[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, [224, 224])
    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 = (224, 224)

# 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 = MobileNetV2(input_shape=(224, 224, 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=0.00002),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

```
model.summary()
```

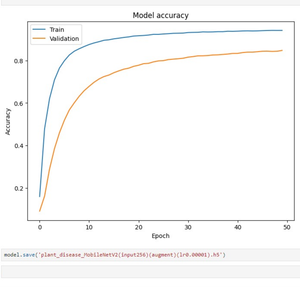
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 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=5, 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 [==============================] - 96s 21ms/step - loss: 3.7273 - accuracy: 0.1600 - val_loss: 3.3745 - val_accuracy: 0.0912 - lr: 2.0000e-05
Epoch 2/50
4394/4394 [==============================] - 92s 21ms/step - loss: 1.9479 - accuracy: 0.4791 - val_loss: 3.0960 - val_accuracy: 0.1622 - lr: 2.0000e-05
Epoch 3/50
4394/4394 [==============================] - 93s 21ms/step - loss: 1.3930 - accuracy: 0.6207 - val_loss: 2.4205 - val_accuracy: 0.2893 - lr: 2.0000e-05
Epoch 4/50
4394/4394 [==============================] - 92s 21ms/step - loss: 1.0541 - accuracy: 0.7091 - val_loss: 1.9955 - val_accuracy: 0.3855 - lr: 2.0000e-05
Epoch 5/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.8537 - accuracy: 0.7644 - val_loss: 1.6840 - val_accuracy: 0.4596 - lr: 2.0000e-05
Epoch 6/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.7201 - accuracy: 0.7990 - val_loss: 1.4763 - val_accuracy: 0.5192 - lr: 2.0000e-05
Epoch 7/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.6316 - accuracy: 0.8261 - val_loss: 1.3125 - val_accuracy: 0.5676 - lr: 2.0000e-05
Epoch 8/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.5630 - accuracy: 0.8438 - val_loss: 1.2017 - val_accuracy: 0.6010 - lr: 2.0000e-05
Epoch 9/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.5132 - accuracy: 0.8595 - val_loss: 1.1017 - val_accuracy: 0.6320 - lr: 2.0000e-05
Epoch 10/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.4748 - accuracy: 0.8658 - val_loss: 1.0220 - val_accuracy: 0.6581 - lr: 2.0000e-05
Epoch 11/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.4430 - accuracy: 0.8753 - val_loss: 0.9615 - val_accuracy: 0.6785 - lr: 2.0000e-05
Epoch 12/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.4171 - accuracy: 0.8826 - val_loss: 0.9044 - val_accuracy: 0.6977 - lr: 2.0000e-05
Epoch 13/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3938 - accuracy: 0.8886 - val_loss: 0.8571 - val_accuracy: 0.7131 - lr: 2.0000e-05
Epoch 14/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3742 - accuracy: 0.8952 - val_loss: 0.8183 - val_accuracy: 0.7245 - lr: 2.0000e-05
Epoch 15/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3579 - accuracy: 0.8975 - val_loss: 0.7958 - val_accuracy: 0.7316 - lr: 2.0000e-05
Epoch 16/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3425 - accuracy: 0.9020 - val_loss: 0.7610 - val_accuracy: 0.7426 - lr: 2.0000e-05
Epoch 17/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3327 - accuracy: 0.9050 - val_loss: 0.7335 - val_accuracy: 0.7513 - lr: 2.0000e-05
Epoch 18/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3195 - accuracy: 0.9085 - val_loss: 0.7130 - val_accuracy: 0.7592 - lr: 2.0000e-05
Epoch 19/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3090 - accuracy: 0.9111 - val_loss: 0.6966 - val_accuracy: 0.7639 - lr: 2.0000e-05
Epoch 20/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2983 - accuracy: 0.9151 - val_loss: 0.6733 - val_accuracy: 0.7726 - lr: 2.0000e-05
Epoch 21/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2920 - accuracy: 0.9166 - val_loss: 0.6588 - val_accuracy: 0.7778 - lr: 2.0000e-05
Epoch 22/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2847 - accuracy: 0.9178 - val_loss: 0.6386 - val_accuracy: 0.7851 - lr: 2.0000e-05
Epoch 23/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2770 - accuracy: 0.9198 - val_loss: 0.6317 - val_accuracy: 0.7867 - lr: 2.0000e-05
