

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра «Высшая школа программной инженерии»

Курсовой проект
**“ Анти-фальсификация видеоизображением и
изображением в системе идентификации по лицам”**
по предмету: «Защита информации»

Выполнил
Студент группы 3530904/60105

Чинь В.Т.

Руководитель

Медведев Б. М.

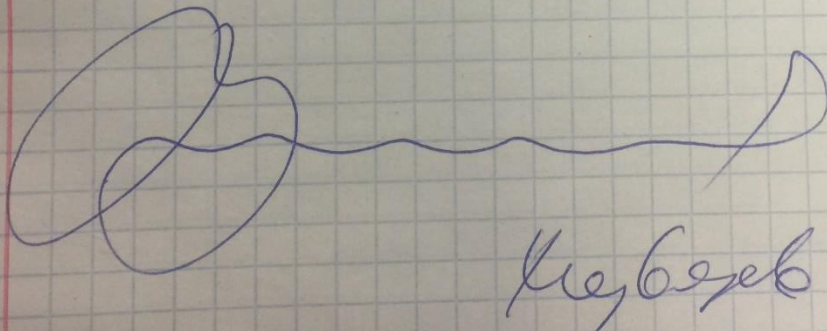
Задание

1. Классификация на входе система идентификация
типов человека или поперечное изображение / видео.
2. На входе системы живой человек, то происх.
идентификация ^(определ. кол-во попыток) (~~3~~ раз) ✓. Если попытка правильно,
система открыта, если нет, блокируется на несколько
минут.

(Замеч. На входе 1 камера).

- 1 Тема. Анти поперечное изображение / видео в
системе идентификации по лицам.

Тинь Ван Тхань ул 3530304160105.



Кебеев

Оглавление

Введение	4
Постановки задачи	5
Выполнение работы	5
1. Описание системы	5
2. Обзор методов обнаружения спуфинга	5
Результат полученный	6
Заключение	8
Список использованной литературы	8
Приложение 1: Код программы	8
1. ResNet.py: Модель обнаружения спуфинга.	8
2. TrainModel.py: Обучения модели идентификации.	9
3. Main.py: Программа.	9

Введение

Биометрическая идентификация человека – это одна из самых старых идей для распознавания людей, которую вообще попытались технически осуществить [1]. Пароли можно украсть, подсмотреть, забыть, ключи – подделать. А вот уникальные характеристики самого человека подделать и потерять намного труднее. Это могут быть отпечатки пальцев, голос, рисунок сосудов сетчатки глаза, походка и прочее, которое используется для решения задач верификации и идентификации профиля человека среди миллионов записей в базе данных.

Наиболее простой с точки зрения считывания биометрических данных является технология лицевой биометрии. Современные системы распознавания личности по лицу способны с высокой точностью работать даже в неконтролируемых условиях. Однако это приводит к повышенному риску взлома, так как для прохождения верификации, системе достаточно предоставить фотографию, сделанную на обычную камеру или взятую из открытых источников. В связи с этим возникает ряд задач по предотвращению попыток подмены биометрических данных, обычно называемых спуфингатаками [2].

Спуфинг-атака на лицевую биометрическую систему может быть осуществлена разными способами. Наиболее эффективным из них является прямая загрузка фотографии в систему, однако для этого злоумышленникам требуется получить доступ к программному обеспечению, что начительное усложняет атаку. В случае доступности ответа системы в виде степени схожести или вероятности верификации, возможно осуществление атаки непосредственно на алгоритмы построения биометрического шаблона, которые, как правило, основаны а глубоких сверточных нейронных сетях (СНС). Как известно СНС уязвимы к так называемым «состязательным атакам» (adversarial attack), что позволяет злоумышленникам использовать специальным образом нанесенный макияж для существенного увеличения вероятности ложного срабатывания системы. Наконец, наиболее простым способом осуществления атаки является атака на уровне сенсора с использованием фотографии зарегистрированного в системе пользователя.

