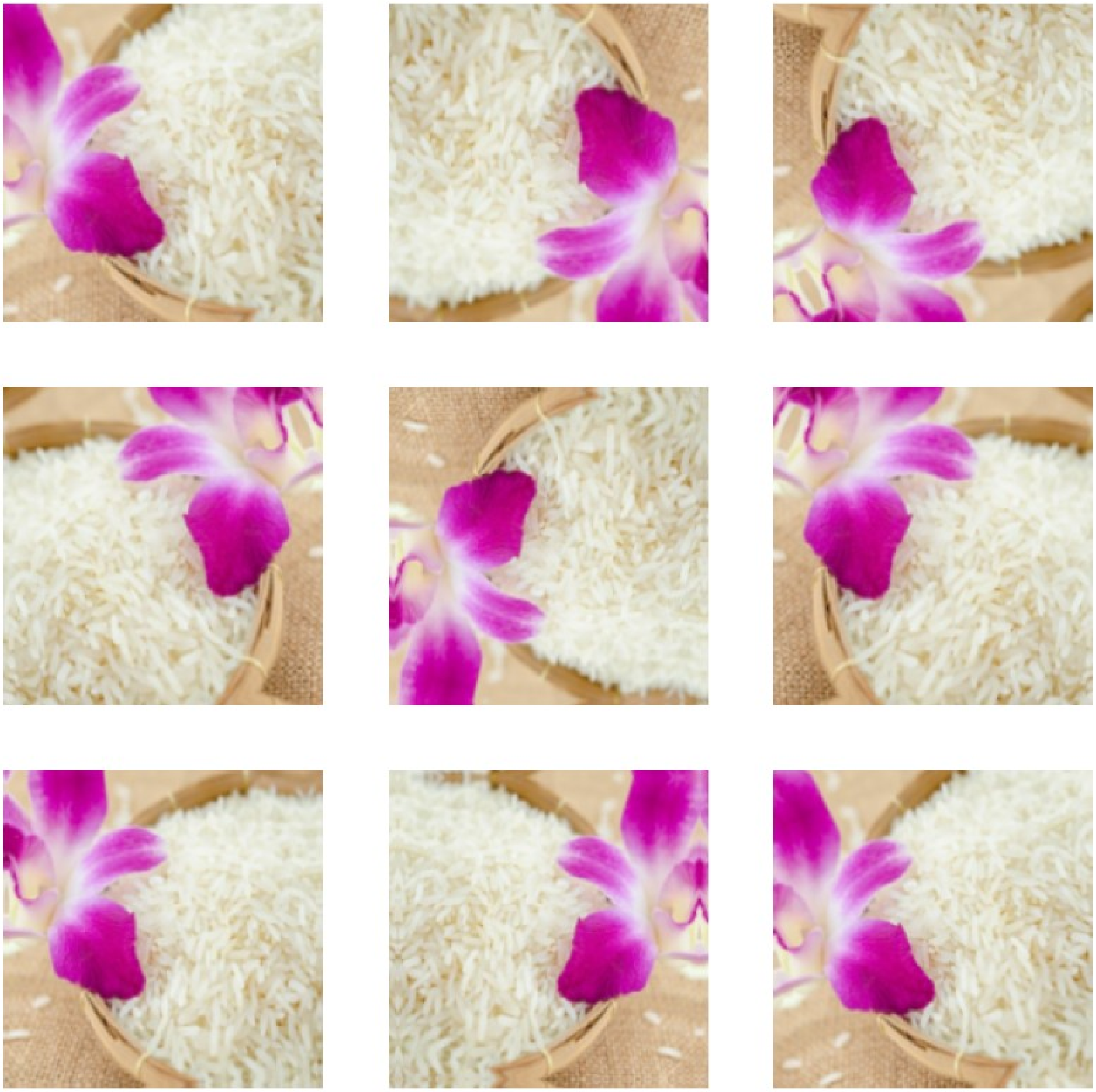
```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout
import matplotlib.pyplot as plt
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from PIL import Image
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
base_dir = '/content/drive/MyDrive/UAS PMDPM 2024/DATASET'

img_size = 180
batch = 32
validation_split = 0.1

dataset = tf.keras.utils.image_dataset_from_directory(
    base_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch,
    validation_split=validation_split,
    subset="training",
    interpolation="bilinear"
)

class_names = ['Beras Hitam', 'Beras Merah', 'Beras Putih']
print("Class Names:", class_names)
```

```
Found 269 files belonging to 3 classes.
Using 243 files for training.
Class Names: ['Beras Hitam', 'Beras Merah', 'Beras Putih']
```

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

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

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

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



```
base_model = MobileNet(include_top=False, input_shape=(img_size,
img_size, 3))
base_model.trainable = False

model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_size, img_size, 3)),
    base_model,
    layers.GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(len(class_names), activation='softmax')
])

train_ds = train_ds.map(lambda x, y: (data_augmentation(x), y))
val_ds = val_ds.map(lambda x, y: (data_augmentation(x), y))

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

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

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

```
<ipython-input-9-b70865b08e16>:1: 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))
```

