APPENDIX

IMPORT LIBRARIES AND PACKAGES

```
1. import numpy as np
2. import pickle
3. import cv2
4. import tensorflow
5. from os import listdir
6. from sklearn.preprocessing import LabelBinarizer
7. from keras.applications import MobileNet
8. from keras.models import Model
9. from tensorflow.keras.layers import GlobalAveragePooling2D
10. from keras.layers.core import Dropout, Dense
11. from keras.preprocessing.image import ImageDataGenerator
12. from tensorflow.keras.optimizers.legacy import Adam
13. from keras.preprocessing import image
14. from tensorflow.keras.utils import img_to_array
15. import matplotlib.pyplot as plt
16. from sklearn.model_selection import KFold
17. from sklearn.model_selection import train_test_split
18. from tensorflow.keras.callbacks import EarlyStopping
```

SET THE PARAMETER

```
19. EPOCHS = 25
20. INIT_LR = 0.001
21. BS = 16
22. default_image_size = tuple((224, 224))
23. image_size = 0
24. directory_root = 'D:\\KULIAH\\SKRIPSI\\Dataset'
25. width=224
26. height=224
27. depth=3
28. num_folds = 10
```

CONVERT IMAGE TO ARRAY

```
29. def convert_image_to_array(image_dir):
30. try:
31. image = cv2.imread(image_dir)
32. if image is not None :
33. image = cv2.resize(image, default_image_size)
34. return img_to_array(image)
35. else :
36. return np.array([])
37. except Exception as e:
38. print(f"Error : {e}")
39. return None
```

LOAD THE DATASET

```
image_list, label_list = [], []
try:
print("[INFO] Loading images ...")
root_dir = listdir(directory_root)
for directory in root_dir :
# remove .DS_Store from list
if directory == ".DS_Store" :
root_dir.remove(directory)
for plant_folder in root_dir :
plant_disease_folder_list = listdir(f"{directory_root}/{plant_folder}")
for disease_folder in plant_disease_folder_list :
# remove .DS_Store from list
if disease_folder == ".DS_Store" :
plant_disease_folder_list.remove(disease_folder)
for plant_disease_folder in plant_disease_folder_list:
print(f"[INFO] Processing {plant_disease_folder} ...")
plant_disease_image_list = listdir(f"{directory_root}/{plant_folder}/{plant_disease_folder}/")
for single_plant_disease_image in plant_disease_image_list :
if single_plant_disease_image == ".DS_Store" :
plant_disease_image_list.remove(single_plant_disease_image)
for image in plant_disease_image_list[:200]:
image_directory = f"{directory_root}/{plant_folder}/{plant_disease_folder}/{image}"
If image_directory.endswith(".jpg") == True or image_directory.endswith(".JPG") == True:
image_list.append(convert_image_to_array(image_directory))
label_list.append(plant_disease_folder)
print("[INFO] Image loading completed")
except Exception as e:
print(f"Error : {e}")
```

CONVERT IMAGE LABEL TO BINARY LEVELS

```
image_size = len(image_list)
label_binarizer = LabelBinarizer()
image_labels = label_binarizer.fit_transform(label_list)
pickle.dump(label_binarizer,open('label_transform.pkl', 'wb'))
n_classes = len(label_binarizer.classes_)
print(label_binarizer.classes_)
```

PRE-PROCESS INPUT DATA

```
np_image_list = np.array(image_list, dtype=np.float32) / 225.0
```

TRAIN TEST SPLIT DATA

```
print("[INFO] Splitting data to train, test")
X_train, X_test, Y_train, Y_test = train_test_split(np_image_list, image_labels, test_size=0.2, random_state = 42)
```

AUGMENTATION DATA

```
aug = ImageDataGenerator(
```

```
78. rotation_range=25,
79. width_shift_range=0.1,
80. height_shift_range=0.1,
81. shear_range=0.2,
82. zoom_range=0.2,
83. horizontal_flip=True,
84. fill_mode="nearest")
```

DEFINE PER-FOLD SCORE CONTAINERS

```
85. acc_per_fold = []
86. loss_per_fold = []
```

K-FOLD CROSS VALIDATION

```
87. kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)
88. fold_no = 1
89. for train, val in kfold.split(X_train, Y_train):
90. print(f'Training for fold {fold_no} ...')
91. X_train_cv, X_val_cv = X_train[train], X_train[val]
92. Y_train_cv, Y_val_cv = Y_train[train], Y_train[val]
```

CREATE THE BASE MODEL OF MOBILENET

```
93. model = MobileNet(weights='imagenet', include_top=False,
    input_shape=(height, width, depth))
```

FREEZE THE LAYERS

```
94. for layer in model.layers:
95. layer.trainable = False
```

ADD NEW LAYERS

```
96. x = model.output
97. x = GlobalAveragePooling2D()(x)
98. x = Dense(1024, activation='relu')(x)
99. x = Dropout(0.5)(x)
100.predictions = Dense(n_classes, activation='softmax')(x)
```

CREATE THE FINAL MODEL

```
101.model = Model(inputs=model.input, outputs=predictions)
```

MODEL COMPILE

```
102.opt = Adam(learning_rate=INIT_LR, decay=INIT_LR / EPOCHS)
103.# distribution
104.model.compile(loss="categorical_crossentropy",
    optimizer=opt,metrics=["accuracy"])
105.early_stopping = EarlyStopping(monitor='val_accuracy', patience=10)
```

TRAIN THE MODEL

```
history = model.fit(
aug.flow(X_train_cv, Y_train_cv, batch_size=BS),
validation_data=(X_val_cv, Y_val_cv),
steps_per_epoch=len(X_train_cv) // BS,
epochs=EPOCHS, verbose=1,
callbacks=[early_stopping])
```

GENERATE GENERALIZATION METRICS

```
scores = model.evaluate(X_val_cv, Y_val_cv, verbose=0)
print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1]*100}%')
print('------------------------------------------------------------------------')
acc_per_fold.append(scores[1] * 100)
loss_per_fold.append(scores[0])
```

PROVIDE AVERAGE SCORE

```
print('------------------------------------------------------------------------')
print('Score per fold')
for i in range(0, len(acc_per_fold)):
  print('------------------------------------------------------------------------')
print(f'> Fold {i+1} - Loss: {loss_per_fold[i]} - Accuracy: {acc_per_fold[i]}%')
print('------------------------------------------------------------------------')
print('Average scores for all folds:')
print(f'> Accuracy: {np.mean(acc_per_fold)} (+- {np.std(acc_per_fold)})')
print(f'> Loss: {np.mean(loss_per_fold)}')
print('------------------------------------------------------------------------')
```

PLOT THE GRAPH

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training & Validation accuracy')
plt.legend()
plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training & Validation loss')
plt.legend()
```

```
143.plt.show()
```

EVALUATE THE MODEL

```
144.test_loss, test_acc = model.evaluate(X_test, Y_test, verbose=0)
145.print(f'Test Loss: {test_loss}')
146.print(f'Test Accuracy: {test_acc*100}%')
```

SAVE THE MODEL

```
147.print("[INFO] Saving model...")
148.save_path = './model.h5'
149.model.save(save_path)
```

LOAD THE MODEL

```
150.loaded_model= tensorflow.keras.models.load_model(save_path)
151.image_dir="D:\\KULIAH\\SKRIPSI\\Dataset\\PlantVillage\\Tomato_Late_bli
   ght\\0a3f65fc-ef1c-4aed-b235-46bae4e5c0e7___GHLB2 Leaf 9065.JPG"
152.im=convert_image_to_array(image_dir)
153.np_image_li = np.array(im, dtype=np.float32) / 225.0
154.npp_image = np.expand_dims(np_image_li, axis=0)
```

MODEL PREDICT

```
155.result=model.predict(npp_image)
156.print(result)
```

PROBABILITY RESULT

```
157.itemindex = np.where(result==np.max(result))
158.print("probability:"+str(np.max(result))+"\n"+label_binarizer.classes_
   [itemindex[1][0]])
```

CONFUSION MATRIX

```
159.results=model.predict(X_test)
160.predicted_classes = np.argmax(results, axis=1)
161.true_classes = np.argmax(Y_test, axis=1)
162.confusion = confusion_matrix(true_classes, predicted_classes)
```

CREATE HEATMAP

```
163.sn.heatmap(confusion, annot=True, cmap='Blues', fmt='d')
```

ADD X AND Y LABELS

```
164.plt.xlabel('Predicted')
165.plt.ylabel('True')
```

SHOW PLOT

```
166.plt.show()
```

CREATE HEATMAP

```
167.acc = accuracy_score(true_classes, predicted_classes)
168.print(f'Test Accuracy: {acc*100}%')
169.f1 = f1_score(true_classes, predicted_classes, average='weighted')
170.print(f'F1 score: {f1}')
171.precision = precision_score(true_classes, predicted_classes,
   average='weighted')
172.print(f'Precision: {precision}')
173.recall = recall_score(true_classes, predicted_classes,
   average='weighted')
174.print(f'Recall: {recall}')
```

PAPER NAME

TA-18.K1.0069.docx

WORD COUNT	CHARACTER COUNT
6853 Words	35673 Characters
PAGE COUNT	FILE SIZE
25 Pages	107.0KB
SUBMISSION DATE	REPORT DATE
Jan 29, 2023 10:44 PM GMT+7	Jan 29, 2023 10:44 PM GMT+7

17% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- 13% Internet database
- 11% Publications database
- Crossref database
- Crossref Posted Content database
- 10% Submitted Works database

Excluded from Similarity Report

- Bibliographic material
- Quoted material
- Cited material
- Small Matches (Less then 10 words)
- Manually excluded text blocks