Epoch 24/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2703 - accuracy: 0.9232 - val_loss: 0.6105 - val_accuracy: 0.7936 - lr: 2.0000e-05
Epoch 25/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2654 - accuracy: 0.9220 - val_loss: 0.5961 - val_accuracy: 0.7981 - lr: 2.0000e-05
Epoch 26/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2610 - accuracy: 0.9249 - val_loss: 0.5876 - val_accuracy: 0.7996 - lr: 2.0000e-05
Epoch 27/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2548 - accuracy: 0.9262 - val_loss: 0.5749 - val_accuracy: 0.8043 - lr: 2.0000e-05
Epoch 28/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2496 - accuracy: 0.9280 - val_loss: 0.5664 - val_accuracy: 0.8065 - lr: 2.0000e-05
Epoch 29/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2473 - accuracy: 0.9284 - val_loss: 0.5607 - val_accuracy: 0.8084 - lr: 2.0000e-05
Epoch 30/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2421 - accuracy: 0.9291 - val_loss: 0.5536 - val_accuracy: 0.8107 - lr: 2.0000e-05
Epoch 31/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2377 - accuracy: 0.9317 - val_loss: 0.5412 - val_accuracy: 0.8158 - lr: 2.0000e-05
Epoch 32/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2341 - accuracy: 0.9323 - val_loss: 0.5309 - val_accuracy: 0.8185 - lr: 2.0000e-05
Epoch 33/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2305 - accuracy: 0.9328 - val_loss: 0.5209 - val_accuracy: 0.8219 - lr: 2.0000e-05
Epoch 34/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2278 - accuracy: 0.9346 - val_loss: 0.5194 - val_accuracy: 0.8223 - lr: 2.0000e-05
Epoch 35/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2230 - accuracy: 0.9342 - val_loss: 0.5125 - val_accuracy: 0.8234 - lr: 2.0000e-05
Epoch 36/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2229 - accuracy: 0.9346 - val_loss: 0.5055 - val_accuracy: 0.8260 - lr: 2.0000e-05
Epoch 37/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2186 - accuracy: 0.9357 - val_loss: 0.5022 - val_accuracy: 0.8267 - lr: 2.0000e-05
Epoch 38/50
4394/4394 [==============================] - 94s 21ms/step - loss: 0.2184 - accuracy: 0.9357 - val_loss: 0.4970 - val_accuracy: 0.8286 - lr: 2.0000e-05
Epoch 39/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.2148 - accuracy: 0.9377 - val_loss: 0.4900 - val_accuracy: 0.8305 - lr: 2.0000e-05
Epoch 40/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2127 - accuracy: 0.9373 - val_loss: 0.4835 - val_accuracy: 0.8333 - lr: 2.0000e-05
Epoch 41/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2116 - accuracy: 0.9380 - val_loss: 0.4818 - val_accuracy: 0.8334 - lr: 2.0000e-05
Epoch 42/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2099 - accuracy: 0.9391 - val_loss: 0.4786 - val_accuracy: 0.8373 - lr: 2.0000e-05
Epoch 43/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2063 - accuracy: 0.9396 - val_loss: 0.4664 - val_accuracy: 0.8393 - lr: 2.0000e-05
Epoch 44/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2060 - accuracy: 0.9390 - val_loss: 0.4623 - val_accuracy: 0.8395 - lr: 2.0000e-05
Epoch 45/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2037 - accuracy: 0.9395 - val_loss: 0.4584 - val_accuracy: 0.8413 - lr: 2.0000e-05
Epoch 46/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2004 - accuracy: 0.9405 - val_loss: 0.4542 - val_accuracy: 0.8436 - lr: 2.0000e-05
Epoch 47/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.2008 - accuracy: 0.9413 - val_loss: 0.4518 - val_accuracy: 0.8439 - lr: 2.0000e-05
Epoch 48/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.1980 - accuracy: 0.9417 - val_loss: 0.4551 - val_accuracy: 0.8428 - lr: 2.0000e-05
Epoch 49/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.1961 - accuracy: 0.9416 - val_loss: 0.4502 - val_accuracy: 0.8436 - lr: 2.0000e-05
Epoch 50/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.1959 - accuracy: 0.9416 - val_loss: 0.4416 - val_accuracy: 0.8475 - lr: 2.0000e-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)(augment)(lr0.00001).h5')
```

[ ]: