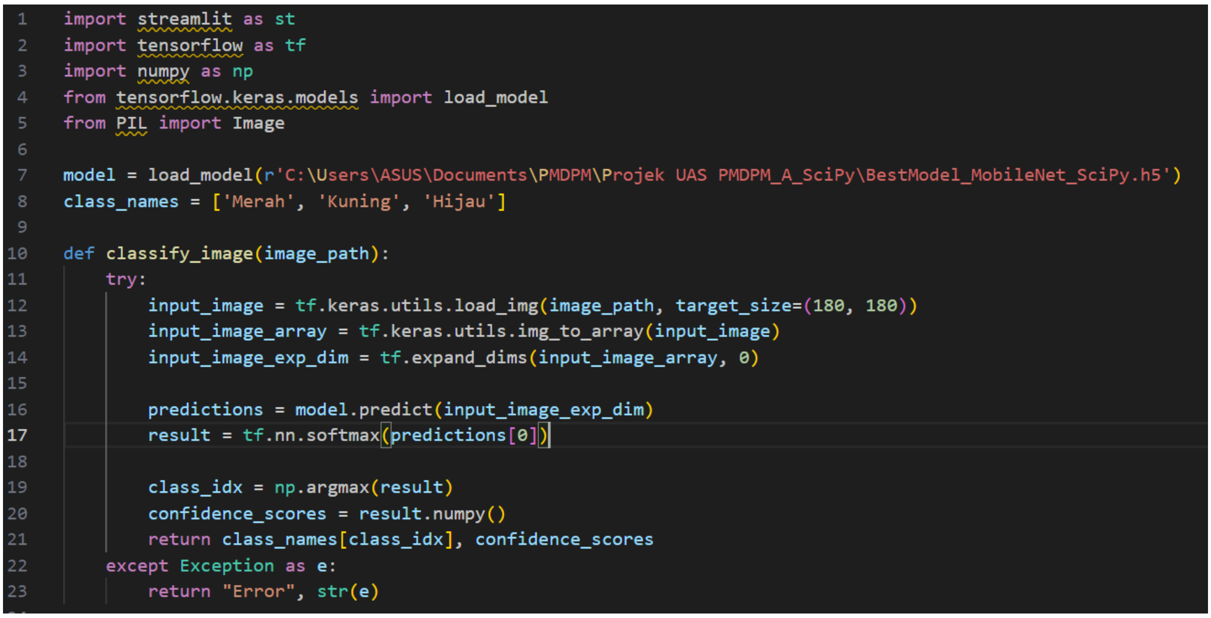
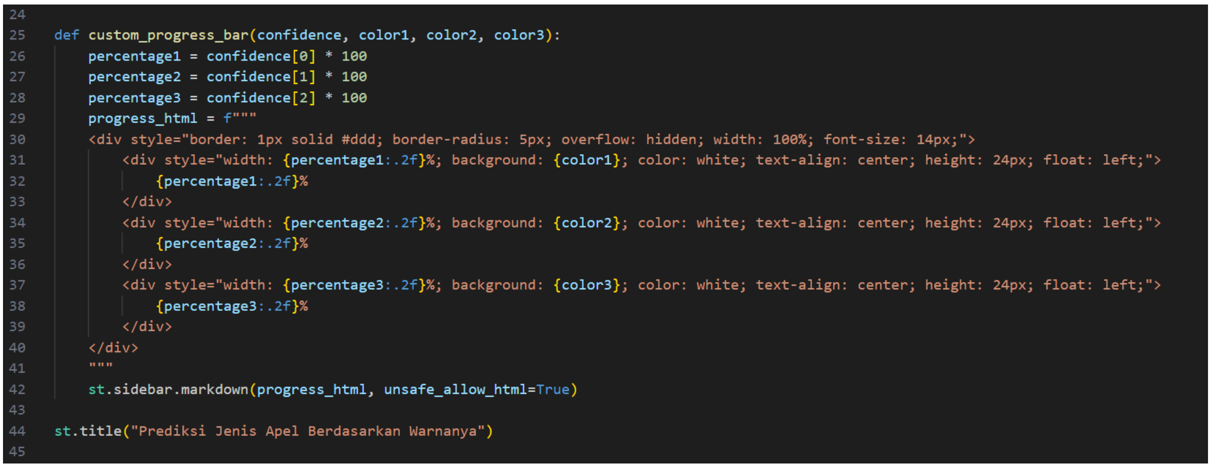
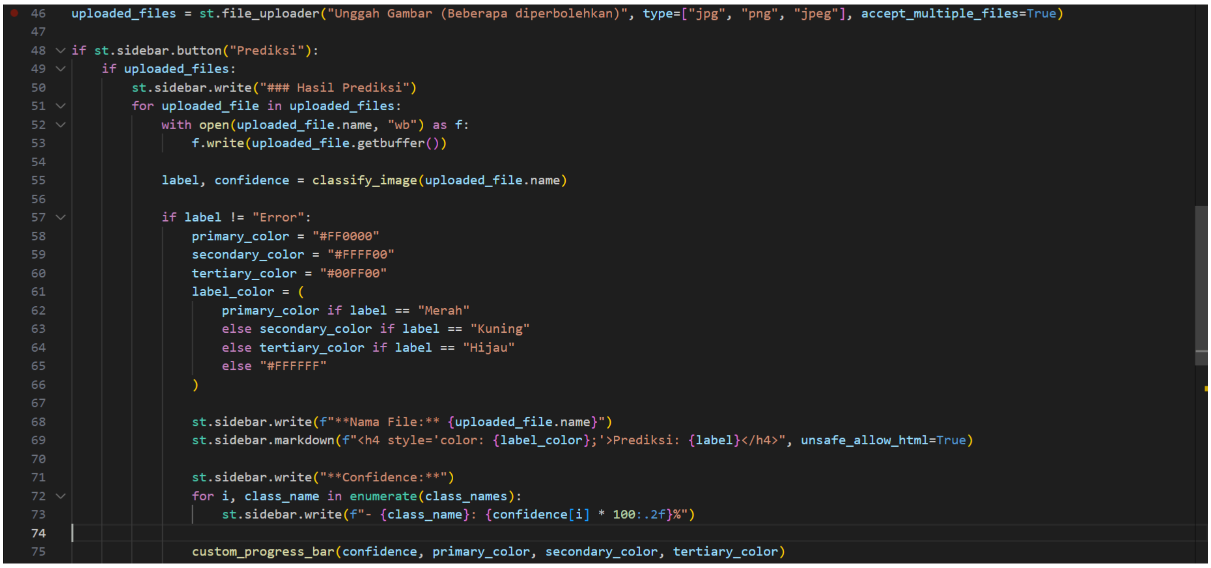
Streamlit



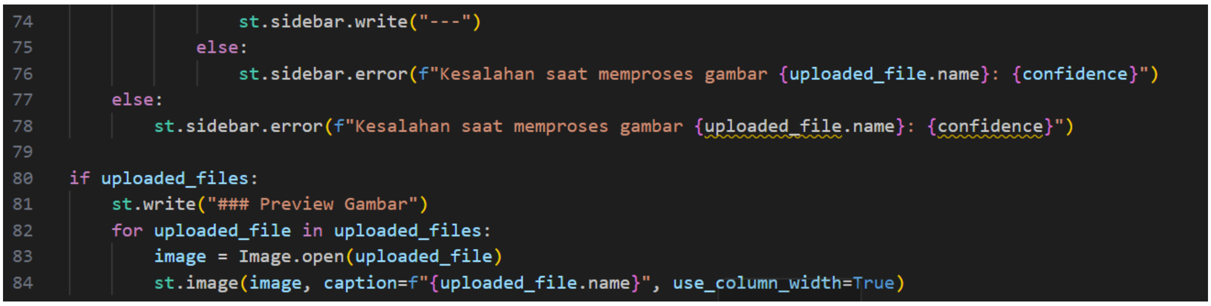
```python
import streamlit as st
import tensorflow as tf
import numpy as np
from tensorflow.keras.models import load_model
from PIL import Image

model = load_model(r'C:\Users\ASUS\Documents\PMDPM\Projek UAS PMDPM_A_SciPy\BestModel_MobileNet_SciPy.h5')
class_names = ['Merah', 'Kuning', 'Hijau']

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)
```



```python

def custom_progress_bar(confidence, color1, color2, color3):
    percentage1 = confidence[0] * 100
    percentage2 = confidence[1] * 100
    percentage3 = confidence[2] * 100
    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)

st.title("Prediksi Jenis Apel Berdasarkan Warnanya")

```

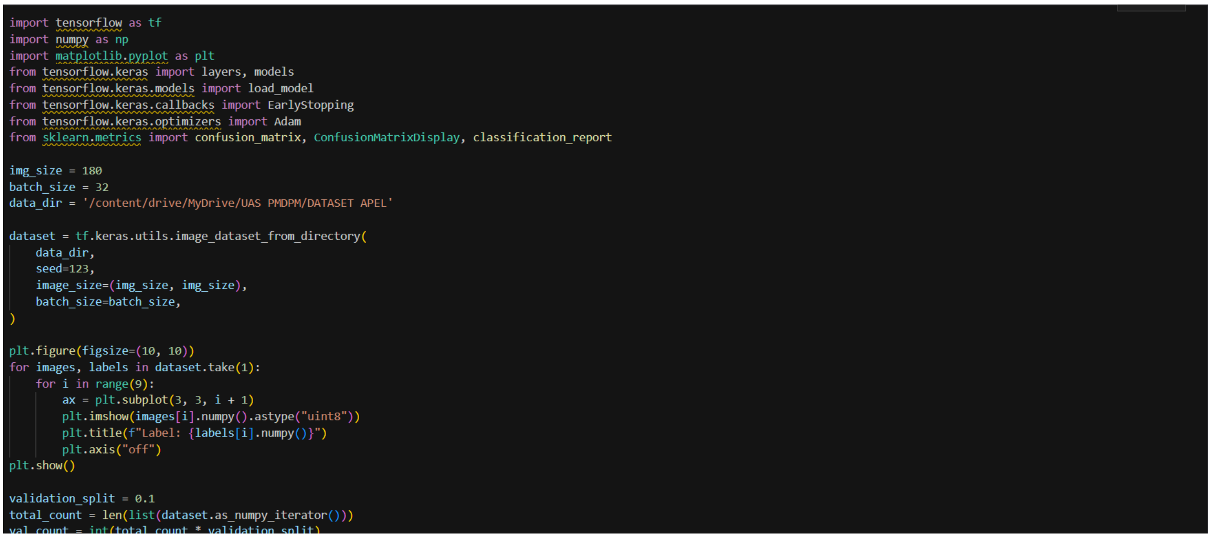


```python
uploaded_files = st.file_uploader("Unggah Gambar (Beberapa diperbolehkan)", type=["jpg", "png", "jpeg"], accept_multiple_files=True)

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":
                primary_color = "#FF0000"
                secondary_color = "#FFFF00"
                tertiary_color = "#00FF00"
                label_color = (
                    primary_color if label == "Merah"
                    else secondary_color if label == "Kuning"
                    else tertiary_color if label == "Hijau"
                    else "#FFFFFF"
                )

                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)

                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, primary_color, secondary_color, tertiary_color)
```

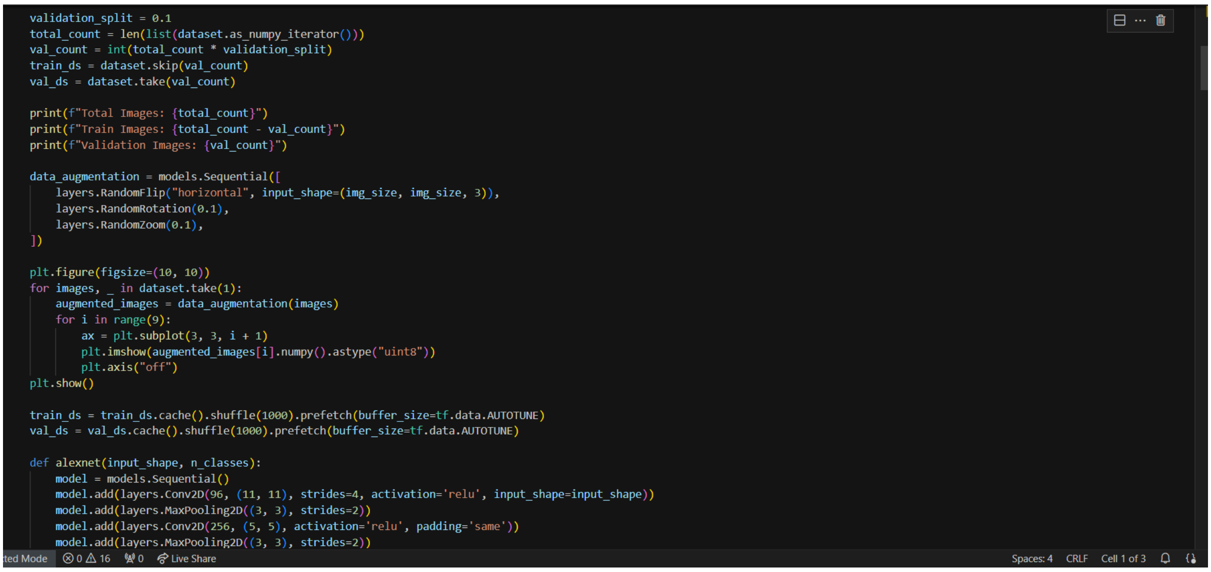


```
                st.sidebar.write("---")
            else:
                st.sidebar.error(f"Kesalahan saat memproses gambar {uploaded_file.name}: {confidence}")
    else:
        st.sidebar.error(f"Kesalahan saat memproses gambar {uploaded_file.name}: {confidence}")

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)
```

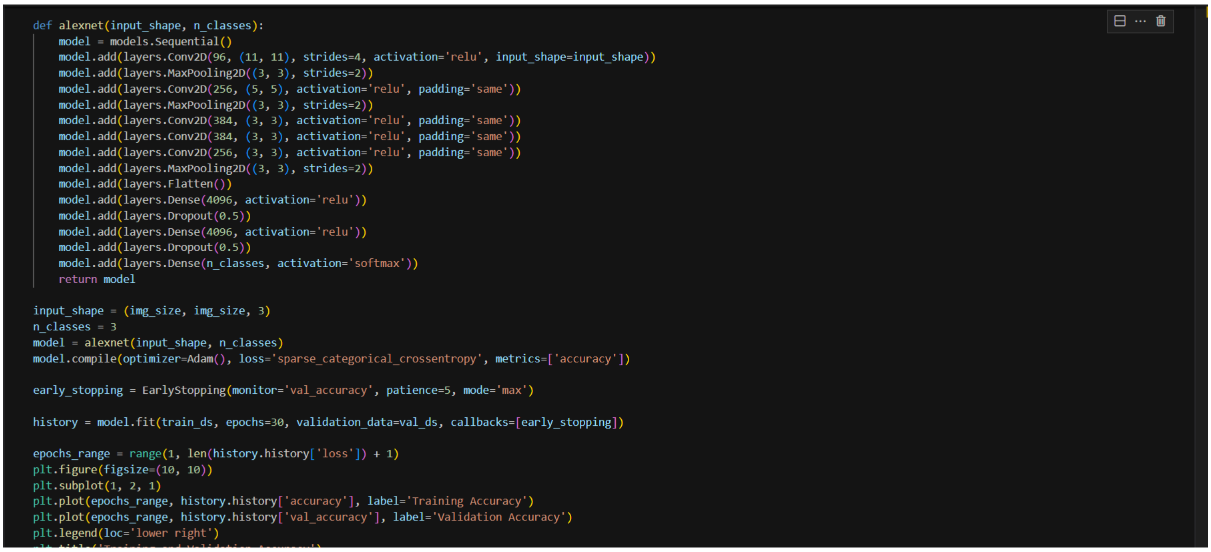
Notebook_AlexNet_A_SciPy_Nathan_Juan.ipynb



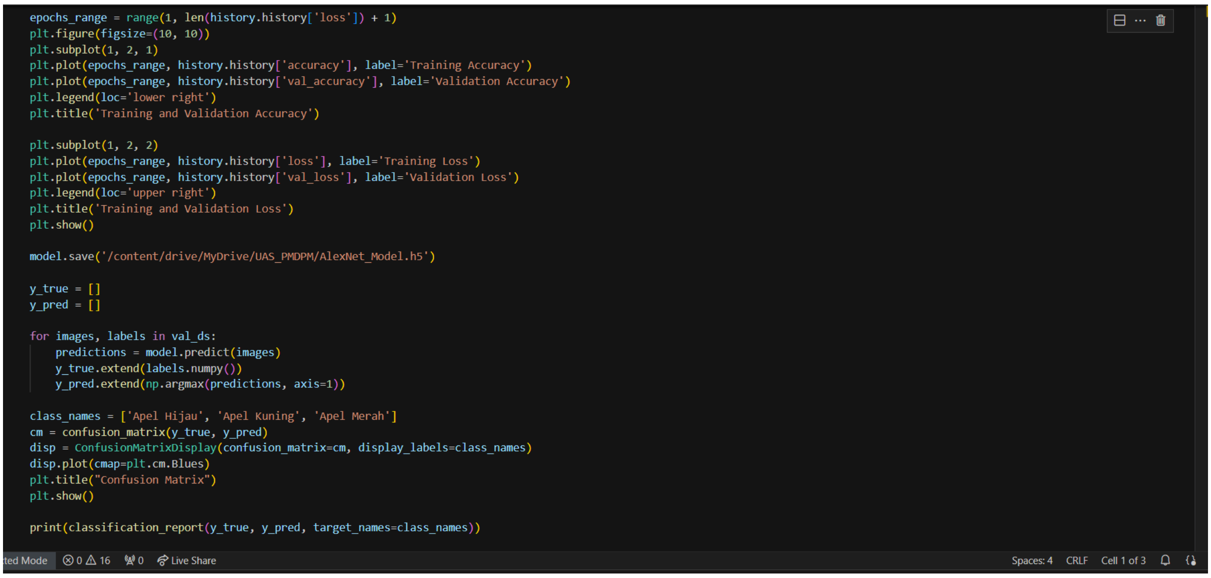
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_report

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

dataset = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size,
)

plt.figure(figsize=(10, 10))
for images, labels in dataset.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(f"Label: {labels[i].numpy()}")
        plt.axis("off")
plt.show()

validation_split = 0.1
total_count = len(list(dataset.as_numpy_iterator()))
val_count = int(total_count * validation_split)
```



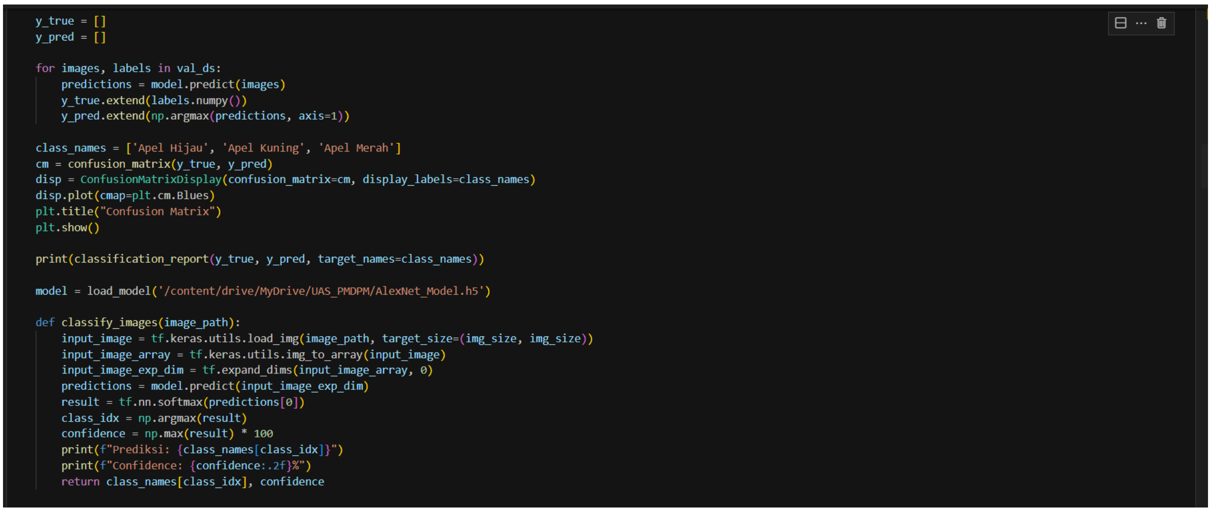
```
validation_split = 0.1
total_count = len(list(dataset.as_numpy_iterator()))
val_count = int(total_count * validation_split)
train_ds = dataset.skip(val_count)
val_ds = dataset.take(val_count)

print(f"Total Images: {total_count}")
print(f"Train Images: {total_count - val_count}")
print(f"Validation Images: {val_count}")

data_augmentation = models.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, _ in dataset.take(1):
    augmented_images = data_augmentation(images)
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[i].numpy().astype("uint8"))
        plt.axis("off")
plt.show()

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

def alexnet(input_shape, n_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(96, (11, 11), strides=4, activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((3, 3), strides=2))
    model.add(layers.Conv2D(256, (5, 5), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((3, 3), strides=2))
```



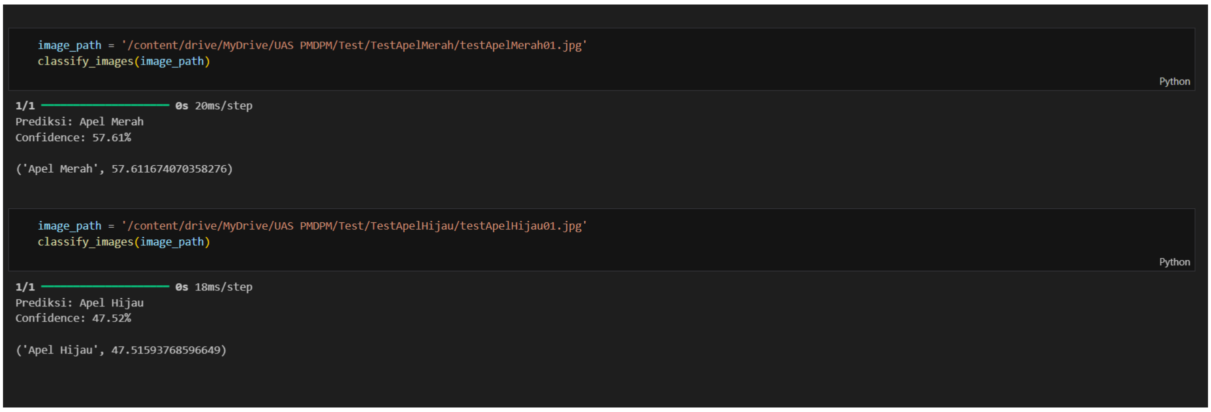
```python
def alexnet(input_shape, n_classes):
    model = models.Sequential()
    model.add(layers.Conv2D(96, (11, 11), strides=4, activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((3, 3), strides=2))
    model.add(layers.Conv2D(256, (5, 5), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((3, 3), strides=2))
    model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(384, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((3, 3), strides=2))
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(n_classes, activation='softmax'))
    return model

input_shape = (img_size, img_size, 3)
n_classes = 3
model = alexnet(input_shape, n_classes)
model.compile(optimizer=Adam(), 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')
```



```python
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('/content/drive/MyDrive/UAS_PMDPM/AlexNet_Model.h5')

y_true = []
y_pred = []

for images, labels in val_ds:
    predictions = model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(predictions, axis=1))

class_names = ['Apel Hijau', 'Apel Kuning', 'Apel Merah']
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

print(classification_report(y_true, y_pred, target_names=class_names))
```



```python
y_true = []
y_pred = []

for images, labels in val_ds:
    predictions = model.predict(images)
    y_true.extend(labels.numpy())
    y_pred.extend(np.argmax(predictions, axis=1))

class_names = ['Apel Hijau', 'Apel Kuning', 'Apel Merah']
cm = confusion_matrix(y_true, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_names)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

print(classification_report(y_true, y_pred, target_names=class_names))

model = load_model('/content/drive/MyDrive/UAS_PMDPM/AlexNet_Model.h5')

def classify_images(image_path):
    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"Confidence: {confidence:.2f}%")
    return class_names[class_idx], confidence
```



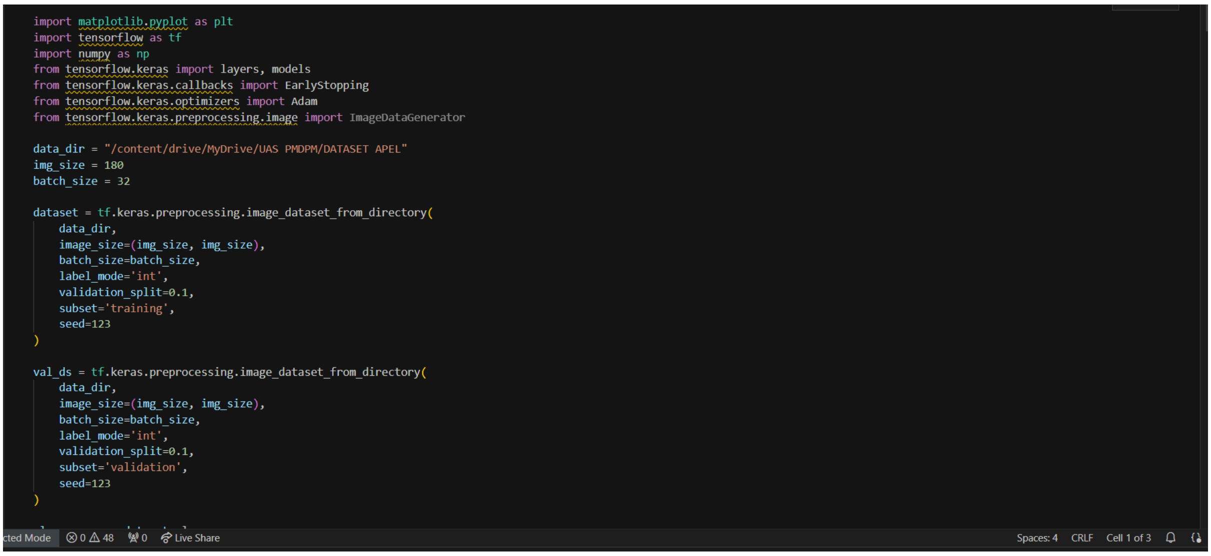
```
image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelMerah/testApelMerah01.jpg'
classify_images(image_path)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
Prediksi: Apel Merah
Confidence: 57.61%

('Apel Merah', 57.611674070358276)
```

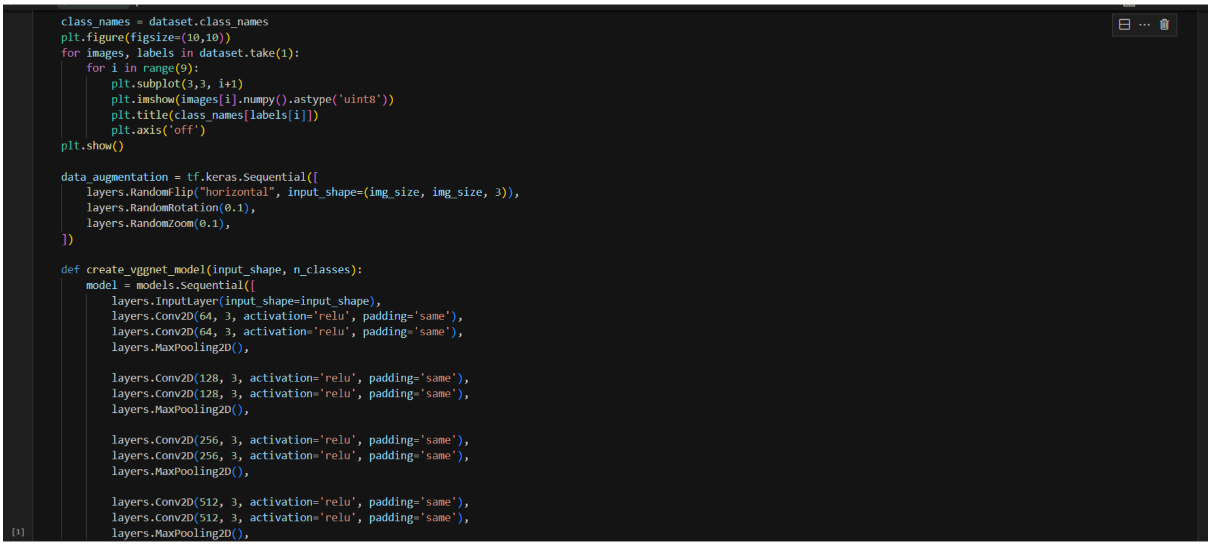
```
image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelHijau/testApelHijau01.jpg'
classify_images(image_path)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Prediksi: Apel Hijau
Confidence: 47.52%

('Apel Hijau', 47.51593768596649)
```

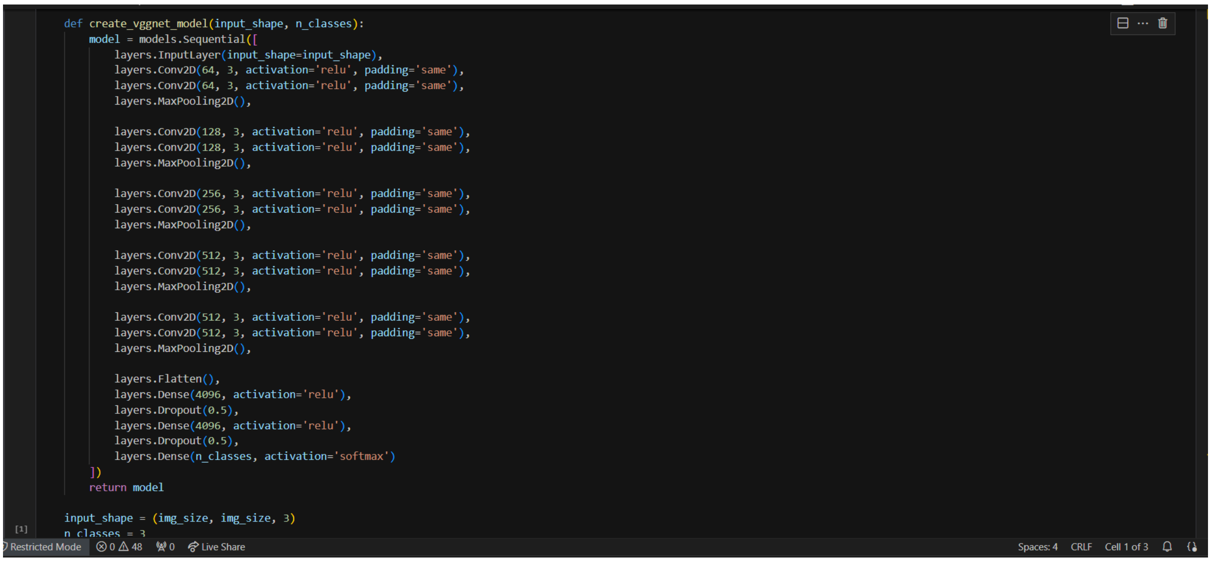
Notebook_VGG-16_A_SciPy_Dito.ipynb



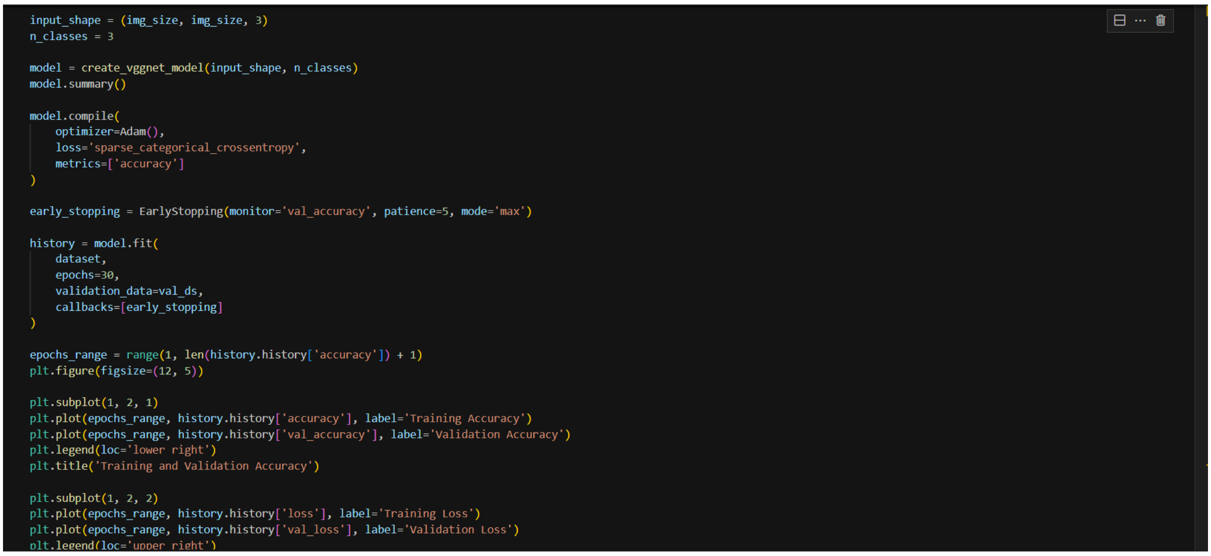
```
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

