Maкет 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 - Perceptron:

Вход – 128-feature vector

Выход – N-вектор уверенности (N – число классов)

**Code:**

**1) SSD-300:**

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

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

dnn\_model='assets/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)

faceboxes.append(

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

self.detection\_result = [faceboxes, confidences]

return confidences, faceboxes

**2) 68-landmarks:**

import sys

import dlib

import cv2

import openface

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

# Take the image file name from the command line

file\_name = sys.argv[1]

# Create a HOG face detector using the built-in dlib class

face\_detector = dlib.get\_frontal\_face\_detector()

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

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

# Take the image file name from the command line

file\_name = sys.argv[1]

# Load the image

image = cv2.imread(file\_name)

# Run the HOG face detector on the image data

detected\_faces = face\_detector(image, 1)

print("Found {} faces in the image file {}".format(len(detected\_faces), file\_name))

# Loop through each face we found in the image

for i, face\_rect in enumerate(detected\_faces):

# Detected faces are returned as an object with the coordinates

# of the top, left, right and bottom edges

print("- Face #{} found at Left: {} Top: {} Right: {} Bottom: {}".format(i, face\_rect.left(), face\_rect.top(), face\_rect.right(), face\_rect.bottom()))

# 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)

# Save the aligned image to a file

cv2.imwrite("aligned\_face\_{}.jpg".format(i), alignedFace)

**3) ResNet**

import face\_recognition\_models

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

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

face\_encodings(face\_image, known\_face\_locations=None, num\_jitters=1):

"""

Given an image, return the 128-dimension face encoding for each face in the image.

:param face\_image: The image that contains one or more faces

:param known\_face\_locations: Optional - the bounding boxes of each face if you already know them.

:param num\_jitters: How many times to re-sample the face when calculating encoding. Higher is more accurate, but slower (i.e. 100 is 100x slower)

:return: A list of 128-dimensional face encodings (one for each face in the image)

"""

raw\_landmarks = \_raw\_face\_landmarks(face\_image, known\_face\_locations, model="small")

return [np.array(face\_encoder.compute\_face\_descriptor(face\_image, raw\_landmark\_set, num\_jitters)) for raw\_landmark\_set in raw\_landmarks]

**Data flow diagram:**

Input Image

SSD-300 Face Detector

dlib.rect

68 Landmarks predictor

Aligned image

ResNet model

Feature vector

Perceptron

Confidence vector