В данной работе предлагается способ защиты от последнего вида спуфинг-атак на системы идентификации по лицам и управления доступом. Ввиду большой популярности социальных сетей весьма просто заполучить фотографию какого-либо человека, работающего на предприятии. При этом она может демонстрироваться сенсору как в распечатанном виде, так и на смартфоне или планшете. Далее будет рассмотрена единая модель, позволяющая детектировать одновременно все варианты спуфинга на уровне сенсора используя только одно изображение при малом количестве ложных отказов.

Постановки задачи

1. Классификация на входе системы идентификации живого человек или поддельное изображение/видеоизображение.
2. Если на входе системы живой человек, то происходит идентификацию (ограничение количество попыток – 10 раз – это значение можно задать самостоятельно). Если получается правильно, системы открыта. А в остальном случае, система блокируется на несколько минуты (5 минут).

Выполнение работы

1. Описание системы

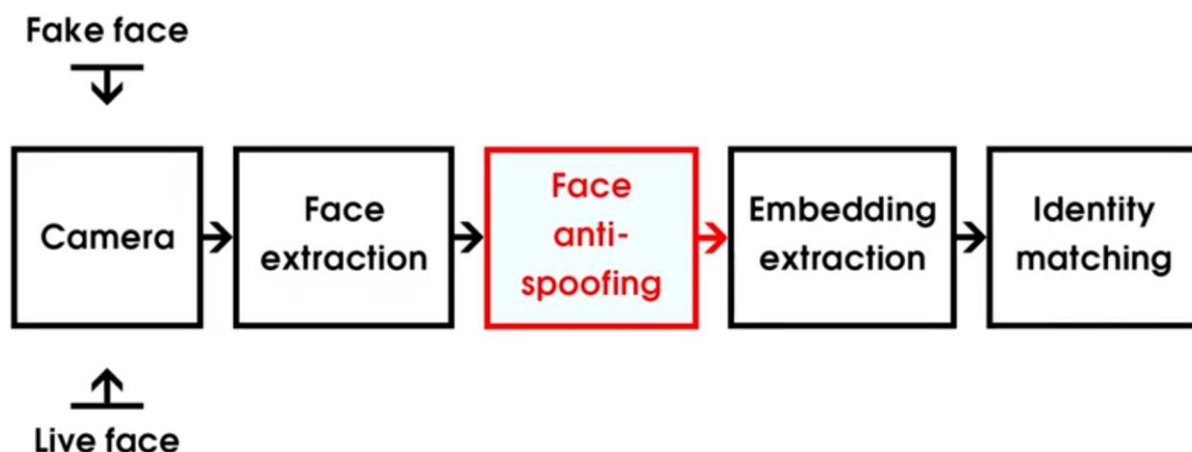


Рис. 1. Схема работы системы

Система работает по следующему алгоритму (на рисунке 1) [3]:

Шаг 1: На вход системы попадает лицо из изображения/видеоизображения/ или само лицо живого человека.

Шаг 2: Распознавание лиц, которые попадают на вход, используя модуль Deep Neural Network обеспечивающую библиотекой opencv на Python [4].

Шаг 3: Проверка распознанных лиц.

Если входное лицо является лицом живого человека, переходить в шаг 4.

Если лицо не является лицом живого человека, возвращать в шаг 2.

Шаг 4: Идентификация распознанных лиц, используя модуль LBPHFaceRecognizer обеспечивающую библиотекой opencv на Python [5].

По результату идентификации лиц, система дает пользователю доступ. Система ограничена количество попыток, она будет блокировать на несколько минут если количество попыток превышает ограничение.

2. Обзор методов обнаружения спуфинга

Проблема детектирования спуфинг-атак или обнаружения витальности (liveness detection) применительно к системам лицевой биометрии, приобрела популярность у исследователей в середине 2000-х годов. К настоящему времени предложено множество методов ее решения, которые условно разделяются на две группы: активные и пассивные.