data_dir = "/content/drive/MyDrive/UAS PMDPM/DATASET APEL"
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
)
```



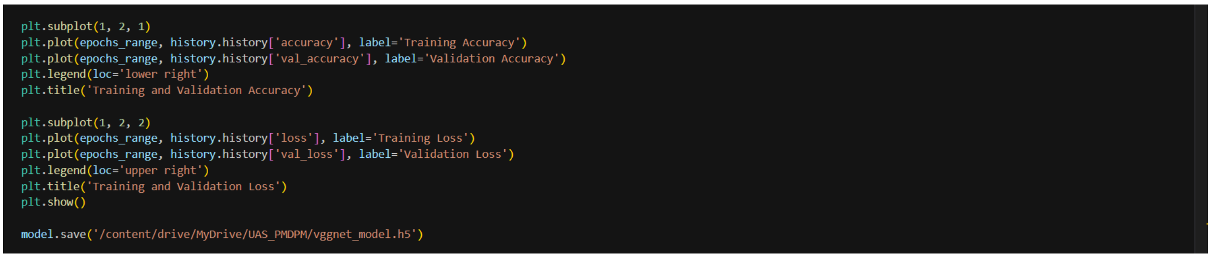
```
class_names = dataset.class_names
plt.figure(figsize=(10,10))
for images, labels in dataset.take(1):
    for i in range(9):
        plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        plt.title(class_names[labels[i]])
        plt.axis('off')
plt.show()

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

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(),
```



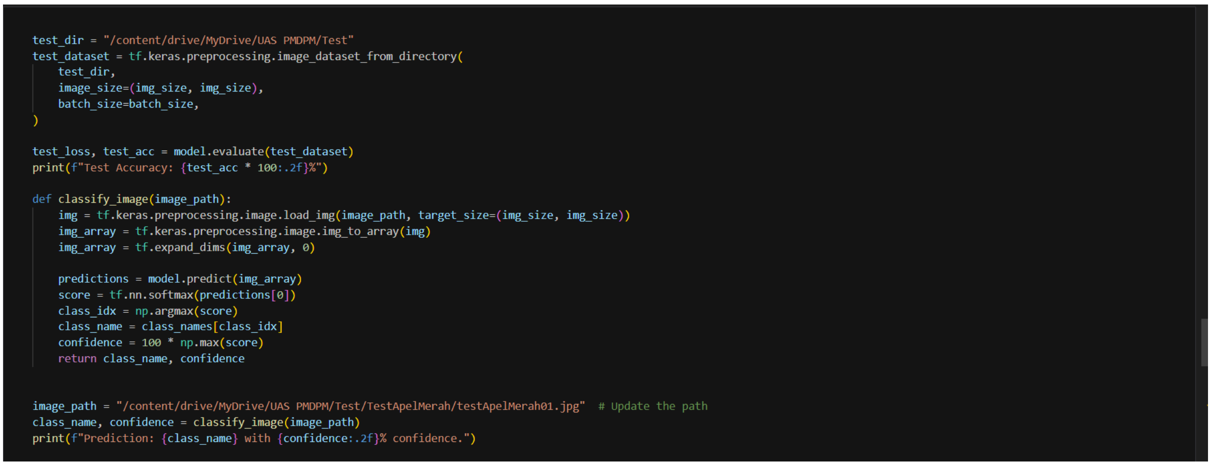
```
def create_vggnet_model(input_shape, n_classes):
    model = models.Sequential([
        layers.InputLayer(input_shape=input_shape),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.Conv2D(64, 3, activation='relu', padding='same'),
        layers.MaxPooling2D(),

        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
```



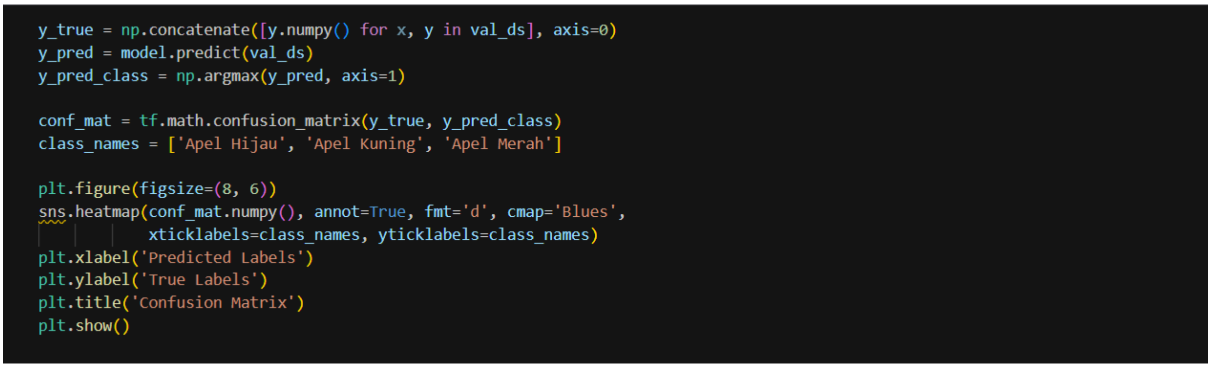
```
input_shape = (img_size, img_size, 3)
n_classes = 3

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))

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

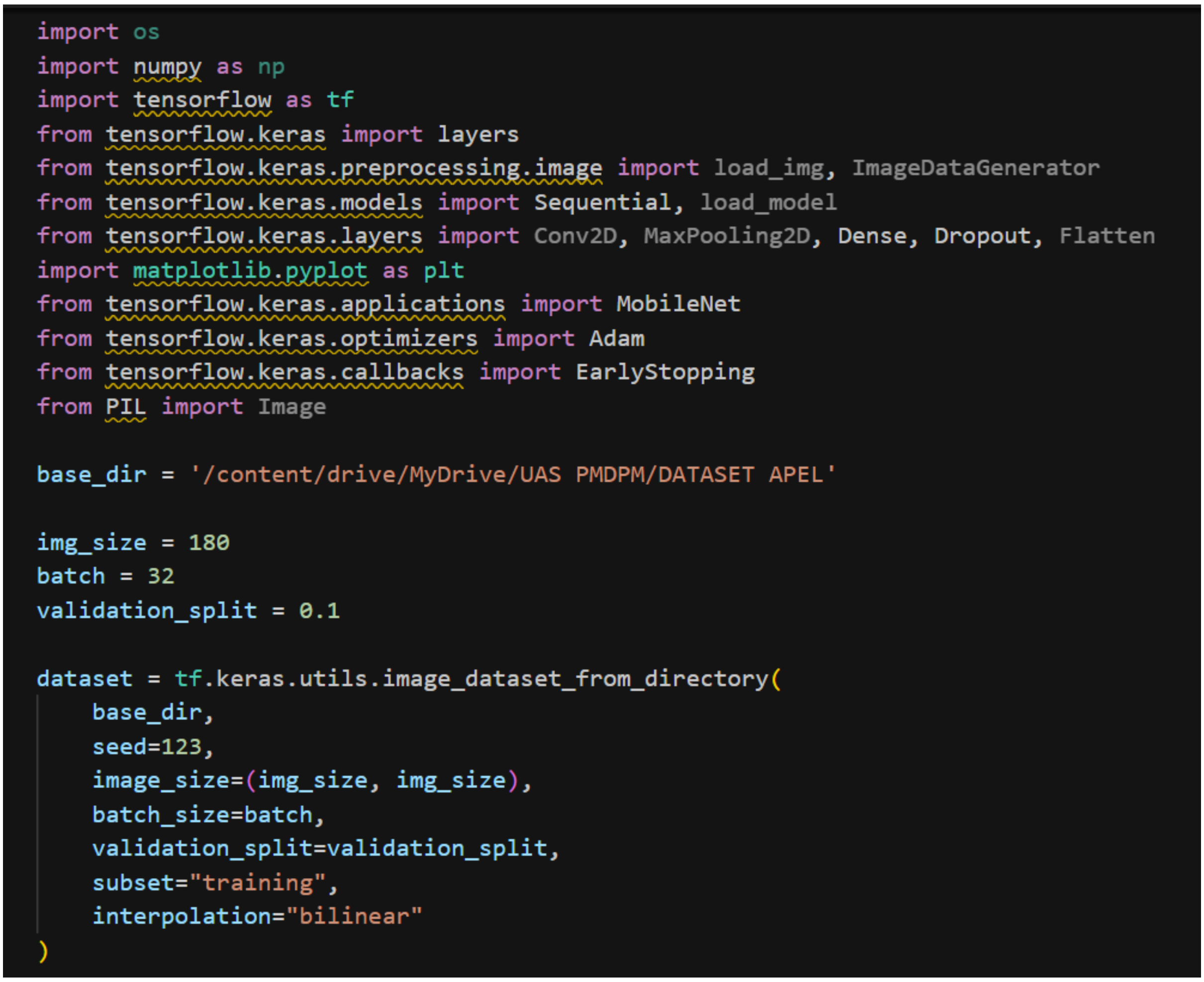


```python
test_dir = "/content/drive/MyDrive/UAS PMDPM/Test"
test_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(img_size, img_size),
    batch_size=batch_size,
)

test_loss, test_acc = model.evaluate(test_dataset)
print(f"Test Accuracy: {test_acc * 100:.2f}%")

def classify_image(image_path):
    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 class_name, confidence


image_path = "/content/drive/MyDrive/UAS PMDPM/Test/TestApelMerah/testApelMerah01.jpg"  # Update the path
class_name, confidence = classify_image(image_path)
print(f"Prediction: {class_name} with {confidence:.2f}% confidence.")
```

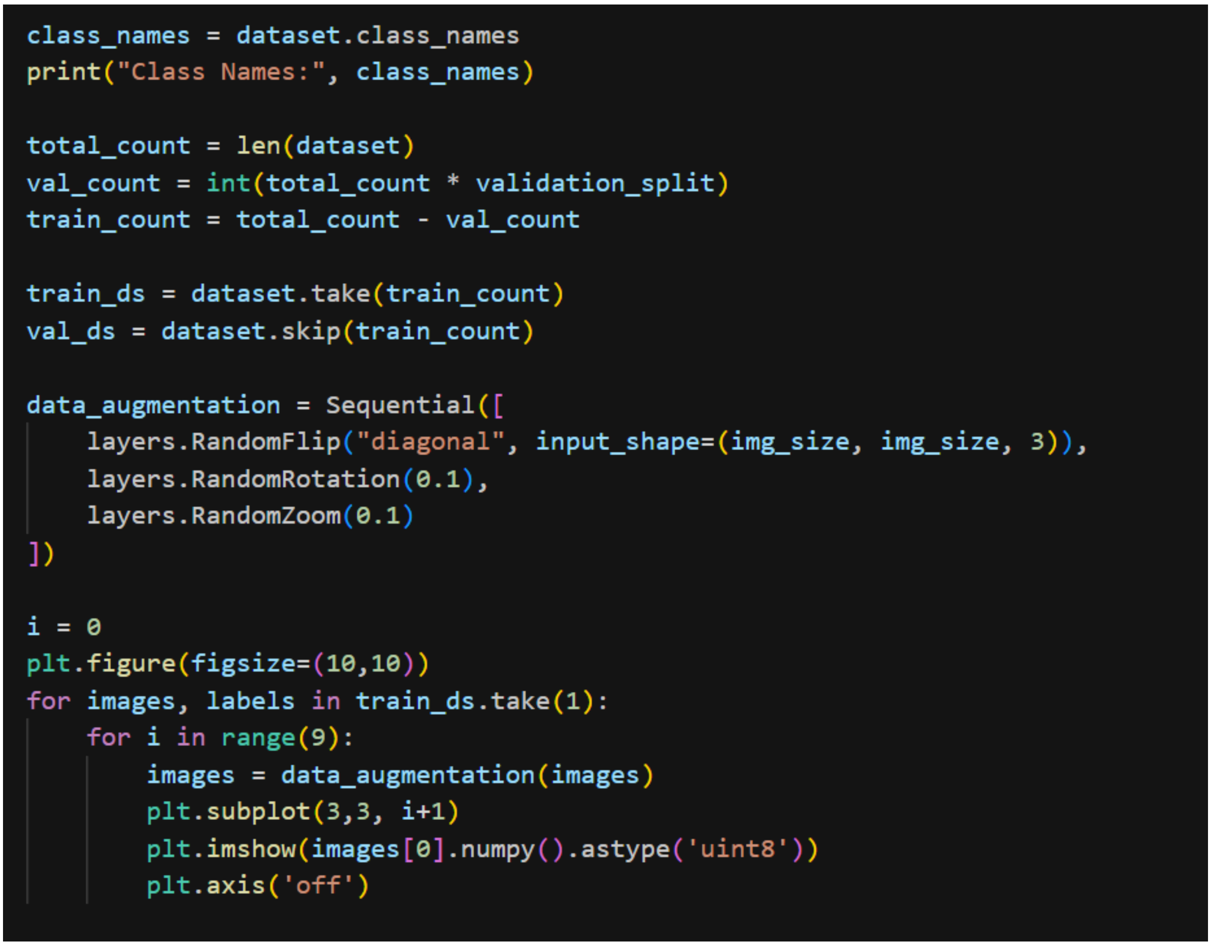


```python
y_true = np.concatenate([y.numpy() for x, y in val_ds], axis=0)
y_pred = model.predict(val_ds)
y_pred_class = np.argmax(y_pred, axis=1)

conf_mat = tf.math.confusion_matrix(y_true, y_pred_class)
class_names = ['Apel Hijau', 'Apel Kuning', 'Apel Merah']

plt.figure(figsize=(8, 6))
sns.heatmap(conf_mat.numpy(), annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

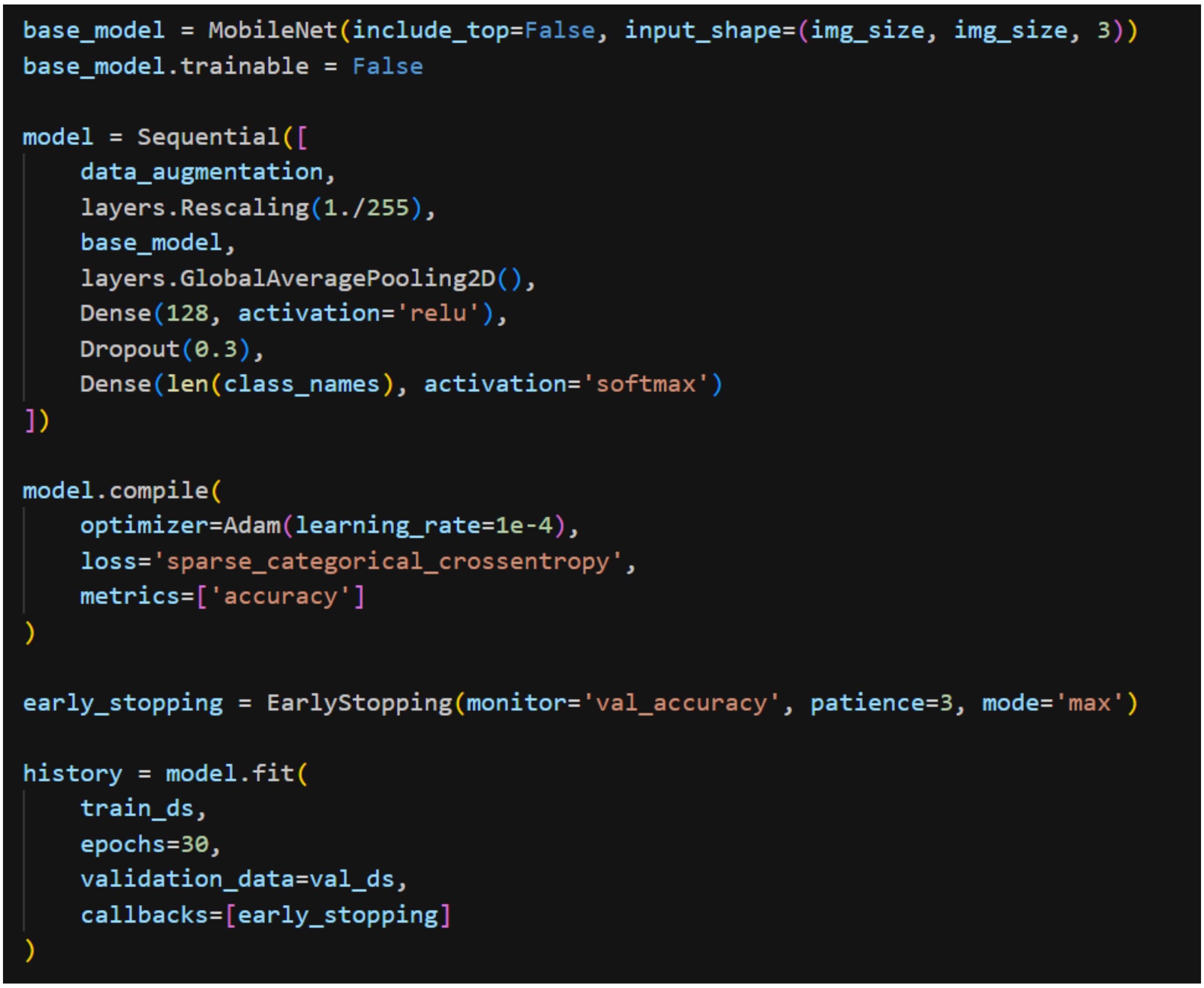
Notebook_MobileNet_A_SciPy_Vivi.ipynb



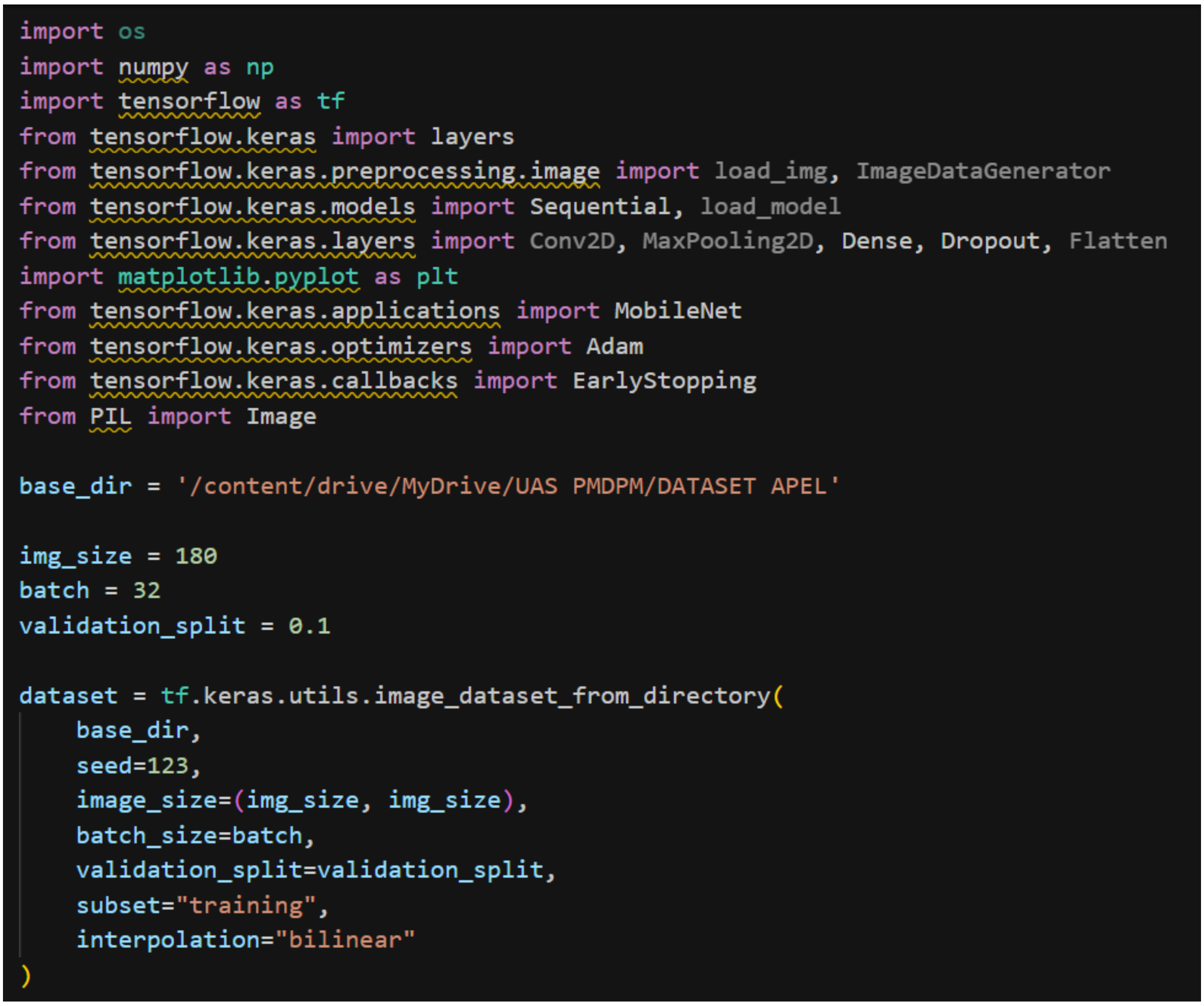
```
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, 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

base_dir = '/content/drive/MyDrive/UAS PMDPM/DATASET APEL'

