ПРИЛОЖЕНИЕ Ж

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

СОГЛАСОВАНО
Профессор департамента
программной инженерии
факультета компьютерных наук
доктор техн. наук

_____________________ Д.В. Александров
«___» _____________ 2018 г.

УТВЕРЖДАЮ
Академический руководитель
образовательной программы
«Программная инженерия»
профессор департамента программной
инженерии, канд. техн. наук

__________________ В.В. Шилов
«___» _____________ 2018 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.04.01-01 12101-1

Сервис распознавания лиц для идентификации личности
Сервер

Текст программы
ЛИСТ УТВЕРЖДЕНИЯ
RU.17701729.04.01-01 12 01-1-ЛУ

Исполнитель
студент группы БПИ 143
_____________________ /Н.О. Константиновский /
«____»_______________________ 2018 г.

Москва 2018

УТВЕРЖДЕН
RU.17701729.04.0-01 51 01-1-ЛУ

Сервис распознавания лиц для идентификации личности
Сервер

Текст программы

RU.17701729.04.01-01 12 01-1

Листов 39

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	RU.17701729.04.01-01 12 01-1

Москва 2018

Содержание

1. ТЕКСТ ПРОГРАММЫ

1.1. MindFaceService.py

```
# coding=utf-8

import os
from config import app



from WebApp.login.login import login_blueprint
from WebApp.persons.persons import persons_blueprint
from WebApp.photos.photos import photos_blueprint

from API.vector.vector import vector_blueprint
from API.load.load import load_blueprint
# from API.image.image import image_blueprint
[redacted]
app.register_blueprint(vector_blueprint)
app.register_blueprint(load_blueprint)
# app.register_blueprint(image_blueprint)



@app.route('/')
def home():
    return 'MindFace'



from flask import flash, redirect, render_template, request, session, abort, escape
from flask import Blueprint
import json
import io
from PIL import Image
from MLCore.face_detect import rewrite_alignet_resize96_face_image
import numpy as np
from config import config_model, pred_clf
# from MLCore.classifier_controller import pred_clf

from DBEngine.dbmodel import Person
```

[redacted]

```
    res = ''
    if img is not None:
        res = pred_clf(img)

        query = Person.query.filter_by(person_id=int(res))
        res = query.first()
        res = str(res.name) + ' ' + str(res.surname)

    return res



if __name__ == '__main__':
    config_model()
    app.secret_key = os.urandom(12)
    app.debug = False
    # app.run()
    # app.run(host='192.168.0.105')
    app.run(host='172.20.10.3')
```

1.2. config.py

```
# coding=utf-8

from flask import Flask

from MLCore.MindFaceKerasSmall2V1 import create_model
import numpy as np
import pickle



app = Flask(__name__)

[redacted]

app.config['UPLOAD_FOLDER'] =
'/Users/apple/Documents/Development/MindFaceProject/MindFaceService/static/uploads/'
app.config['CELERY_BROKER_URL'] = 'redis://localhost:6379/0'
app.config['CELERY_RESULT_BACKEND'] = 'redis://localhost:6379/0'

predictor_face_landmark_model = \

"/Users/apple/Documents/Development/MindFaceProject/MindFaceService/MLCore/shape_predictor_
68_face_landmarks.dat"

model_path =
'/Users/apple/Documents/Development/MindFaceProject/MindFaceService/MLCore/MindFaceKerasS
mall2V1.h5'

clf_path =
'/Users/apple/Documents/Development/MindFaceProject/MindFaceService/MLCore/classifier.sav'
```

```
clf_path_local =
'/Users/apple/Documents/Development/MindFaceProject/MindFaceService/static/classifier_local.sav'


MindFaceKerasSmall2V1 = None
main_clf = pickle.load(open(clf_path, 'rb'))
main_clf_local = pickle.load(open(clf_path_local, 'rb'))
```

