[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=0.000009),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.summary()
```

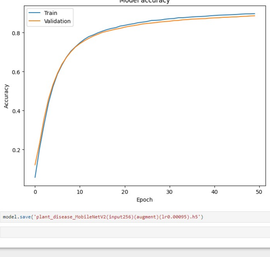
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 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 [==============================] - 94s 21ms/step - loss: 4.2589 - accuracy: 0.0576 - val_loss: 3.2624 - val_accuracy: 0.1218
Epoch 2/50
4394/4394 [==============================] - 91s 21ms/step - loss: 3.0027 - accuracy: 0.2047 - val_loss: 2.7497 - val_accuracy: 0.2352
Epoch 3/50
4394/4394 [==============================] - 91s 21ms/step - loss: 2.4451 - accuracy: 0.3274 - val_loss: 2.2642 - val_accuracy: 0.3574
Epoch 4/50
4394/4394 [==============================] - 91s 21ms/step - loss: 2.0023 - accuracy: 0.4387 - val_loss: 1.9000 - val_accuracy: 0.4591
Epoch 5/50
4394/4394 [==============================] - 91s 21ms/step - loss: 1.6974 - accuracy: 0.5228 - val_loss: 1.6215 - val_accuracy: 0.5333
Epoch 6/50
4394/4394 [==============================] - 91s 21ms/step - loss: 1.4603 - accuracy: 0.5872 - val_loss: 1.4288 - val_accuracy: 0.5915
Epoch 7/50
4394/4394 [==============================] - 91s 21ms/step - loss: 1.2928 - accuracy: 0.6339 - val_loss: 1.2821 - val_accuracy: 0.6305
Epoch 8/50
4394/4394 [==============================] - 91s 21ms/step - loss: 1.1563 - accuracy: 0.6744 - val_loss: 1.1603 - val_accuracy: 0.6731
Epoch 9/50
4394/4394 [==============================] - 92s 21ms/step - loss: 1.0493 - accuracy: 0.7051 - val_loss: 1.0432 - val_accuracy: 0.7020
Epoch 10/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.9650 - accuracy: 0.7274 - val_loss: 0.9594 - val_accuracy: 0.7259
Epoch 11/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.8878 - accuracy: 0.7480 - val_loss: 0.8969 - val_accuracy: 0.7436
Epoch 12/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.8315 - accuracy: 0.7657 - val_loss: 0.8410 - val_accuracy: 0.7584
Epoch 13/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.7801 - accuracy: 0.7798 - val_loss: 0.7944 - val_accuracy: 0.7712
Epoch 14/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.7419 - accuracy: 0.7884 - val_loss: 0.7527 - val_accuracy: 0.7827
Epoch 15/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.7072 - accuracy: 0.7989 - val_loss: 0.7182 - val_accuracy: 0.7918
Epoch 16/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.6736 - accuracy: 0.8075 - val_loss: 0.6882 - val_accuracy: 0.8013
Epoch 17/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.6451 - accuracy: 0.8152 - val_loss: 0.6623 - val_accuracy: 0.8070
Epoch 18/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.6223 - accuracy: 0.8214 - val_loss: 0.6365 - val_accuracy: 0.8127
Epoch 19/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5987 - accuracy: 0.8258 - val_loss: 0.6145 - val_accuracy: 0.8181
Epoch 20/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5756 - accuracy: 0.8339 - val_loss: 0.5961 - val_accuracy: 0.8223
Epoch 21/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5575 - accuracy: 0.8375 - val_loss: 0.5783 - val_accuracy: 0.8282
Epoch 22/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5402 - accuracy: 0.8424 - val_loss: 0.5612 - val_accuracy: 0.8328
Epoch 23/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5302 - accuracy: 0.8459 - val_loss: 0.5458 - val_accuracy: 0.8362
Epoch 24/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5118 - accuracy: 0.8512 - val_loss: 0.5334 - val_accuracy: 0.8401
Epoch 25/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.5014 - accuracy: 0.8540 - val_loss: 0.5182 - val_accuracy: 0.8449
Epoch 26/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.4895 - accuracy: 0.8576 - val_loss: 0.5080 - val_accuracy: 0.8471
Epoch 27/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.4753 - accuracy: 0.8622 - val_loss: 0.4983 - val_accuracy: 0.8495
Epoch 28/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4662 - accuracy: 0.8633 - val_loss: 0.4900 - val_accuracy: 0.8520
Epoch 29/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.4587 - accuracy: 0.8650 - val_loss: 0.4783 - val_accuracy: 0.8552
Epoch 30/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4473 - accuracy: 0.8690 - val_loss: 0.4702 - val_accuracy: 0.8571
Epoch 31/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4413 - accuracy: 0.8714 - val_loss: 0.4621 - val_accuracy: 0.8586
Epoch 32/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4318 - accuracy: 0.8727 - val_loss: 0.4511 - val_accuracy: 0.8621
Epoch 33/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4199 - accuracy: 0.8767 - val_loss: 0.4455 - val_accuracy: 0.8627
Epoch 34/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4163 - accuracy: 0.8769 - val_loss: 0.4352 - val_accuracy: 0.8662
Epoch 35/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4082 - accuracy: 0.8787 - val_loss: 0.4309 - val_accuracy: 0.8672
Epoch 36/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.4034 - accuracy: 0.8806 - val_loss: 0.4234 - val_accuracy: 0.8696
Epoch 37/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3961 - accuracy: 0.8822 - val_loss: 0.4187 - val_accuracy: 0.8703
Epoch 38/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3918 - accuracy: 0.8830 - val_loss: 0.4129 - val_accuracy: 0.8716
Epoch 39/50
4394/4394 [==============================] - 93s 21ms/step - loss: 0.3876 - accuracy: 0.8841 - val_loss: 0.4099 - val_accuracy: 0.8721
Epoch 40/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3814 - accuracy: 0.8864 - val_loss: 0.4029 - val_accuracy: 0.8747
Epoch 41/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3778 - accuracy: 0.8881 - val_loss: 0.3983 - val_accuracy: 0.8759
Epoch 42/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.3694 - accuracy: 0.8896 - val_loss: 0.3936 - val_accuracy: 0.8765
Epoch 43/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.3660 - accuracy: 0.8905 - val_loss: 0.3881 - val_accuracy: 0.8788
Epoch 44/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.3646 - accuracy: 0.8914 - val_loss: 0.3858 - val_accuracy: 0.8791
Epoch 45/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.3575 - accuracy: 0.8924 - val_loss: 0.3814 - val_accuracy: 0.8807
Epoch 46/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3546 - accuracy: 0.8934 - val_loss: 0.3777 - val_accuracy: 0.8811
Epoch 47/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3518 - accuracy: 0.8953 - val_loss: 0.3718 - val_accuracy: 0.8832
Epoch 48/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3450 - accuracy: 0.8965 - val_loss: 0.3699 - val_accuracy: 0.8839
Epoch 49/50
4394/4394 [==============================] - 91s 21ms/step - loss: 0.3440 - accuracy: 0.8966 - val_loss: 0.3643 - val_accuracy: 0.8857
Epoch 50/50
4394/4394 [==============================] - 92s 21ms/step - loss: 0.3406 - accuracy: 0.8975 - val_loss: 0.3597 - val_accuracy: 0.8865
```

[9]
```
# # Evaluate the model
# loss, accuracy = model.evaluate(val_ds)
# print('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)(lr0.00009).h5')
```

[ ]