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"
)
```



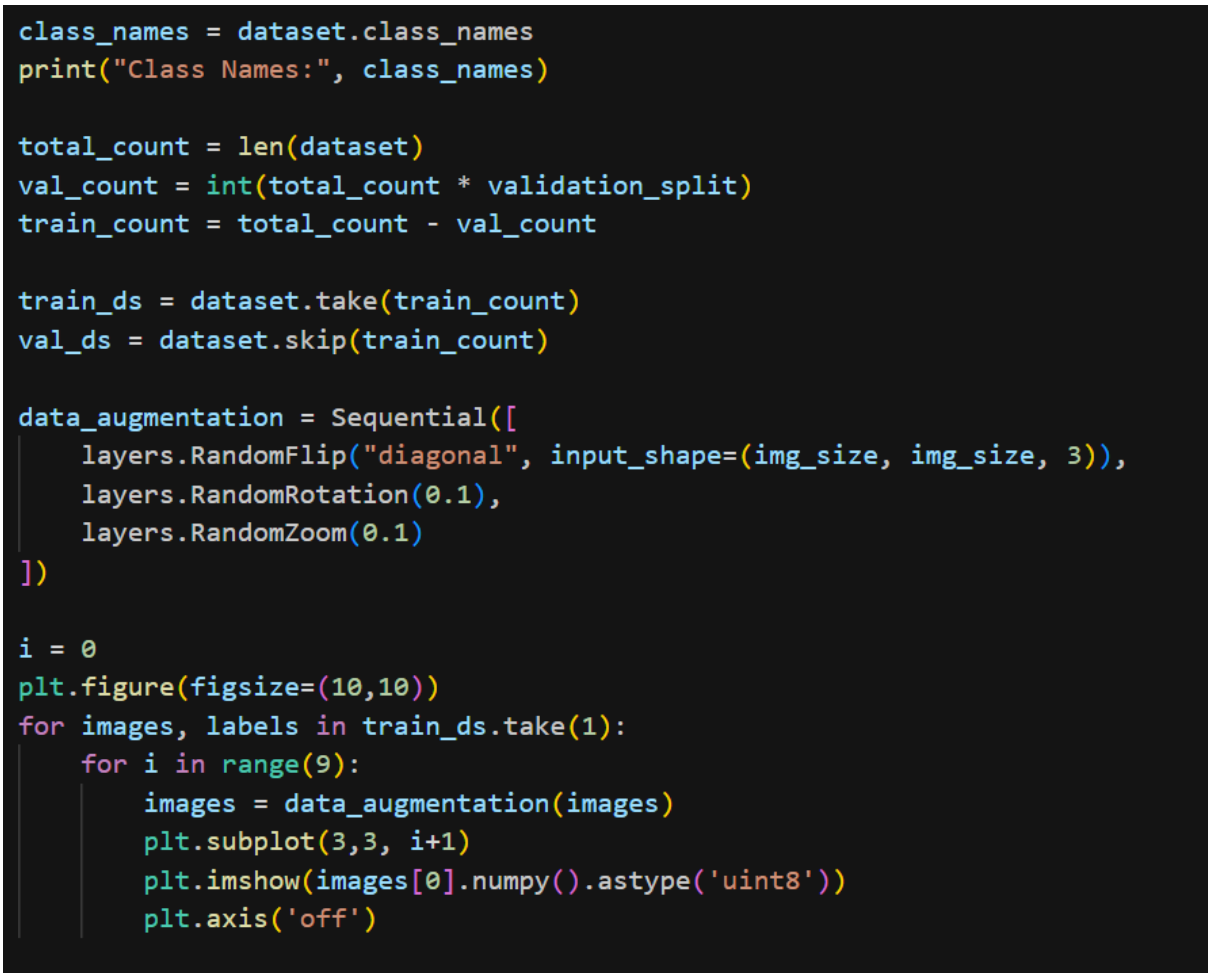
```python
class_names = dataset.class_names
print("Class Names:", class_names)

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)
])

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')
```



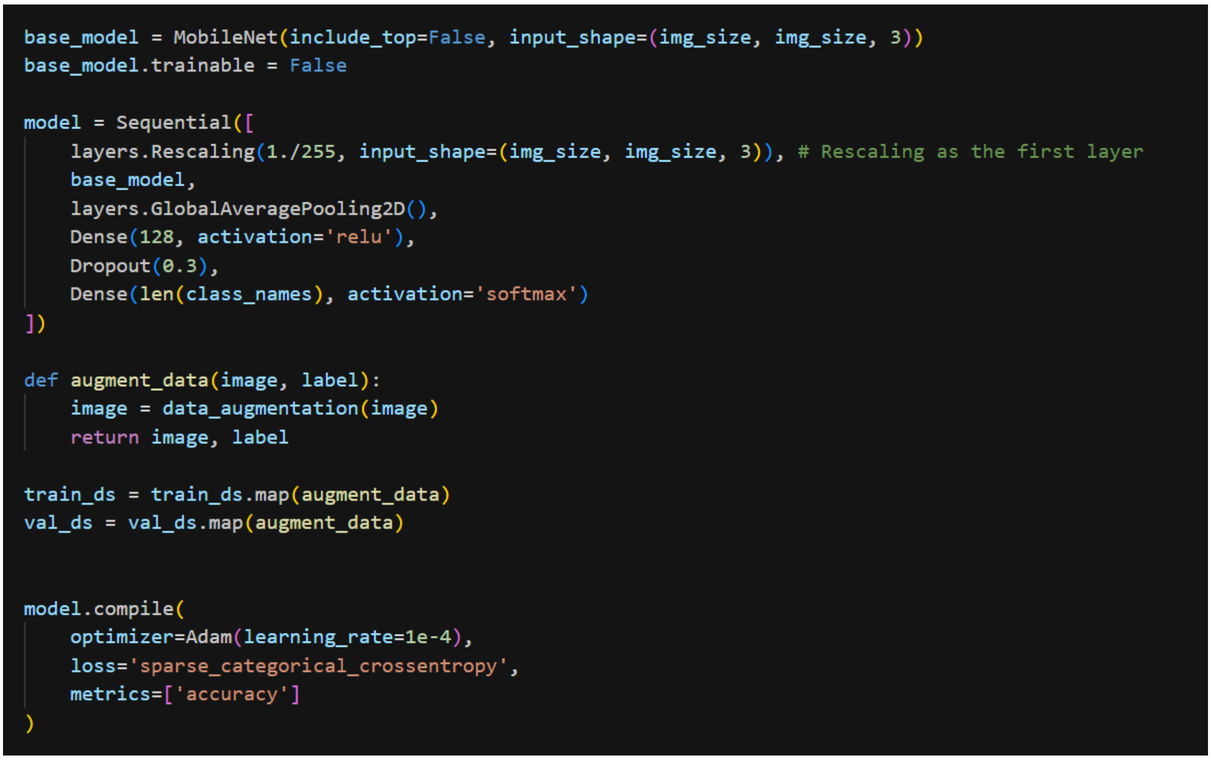
```python
base_model = MobileNet(include_top=False, input_shape=(img_size, img_size, 3))
base_model.trainable = False

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')
])

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]
)
```



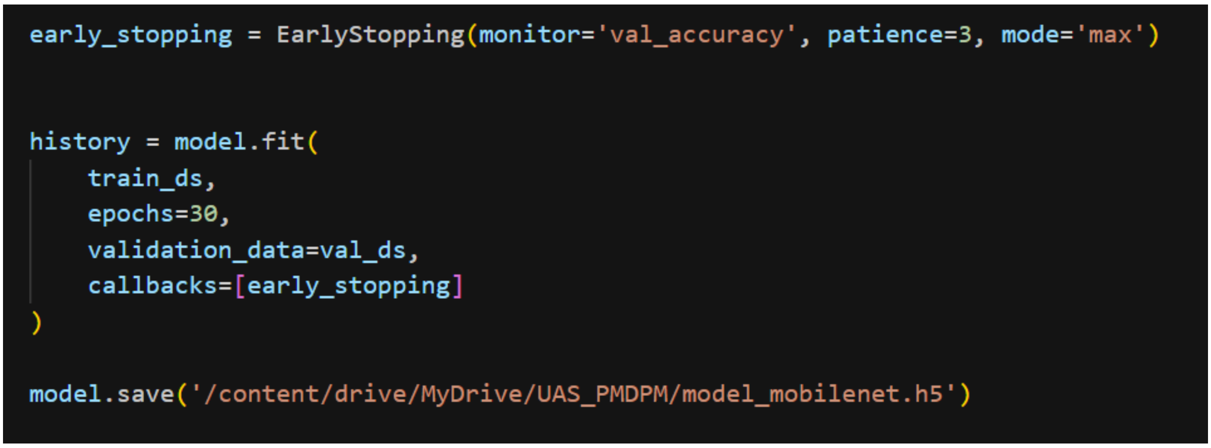
```python
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.preprocessing.image import load_img, ImageDataGenerator
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, 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

base_dir = '/content/drive/MyDrive/UAS PMDPM/DATASET APEL'

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"
)
```



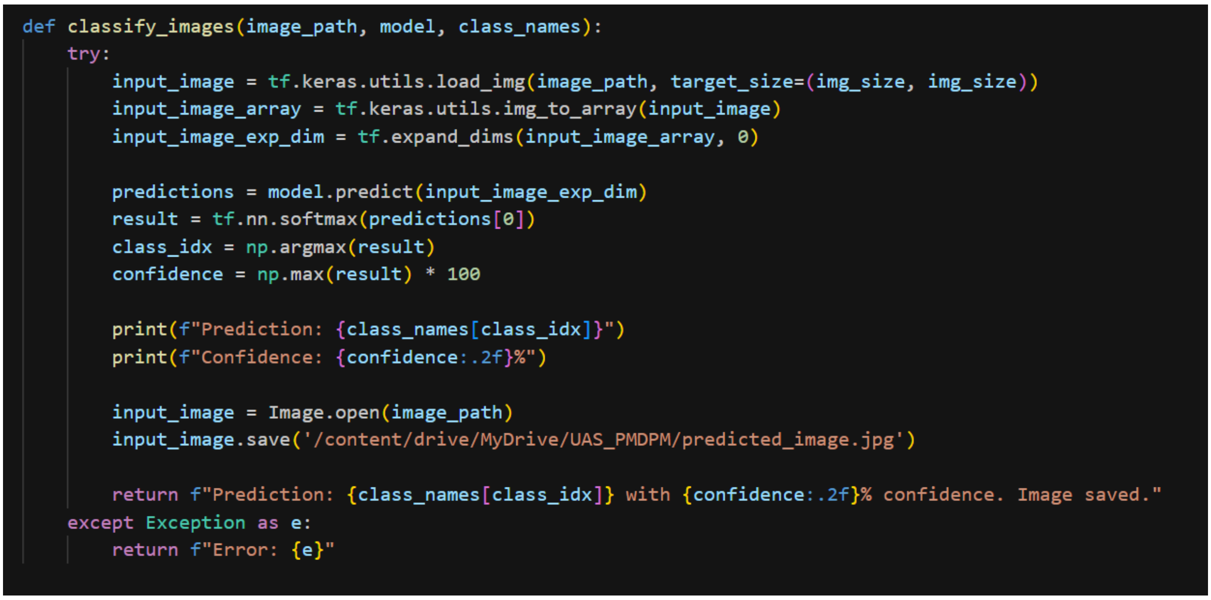
```
class_names = dataset.class_names
print("Class Names:", class_names)

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)
])

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')
```



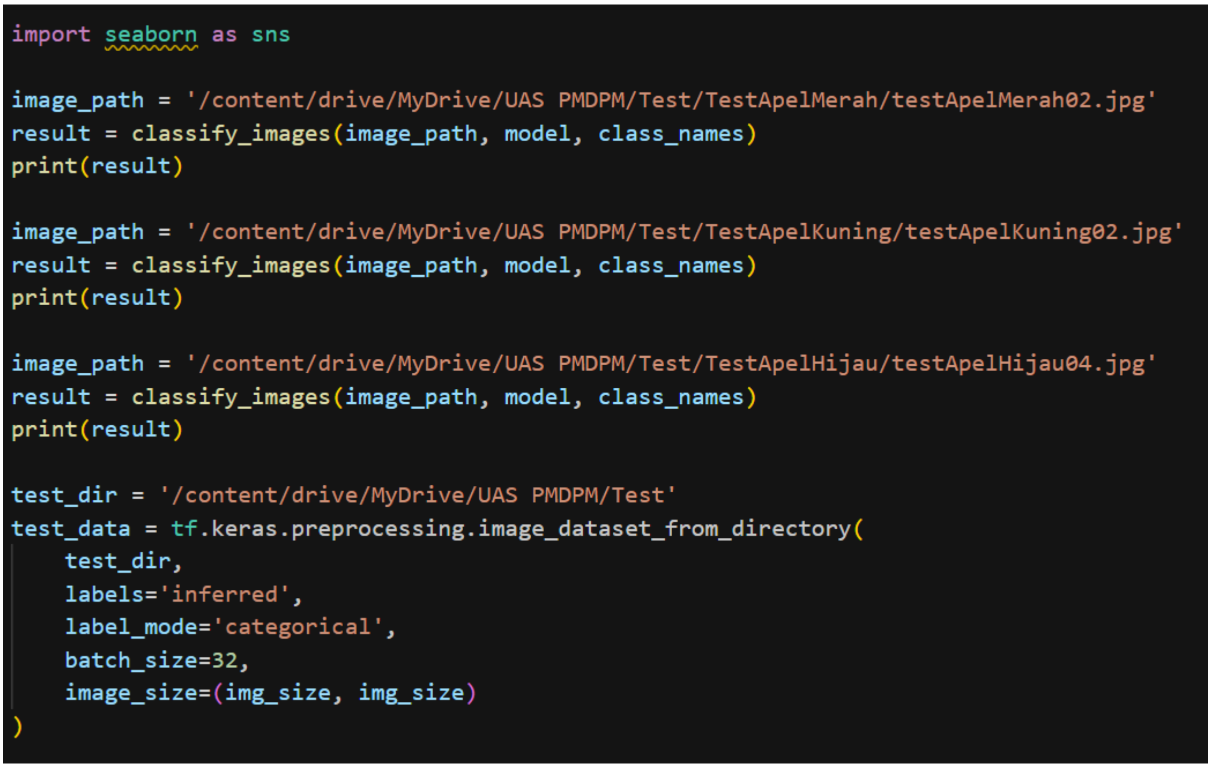
```
base_model = MobileNet(include_top=False, input_shape=(img_size, img_size, 3))
base_model.trainable = False

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

def augment_data(image, label):
    image = data_augmentation(image)
    return image, label

train_ds = train_ds.map(augment_data)
val_ds = val_ds.map(augment_data)


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



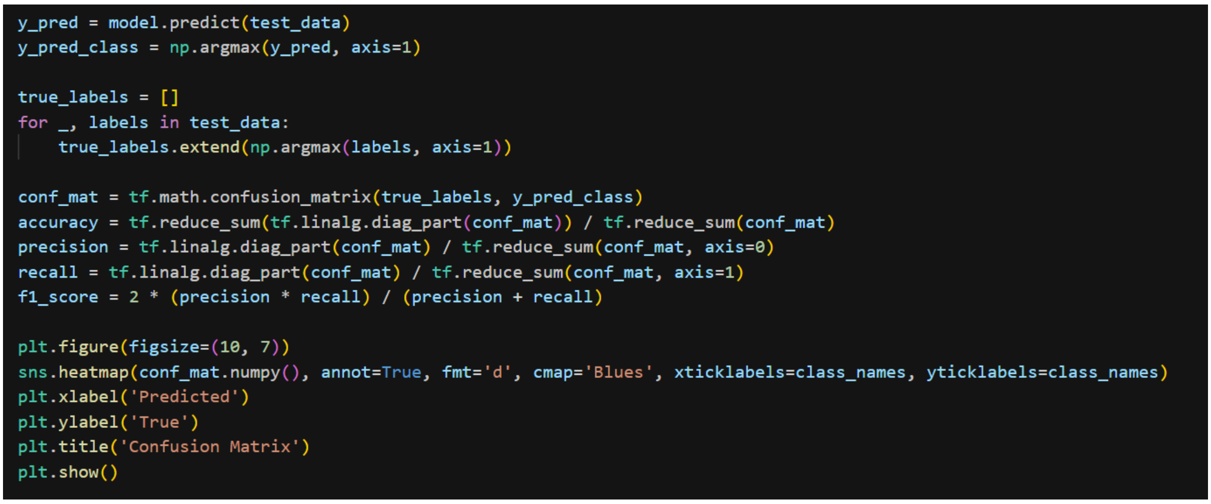
```python
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, mode='max')


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

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



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

        input_image = Image.open(image_path)
        input_image.save('/content/drive/MyDrive/UAS_PMDPM/predicted_image.jpg')

        return f"Prediction: {class_names[class_idx]} with {confidence:.2f}% confidence. Image saved."
    except Exception as e:
        return f"Error: {e}"
```

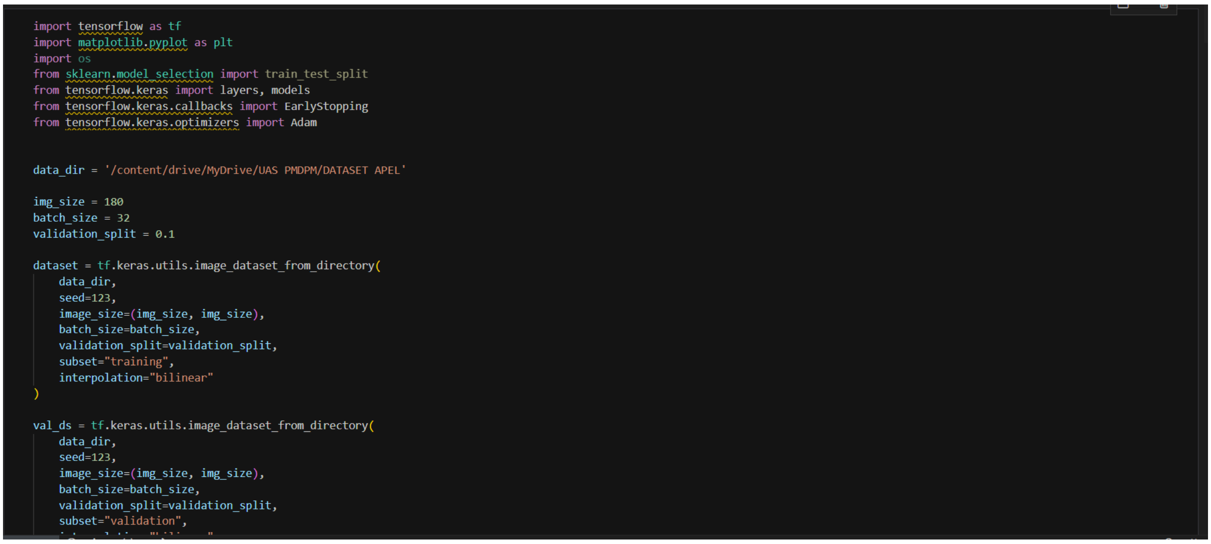


```
import seaborn as sns

image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelMerah/testApelMerah02.jpg'
result = classify_images(image_path, model, class_names)
print(result)

image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelKuning/testApelKuning02.jpg'
result = classify_images(image_path, model, class_names)
print(result)

image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelHijau/testApelHijau04.jpg'
result = classify_images(image_path, model, class_names)
print(result)

test_dir = '/content/drive/MyDrive/UAS PMDPM/Test'
test_data = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    labels='inferred',
    label_mode='categorical',
    batch_size=32,
    image_size=(img_size, img_size)
)
```

