[2]:
```
import tensorflow as tf
import os
import numpy as np
from tensorflow.keras.applications import MobileNetV2
# from tensorflow.keras.applications import MobileNetV3Large
# from tensorflow.keras.applications import EfficientNetV2B1
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
```

[3]:
```
# Konfigurasi penggunaan memori GPU agar lebih efisien
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
    except RuntimeError as e:
        print(e)
```

[4]:
```
def _bytes_feature(value):
    """Returns a bytes_list from a string / byte."""
    if isinstance(value, type(tf.constant(0))):
        value = value.numpy()
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))

def _float_feature(value):
    """Returns a float_list from a float / double."""
    return tf.train.Feature(float_list=tf.train.FloatList(value=[value]))

def _int64_feature(value):
    """Returns an int64_list from a bool / enum / int / uint."""
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))

def serialize_example(image, label):
    image = tf.cast(image * 255.0, tf.uint8)
    feature = {
        'image': _bytes_feature(tf.io.encode_jpeg(image).numpy()),
        'label': _int64_feature(label),
    }
    example_proto = tf.train.Example(features=tf.train.Features(feature=feature))
    return example_proto.SerializeToString()

def write_tfrecords(file_path, dataset, class_names):
    with tf.io.TFRecordWriter(file_path) as writer:
        for image_path in dataset:
            image = tf.io.read_file(image_path)
            image = tf.image.decode_jpeg(image, channels=3)
            image = tf.image.resize(image, [256, 256])
            image /= 255.0
            label_name = tf.strings.split(image_path, os.path.sep)[-2]
            label = tf.argmax(tf.cast(tf.equal(class_names, label_name), tf.float32))
            tf_example = serialize_example(image, label)
            writer.write(tf_example)

train_dir = 'New_Plant_Diseases_Dataset(Augmented)/train'
validation_dir = 'New_Plant_Diseases_Dataset(Augmented)/valid'
class_names = np.array(sorted([item for item in os.listdir(train_dir) if os.path.isdir(os.path.join(train_dir, item))]))

train_paths = tf.io.gfile.glob(f"{train_dir}/*/*")
valid_paths = tf.io.gfile.glob(f"{validation_dir}/*/*")

write_tfrecords('train256.tfrecord', train_paths, class_names)
write_tfrecords('valid256.tfrecord', valid_paths, class_names)
```

[5]:
```
def _parse_image_function(proto):
    keys_to_features = {
        'image': tf.io.FixedLenFeature([], tf.string),
        'label': tf.io.FixedLenFeature([], tf.int64),
    }
    parsed_features = tf.io.parse_single_example(proto, keys_to_features)
    image = tf.io.decode_jpeg(parsed_features['image'], channels=3)
    # image = tf.image.resize(image, [128, 128])
    image = tf.image.resize(image, [256, 256])
    image /= 255.0
    return image, parsed_features['label']

def load_dataset(file_path, batch_size):
    dataset = tf.data.TFRecordDataset(file_path)
    dataset = dataset.map(_parse_image_function, num_parallel_calls=tf.data.experimental.AUTOTUNE)
    dataset = dataset.shuffle(buffer_size=1000)
    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
    return dataset
```

[6]:
```
def augment(image, label):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    return image, label
```

[7]:
```
# Parameters
batch_size = 16
# img_size = (128, 128)
img_size = (256, 256)

# Load dataset from TFRecords
train_ds = load_dataset('train256.tfrecord', batch_size)
# train_ds = train_ds.map(augment, num_parallel_calls=tf.data.experimental.AUTOTUNE)
val_ds = load_dataset('valid256.tfrecord', batch_size)

# Define and compile the model
# base_model = MobileNetV2(input_shape=(128, 128, 3), include_top=False, weights='imagenet')
base_model = MobileNetV2(input_shape=(256, 256, 3), include_top=False, weights='imagenet')
# base_model = EfficientNetB3(input_shape=(128, 128, 3), include_top=False, weights='imagenet')
# base_model = EfficientNetV2B1(input_shape=(128, 128, 3), include_top=False, weights='imagenet')

base_model.trainable = False

model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.5),
    Dense(len(class_names), activation='softmax')
])
# model = Sequential([
#     base_model,
#     GlobalAveragePooling2D(),
#     Dropout(0.5),
#     Dense(256, activation='relu', kernel_regularizer=tf.keras.regularizers.l2(0.01)),
#     Dropout(0.5),
#     Dense(len(class_names), activation='softmax')
# ])
# model = Sequential([
#     Conv2D(32, (3, 3), activation='relu', input_shape=(128, 128, 3)),
#     MaxPooling2D((2, 2)),
#     Conv2D(64, (3, 3), activation='relu'),
#     MaxPooling2D((2, 2)),
#     Conv2D(128, (3, 3), activation='relu'),
#     MaxPooling2D((2, 2)),
#     Flatten(),
#     Dense(128, activation='relu'),
#     Dropout(0.5),
#     Dense(len(class_names), activation='softmax')
# ])

model.compile(optimizer=Adam(learning_rate=0.00001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

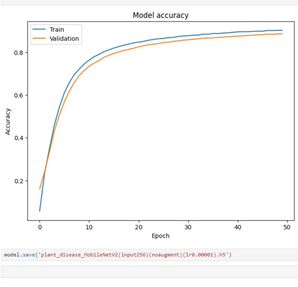
```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loade
d as the default.
Model: "sequential1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 8, 8, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 38)                48678

=================================================================
Total params: 2,306,662
Trainable params: 48,678
Non-trainable params: 2,257,984
_________________________________________________________________
```

[8]:
```
# Train the model
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=3, min_lr=0.00001)
# history = model.fit(train_ds, validation_data=val_ds, epochs=50, callbacks=[early_stopping])
history = model.fit(train_ds, validation_data=val_ds, epochs=50, callbacks=[early_stopping, reduce_lr])
# history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
4394/4394 [==============================] - 116s 26ms/step - loss: 4.2174 - accuracy: 0.0570 - val_loss: 3.1947 - val_accuracy: 0.1605 - lr: 1.0000e-05
Epoch 2/50
4394/4394 [==============================] - 112s 25ms/step - loss: 2.8528 - accuracy: 0.2335 - val_loss: 2.7581 - val_accuracy: 0.2412 - lr: 1.0000e-05
Epoch 3/50
4394/4394 [==============================] - 112s 26ms/step - loss: 2.3538 - accuracy: 0.3524 - val_loss: 2.2970 - val_accuracy: 0.3354 - lr: 1.0000e-05
Epoch 4/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.9373 - accuracy: 0.4617 - val_loss: 1.9357 - val_accuracy: 0.4343 - lr: 1.0000e-05
Epoch 5/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.6462 - accuracy: 0.5436 - val_loss: 1.6686 - val_accuracy: 0.5078 - lr: 1.0000e-05
Epoch 6/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.4006 - accuracy: 0.6112 - val_loss: 1.4660 - val_accuracy: 0.5681 - lr: 1.0000e-05
Epoch 7/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.2444 - accuracy: 0.6580 - val_loss: 1.3073 - val_accuracy: 0.6183 - lr: 1.0000e-05
Epoch 8/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.1120 - accuracy: 0.6957 - val_loss: 1.1838 - val_accuracy: 0.6588 - lr: 1.0000e-05
Epoch 9/50
4394/4394 [==============================] - 112s 26ms/step - loss: 1.0161 - accuracy: 0.7214 - val_loss: 1.0822 - val_accuracy: 0.6909 - lr: 1.0000e-05
Epoch 10/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.9289 - accuracy: 0.7449 - val_loss: 1.0007 - val_accuracy: 0.7143 - lr: 1.0000e-05
Epoch 11/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.8653 - accuracy: 0.7634 - val_loss: 0.9319 - val_accuracy: 0.7349 - lr: 1.0000e-05
Epoch 12/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.8035 - accuracy: 0.7790 - val_loss: 0.8762 - val_accuracy: 0.7486 - lr: 1.0000e-05
Epoch 13/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.7583 - accuracy: 0.7901 - val_loss: 0.8281 - val_accuracy: 0.7616 - lr: 1.0000e-05
Epoch 14/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.7171 - accuracy: 0.8025 - val_loss: 0.7880 - val_accuracy: 0.7764 - lr: 1.0000e-05
Epoch 15/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.6810 - accuracy: 0.8110 - val_loss: 0.7448 - val_accuracy: 0.7859 - lr: 1.0000e-05
Epoch 16/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.6502 - accuracy: 0.8198 - val_loss: 0.7122 - val_accuracy: 0.7944 - lr: 1.0000e-05
Epoch 17/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.6235 - accuracy: 0.8265 - val_loss: 0.6851 - val_accuracy: 0.8012 - lr: 1.0000e-05
Epoch 18/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5987 - accuracy: 0.8329 - val_loss: 0.6585 - val_accuracy: 0.8080 - lr: 1.0000e-05
Epoch 19/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5775 - accuracy: 0.8380 - val_loss: 0.6356 - val_accuracy: 0.8136 - lr: 1.0000e-05
Epoch 20/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5590 - accuracy: 0.8437 - val_loss: 0.6156 - val_accuracy: 0.8186 - lr: 1.0000e-05
Epoch 21/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5395 - accuracy: 0.8473 - val_loss: 0.5956 - val_accuracy: 0.8253 - lr: 1.0000e-05
Epoch 22/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5235 - accuracy: 0.8510 - val_loss: 0.5779 - val_accuracy: 0.8301 - lr: 1.0000e-05
Epoch 23/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.5090 - accuracy: 0.8561 - val_loss: 0.5601 - val_accuracy: 0.8351 - lr: 1.0000e-05
Epoch 24/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4965 - accuracy: 0.8599 - val_loss: 0.5487 - val_accuracy: 0.8376 - lr: 1.0000e-05
Epoch 25/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4830 - accuracy: 0.8631 - val_loss: 0.5346 - val_accuracy: 0.8409 - lr: 1.0000e-05
Epoch 26/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4744 - accuracy: 0.8647 - val_loss: 0.5215 - val_accuracy: 0.8455 - lr: 1.0000e-05
Epoch 27/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4619 - accuracy: 0.8683 - val_loss: 0.5126 - val_accuracy: 0.8466 - lr: 1.0000e-05
Epoch 28/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4518 - accuracy: 0.8695 - val_loss: 0.4996 - val_accuracy: 0.8500 - lr: 1.0000e-05
Epoch 29/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4403 - accuracy: 0.8736 - val_loss: 0.4908 - val_accuracy: 0.8532 - lr: 1.0000e-05
Epoch 30/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4322 - accuracy: 0.8761 - val_loss: 0.4797 - val_accuracy: 0.8563 - lr: 1.0000e-05
Epoch 31/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.4250 - accuracy: 0.8773 - val_loss: 0.4714 - val_accuracy: 0.8585 - lr: 1.0000e-05
Epoch 32/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4173 - accuracy: 0.8800 - val_loss: 0.4636 - val_accuracy: 0.8607 - lr: 1.0000e-05
Epoch 33/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4108 - accuracy: 0.8802 - val_loss: 0.4561 - val_accuracy: 0.8626 - lr: 1.0000e-05
Epoch 34/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.4040 - accuracy: 0.8846 - val_loss: 0.4481 - val_accuracy: 0.8650 - lr: 1.0000e-05
Epoch 35/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3974 - accuracy: 0.8841 - val_loss: 0.4397 - val_accuracy: 0.8666 - lr: 1.0000e-05
Epoch 36/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3887 - accuracy: 0.8879 - val_loss: 0.4364 - val_accuracy: 0.8672 - lr: 1.0000e-05
Epoch 37/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3868 - accuracy: 0.8875 - val_loss: 0.4291 - val_accuracy: 0.8691 - lr: 1.0000e-05
Epoch 38/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3819 - accuracy: 0.8894 - val_loss: 0.4228 - val_accuracy: 0.8702 - lr: 1.0000e-05
Epoch 39/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3753 - accuracy: 0.8908 - val_loss: 0.4147 - val_accuracy: 0.8732 - lr: 1.0000e-05
Epoch 40/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3697 - accuracy: 0.8931 - val_loss: 0.4086 - val_accuracy: 0.8739 - lr: 1.0000e-05
Epoch 41/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3641 - accuracy: 0.8954 - val_loss: 0.4053 - val_accuracy: 0.8750 - lr: 1.0000e-05
Epoch 42/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3582 - accuracy: 0.8959 - val_loss: 0.3994 - val_accuracy: 0.8768 - lr: 1.0000e-05
Epoch 43/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3563 - accuracy: 0.8959 - val_loss: 0.3950 - val_accuracy: 0.8783 - lr: 1.0000e-05
Epoch 44/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3526 - accuracy: 0.8975 - val_loss: 0.3904 - val_accuracy: 0.8800 - lr: 1.0000e-05
Epoch 45/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3485 - accuracy: 0.8985 - val_loss: 0.3859 - val_accuracy: 0.8812 - lr: 1.0000e-05
Epoch 46/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3459 - accuracy: 0.8986 - val_loss: 0.3826 - val_accuracy: 0.8819 - lr: 1.0000e-05
Epoch 47/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3388 - accuracy: 0.9016 - val_loss: 0.3761 - val_accuracy: 0.8844 - lr: 1.0000e-05
Epoch 48/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3357 - accuracy: 0.9017 - val_loss: 0.3749 - val_accuracy: 0.8844 - lr: 1.0000e-05
Epoch 49/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3351 - accuracy: 0.9021 - val_loss: 0.3707 - val_accuracy: 0.8856 - lr: 1.0000e-05
Epoch 50/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3318 - accuracy: 0.9027 - val_loss: 0.3658 - val_accuracy: 0.8871 - lr: 1.0000e-05
```

[9]:
```
# # Evaluate the model
# loss, accuracy = model.evaluate(val_ds)
# print(f'Validation accuracy: {accuracy}')
```

[10]:
```
# Plot training & validation accuracy values
plt.figure(figsize=(8, 6))
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



[11]:
```
model.save('plant_disease_MobileNetV2(input256)(noaugment)(lr0.00001).h5')
```

[ ]: