APPENDIX

Jika Anda punya lampiran dari project, silahkan dilampirkan di bagian ini. Yang wajib Anda lampirkan adalah kode program (coding) lengkap dan diberikan keterangan terlebih dahulu pada bagian atas dari coding tersebut, koding ditulis dengan format font yang berbeda. Contoh:

IMAGE LABELING

```
#https://pythonprogramming.net/loading-custom-data-deep-learning-
python-tensorflow-keras/

#https://machinelearningmastery.com/save-load-machine-learning-
models-python-scikit-learn/

try:

    import tensorflow as tf
    import cv2
    import os
    import pickle
    import numpy as np
    print("Library Loaded Successfully ..........")
except:
    print("Library not Found ! ")

#load image and resize image to size 50

class MasterImage(object):

    def __init__(self,PATH='', IMAGE_SIZE = 50):
        self.PATH = PATH
        self.IMAGE_SIZE = IMAGE_SIZE

        self.image_data = []
        self.x_data = []
        self.y_data = []
        self.CATEGORIES = []

        # This will get List of categories
        self.list_categories = []

    def get_categories(self):
        for path in os.listdir(self.PATH):
            if '.DS_Store' in path:
                pass
            else:
                self.list_categories.append(path)
        print("Found Categories ",self.list_categories,'\n')
        return self.list_categories

    def Process_Image(self):
        try:
            """
            Return Numpy array of image
```

```
            :return: X_Data, Y_Data
            """
            self.CATEGORIES = self.get_categories()
            for categories in self.CATEGORIES:
                      train_folder_path  =  os.path.join(self.PATH,
categories)
                class_index = self.CATEGORIES.index(categories)

                for img in os.listdir(train_folder_path):
                        new_path  =  os.path.join(train_folder_path,
img)
                    try:
                                                        image_data_temp  =
cv2.imread(new_path,cv2.IMREAD_GRAYSCALE)
                                                      image_temp_resize  =
cv2.resize(image_data_temp,(self.IMAGE_SIZE,self.IMAGE_SIZE))
                         self.image_data.append([image_temp_resize,
class_index])
                    except:
                        pass

            data = np.asanyarray(self.image_data) #data convert ke
array


            for x in data:
                self.x_data.append(x[0])          # Get the X_Data
                self.y_data.append(x[1])          # get the label

               X_Data  =  np.asarray(self.x_data)  /  (255.0)        #
Normalize Data
            Y_Data = np.asarray(self.y_data)



                    X_Data  =  X_Data.reshape(-1,  self.IMAGE_SIZE,
self.IMAGE_SIZE, 1)

            return X_Data, Y_Data
        except:
            print("Failed to run Function Process Image ")

    def pickle_image(self):

        """
        :return: None Creates a Pickle Object of DataSet
        """
        # Call the Function and Get the Data
        X_Data,Y_Data = self.Process_Image()

        # Write the Entire Data into a Pickle File
        pickle_out = open('X_Data','wb')
        pickle.dump(X_Data, pickle_out)
        pickle_out.close()

        # Write the Y Label Data
        pickle_out = open('Y_Data', 'wb')
        pickle.dump(Y_Data, pickle_out)
        pickle_out.close()
```

```
        print("Pickled Image Successfully ")
        return X_Data,Y_Data

    def load_dataset(self):

        try:
            # Read the Data from Pickle Object
            X_Temp = open('X_Data','rb')
            X_Data = pickle.load(X_Temp)

            Y_Temp = open('Y_Data','rb')
            Y_Data = pickle.load(Y_Temp)

            print('Reading Dataset from PIckle Object')

            return X_Data,Y_Data

        except:
            print('Could not Found Pickle File ')
            print('Loading File and Dataset  ..........')

            X_Data,Y_Data = self.pickle_image()
            return X_Data,Y_Data


if __name__ == "__main__":
    path = '/home/kevinew/Documents/East_project/IIIT5K'
    a = MasterImage(PATH=path,
                    IMAGE_SIZE=80)

    X_Data,Y_Data = a.load_dataset()
    print(X_Data.shape)



#
```

IMAGE EXCTRACTION AND TRAIN MODEL

```
#https://pythonprogramming.net/loading-custom-data-deep-learning-
python-tensorflow-keras/?completed=/introduction-deep-learning-
python-tensorflow-keras/


#https://www.tensorflow.org/tutorials/images/cnn


#https://machinelearningmastery.com/convolutional-layers-for-deep-
learning-neural-networks/




from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Activation,
Flatten
```