Активные методы запрашивают от пользователя совершения определенного действия, например: улыбнуться, моргнуть, наклонить или повернуть голову и др. Пассивные методы обычно используют для анализа только одного изображения, по которому непосредственно строится биометрический шаблон. В связи с этим они удобнее в использовании, а также позволяют исключить ситуацию демонстрации фотографии в промежуток между процессами определения витальности лица и верификации. Большое количество работ, посвященных алгоритмам пассивного обнаружения спуфинг-атак, основаны на анализе текстуры области лица [2].

В моей работе, чтобы предотвратить подделку обоих видеоизображений, содержащих движения пользователя, я использую пассивный метод, основанный на анализе текстуры изображения лица. Для этого, я построил сверточную нейронную сеть по архитектуре ResNet длины 18. Согласно следующей структуре [6]:

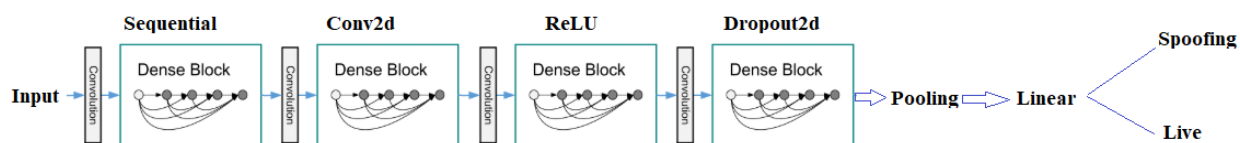


Рис. 2. Архитектура сети.

Результат полученный

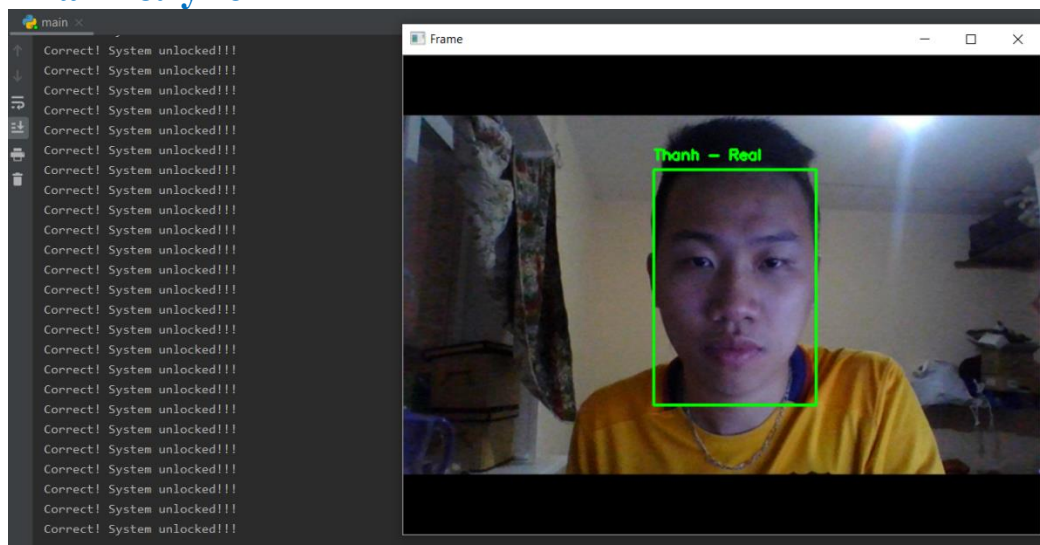


Рис 3. Результат лица живого ползователя, имеющего доступ к системе.

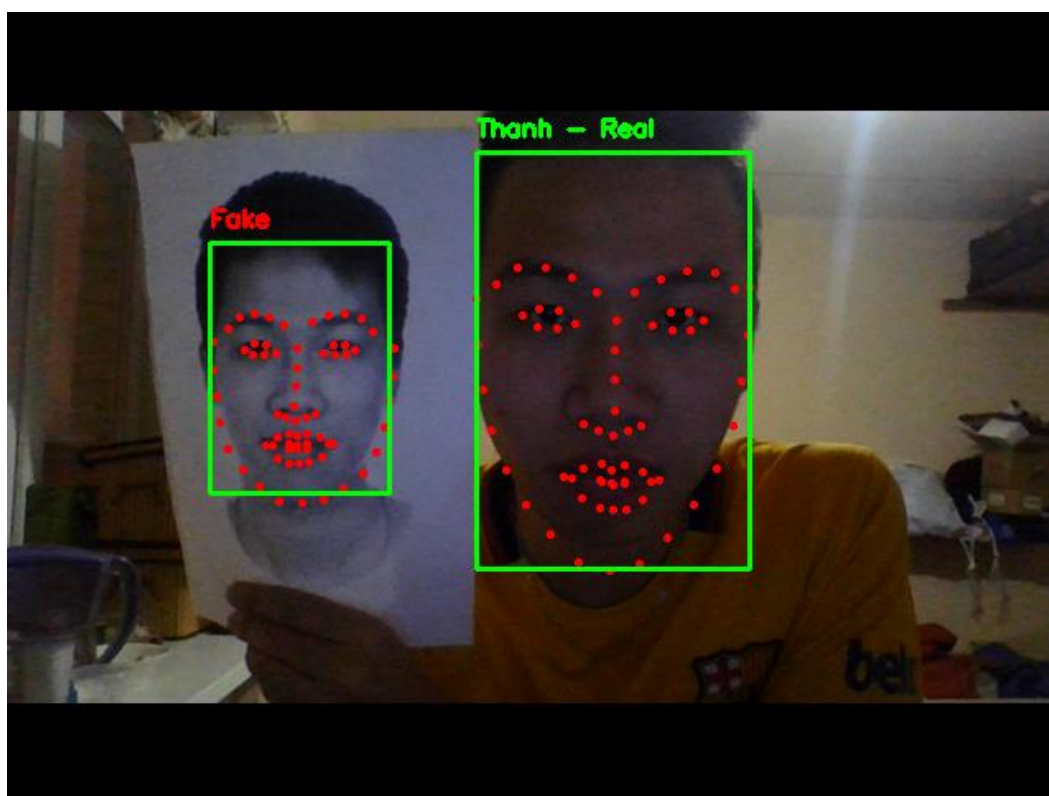


Рис 4. Результат лица живого ползователя и лица из изображения.



Рис 3. Результат лица живого ползователя, и лица из видеоизображения.