```
Epoch 1/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 13s 944ms/step - accuracy: 0.4542 - loss:
1.3815
Epoch 2/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 615ms/step - accuracy: 0.4805 - loss:
1.2338
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 544ms/step - accuracy: 0.5436 - loss:
0.9984
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 789ms/step - accuracy: 0.6291 - loss:
0.8083
```

```
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 9s 560ms/step - accuracy: 0.7278 - loss:
0.7364
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 805ms/step - accuracy: 0.7564 - loss:
0.5986
Epoch 7/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 574ms/step - accuracy: 0.7983 - loss:
0.5198
Epoch 8/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 706ms/step - accuracy: 0.8442 - loss:
0.3932
Epoch 9/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 9s 570ms/step - accuracy: 0.8465 - loss:
0.4591
Epoch 10/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 753ms/step - accuracy: 0.8701 - loss:
0.3608
Epoch 11/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 9s 552ms/step - accuracy: 0.8735 - loss:
0.3172
Epoch 12/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 811ms/step - accuracy: 0.8877 - loss:
0.3225
Epoch 13/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 553ms/step - accuracy: 0.8650 - loss:
0.3495
Epoch 14/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 551ms/step - accuracy: 0.8983 - loss:
0.2874
Epoch 15/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 7s 784ms/step - accuracy: 0.9503 - loss:
0.2034
Epoch 16/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 558ms/step - accuracy: 0.9090 - loss:
0.2873
Epoch 17/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 607ms/step - accuracy: 0.9066 - loss:
0.2490
Epoch 18/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 696ms/step - accuracy: 0.9067 - loss:
0.2343
Epoch 19/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 693ms/step - accuracy: 0.9248 - loss:
0.2236
Epoch 20/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 825ms/step - accuracy: 0.9368 - loss:
0.2253
Epoch 21/30
```

```
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 564ms/step - accuracy: 0.9532 - loss:
0.1298
Epoch 22/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 556ms/step - accuracy: 0.9701 - loss:
0.1190
Epoch 23/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 808ms/step - accuracy: 0.9398 - loss:
0.1853
Epoch 24/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 561ms/step - accuracy: 0.9560 - loss:
0.1480
Epoch 25/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 639ms/step - accuracy: 0.9569 - loss:
0.1377
Epoch 26/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 685ms/step - accuracy: 0.9758 - loss:
0.1153
Epoch 27/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 556ms/step - accuracy: 0.9439 - loss:
0.1576
Epoch 28/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 709ms/step - accuracy: 0.9455 - loss:
0.1520
Epoch 29/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 9s 546ms/step - accuracy: 0.9825 - loss:
0.0939
Epoch 30/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 6s 815ms/step - accuracy: 0.9715 - loss:
0.1037
```

```python
model.save('/content/drive/MyDrive/DATASET/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')`.
```

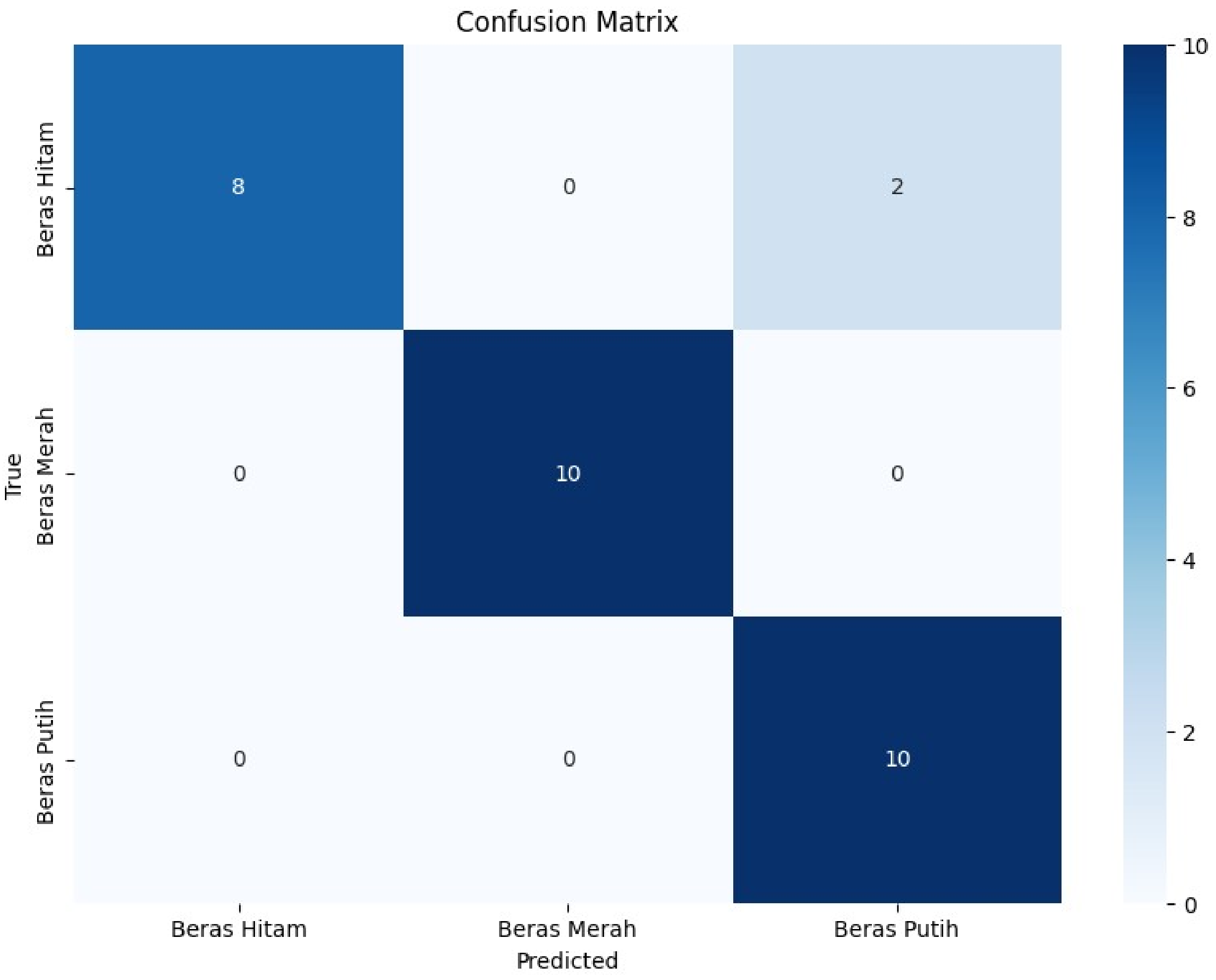
```python
def classify_images(image_path, model, class_names):
    input_image = tf.keras.utils.load_img(image_path,
target_size=(img_size, img_size))
    input_image_array = tf.keras.utils.img_to_array(input_image)
    input_image_exp_dim = tf.expand_dims(input_image_array, 0)

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

    input_image = Image.open(image_path)
    input_image.save('/content/drive/MyDrive/UAS PMDPM
2024/Test/Hitam/images (39).jpg')

    return f"Prediction: {class_names[class_idx]} with
{confidence:.2f}% confidence. Image saved."

image_path = '/content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg'
result = classify_images(image_path, model, class_names)
print(result)

image_path = '/content/drive/MyDrive/UAS PMDPM
2024/Test/Merah/Image_86.jpg'
result = classify_images(image_path, model, class_names)
print(result)

image_path = '/content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images
(32).jpg'
result = classify_images(image_path, model, class_names)
print(result)
```

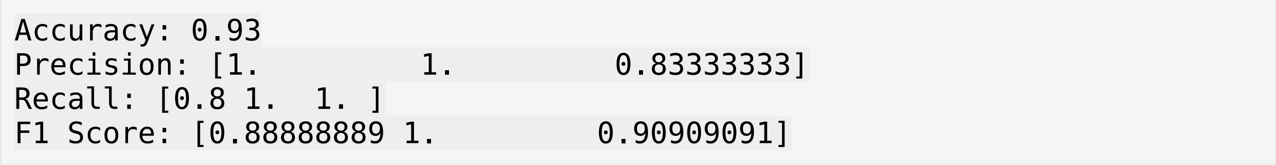
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Prediction: Beras Hitam
Confidence: 51.86%
Prediction: Beras Hitam with 51.86% confidence. Image saved.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Prediction: Beras Merah
Confidence: 50.90%
Prediction: Beras Merah with 50.90% confidence. Image saved.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Prediction: Beras Putih
Confidence: 56.28%
Prediction: Beras Putih with 56.28% confidence. Image saved.
```

```
import seaborn as sns
test_dir = '/content/drive/MyDrive/UAS PMDPM 2024/Test'
test_data = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(img_size, img_size),
    shuffle=False
)

y_pred = model.predict(test_data)
y_pred_class = np.argmax(y_pred, axis=1)
```

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

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=(10, 7))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print(f"Accuracy: {accuracy.numpy():.2f}")
print(f"Precision: {precision.numpy()}")
print(f"Recall: {recall.numpy()}")
print(f"F1 Score: {f1_score.numpy()}")
```

```
Found 30 files belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 289ms/step
```





```
Accuracy: 0.93
Precision: [1.         1.         0.83333333]
Recall: [0.8 1.  1. ]
F1 Score: [0.88888889 1.         0.90909091]
```

```
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 matplotlib.pyplot as plt
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from PIL import Image
import seaborn as sns
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
dataset_dir = '/content/drive/MyDrive/UAS PMDPM 2024/DATASET'
test_dir = '/content/drive/MyDrive/UAS PMDPM 2024/Test'

img_size = 180
batch_size = 32
validation_split = 0.1

dataset = tf.keras.utils.image_dataset_from_directory(
    dataset_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size,
    validation_split=validation_split,
    subset="training",
    interpolation="bilinear"
)

class_names = dataset.class_names
print("Nama Kelas:", class_names)
```

```
Found 269 files belonging to 3 classes.
Using 243 files for training.
Nama Kelas: ['Beras Hitam', 'Beras Merah', 'Beras Putih']
```

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

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

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

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype('uint8'))
        plt.axis('off')
plt.show()
```

```
base_model = MobileNet(include_top=False, input_shape=(img_size,
img_size, 3))
base_model.trainable = False

model = Sequential([
    layers.Rescaling(1./255, input_shape=(img_size, img_size, 3)),
    base_model,
    layers.GlobalAveragePooling2D(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(len(class_names), activation='softmax')
])
```

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


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


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


<ipython-input-20-d135df2c45a6>:1: 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))

Epoch 1/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 623ms/step - accuracy: 0.2985 - loss:
1.9274

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out
of data; interrupting training. Make sure that your dataset or
generator can generate at least `steps_per_epoch * epochs` batches.
You may need to use the `.repeat()` function when building your
dataset.
  self.gen.throw(typ, value, traceback)

 8/8 ━━━━━━━━━━━━━━━━━━━━ 17s 1s/step - accuracy: 0.3019 - loss:
1.9087
Epoch 2/30

/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/
early_stopping.py:155: UserWarning: Early stopping conditioned on
metric `val_accuracy` which is not available. Available metrics are:
accuracy,loss
  current = self.get_monitor_value(logs)

8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 584ms/step - accuracy: 0.4879 - loss:
1.1843
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 542ms/step - accuracy: 0.5341 - loss:
```

```
1.0105
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 414ms/step - accuracy: 0.6575 - loss:
0.7569
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 555ms/step - accuracy: 0.7518 - loss:
0.6252
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 480ms/step - accuracy: 0.7398 - loss:
0.5907
Epoch 7/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 403ms/step - accuracy: 0.7755 - loss:
0.4802
Epoch 8/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 395ms/step - accuracy: 0.8495 - loss:
0.4521
Epoch 9/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 560ms/step - accuracy: 0.9004 - loss:
0.3162
Epoch 10/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 487ms/step - accuracy: 0.8727 - loss:
0.3451
Epoch 11/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 403ms/step - accuracy: 0.8960 - loss:
0.2841
Epoch 12/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 410ms/step - accuracy: 0.9224 - loss:
0.2548
Epoch 13/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 476ms/step - accuracy: 0.9494 - loss:
0.2323
Epoch 14/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 405ms/step - accuracy: 0.9361 - loss:
0.2343
Epoch 15/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 534ms/step - accuracy: 0.9592 - loss:
0.2069
Epoch 16/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 519ms/step - accuracy: 0.9636 - loss:
0.1774
Epoch 17/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 500ms/step - accuracy: 0.9583 - loss:
0.1766
Epoch 18/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 414ms/step - accuracy: 0.9805 - loss:
0.1378
Epoch 19/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 384ms/step - accuracy: 0.9538 - loss:
0.1616
```

```
Epoch 20/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 486ms/step - accuracy: 0.9452 - loss:
0.1731
Epoch 21/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 536ms/step - accuracy: 0.9507 - loss:
0.1589
Epoch 22/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 391ms/step - accuracy: 0.9851 - loss:
0.0846
Epoch 23/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 390ms/step - accuracy: 0.9728 - loss:
0.0985
Epoch 24/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 646ms/step - accuracy: 0.9917 - loss:
0.0695
Epoch 25/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 669ms/step - accuracy: 0.9904 - loss:
0.0839
Epoch 26/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 384ms/step - accuracy: 0.9802 - loss:
0.0985
Epoch 27/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 557ms/step - accuracy: 0.9822 - loss:
0.0890
Epoch 28/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 648ms/step - accuracy: 0.9869 - loss:
0.0741
Epoch 29/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 391ms/step - accuracy: 0.9898 - loss:
0.0793
Epoch 30/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 381ms/step - accuracy: 0.9865 - loss:
0.0810
```

```
model.save('/content/drive/MyDrive/UAS PMDPM 2024/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')`.
```

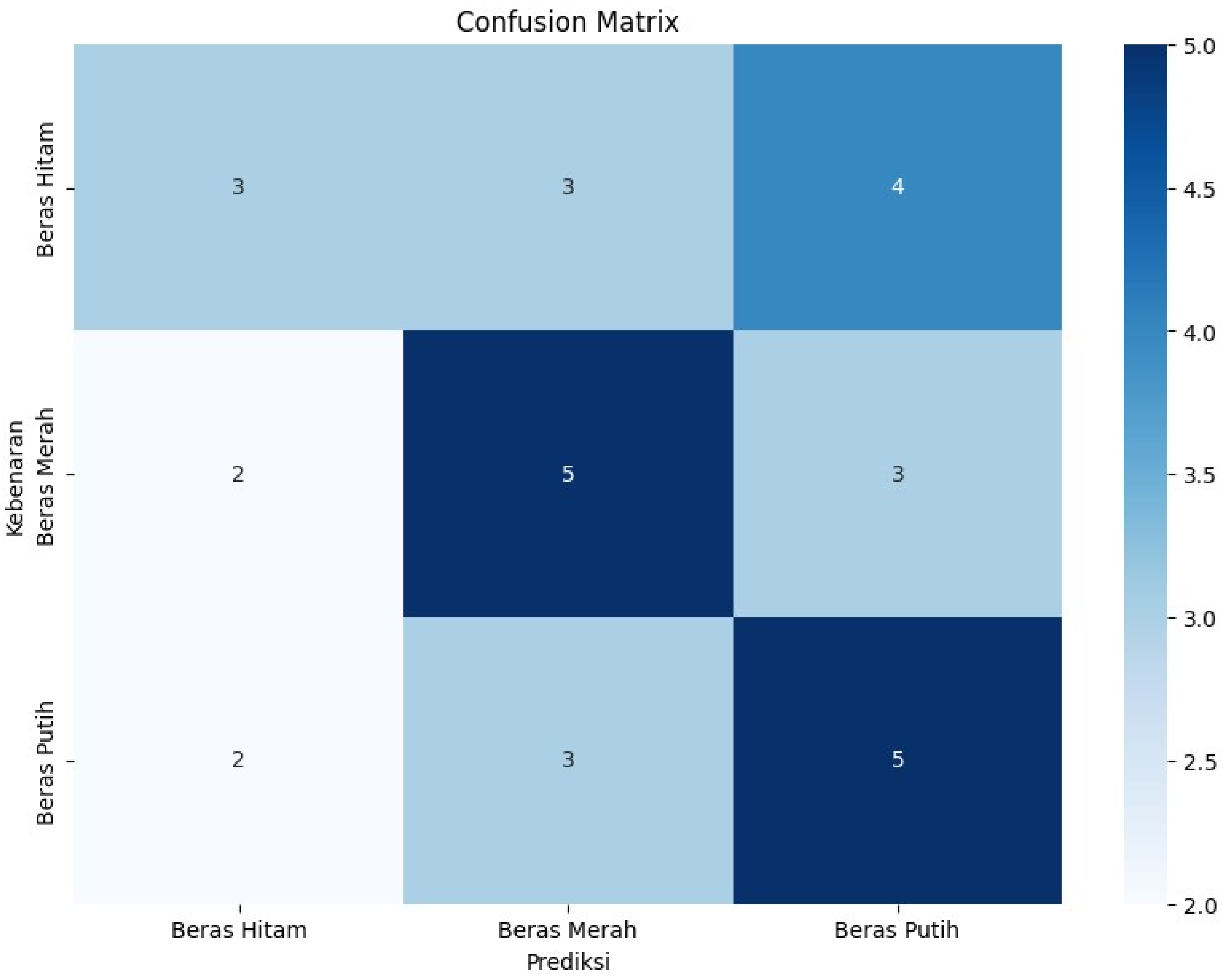
```
def classify_images(image_path, model, class_names):
    try:
        input_image = tf.keras.utils.load_img(image_path,
target_size=(img_size, img_size))
        input_image_array = tf.keras.utils.img_to_array(input_image)
        input_image_exp_dim = tf.expand_dims(input_image_array, 0)

        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"Kepercayaan: {confidence:.2f}%")

        input_image = Image.open(image_path)
        input_image.save('/content/drive/MyDrive/UAS PMDPM
2024/Test/Hitam/images (39).jpg')

        return f"Prediksi: {class_names[class_idx]} dengan kepercayaan
{confidence:.2f}%. Gambar disimpan."
    except Exception as e:
        return f"Error: {e}"


test_images = [
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Merah/Image_86.jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images (32).jpg'
]

for img_path in test_images:
    result = classify_images(img_path, model, class_names)
    print(result)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
Prediksi: Beras Hitam
Kepercayaan: 40.64%
Prediksi: Beras Hitam dengan kepercayaan 40.64%. Gambar disimpan.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Prediksi: Beras Merah
Kepercayaan: 42.80%
Prediksi: Beras Merah dengan kepercayaan 42.80%. Gambar disimpan.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
Prediksi: Beras Putih
Kepercayaan: 56.93%
Prediksi: Beras Putih dengan kepercayaan 56.93%. Gambar disimpan.
```

```
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=batch_size,
    image_size=(img_size, img_size)
)
```

```python
y_pred = model.predict(test_data)
y_pred_class = np.argmax(y_pred, axis=1)

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

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=(10, 7))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Prediksi')
plt.ylabel('Kebenaran')
plt.title('Confusion Matrix')
plt.show()
```

```
Found 30 files belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 9s 9s/step
```



Kebenaran \ Prediksi	Beras Hitam	Beras Merah	Beras Putih
Beras Hitam	3	3	4
Beras Merah	2	5	3
Beras Putih	2	3	5

```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import seaborn as sns

data_dir = '/content/drive/MyDrive/UAS PMDPM 2024/DATASET'
img_size = 180
batch_size = 32

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    image_size=(img_size, img_size),
    batch_size=batch_size,
    label_mode='int',
    validation_split=0.1,
    subset='training',
    seed=123
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,
    image_size=(img_size, img_size),
    batch_size=batch_size,
    label_mode='int',
    validation_split=0.1,
    subset='validation',
    seed=123
)

class_names = ['Beras Hitam', 'Beras Merah', 'Beras Putih']
```

```
Found 269 files belonging to 3 classes.
Using 243 files for training.
Found 269 files belonging to 3 classes.
Using 26 files for validation.
```

```
plt.figure(figsize=(10, 10))
for images, labels in dataset.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')
plt.show()

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

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

for images, labels in dataset.take(1):
    augmented_images = data_augmentation(images)

    for i in range(9):
        plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[i].numpy().astype('uint8'))
        plt.title(class_names[labels[i]])
        plt.axis('off')
plt.show()
```

Beras Hitam

Beras Putih

Beras Putih

Beras Hitam

Beras Putih

Beras Merah

Beras Putih

Beras Hitam

Beras Merah

```python
def create_vggnet_model(input_shape, n_classes):
    model = models.Sequential([
        layers.InputLayer(input_shape=input_shape),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),

        layers.Conv2D(128, 3, activation='relu', padding='same'),
        layers.Conv2D(128, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),
        layers.Conv2D(256, 3, activation='relu', padding='same'),
        layers.Conv2D(256, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),

        layers.Conv2D(512, 3, activation='relu', padding='same'),
        layers.Conv2D(512, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),

        layers.Conv2D(512, 3, activation='relu', padding='same'),
        layers.Conv2D(512, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),

        layers.Flatten(),
        layers.Dense(4096, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(4096, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(n_classes, activation='softmax')
    ])
    return model


input_shape = (img_size, img_size, 3)
n_classes = 3

model = create_vggnet_model(input_shape, n_classes)
model.summary()

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

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

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

epochs_range = range(1, len(history.history['accuracy']) + 1)
plt.figure(figsize=(12, 5))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/
input_layer.py:26: UserWarning: Argument `input_shape` is deprecated.
```

```
Use `shape` instead.
  warnings.warn(

Model: "sequential_1"

┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━
━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━
━━━━━━━━━━━━━━━━━┩
│ conv2d (Conv2D)                      │ (None, 180, 180, 64)        │
1,792 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_1 (Conv2D)                    │ (None, 180, 180, 64)        │
36,928 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ max_pooling2d (MaxPooling2D)         │ (None, 90, 90, 64)          │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_2 (Conv2D)                    │ (None, 90, 90, 128)         │
73,856 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_3 (Conv2D)                    │ (None, 90, 90, 128)         │
147,584 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ max_pooling2d_1 (MaxPooling2D)       │ (None, 45, 45, 128)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_4 (Conv2D)                    │ (None, 45, 45, 256)         │
295,168 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_5 (Conv2D)                    │ (None, 45, 45, 256)         │
590,080 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ max_pooling2d_2 (MaxPooling2D)       │ (None, 22, 22, 256)         │
0 │
├──────────────────────────────────────┼─────────────────────────────┼─
─────────────────┤
│ conv2d_6 (Conv2D)                    │ (None, 22, 22, 512)         │
1,180,160 │
├──────────────────────────────────────┼─────────────────────────────┼─
```

```
─────────────────┤
│ conv2d_7 (Conv2D)                    │ (None, 22, 22, 512)         │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ max_pooling2d_3 (MaxPooling2D)       │ (None, 11, 11, 512)         │
0 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ conv2d_8 (Conv2D)                    │ (None, 11, 11, 512)         │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ conv2d_9 (Conv2D)                    │ (None, 11, 11, 512)         │
2,359,808 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ max_pooling2d_4 (MaxPooling2D)       │ (None, 5, 5, 512)           │
0 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ flatten (Flatten)                    │ (None, 12800)               │
0 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ dense (Dense)                        │ (None, 4096)                │
52,432,896 │
├──────────────────────────────────────┼──────────────────────────────┤
─────────────────┤
│ 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: 78,631,491 (299.96 MB)

 Trainable params: 78,631,491 (299.96 MB)
 Non-trainable params: 0 (0.00 B)
```

```
Epoch 1/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 122s 11s/step - accuracy: 0.3306 - loss:
365.3763 - val_accuracy: 0.4231 - val_loss: 1.0832
Epoch 2/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 28s 403ms/step - accuracy: 0.3482 - loss:
1.2344 - val_accuracy: 0.4231 - val_loss: 1.0755
Epoch 3/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 4s 307ms/step - accuracy: 0.3420 - loss:
1.1105 - val_accuracy: 0.2692 - val_loss: 1.1023
Epoch 4/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 309ms/step - accuracy: 0.3998 - loss:
1.0905 - val_accuracy: 0.3462 - val_loss: 1.0817
Epoch 5/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 5s 291ms/step - accuracy: 0.3525 - loss:
1.0654 - val_accuracy: 0.3077 - val_loss: 1.0974
Epoch 6/30
8/8 ━━━━━━━━━━━━━━━━━━━━ 3s 318ms/step - accuracy: 0.3341 - loss:
1.8343 - val_accuracy: 0.3077 - val_loss: 1.1351
```

```
<Figure size 1200x500 with 0 Axes>
```

```
<Figure size 1200x500 with 0 Axes>
```

```python
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()
```

```python
model.save('/content/drive/MyDrive/UAS_PMDPM/vggnet_model.h5')
```

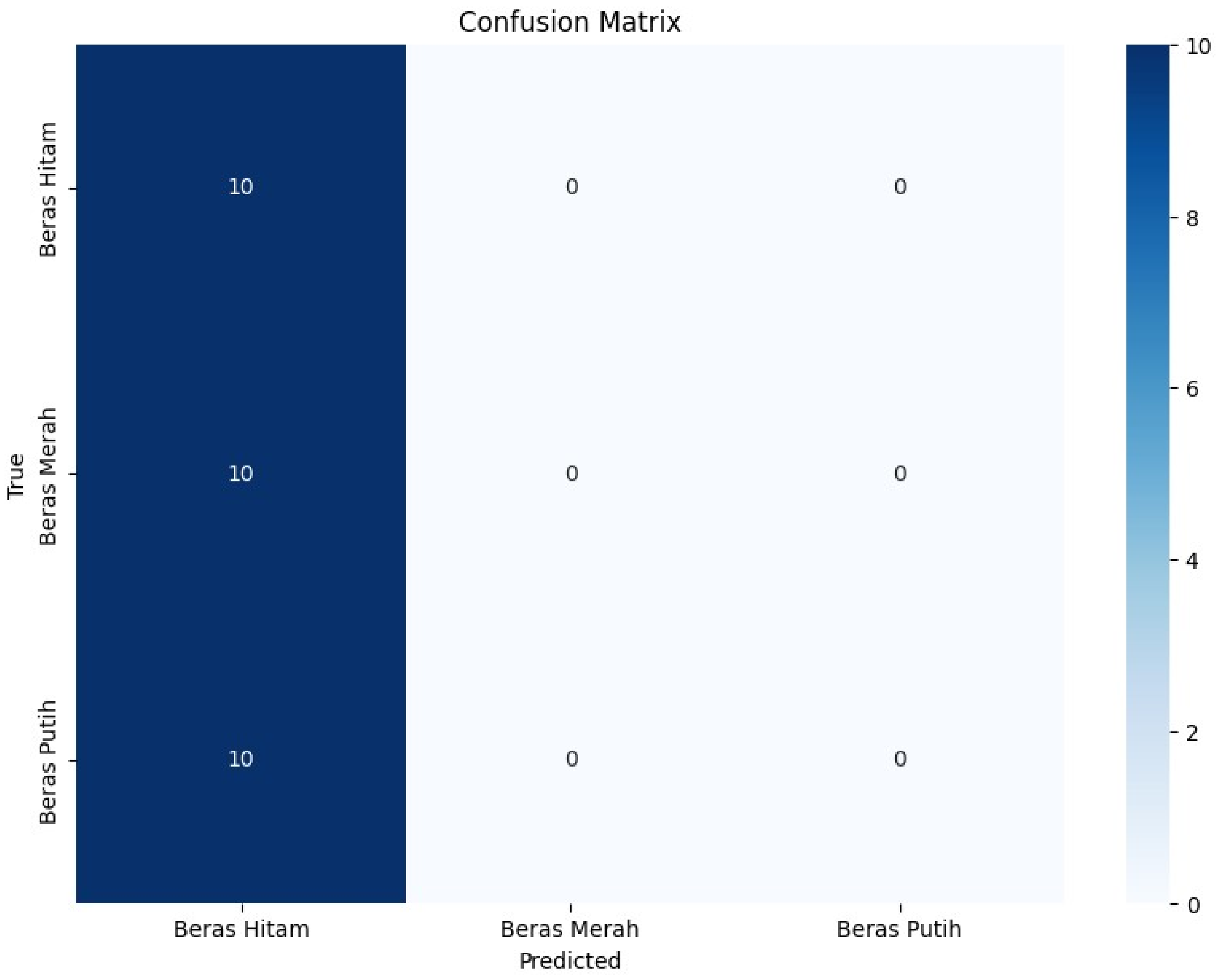
```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.
```

```python
image_paths = [
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Merah/Image_86.jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images (37).jpg'
]

def classify_images(image_path, model, class_names):
    img = tf.keras.preprocessing.image.load_img(image_path,
target_size=(img_size, img_size))
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)
    score = tf.nn.softmax(predictions[0])
    class_idx = np.argmax(score)
    class_name = class_names[class_idx]
    confidence = 100 * np.max(score)
    return f"Prediction: {class_name} with {confidence:.2f}%
confidence."

for path in image_paths:
    result = classify_images(path, model, class_names)
    print(f"Image Path: {path}")
    print(result)
    print()
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Image Path: /content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg
Prediction: Beras Hitam with 37.03% confidence.

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
Image Path: /content/drive/MyDrive/UAS PMDPM
2024/Test/Merah/Image_86.jpg
Prediction: Beras Hitam with 37.08% confidence.

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 25ms/step
Image Path: /content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images
(37).jpg
Prediction: Beras Hitam with 36.66% confidence.
```

```
import seaborn as sns
test_dir = '/content/drive/MyDrive/UAS PMDPM 2024/Test'
test_data = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(img_size, img_size),
    shuffle=False
)

y_pred = model.predict(test_data)
y_pred_class = np.argmax(y_pred, axis=1)

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

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=(10, 7))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

print(f"Accuracy: {accuracy.numpy():.2f}")
print(f"Precision: {precision.numpy()}")
print(f"Recall: {recall.numpy()}")
print(f"F1 Score: {f1_score.numpy()}")
```

```
Found 30 files belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 21s 21s/step
```



```
Accuracy: 0.33
Precision: [0.33333333        nan        nan]
Recall: [1. 0. 0.]
F1 Score: [0.5 nan nan]
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
import os
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import load_img,
img_to_array
from tensorflow.keras.models import load_model

img_size = 224
class_names = ['Beras Hitam', 'Beras Merah', 'Beras Putih']
dataset_dir = '/content/drive/MyDrive/UAS PMDPM 2024/DATASET'

def preprocess_image(image_path):
    img = load_img(image_path, target_size=(img_size, img_size))
    img_array = img_to_array(img) / 255.0
    return np.expand_dims(img_array, axis=0)

def create_model():
    model = models.Sequential()
    model.add(layers.Conv2D(96, (11, 11), strides=(4, 4),
activation='relu', input_shape=(img_size, img_size, 3)))
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
    model.add(layers.Conv2D(256, (5, 5), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
    model.add(layers.Conv2D(384, (3, 3), activation='relu'))
    model.add(layers.Conv2D(384, (3, 3), activation='relu'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(3, activation='softmax'))
    model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model

def load_dataset():
    train_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
validation_split=0.2)
    train_generator = train_datagen.flow_from_directory(
        dataset_dir,
        target_size=(img_size, img_size),
        batch_size=32,
        class_mode='sparse',
        subset='training'
    )
    validation_generator = train_datagen.flow_from_directory(
        dataset_dir,
        target_size=(img_size, img_size),
        batch_size=32,
        class_mode='sparse',
        subset='validation'
    )
    return train_generator, validation_generator

def train_model(model, train_generator, validation_generator):
    model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples //
train_generator.batch_size,
        epochs=10,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples //
validation_generator.batch_size
    )
    model.save('/content/drive/MyDrive/UAS PMDPM
2024/alexnet_model.h5')

def classify_images(image_path, model, class_names):
    img = preprocess_image(image_path)
    predictions = model.predict(img)
    predicted_class = class_names[np.argmax(predictions)]
    return predicted_class

train_generator, validation_generator = load_dataset()
model = create_model()
train_model(model, train_generator, validation_generator)

model = load_model('/content/drive/MyDrive/UAS PMDPM
2024/alexnet_model.h5')

image_paths = [
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Merah/Image_86.jpg',
    '/content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images (37).jpg'
]

for image_path in image_paths:
    result = classify_images(image_path, model, class_names)
    print(f"Prediction for {image_path}: {result}")
```

```
Found 216 images belonging to 3 classes.
Found 53 images belonging to 3 classes.
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/
convolutional/base_conv.py:107: 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__(activity_regularizer=activity_regularizer,
**kwargs)

Epoch 1/10

/usr/local/lib/python3.10/dist-packages/keras/src/trainers/
data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset`
class should call `super().__init__(**kwargs)` in its constructor.
`**kwargs` can include `workers`, `use_multiprocessing`,
`max_queue_size`. Do not pass these arguments to `fit()`, as they will
be ignored.
  self._warn_if_super_not_called()

6/6 ━━━━━━━━━━━━━━━━━━━━ 87s 5s/step - accuracy: 0.3187 - loss: 1.2181
- val_accuracy: 0.4062 - val_loss: 1.0864
Epoch 2/10
1/6 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.2188 - loss:
1.1028

/usr/lib/python3.10/contextlib.py:153: UserWarning: Your input ran out
of data; interrupting training. Make sure that your dataset or
generator can generate at least `steps_per_epoch * epochs` batches.
You may need to use the `.repeat()` function when building your
dataset.
  self.gen.throw(typ, value, traceback)

 6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 257ms/step - accuracy: 0.2188 - loss:
1.1028 - val_accuracy: 0.2857 - val_loss: 1.0965
Epoch 3/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 58s 44ms/step - accuracy: 0.4441 - loss:
1.0905
Epoch 4/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 1s 102ms/step - accuracy: 0.3750 - loss:
1.1584 - val_accuracy: 0.3438 - val_loss: 1.0921
Epoch 5/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 50ms/step - accuracy: 0.4020 - loss:
1.0835 - val_accuracy: 0.3333 - val_loss: 0.9750
Epoch 6/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 5s 1s/step - accuracy: 0.4688 - loss: 0.9137

Epoch 7/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 3s 139ms/step - accuracy: 0.5522 - loss:
0.8488 - val_accuracy: 0.6562 - val_loss: 0.6842
Epoch 8/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 51ms/step - accuracy: 0.6250 - loss:
0.7116 - val_accuracy: 0.6190 - val_loss: 1.7610
Epoch 9/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 2s 41ms/step - accuracy: 0.5972 - loss:
```

```
0.8001
Epoch 10/10
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 0.6250 - loss:
0.7447 - val_accuracy: 0.6250 - val_loss: 1.0086

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')`.
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 1s/step
Prediction for /content/drive/MyDrive/UAS PMDPM 2024/Test/Hitam/images
(42).jpg: Beras Putih
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Prediction for /content/drive/MyDrive/UAS PMDPM
2024/Test/Merah/Image_86.jpg: Beras Merah
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step
Prediction for /content/drive/MyDrive/UAS PMDPM 2024/Test/Putih/images
(37).jpg: Beras Putih
```