[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=0.0001),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

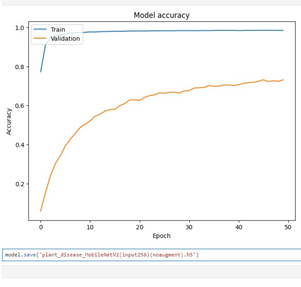
```
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 224]. Weights for input shape (224, 224) will be loade
d as the default.
Model: "sequential1"
_________________________________________________________________
 Layer (type)                Output 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=3, 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=30, callbacks=[early_stopping])
# history = model.fit(train_generator, validation_data=validation_generator, epochs=50, callbacks=[early_stopping, reduce_lr])
history = model.fit(train_ds, validation_data=val_ds, epochs=50)
```

```
Epoch 1/50
4394/4394 [==============================] - 118s 26ms/step - loss: 1.0490 - accuracy: 0.7723 - val_loss: 5.4872 - val_accuracy: 0.0583
Epoch 2/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.3635 - accuracy: 0.9121 - val_loss: 4.0903 - val_accuracy: 0.1547
Epoch 3/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.2193 - accuracy: 0.9441 - val_loss: 3.3309 - val_accuracy: 0.2415
Epoch 4/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.1673 - accuracy: 0.9558 - val_loss: 2.8341 - val_accuracy: 0.3030
Epoch 5/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.1383 - accuracy: 0.9622 - val_loss: 2.5470 - val_accuracy: 0.3425
Epoch 6/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.1210 - accuracy: 0.9658 - val_loss: 2.2219 - val_accuracy: 0.3938
Epoch 7/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.1082 - accuracy: 0.9697 - val_loss: 2.0567 - val_accuracy: 0.4264
Epoch 8/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.1008 - accuracy: 0.9721 - val_loss: 1.9079 - val_accuracy: 0.4570
Epoch 9/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0940 - accuracy: 0.9734 - val_loss: 1.7722 - val_accuracy: 0.4874
Epoch 10/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0897 - accuracy: 0.9747 - val_loss: 1.6914 - val_accuracy: 0.5041
Epoch 11/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.0832 - accuracy: 0.9770 - val_loss: 1.6243 - val_accuracy: 0.5210
Epoch 12/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0815 - accuracy: 0.9765 - val_loss: 1.5040 - val_accuracy: 0.5447
Epoch 13/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.0755 - accuracy: 0.9782 - val_loss: 1.4938 - val_accuracy: 0.5552
Epoch 14/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.0741 - accuracy: 0.9788 - val_loss: 1.4120 - val_accuracy: 0.5710
Epoch 15/50
4394/4394 [==============================] - 112s 26ms/step - loss: 0.0726 - accuracy: 0.9794 - val_loss: 1.3767 - val_accuracy: 0.5790
Epoch 16/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0688 - accuracy: 0.9803 - val_loss: 1.4042 - val_accuracy: 0.5806
Epoch 17/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.0690 - accuracy: 0.9799 - val_loss: 1.3093 - val_accuracy: 0.5982
Epoch 18/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0681 - accuracy: 0.9804 - val_loss: 1.2833 - val_accuracy: 0.6103
Epoch 19/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0651 - accuracy: 0.9818 - val_loss: 1.2089 - val_accuracy: 0.6282
Epoch 20/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0646 - accuracy: 0.9812 - val_loss: 1.2068 - val_accuracy: 0.6288
Epoch 21/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0621 - accuracy: 0.9819 - val_loss: 1.2348 - val_accuracy: 0.6261
Epoch 22/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0631 - accuracy: 0.9815 - val_loss: 1.1877 - val_accuracy: 0.6416
Epoch 23/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0612 - accuracy: 0.9821 - val_loss: 1.1298 - val_accuracy: 0.6501
Epoch 24/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0607 - accuracy: 0.9827 - val_loss: 1.1262 - val_accuracy: 0.6543
Epoch 25/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0603 - accuracy: 0.9826 - val_loss: 1.0898 - val_accuracy: 0.6645
Epoch 26/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0599 - accuracy: 0.9829 - val_loss: 1.1026 - val_accuracy: 0.6627
Epoch 27/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0581 - accuracy: 0.9829 - val_loss: 1.0842 - val_accuracy: 0.6667
Epoch 28/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0587 - accuracy: 0.9827 - val_loss: 1.0916 - val_accuracy: 0.6681
Epoch 29/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0554 - accuracy: 0.9838 - val_loss: 1.1058 - val_accuracy: 0.6633
Epoch 30/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0569 - accuracy: 0.9835 - val_loss: 1.0822 - val_accuracy: 0.6744
Epoch 31/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.0561 - accuracy: 0.9837 - val_loss: 1.0540 - val_accuracy: 0.6762
Epoch 32/50
4394/4394 [==============================] - 112s 25ms/step - loss: 0.0562 - accuracy: 0.9834 - val_loss: 1.0110 - val_accuracy: 0.6891
Epoch 33/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0579 - accuracy: 0.9835 - val_loss: 1.0195 - val_accuracy: 0.6906
Epoch 34/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0563 - accuracy: 0.9838 - val_loss: 1.0100 - val_accuracy: 0.6921
Epoch 35/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0561 - accuracy: 0.9837 - val_loss: 0.9790 - val_accuracy: 0.7016
Epoch 36/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.0548 - accuracy: 0.9842 - val_loss: 1.0043 - val_accuracy: 0.6978
Epoch 37/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0522 - accuracy: 0.9850 - val_loss: 0.9928 - val_accuracy: 0.6990
Epoch 38/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0539 - accuracy: 0.9844 - val_loss: 0.9726 - val_accuracy: 0.7048
Epoch 39/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0516 - accuracy: 0.9847 - val_loss: 0.9787 - val_accuracy: 0.7049
Epoch 40/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0518 - accuracy: 0.9846 - val_loss: 0.9851 - val_accuracy: 0.7021
Epoch 41/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0543 - accuracy: 0.9839 - val_loss: 0.9696 - val_accuracy: 0.7056
Epoch 42/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0524 - accuracy: 0.9843 - val_loss: 0.9467 - val_accuracy: 0.7139
Epoch 43/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0526 - accuracy: 0.9848 - val_loss: 0.9239 - val_accuracy: 0.7168
Epoch 44/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0515 - accuracy: 0.9845 - val_loss: 0.9187 - val_accuracy: 0.7186
Epoch 45/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0514 - accuracy: 0.9848 - val_loss: 0.9140 - val_accuracy: 0.7233
Epoch 46/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0511 - accuracy: 0.9850 - val_loss: 0.8816 - val_accuracy: 0.7314
Epoch 47/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0501 - accuracy: 0.9852 - val_loss: 0.9335 - val_accuracy: 0.7229
Epoch 48/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0507 - accuracy: 0.9850 - val_loss: 0.9094 - val_accuracy: 0.7263
Epoch 49/50
4394/4394 [==============================] - 114s 26ms/step - loss: 0.0514 - accuracy: 0.9848 - val_loss: 0.9176 - val_accuracy: 0.7240
Epoch 50/50
4394/4394 [==============================] - 113s 26ms/step - loss: 0.0520 - accuracy: 0.9845 - val_loss: 0.8921 - val_accuracy: 0.7307
```

[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()
```



[12]:
```
model.save('plant_disease_MobileNetV2(input256)(noaugment).h5')
```

[ ]: