LAMPIRAN

Lampiran 1: source code register_face.py

```
import numpy as np
import cv2
import mysql.connector
import datetime
import time

face_id = 0

# For each person, enter one numeric face id
type = input('\n enter type, 1 for real and 0 for fake. End press
<return> ==>  ')
user_name = input('\n enter user name end press <return> ==>  ')
file_name = 'fake-' + user_name

if type == '1':
    face_id = input('\n enter user id end press <return> ==>  ')
    file_name = 'real-' + user_name

filename = 'videos/' + file_name + '.avi'

# set start time
start_time = time.time()

# make connector
mydb = mysql.connector.connect(
    host="localhost",
    user="admin",
    password="admin",
    database="face_recognition"
)

cap = cv2.VideoCapture(0)
# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')
out = cv2.VideoWriter(filename, fourcc, 20, (640, 480))

while (cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
        out.write(frame)
        cv2.imshow('frame', frame)
        # wait q for exit
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
        # set duration
        end_time = time.time()
        elapsed = end_time - start_time
        if elapsed > 20:
            break
    else:
        break

# Release everything if job is finished
cap.release()
out.release()
cv2.destroyAllWindows()

# insert data to table
if type == '1':
    mycursor = mydb.cursor()
    sql = "INSERT INTO users (id, name, allowed, created_at,
updated_at) VALUES (%s, %s, %s, %s, %s)"
    val = (face_id, user_name, 1, datetime.datetime.now(),
datetime.datetime.now())

    mycursor.execute(sql, val)
    mydb.commit()

    print(mycursor.rowcount, "record inserted.")
# end insert data to table
```

Lampiran 2: source code gather_examples.py

```
# USAGE
# python gather_examples.py --input videos/real.mov --output
dataset/real --detector face_detector --skip 1
# python gather_examples.py --input videos/fake.mp4 --output
dataset/fake --detector face_detector --skip 1

# TODO:
#  0. give input to user (id and name)
#  1. connect to database
#  2. add authorize user to database
#  3. add videostream and save video to videos directory
#


# import the necessary packages
import numpy as np
import argparse
import cv2
import os

cascade =
cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_defau
lt.xml')

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--input", type=str, required=True,
   help="path to input video")
ap.add_argument("-o", "--output", type=str, required=True,
   help="path to output directory of cropped faces")
ap.add_argument("-d", "--detector", type=str, required=True,
   help="path to OpenCV's deep learning face detector")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
   help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip", type=int, default=16,
   help="# of frames to skip before applying face detection")
ap.add_argument("-r", "--recognize", type=int, default=16,
   help="# reconize to set as human and name")
args = vars(ap.parse_args())

if args["recognize"] == 1:
   face_id = input('\n enter user id end press <return> ==>  ')

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"],
"deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
   "res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# open a pointer to the video file stream and initialize the total
# number of frames read and saved thus far
vs = cv2.VideoCapture(args["input"])
read = 0
saved = 0

# loop over frames from the video file stream
while True:
   # grab the frame from the file
   (grabbed, frame) = vs.read()

   # if the frame was not grabbed, then we have reached the end
   # of the stream
   if not grabbed:
      break

   # increment the total number of frames read thus far
   read += 1

   # check to see if we should process this frame
   if read % args["skip"] != 0:
      continue

   # grab the frame dimensions and construct a blob from the frame
   (h, w) = frame.shape[:2]
   blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
1.0,
      (300, 300), (104.0, 177.0, 123.0))

   # pass the blob through the network and obtain the detections
and
   # predictions
   net.setInput(blob)
   detections = net.forward()
   # ensure at least one face was found
   if len(detections) > 0:
      # we're making the assumption that each image has only ONE
      # face, so find the bounding box with the largest
probability
      i = np.argmax(detections[0, 0, :, 2])
      confidence = detections[0, 0, i, 2]

      # ensure that the detection with the largest probability
also
      # means our minimum probability test (thus helping filter
out
      # weak detections)
      if confidence > args["confidence"]:
         # compute the (x, y)-coordinates of the bounding box for
         # the face and extract the face ROI
         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
         (startX, startY, endX, endY) = box.astype("int")
         face = frame[startY:endY, startX:endX]

         # write the frame to disk
         p = os.path.sep.join([args["output"],
            "{}.png".format(saved)])
         cv2.imwrite(p, face)
         if args["recognize"] == 1:
            gray = cv2.cvtColor(face, cv2.COLOR_BGR2GRAY)
            cv2.imwrite("face_dataset/User." + str(face_id) + '.'
+ str(saved) + ".png", gray)

         saved += 1
         print("[INFO] saved {} to disk".format(p))

# do a bit of cleanup
vs.release()
cv2.destroyAllWindows()
```