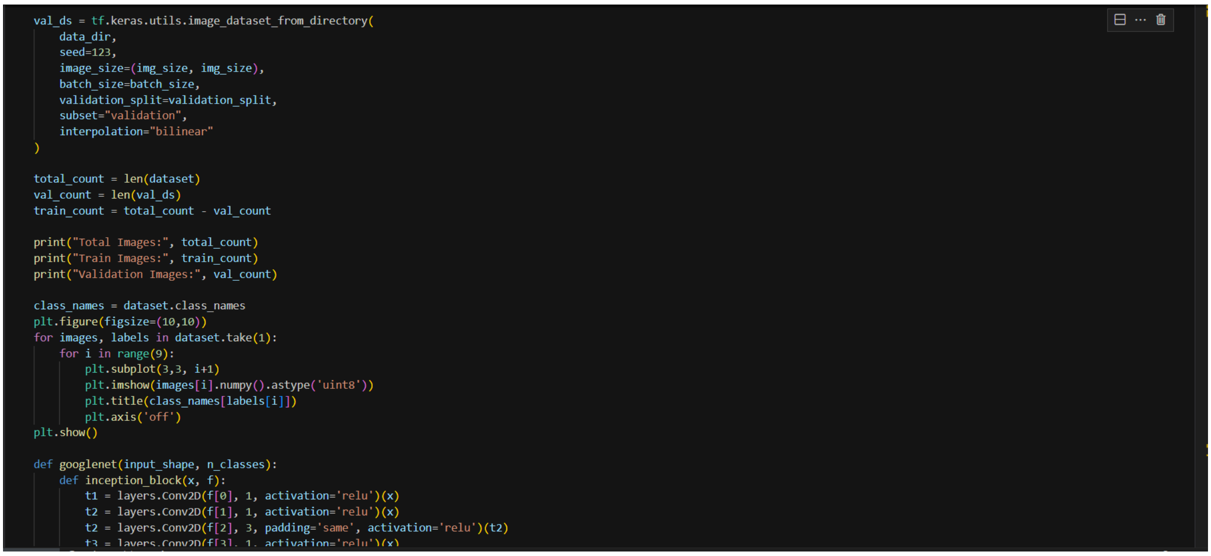


```
y_pred = model.predict(test_data)
y_pred_class = np.argmax(y_pred, axis=1)

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('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

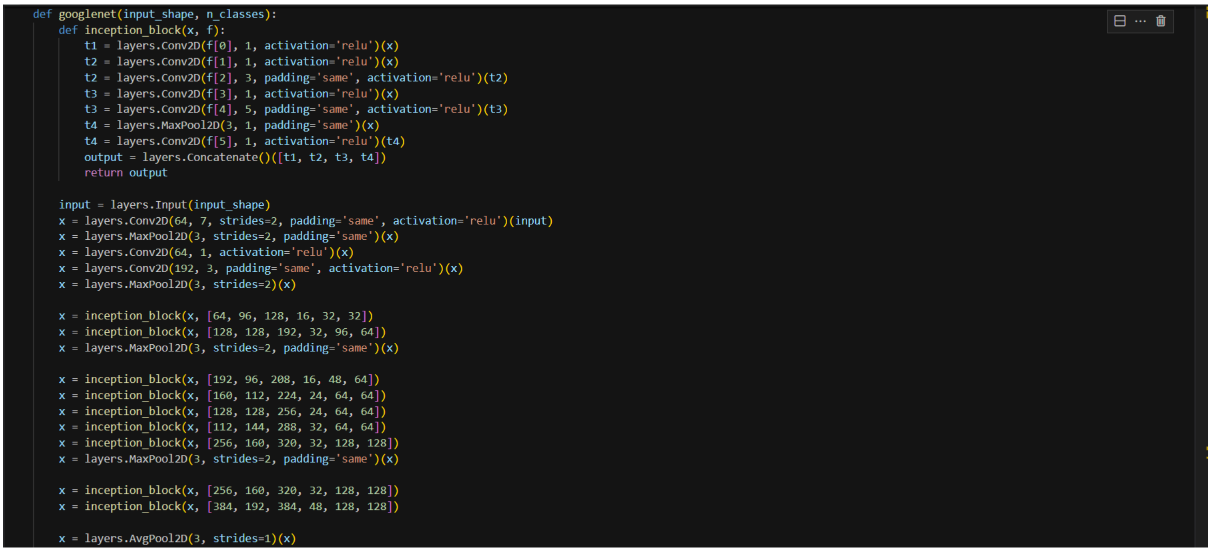
Notebook_GoogleNet_A_SciPy_Galih.ipynb



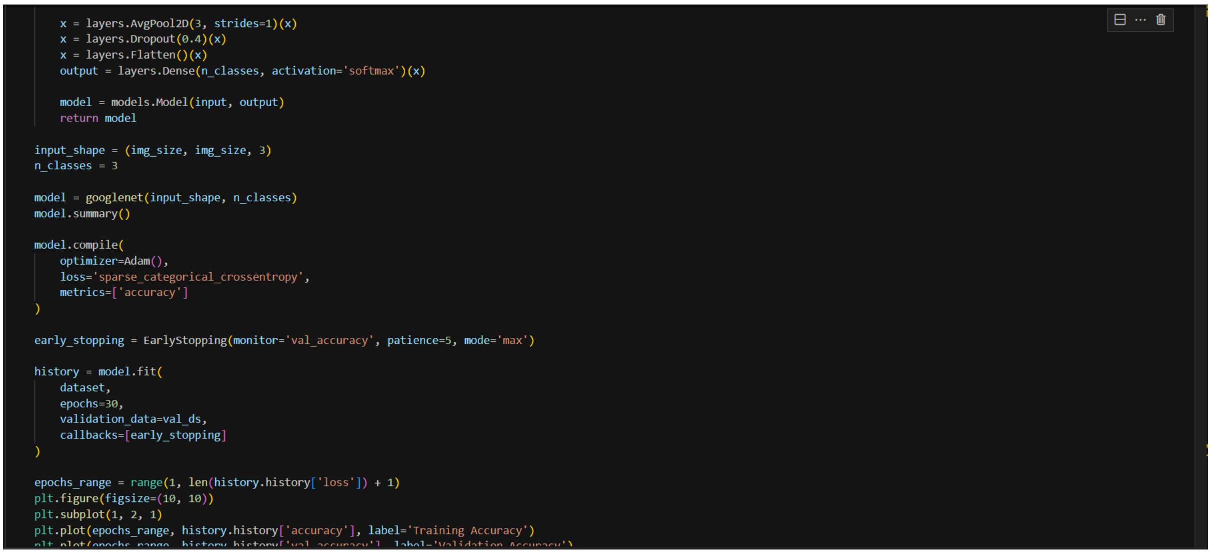
```
import tensorflow as tf
import matplotlib.pyplot as plt
import os
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam


data_dir = '/content/drive/MyDrive/UAS PMDPM/DATASET APEL'

img_size = 180
batch_size = 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_size,
    validation_split=validation_split,
    subset="training",
    interpolation="bilinear"
)

val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size,
    validation_split=validation_split,
    subset="validation",
    interpolation="bilinear"
```



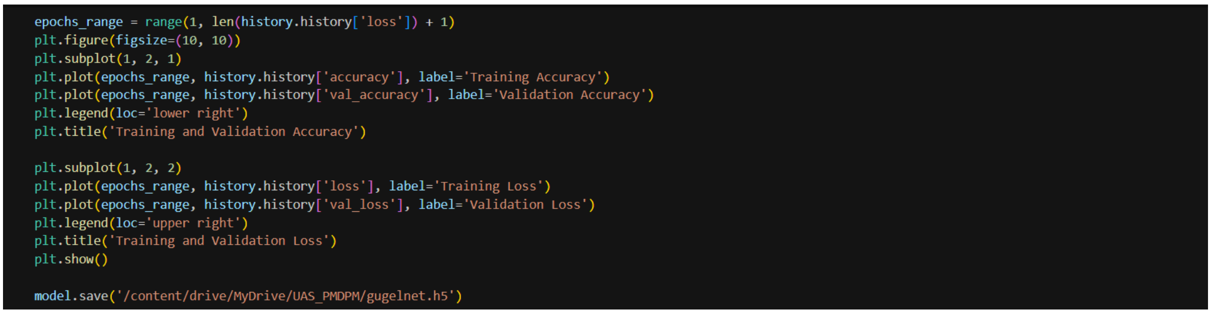
```
val_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    seed=123,
    image_size=(img_size, img_size),
    batch_size=batch_size,
    validation_split=validation_split,
    subset="validation",
    interpolation="bilinear"
)

total_count = len(dataset)
val_count = len(val_ds)
train_count = total_count - val_count

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

class_names = dataset.class_names
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()

def googlenet(input_shape, n_classes):
    def inception_block(x, f):
        t1 = layers.Conv2D(f[0], 1, activation='relu')(x)
        t2 = layers.Conv2D(f[1], 1, activation='relu')(x)
        t2 = layers.Conv2D(f[2], 3, padding='same', activation='relu')(t2)
        t3 = layers.Conv2D(f[3], 1, activation='relu')(x)
```



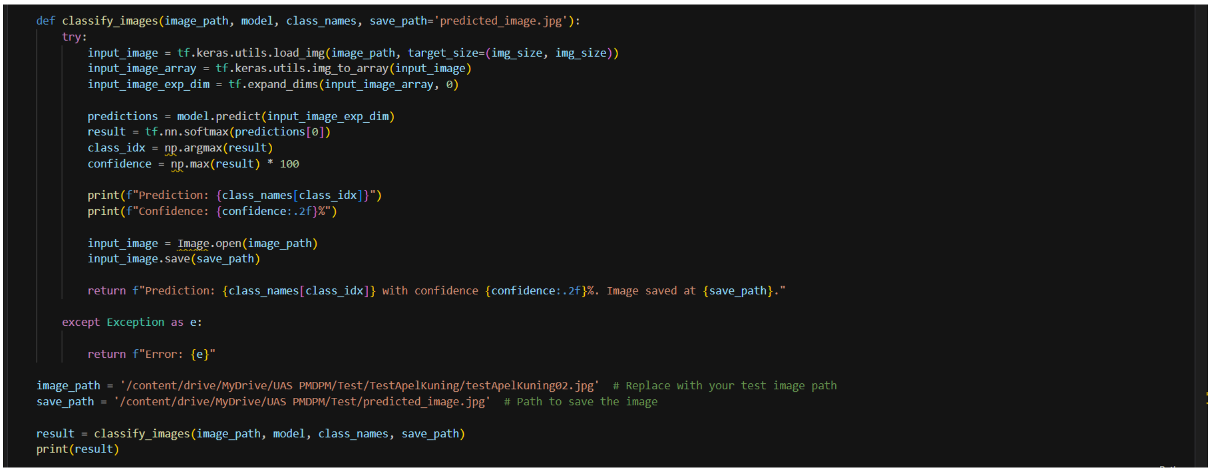
```
def googlenet(input_shape, n_classes):
    def inception_block(x, f):
        t1 = layers.Conv2D(f[0], 1, activation='relu')(x)
        t2 = layers.Conv2D(f[1], 1, activation='relu')(x)
        t2 = layers.Conv2D(f[2], 3, padding='same', activation='relu')(t2)
        t3 = layers.Conv2D(f[3], 1, activation='relu')(x)
        t3 = layers.Conv2D(f[4], 5, padding='same', activation='relu')(t3)
        t4 = layers.MaxPool2D(3, 1, padding='same')(x)
        t4 = layers.Conv2D(f[5], 1, activation='relu')(t4)
        output = layers.Concatenate()([t1, t2, t3, t4])
        return output

    input = layers.Input(input_shape)
    x = layers.Conv2D(64, 7, strides=2, padding='same', activation='relu')(input)
    x = layers.MaxPool2D(3, strides=2, padding='same')(x)
    x = layers.Conv2D(64, 1, activation='relu')(x)
    x = layers.Conv2D(192, 3, padding='same', activation='relu')(x)
    x = layers.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 = layers.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 = layers.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 = layers.AvgPool2D(3, strides=1)(x)
```



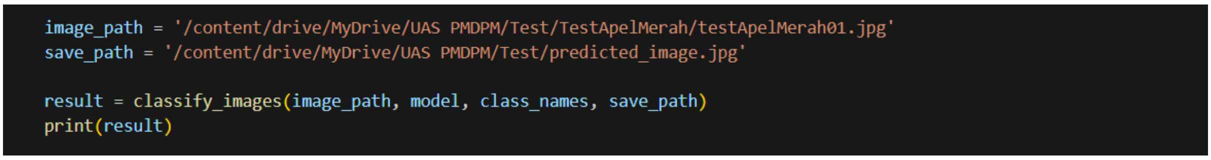
```
    x = layers.AvgPool2D(3, strides=1)(x)
    x = layers.Dropout(0.4)(x)
    x = layers.Flatten()(x)
    output = layers.Dense(n_classes, activation='softmax')(x)

    model = models.Model(input, output)
    return model

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

model = googlenet(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['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')
```



```
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('/content/drive/MyDrive/UAS_PMDPM/gugelnet.h5')
```



```
def classify_images(image_path, model, class_names, save_path='predicted_image.jpg'):
    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"Prediction: {class_names[class_idx]}")
        print(f"Confidence: {confidence:.2f}%")

        input_image = Image.open(image_path)
        input_image.save(save_path)

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

    except Exception as e:

        return f"Error: {e}"

image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelKuning/testApelKuning02.jpg'  # Replace with your test image path
save_path = '/content/drive/MyDrive/UAS PMDPM/Test/predicted_image.jpg'  # Path to save the image

result = classify_images(image_path, model, class_names, save_path)
print(result)
```



```
image_path = '/content/drive/MyDrive/UAS PMDPM/Test/TestApelMerah/testApelMerah01.jpg'
save_path = '/content/drive/MyDrive/UAS PMDPM/Test/predicted_image.jpg'

result = classify_images(image_path, model, class_names, save_path)
print(result)
```