Проект high level design

**Технологический стек:**

1) Face detection – SSD-300 (Single Shot Detector)

Вход – изображение в формате jpeg или png.

Выход – M-confidence-vector; M-boundbox-array;

M – число задетекченных лиц с уверенностью не менее 0.5

Производительность - 45 кадров/сек.

2) Face alignment – 68-landmarks

Вход – Изображение и рамка (dlib.rectangle)

Выход – Изображение с выравненным лицом

Производительность – 0.2 секунд на выравнивание одного лица

3) Feature extraction – ResNet-200

Вход – изображение лица 224х224px

Выход – 128-feature vector

4) Classification – L2-distance с порогом 0.6

**Code:**

import numpy as np

import sys

import cv2

import dlib

import openface

import face\_recognition\_models

class FaceDetector:

"""Detect human face from image"""

def \_\_init\_\_(self,

dnn\_proto\_text='models/deploy.prototxt',

dnn\_model='models/res10\_300x300\_ssd\_iter\_140000.caffemodel'):

"""Initialization"""

self.face\_net = cv2.dnn.readNetFromCaffe(dnn\_proto\_text, dnn\_model)

self.detection\_result = None

def get\_faceboxes(self, image, threshold=0.5):

"""

Get the bounding box of faces in image using dnn.

"""

rows, cols, \_ = image.shape

confidences = []

faceboxes = []

self.face\_net.setInput(cv2.dnn.blobFromImage(

image, 1.0, (300, 300), (104.0, 177.0, 123.0), False, False))

detections = self.face\_net.forward()

for result in detections[0, 0, :, :]:

confidence = result[2]

if confidence > threshold:

x\_left\_bottom = int(result[3] \* cols)

y\_left\_bottom = int(result[4] \* rows)

x\_right\_top = int(result[5] \* cols)

y\_right\_top = int(result[6] \* rows)

confidences.append(confidence)

#dlib rectangle for alignment

faceboxes.append(

[x\_left\_bottom, y\_left\_bottom, x\_right\_top, y\_right\_top])

self.detection\_result = [faceboxes, confidences]

return confidences, faceboxes

def draw\_all\_result(self, image):

"""Draw the detection result on image"""

for facebox, conf in self.detection\_result:

cv2.rectangle(image, (facebox[0], facebox[1]),

(facebox[2], facebox[3]), (0, 255, 0))

label = "face: %.4f" % conf

label\_size, base\_line = cv2.getTextSize(

label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, 1)

cv2.rectangle(image, (facebox[0], facebox[1] - label\_size[1]),

(facebox[0] + label\_size[0],

facebox[1] + base\_line),

(0, 255, 0), cv2.FILLED)

cv2.putText(image, label, (facebox[0], facebox[1]),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 0))

@staticmethod

def draw\_marks(image, marks, color=(0, 255, 0)):

"""Draw mark points on image"""

for mark in marks:

cv2.circle(image, (int(mark.x), int(

mark.y)), 1, color, -1, cv2.LINE\_AA)

def compare\_2\_faces(known\_face\_encoding, face\_encoding\_to\_check):

return (np.linalg.norm(known\_face\_encoding - face\_encoding\_to\_check))

def compare\_faces(known\_faces, face\_to\_check):

tolerance=0.6

ind, length = -1, sys.float\_info.max

for i in range(len(known\_faces)):

for face in known\_faces[i]:

cur = compare\_2\_faces(face, face\_to\_check)

if (cur < length):

length = cur

ind = i

if (length <= tolerance):

return ind

else:

return -1

def main():

classes = ['MXG', 'Sanaken', 'Zofinka', 'Toalk', 'Zissxzirsziiss', 'kiasummer']

known\_face\_encodes = [

np.loadtxt('persons/MXG/fv.txt'),

np.loadtxt('persons/Sanaken/fv.txt'),

np.loadtxt('persons/Zofinka/fv.txt'),

np.loadtxt('persons/Toalk/fv.txt'),

np.loadtxt('persons/Zissxzirsziiss/fv.txt'),

np.loadtxt('persons/kiasummer/fv.txt')

]

known\_face\_encodes = np.reshape(known\_face\_encodes, (6, 5, 128))

#get image

image = cv2.imread('team.jpg', 1)

#get bboxes

fd = FaceDetector()

\_, faceboxes = fd.get\_faceboxes(image)

#get alignment model

predictor\_model = "models/shape\_predictor\_68\_face\_landmarks.dat"

face\_pose\_predictor = dlib.shape\_predictor(predictor\_model)

face\_aligner = openface.AlignDlib(predictor\_model)

for i in range(len(faceboxes)):

# print(faceboxes[i][0], faceboxes[i][1], faceboxes[i][2], faceboxes[i][3])

face\_rect = dlib.rectangle(faceboxes[i][0], faceboxes[i][1], faceboxes[i][2], faceboxes[i][3])

# Get the the face's pose

pose\_landmarks = face\_pose\_predictor(image, face\_rect)

# Use openface to calculate and perform the face alignment

alignedFace = face\_aligner.align(534, image, face\_rect, landmarkIndices=openface.AlignDlib.OUTER\_EYES\_AND\_NOSE)

output = cv2.resize(alignedFace, (faceboxes[i][2] - faceboxes[i][0], faceboxes[i][3] - faceboxes[i][1]))

#draw marks

parts = dlib.full\_object\_detection.parts(pose\_landmarks)

FaceDetector.draw\_marks(image, parts)

#uncomment to see aligned faces

#image[faceboxes[i][1]:faceboxes[i][3], faceboxes[i][0]:faceboxes[i][2]] = output

#init predection model

predictor\_5\_point\_model = face\_recognition\_models.pose\_predictor\_five\_point\_model\_location()

pose\_predictor\_5\_point = dlib.shape\_predictor(predictor\_5\_point\_model)

face\_recognition\_model = face\_recognition\_models.face\_recognition\_model\_location()

face\_encoder = dlib.face\_recognition\_model\_v1(face\_recognition\_model)

#get face landmarks for feature extraction

landmark\_set = pose\_predictor\_5\_point(alignedFace, dlib.rectangle(0, 0, alignedFace.shape[0], alignedFace.shape[1]))

#get feature vector

feature\_vector = np.array(face\_encoder.compute\_face\_descriptor(alignedFace, landmark\_set, 1))

#uncomment for adding feature\_vector. 'w' for write, 'a' for append

#with open('persons/Sanaken/fv.txt', 'w') as outfile:

# np.savetxt(outfile, feature\_vector)

#known\_face\_encode = np.loadtxt('persons/MXG/fv.txt')

ind = compare\_faces(known\_face\_encodes, feature\_vector)

if (ind != -1):

cv2.putText(image, classes[ind], (faceboxes[i][0], faceboxes[i][1] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 0, 255), 1)

else:

cv2.putText(image, "Unknown", (faceboxes[i][0], faceboxes[i][1] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 1)

cv2.rectangle(image, (faceboxes[i][0], faceboxes[i][1]), (faceboxes[i][2], faceboxes[i][3]), (255, 0, 0))

cv2.imshow("Preview", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**Data flow diagram:**

Input Image

SSD-300 Face Detector

dlib.rect

68 Landmarks predictor

Aligned image

ResNet-200 feature extractor

Feature vector

L2-distance

Confidence vector