Lampiran 1: source code face_training.py

```
''''
Training Multiple Faces stored on a DataBase:
   ==> Each face should have a unique numeric integer ID as 1, 2,
3, etc
   ==> LBPH computed model will be saved on trainer/ directory.
(if it does not exist, pls create one)
   ==> for using PIL, install pillow library with "pip install
pillow"
'''
import cv2
import numpy as np
from PIL import Image
import os
```

```
# Path for face image database
path = 'face_dataset'

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector =
cv2.CascadeClassifier("cascades/data/haarcascade_frontalface_defau
lt.xml")

# function to get the images and label data
def getImagesAndLabels(path):

    imagePaths = [os.path.join(path,f) for f in os.listdir(path)]
    faceSamples=[]
    ids = []

    for imagePath in imagePaths:

        PIL_img = Image.open(imagePath).convert('L') # convert it
to grayscale
        img_numpy = np.array(PIL_img,'uint8')

        id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces = detector.detectMultiScale(img_numpy)

        for (x,y,w,h) in faces:
            faceSamples.append(img_numpy[y:y+h,x:x+w])
            ids.append(id)

    return faceSamples,ids

print ("\n [INFO] Training faces. It will take a few seconds. Wait
...")
faces,ids = getImagesAndLabels(path)
recognizer.train(faces, np.array(ids))

# Save the model into trainer/trainer.yml
recognizer.write('trainer/trainer.yml') # recognizer.save() worked
on Mac, but not on Pi

# Print the numer of faces trained and end program
print("\n [INFO] {0} faces trained. Exiting
Program".format(len(np.unique(ids))))
```

Lampiran 1: source code train.py

```
# USAGE
# python train.py --dataset dataset --model liveness.model --le
le.pickle

# set the matplotlib backend so figures can be saved in the
background
import matplotlib
matplotlib.use("Agg")
```

```
# import the necessary packages
from pyimagesearch.livenessnet import LivenessNet
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from tensorflow.keras.preprocessing.image import
ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import pickle
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
   help="path to input dataset")
ap.add_argument("-m", "--model", type=str, required=True,
   help="path to trained model")
ap.add_argument("-l", "--le", type=str, required=True,
   help="path to label encoder")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
   help="path to output loss/accuracy plot")
args = vars(ap.parse_args())

# initialize the initial learning rate, batch size, and number of
# epochs to train for
INIT_LR = 1e-4
BS = 8
EPOCHS = 50

# grab the list of images in our dataset directory, then
initialize
# the list of data (i.e., images) and class images
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# loop over all image paths
for imagePath in imagePaths:
   # extract the class label from the filename, load the image and
   # resize it to be a fixed 32x32 pixels, ignoring aspect ratio
   label = imagePath.split(os.path.sep)[-2]
   image = cv2.imread(imagePath)
   image = cv2.resize(image, (32, 32))

   # update the data and labels lists, respectively
   data.append(image)
   labels.append(label)

# convert the data into a NumPy array, then preprocess it by
```

```
scaling
# all pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0

# encode the labels (which are currently strings) as integers and
then
# one-hot encode them
le = LabelEncoder()
labels = le.fit_transform(labels)
labels = to_categorical(labels, 2)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
   test_size=0.25, random_state=42)

# construct the training image generator for data augmentation
aug = ImageDataGenerator(rotation_range=20, zoom_range=0.15,
   width_shift_range=0.2, height_shift_range=0.2,
shear_range=0.15,
   horizontal_flip=True, fill_mode="nearest")

# initialize the optimizer and model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model = LivenessNet.build(width=32, height=32, depth=3,
   classes=len(le.classes_))
model.compile(loss="binary_crossentropy", optimizer=opt,
   metrics=["accuracy"])

# train the network
print("[INFO] training network for {} epochs...".format(EPOCHS))
H = model.fit(x=aug.flow(trainX, trainY, batch_size=BS),
   validation_data=(testX, testY), steps_per_epoch=len(trainX) //
BS,
   epochs=EPOCHS)

# evaluate the network
print("[INFO] evaluating network...")
predictions = model.predict(x=testX, batch_size=BS)
print(classification_report(testY.argmax(axis=1),
   predictions.argmax(axis=1), target_names=le.classes_))

# save the network to disk
print("[INFO] serializing network to
'{}'...".format(args["model"]))
model.save(args["model"], save_format="h5")

# save the label encoder to disk
f = open(args["le"], "wb")
f.write(pickle.dumps(le))
f.close()

# plot the training loss and accuracy
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, EPOCHS), H.history["loss"],
label="train_loss")
plt.plot(np.arange(0, EPOCHS), H.history["val_loss"],
label="val_loss")
plt.plot(np.arange(0, EPOCHS), H.history["accuracy"],
label="train_acc")
plt.plot(np.arange(0, EPOCHS), H.history["val_accuracy"],
label="val_acc")
plt.title("Training Loss and Accuracy on Dataset")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Lampiran 1: source code livenes_demo.py

```
# USAGE
# python liveness_demo.py --model liveness.model --le le.pickle --
detector face_detector