[redacted]

```
def config_model():
    create_nn()


def pred_clf(img):
    global main_clf

    img = img[..., ::-1]
    img = np.around(img / 255.0, decimals=12)
    img = np.array([img])
    vec = MindFaceKerasSmall2V1.predict(img)

    res = main_clf.predict(vec)
    return res


from DBEngine.dbmodel import Person
```

```
def pred_clf_vec(vec):
    global main_clf


    res = main_clf.predict([vec])
    # res = main_bin_clf.predict(vec)


    query = Person.query.filter_by(person_id=int(res))
    res = query.first()
    res = str(res.name) + ' ' + str(res.surname)


    return res



def reload_clf():
    global main_clf
    main_clf = pickle.load(open(clf_path, 'rb'))



def reload_clf_local():
    global main_clf
    main_clf_local = pickle.load(open(clf_path_local, 'rb'))
```

[redacted]

```
main_bin_clf = None


def retrain_bin_classifier():
    global main_bin_clf
    vecs = get_marked_vecs(1)
    codes = []
    labels = []
    for vec in vecs:
        codes.append(vec[0])
        labels.append(vec[1])

    main_bin_clf = SVMBinaryClassificator(codes, labels)

    for person in Person.query.filter_by(user_id=1):
        person.is_train = True
        db.session.commit()
```

[redacted]

```
def get_marked_vecs(user_id):
    query = Photo.query.join(Photo.persons_list).filter_by(user_id=user_id)\
        .with_entities(Photo.features, Photo.person_id)
    result = query.all()
    return result



retrain_bin_classifier()
```

1.3. photos.py

```
# coding=utf-8

from flask import flash, redirect, render_template, request, session, abort, escape
from DBEngine.dbmodel import Photo
from DBEngine.dbmodel import Person
from flask import Blueprint
from Model.model import WebPhoto
from MLCore import face_detect
import os

from config import app


from DBEngine import db
import random
import string

photos_blueprint = Blueprint('photos', __name__)

ALLOWED_EXTENSIONS = set(['png', 'jpg', 'jpeg'])


```

[redacted]

```
    else:
        if request.args.get('pid') and request.args.get('del'):
            pid = request.args.get('pid')
            person_id = request.args.get('id')

            query = Person.query.filter_by(user_id=session.get('user_id'), person_id=person_id)
            result = query.first()
            if result:
                query = Photo.query.filter_by(person_id=person_id, photo_id=pid)
                path = query.first().path

                db.session.query(Photo).filter(Photo.photo_id == pid).delete()
```

```
                    db.session.commit()

                    path = app.config['UPLOAD_FOLDER'] + str(person_id) + '/' + str(path) + '.png'

                    if os.path.exists(path):
                        os.remove(path=path)

                return redirect("/photos?id=" + person_id)
            else:
                person_id = escape(request.args.get('id'))
                query = Person.query.filter_by(user_id=session.get('user_id'), person_id=person_id)
```

[redacted]

```
                    if result is not None:
                        photos = []
                        i = 1
                        for res in result:
                            path = 'static/uploads/' + str(person_id) + '/' + res.path + '.png'
                            photos.append(WebPhoto(str(i), path, res.is_train, res.photo_id))
                            i += 1

                        return render_template("photos.html", photos=photos, person=person_name,
id=person_id)
                    else:
                        return redirect("/persons")
                else:
                    return redirect("/persons")



def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


def random_filename_generator(size=16, chars=string.ascii_uppercase + string.digits):
    return ".join(random.choice(chars) for x in range(size))


@photos_blueprint.route("/photos/addphoto", methods=['GET'])
def add_photo():
    if not session.get('logged_in'):
        return redirect("/login")
    else:
        if not request.args.get('id'):
            return redirect("/photos")
        else:
            id = escape(request.args.get('id'))
            return render_template('addphoto.html', id=id)

[redacted]
        if person_id is " or 'photo' not in request.files:
            flash('No selected file!')
            return redirect("/photos/addphoto?id=" + person_id)
        else:
            file = request.files['photo']
            if file.filename == ":
                flash('No selected file')
                return redirect("/photos/addphoto?id=" + person_id)
```

```
if file and allowed_file(file.filename):
    query = Person.query.filter_by(user_id=session.get('user_id'), person_id=person_id)
    result = query.first()
    if result:
        filename = random_filename_generator()
        while Photo.query.filter_by(path=filename).limit(1).first() is not None:
            filename = random_filename_generator()
```