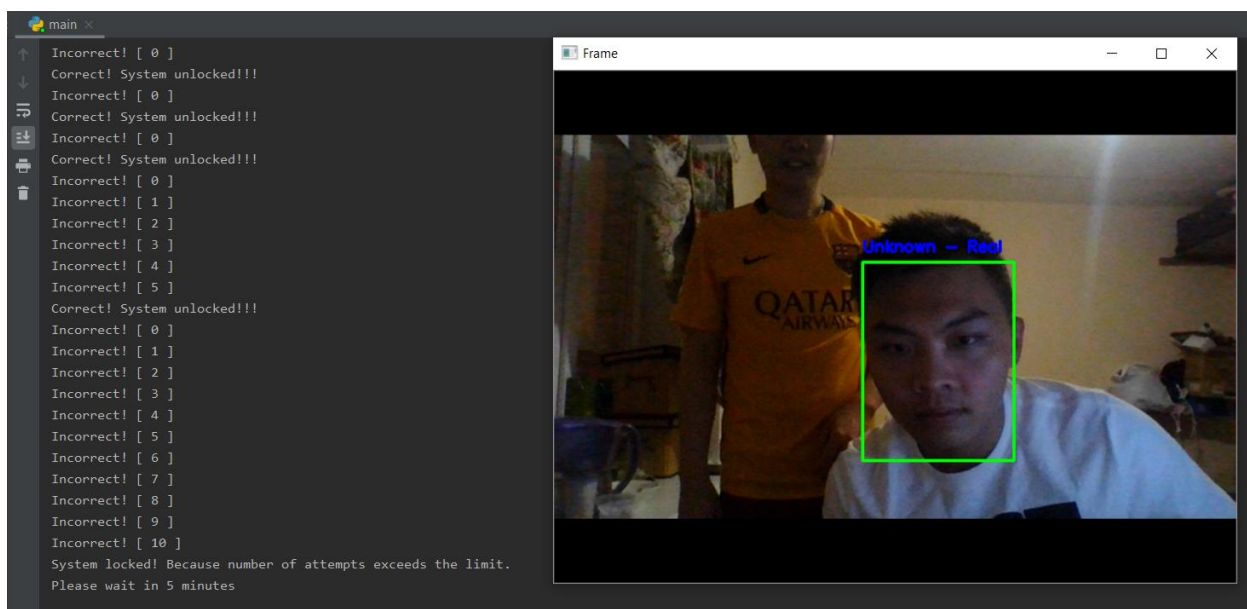


Рис 3. Результат лица живого ползователя, не имеющего доступ к системе.

Заключение

В ходе выполнения курсовой работы были изучены система идентификации по лицам, метод обнаружения спуфинга, и построение нейронной сети. Получено опыт работы на Python с открытой библиотекой opencv.

Полученный результат зависит от устройства, и рабочей сферы. Так как текстуры изображения зависит от этого. В будущем, предлагается исправить систему, чтобы уменьшать ошибку и зависимость от устройства и рабочей сферы.

Список использованной литературы

1. Статья *Face Anti-Spoofing* или технологично узнаём обманищика из тысячи по лицу. <https://habr.com/ru/company/ods/blog/452894/>
2. И. А. Калиновский, Г.М. Лаврентьева. *Обнаружение спуфинг-атак на систему лицевой биометрии*. В GraphiCon 2018. <https://www.graphicon.ru/html/2018/papers/204-207.pdf>
3. Serhii Maskymenko. *Anti-Spoofing Techniques For Face Recognition Solutions*. <https://towardsdatascience.com/anti-spoofing-techniques-for-face-recognition-solutions-4257c5b1dfc9>
4. Материал: https://docs.opencv.org/3.4/d6/d0f/group_dnn.html
5. Материал: https://docs.opencv.org/3.4/df/d25/classcv_1_1face_1_1LBPHFaceRecognizer.html
6. Connor Shorten. Introduction to ResNets. <https://towardsdatascience.com/introduction-to-resnets-c0a830a288a4>

Приложение 1: Код программы

1. [ResNet.py: Модель обнаружения спуфинга.](#)

```
import torch
import torch.nn as nn
import time
from torchvision import models

class BasicModule(nn.Module):
    def __init__(self, opt=None):
        super(BasicModule, self).__init__()
        self.model_name = str(type(self))

    def load(self, path):
        self.load_state_dict(torch.load(path, map_location=torch.device('cpu')))

    def save(self, name=None):
        if name is None:
            prefix = './checkpoints/' + self.model_name + '_'
            name = time.strftime(prefix + '%m%d_%H:%M:%S.pth')
            torch.save(self.state_dict(), name)
        return name

class MyresNet18(BasicModule):
    def __init__(self):
        super(MyresNet18, self).__init__()
        model = models.resnet18(pretrained=True)
        self.resnet_layer = nn.Sequential(*list(model.children())[:-2])
        self.conv1_layer = nn.Conv2d(512, 256, kernel_size=(1,1), stride=(1,1))
        self.relu_layer = nn.ReLU(inplace=True)
        self.dropout_layer = nn.Dropout2d(0.5)
        self.global_average = nn.AdaptiveAvgPool2d((1,1))
```



```

        self.fc_Linear_lay2 = nn.Linear(256, 2)

    def forward(self, x):
        x = self.resnet_lay(x)
        x = self.conv1_lay(x)
        x = self.relu1_lay(x)
        x = self.drop_lay(x)
        x = self.global_average(x)
        x = x.view(x.size(0), -1)
        x = self.fc_Linear_lay2(x)
        return x

```

2. TrainModel.py: Обучения модели идентификации.

```

import cv2
import os
import numpy as np
from PIL import Image

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

def getFaces(path):
    # Get all file in folder
    imgPaths = [os.path.join(path, f) for f in os.listdir(path)]
    # Create empty list face and ID
    faceSamples = []
    ids = []
    # Looping through all the image paths and loading faceSamples and ids
    idx = 1
    for imgPath in imgPaths:
        print("Trained file: ", imgPath)
        # loading image and converting to gray scale
        #img = cv2.imread(imgPath, cv2.IMREAD_GRAYSCALE)
        img = Image.open(imgPath).convert('L')
        # convert image to array
        imgNp = np.array(img, 'uint8')
        #print(imgNp)
        # extract the face from the image
        faces = detector.detectMultiScale(imgNp)
        fc_id = 1
        for (x,y,w,h) in faces:
            faceSamples.append(imgNp[y:y+h, x:x+w])
            ids.append(1)
            print("    Face: ", fc_id)
            fc_id = fc_id + 1
    return faceSamples, ids

# Get face and id from dataset
faceSamples, ids =
getFaces('F:/7th/Medvedev/AntiSpoofingFaceID/face_identification/dataset')
# Train model
recognizer.train(faceSamples, np.array(ids))
# Save model
recognizer.save('trained.yml')
print('Train finished!')

```

3. Main.py: Программа.

```

import ResNet
import argparse
import cv2
import time
import numpy as np

```

```

import torch

ap = argparse.ArgumentParser()
ap.add_argument("-c", "--confidence", type=float, default=0.5,
                help="minimum probability to filter weak detections")
ap.add_argument("-d", "--draw", type=bool, default=False,
                help="draw dots on face")
ap.add_argument("-n", "--attempt", type=int, default=10,
                help="number of attempts")
args = vars(ap.parse_args())

protoPath = "./face_detector/deploy.prototxt"
modelPath = "./face_detector/res10_300x300_ssd_iter_140000.caffemodel"
netFaceDet = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

protoPath2 = "./face_alignment/2_deploy.prototxt"
modelPath2 = "./face_alignment/2_solver_iter_800000.caffemodel"
netFaceDet2 = cv2.dnn.readNetFromCaffe(protoPath2, modelPath2)

def crop_by_mark(image, landmark):
    scale = 3.5
    image_size = 224
    ct_x, std_x = landmark[:, 0].mean(), landmark[:, 0].std()
    ct_y, std_y = landmark[:, 1].mean(), landmark[:, 1].std()

    std_x, std_y = scale * std_x, scale * std_y

    src = np.float32([(ct_x, ct_y), (ct_x + std_x, ct_y + std_y), (ct_x + std_x,
ct_y)])
    dst = np.float32([(image_size - 1) / 2.0, (image_size - 1) / 2.0),
                      ((image_size - 1), (image_size - 1)),
                      ((image_size - 1), (image_size - 1) / 2.0)])
    retval = cv2.getAffineTransform(src, dst)
    result = cv2.warpAffine(image, retval, (image_size, image_size),
flags=cv2.INTER_LINEAR,
                                borderMode=cv2.BORDER_CONSTANT)

    return result

def check_real_fake(img):
    data = np.transpose(np.array(img, dtype=np.float32), (2, 0, 1))
    data = data[np.newaxis, :]
    data = torch.FloatTensor(data)
    with torch.no_grad():
        outputs = modelNetRF(data)
        outputs = torch.softmax(outputs, dim=-1)
        preds = outputs.to('cpu').numpy()
        fake_prob = preds[:, FAKE]
    return fake_prob

if __name__ == '__main__':
    print("[INFO] Load ResNet anti spoofing...")
    net_path = "a8.pth"
    modelNetRF = getattr(ResNet, "MyresNet18")().eval()
    modelNetRF.load(net_path)
    modelNetRF.train(False)

    FAKE = 1
    thresh_probability = 0.9 #0.85
    REAL = 0

    print("[INFO] Load face recognizer...")

```

```

faceRecognizer = cv2.face.LBPHFaceRecognizer_create()
faceRecognizer.read('./face_identification/trained.yml')

print("[INFO] Start video stream....")
vd = cv2.VideoCapture(0)
numIncorrect = 0
while True:
    ret, frame = vd.read()
    if ret is None:
        break
    # Face detection
    timeStart = time.time()
    (h, w) = frame.shape[:2]
    blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0, (300, 300),
(104.0, 177.0, 123.0))
    netFaceDet.setInput(blob)
    detectionFaces = netFaceDet.forward()
    timeEnd = time.time()
    # print('Detect times : %.3f ms' % ((timeEnd - timeStart) * 1000))
    for i in range(0, detectionFaces.shape[2]):
        confidence = detectionFaces[0, 0, i, 2]
        if confidence > args["confidence"]:
            # Bounding detection face from frame
            box = detectionFaces[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int")
            (sx, sy, ex, ey) = (startX, startY, endX, endY)
            # Find borders of face: Loai bo phan tran, lay phan mat, tai
            ww = (endX - startX) // 10
            hh = (endY - startY) // 5
            startX = startX - ww
            startY = startY + hh
            endX = endX + ww
            endY = endY + hh
            # If startX, Y and endX, y are not in range of frame
            startX = max(0, startX)
            startY = max(0, startY)
            endX = min(w, endX)
            endY = min(h, endY)
            # cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 0, 255),
2)

            # Find dots on Face (dots are eye, noise, lip ...)
            x1 = int(startX)
            y1 = int(startY)
            x2 = int(endX)
            y2 = int(endY)
            # Crop Face
            roi = frame[y1:y2, x1:x2]
            grayFace = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)
            # Histogram of brightness
            matrixFace = np.float32(grayFace)
            m = np.zeros((40, 40))
            sd = np.zeros((40, 40))
            mean, std_dev = cv2.meanStdDev(matrixFace, m, sd) # gives you a mean
for each channel and a standard deviation for each channel as arrays
            new_mean = mean[0][0]
            new_std = std_dev[0][0]
            matrixFace = (matrixFace - new_mean) / (0.000001 + new_std)
            # cv2.imshow("Brightness Face", matrixFace)
            blob = cv2.dnn.blobFromImage(cv2.resize(matrixFace, (40, 40)), 1.0,
(40, 40), (0, 0, 0))
            netFaceDet2.setInput(blob)

```

```

alignFace = netFaceDet2.forward()
aligns = [] # list of dots on one face
alignss = []
# Calculate coordinate of dots on Face
for k in range(0, 68):
    dotTMP = []
    x = alignFace[0][2 * k] * (x2 - x1) + x1
    y = alignFace[0][2 * k + 1] * (y2 - y1) + y1
    if (args["draw"] == True):
        cv2.circle(frame, (int(x), int(y)), 1, (0, 0, 255), 2)
    dotTMP.append(int(x))
    dotTMP.append(int(y))
    aligns.append(dotTMP)
cv2.rectangle(frame, (sx, sy), (ex, ey), (0, 255, 0), 2)
alignss.append(aligns)

mark = np.asarray(alignss, dtype=np.float32)
mark = mark[np.argsort(np.std(mark[:, :, 1], axis=1))[-1]]
img = crop_by_mark(frame, mark)
fake_probability = check_real_fake(img)
if fake_probability > thresh_probability:
    cv2.putText(frame, "Fake", (sx, sy - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    (0, 0, 255), 2)
else:
    gray = cv2.cvtColor(frame[sy:ey, sx:ex], cv2.COLOR_RGB2GRAY)
    id, accuracy = faceRecognizer.predict(gray)
    if accuracy <= 50:
        print("Correct! System unlocked!!!")
        cv2.putText(frame, "Thanh - Real", (sx, sy - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    (0, 255, 0), 2)
        numIncorrect = 0
    else:
        print("Incorrect! [", numIncorrect, "]")
        numIncorrect = numIncorrect + 1;
        if numIncorrect > args["attempt"]:
            print("System locked! Because number of attempts exceeds
the limit.")
            print("Please wait in 5 minutes")
            cv2.waitKey(300000)
            numIncorrect = 0
        cv2.putText(frame, "Unknown - Real", (sx, sy - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
                    (255, 0, 0), 2)

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF
    if key == ord("q"):
        break

print("[INFO] End process...")
cv2.destroyAllWindows()

```