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D
# more info on callbakcs: https://keras.io/callbacks/ model saver
is cool too.
from tensorflow.keras.callbacks import TensorBoard
import pickle
import time
#load data yg sudah terlabel
pickle_in = open("X_Data","rb")
X = pickle.load(pickle_in)

pickle_in = open("Y_Data","rb")
y = pickle.load(pickle_in)
#di normalisasi lagi
X = X/255.0

dense_layers = [0, 1, 2]
layer_sizes = [32, 64, 128]
conv_layers = [1, 2, 3]
#memasukan data X utk dilakukan image Ekstraktor
for dense_layer in dense_layers:
    for layer_size in layer_sizes:
        for conv_layer in conv_layers:
                                NAME  =  "{}-conv-{}-nodes-{}-dense-
{}".format(conv_layer, layer_size, dense_layer, int(time.time()))
            print(NAME)

            model = Sequential()

                            model.add(Conv2D(layer_size,  (3,  3),
input_shape=X.shape[1:]))
            model.add(Activation('relu'))
            model.add(MaxPooling2D(pool_size=(2, 2)))

            for l in range(conv_layer-1):
                model.add(Conv2D(layer_size, (3, 3)))
                model.add(Activation('relu'))
                model.add(MaxPooling2D(pool_size=(2, 2)))

            model.add(Flatten())
```

```
            for _ in range(dense_layer):
                model.add(Dense(layer_size))
                model.add(Activation('relu'))

            model.add(Dense(1))
            model.add(Activation('sigmoid'))

                                                tensorboard    =
TensorBoard(log_dir="logs/{}".format(NAME))

            model.compile(loss='binary_crossentropy',
                          optimizer='adam',
                          metrics=['accuracy'],
                          )

            model.fit(X, y,
                      batch_size=32,
                      epochs=10,
                      validation_split=0.3,
                      callbacks=[tensorboard])

model.save('East_text4.model')
```

IMAGE TEXT RECOGNATION

```
#https://machinelearningmastery.com/save-load-keras-deep-learning-
models/

#https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-
east-text-detector/


from imutils.object_detection import non_max_suppression
import numpy as np
import time
import cv2
import pytesseract
```

```
net = cv2.dnn.readNet("East_text5.pb")


def text_detector(image):
      orig = image
      (H, W) = image.shape[:2]

      (newW, newH) = (320, 320)
      rW = W / float(newW)
      rH = H / float(newH)

      image = cv2.resize(image, (newW, newH))
      (H, W) = image.shape[:2]

      layerNames = [
            "feature_fusion/Conv_7/Sigmoid",
            "feature_fusion/concat_3"]


      blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
            (123.68, 116.78, 103.94), swapRB=True, crop=False)

      net.setInput(blob)
      (scores, geometry) = net.forward(layerNames)

      (numRows, numCols) = scores.shape[2:4]
      rects = []
      confidences = []

      for y in range(0, numRows):

            scoresData = scores[0, 0, y]
            xData0 = geometry[0, 0, y]
            xData1 = geometry[0, 1, y]
            xData2 = geometry[0, 2, y]
            xData3 = geometry[0, 3, y]
            anglesData = geometry[0, 4, y]
```

```
            # loop over the number of columns
            for x in range(0, numCols):
                  # if our score does not have sufficient
probability, ignore it
                  if scoresData[x] < 0.5:
                        continue

                  # compute the offset factor as our resulting
feature maps will
                  # be 4x smaller than the input image
                  (offsetX, offsetY) = (x * 4.0, y * 4.0)

                  # extract the rotation angle for the prediction
and then
                  # compute the sin and cosine
                  angle = anglesData[x]
                  cos = np.cos(angle)
                  sin = np.sin(angle)

                  # use the geometry volume to derive the width
and height of
                  # the bounding box
                  h = xData0[x] + xData2[x]
                  w = xData1[x] + xData3[x]

                  # compute both the starting and ending (x, y)-
coordinates for
                  # the text prediction bounding box
                  endX = int(offsetX + (cos * xData1[x]) + (sin *
xData2[x]))
                  endY = int(offsetY - (sin * xData1[x]) + (cos *
xData2[x]))
                  startX = int(endX - w)
                  startY = int(endY - h)

                  # add the bounding box coordinates and
probability score to
                  # our respective lists
                  rects.append((startX, startY, endX, endY))
                  confidences.append(scoresData[x])
```

```
    boxes = non_max_suppression(np.array(rects), probs=confidences)

    for (startX, startY, endX, endY) in boxes:

        startX = int(startX * rW)
        startY = int(startY * rH)
        endX = int(endX * rW)
        endY = int(endY * rH)
        boundary = 2

        text = orig[startY-boundary:endY+boundary, startX - boundary:endX + boundary]
        text = cv2.cvtColor(text.astype(np.uint8), cv2.COLOR_BGR2GRAY)
        textRecongized = pytesseract.image_to_string(text)
        print(textRecongized)
        cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 255, 0), 3)
        orig = cv2.putText(orig, textRecongized, (endX,endY+5), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255), 2, cv2.LINE_AA)
    return orig

#unnamed = cv2.imread('/home/kevinew/Documents/East_project/panin.jpeg')
#image1 = cv2.imread('/home/kevinew/Documents/East_project/matahri.jpeg')
#image2 = cv2.imread('/home/kevinew/Documents/East_project/tes.jpeg')
#image3 = cv2.imread('/home/kevinew/Documents/East_project/rf.png')
unnamed = cv2.imread('/home/kevinew/Documents/East_project/paragon.jpg')
#image5 = cv2.imread('/home/kevinew/Documents/East_project/miele.jpg')
array = [unnamed]#,image1,image2,image3,image4,image5]

for i in range(0,2):
    for img in array:
        image0 = cv2.resize(img, (640,320), interpolation = cv2.INTER_AREA)
```

```
            orig = cv2.resize(img, (640,320), interpolation =
cv2.INTER_AREA)
            orig_gray = cv2.cvtColor(orig, cv2.COLOR_BGR2GRAY)
            textDetected = text_detector(imageO)
            textDetected_gray = cv2.cvtColor(textDetected,
cv2.COLOR_BGR2GRAY)
            cv2.imshow("Orig Image",orig_gray)
            cv2.imshow("Text Detection", textDetected_gray)
            time.sleep(2)
            k = cv2.waitKey(30)
            if k == 27:
                  break
cv2.destroyAllWindows()
```

EAST TEXT DETECTION

```
from PIL import Image
import pytesseract
import argparse
import cv2
import os

net = cv2.dnn.readNet("frozen_east_text_detection.pb")

ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
      help="path to input image to be OCR'd")
ap.add_argument("-p", "--preprocess", type=str, default="thresh",
      help="type of preprocessing to be done")
args = vars(ap.parse_args())

image = cv2.imread(args["image"])
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# check to see if we should apply thresholding to preprocess the
# image
if args["preprocess"] == "thresh":
      gray = cv2.threshold(gray, 0, 255,
            cv2.THRESH_BINARY | cv2.THRESH_OTSU)[1]
# make a check to see if median blurring should be done to remove
# noise
```

```
elif args["preprocess"] == "blur":
      gray = cv2.medianBlur(gray, 3)
filename = "{}.png".format(os.getpid())
cv2.imwrite(filename, gray)
text = pytesseract.image_to_string(Image.open(filename))
os.remove(filename)
print(text)
# show the output images
cv2.imshow("Image", image)
cv2.imshow("Output", gray)
cv2.waitKey(0)
```

PLAGIARISM CHECK.ORG

1.24% PLAGIARISM APPROXIMATELY | **8.98%** IN QUOTES

Report #12289661

CHAPTER 1 Introduction 1.1 Background In this era that is increasingly advanced and rapidly developing in terms of development and information technology, we are increasingly confused by new places that we have never visited, or even passed by,but we never know what place or building it really is. As in big cities, many buildings have a specific purpose. Buildings have various kinds such as public buildings, public buildings in the form of places to eat, entertainment places, places of business. Thus, the place has a unique name to attract attention and boost popularity among the people, but they do not know exactly what the place is. So the community tries to find information from the name of the place through social media such as Instagram, Facebook, Twitter, but sometimes these places do not create accounts on social media, making searches more difficult, then another way is to ask other people about the information about the building. With the development of technology and information systems, this can be overcome by using image processing which will find out information from that place. The problem in this case is to find a place description of a building that has a unique name. East (Efficient and Accurate Scene Text) is a text detector in natural scenarios in open CV is a deep learning model, based on a novel architecture and training pattern as its main purpose as