# TODO:
#  0. read data from db
#  1. print name from db

# import the necessary packages
from imutils.video import VideoStream
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model

import mysql.connector
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-m", "--model", type=str, required=True,
   help="path to trained model")
ap.add_argument("-l", "--le", type=str, required=True,
   help="path to label encoder")
ap.add_argument("-d", "--detector", type=str, required=True,
   help="path to OpenCV's deep learning face detector")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
   help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# Face Detection
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
```

```
cascadePath = "cascades/data/haarcascade_frontalface_alt.xml"
faceCascade = cv2.CascadeClassifier(cascadePath)

font = cv2.FONT_HERSHEY_SIMPLEX

#iniciate id counter
id = 0


# mysql connector
mydb = mysql.connector.connect(
   host="localhost",
   user="admin",
   password="admin",
   database="face_recognition"
)
# fetch all users
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM users")
users = mycursor.fetchall()

# names related to ids: example ==> Marcelo: id=1,  etc
names = ['None']
for user in users:
   names.append(user[1])
print(names)
# names = ['None', 'Bryan', 'Paula', 'Ilza', 'Z', 'W']
#################

# load our serialized face detector from disk
print("[INFO] loading face detector...")
protoPath = os.path.sep.join([args["detector"],
"deploy.prototxt"])
modelPath = os.path.sep.join([args["detector"],
   "res10_300x300_ssd_iter_140000.caffemodel"])
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# load the liveness detector model and label encoder from disk
print("[INFO] loading liveness detector...")
model = load_model(args["model"])
le = pickle.loads(open(args["le"], "rb").read())

# initialize the video stream and allow the camera sensor to
warmup
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

# Define min window size to be recognized as a face

# loop over the frames from the video stream
while True:
   # grab the frame from the threaded video stream and resize it
   # to have a maximum width of 600 pixels
   real_indicator = 0
   user_indicator = 0
```

```
    frame = vs.read()
    frame = imutils.resize(frame, width=600)

    #FACE RECOGNITION
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
       gray,
       scaleFactor=1.2,
       minNeighbors=5,
       minSize=(50, 50),
       flags=cv2.CASCADE_SCALE_IMAGE
    )

    for (x, y, w, h) in faces:

       cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

       id, confidence = recognizer.predict(gray[y:y + h, x:x + w])

       # Check if confidence is less them 100 ==> "0" is perfect
match
       if (confidence < 100):
          name_user = names[id]
          confidence = "  {0}%".format(round(100 - confidence))
          user_indicator = 1
       else:
          name_user = "unknown"
          confidence = "  {0}%".format(round(100 - confidence))
          user_indicator = 0

       cv2.putText(frame, str(name_user), (x + 5, y - 5), font, 1,
(255, 255, 255), 2)
       cv2.putText(frame, str(confidence), (x + 5, y + h - 5),
font, 1, (255, 255, 0), 1)

    # grab the frame dimensions and convert it to a blob
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
1.0,
       (300, 300), (104.0, 177.0, 123.0))

    # pass the blob through the network and obtain the detections
and
    # predictions
    net.setInput(blob)
    detections = net.forward()

    # loop over the detections
    for i in range(0, detections.shape[2]):
       # extract the confidence (i.e., probability) associated with
the
       # prediction
       confidence = detections[0, 0, i, 2]

       # filter out weak detections
```

```
      if confidence > args["confidence"]:
         #real_indicator = 1
         # compute the (x, y)-coordinates of the bounding box for
         # the face and extract the face ROI
         box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
         (startX, startY, endX, endY) = box.astype("int")

         # ensure the detected bounding box does fall outside the
         # dimensions of the frame
         startX = max(0, startX)
         startY = max(0, startY)
         endX = min(w, endX)
         endY = min(h, endY)

         # extract the face ROI and then preproces it in the exact
         # same manner as our training data
         face = frame[startY:endY, startX:endX]
         face = cv2.resize(face, (32, 32))
         face = face.astype("float") / 255.0
         face = img_to_array(face)
         face = np.expand_dims(face, axis=0)

         # pass the face ROI through the trained liveness detector
         # model to determine if the face is "real" or "fake"
         preds = model.predict(face)[0]
         j = np.argmax(preds)
         label = le.classes_[j]
         if label == 'real' :
            real_indicator = 1

         # draw the label and bounding box on the frame
         label = "{}: {:.4f}".format(label, preds[j])
         cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
         cv2.rectangle(frame, (startX, startY), (endX, endY),
            (0, 0, 255), 2)

   if real_indicator == 1 and user_indicator == 1:
            # TODO:
            #open the door
         exec(open('/home/pi/face-recog/buka_pintu.py').read())
   # show the output frame and wait for a key press
   cv2.imshow("Frame", frame)
   key = cv2.waitKey(1) & 0xFF

   # if the `q` key was pressed, break from the loop
   if key == ord("q"):
      break

# do a bit of cleanup
cv2.destroyAllWindows()
vs.stop()
```