[redacted]

```
        file.save(path)

        face_num = face_detect.face_num(path)

        if face_num == 1:
            photo = Photo(filename, person_id)
            db.session.add(photo)
            db.session.commit()
        elif face_num == 0:
            if os.path.exists(path):
                os.remove(path=path)
            flash('No faces on photo!')
            return redirect("/photos/addphoto?id=" + person_id)
        else:
            if os.path.exists(path):
                os.remove(path=path)
            flash('More then one face on photo!')
            return redirect("/photos/addphoto?id=" + person_id)

        return redirect("/photos?id=" + person_id)
    else:
```

```
                    flash('Incorrect person!')
                    return redirect("/photos?id=" + person_id)
            else:
                flash('File not allowed')
                return redirect("/photos/addphoto?id=" + person_id)
```

1.4. persons.py

```
# coding=utf-8
```

[redacted]

```
from DBEngine import db
from config import app
import shutil
import os
from MLCore.classifier_controller import retrain_classifier
from multiprocessing import Pool

persons_blueprint = Blueprint('persons', __name__)



@persons_blueprint.route("/persons", methods=['GET'])
def persons():
    if not session.get('logged_in'):
        return redirect("/login")
    else:
        if request.args.get('id') and request.args.get('del'):
            person_id = request.args.get('id')

            query = Person.query.filter_by(user_id=session.get('user_id'), person_id=person_id)
            result = query.first()
    if result:
        db.session.query(Photo).filter(Photo.person_id == result.person_id).delete()
        db.session.commit()
        db.session.query(Person).filter(Person.person_id == person_id).delete()
        db.session.commit()

        path = app.config['UPLOAD_FOLDER'] + str(person_id)

        if os.path.exists(path):
            shutil.rmtree(path=path)

    return redirect("/persons")
```

[redacted]

```
    # pool.apply_async(retrain_classifier, [session.get('user_id')])
    retrain_classifier(session.get('user_id'))

    return redirect("/persons")
else:
    query = Person.query.filter_by(user_id=session.get('user_id'))
    result = query.all()

    if result is not None:
        persons = []
        i = 1
        for res in result:
            persons.append(WebPerson(str(i), res.name, res.surname, res.is_train, res.person_id))
            i += 1
        query = Classifier.query.filter_by(user_id=session.get('user_id'))
        result = query.first()
```

```
            return render_template("persons.html", persons=persons, is_ready=result.is_ready)
        else:
            return redirect("/login")



@persons_blueprint.route("/persons/addperson", methods=['GET'])
def add_person():
    if not session.get('logged_in'):
        return redirect("/login")
    else:
        return render_template('addperson.html')


[redacted]
        surname = str(escape(request.form['surname']))

        if name is '' or surname is '':
            flash('fill all fields!')
            return redirect("/persons/addperson")
        else:
            person = Person(name=name, surname=surname, user_id=session['user_id'])
            db.session.add(person)
            db.session.commit()
            return redirect("/persons")
```

1.5. login.py

```
# coding=utf-8

from flask import flash, redirect, render_template, request, session, abort
from DBEngine.dbmodel import User
from flask import Blueprint

login_blueprint = Blueprint('login', __name__)


@login_blueprint.route("/login", methods=['GET'])
def login():
    if not session.get('logged_in'):
        return render_template('login.html')
    else:
        return redirect("/persons")


[redacted]
POST_PASSWORD)
    result = query.first()
    if result:
        session['logged_in'] = True
        session['user_id'] = int(result.user_id)
        session['username'] = result.username
    else:
        flash('Wrong password!')
    return login()


@login_blueprint.route("/logout")
def logout():
    session['logged_in'] = False
    return login()
```

1.6. model.py

```
# coding=utf-8


class WebPerson:
    num = str()
    name = str()
    surname = str()
    is_train = bool()
    id = int()
```

[redacted]

```
class WebPhoto:
    num = str()
    path = str()
    is_train = bool()
    id = str()

    def __init__(self, num, path, is_train, id):
        self.num = num
        self.path = path
        self.is_train = is_train
        self.id = id
```

1.7. MindFaceKerasSmall2V1.py

```
from keras.layers import Activation
from keras.layers import AveragePooling2D
from keras.layers import BatchNormalization
from keras.layers import Convolution2D
from keras.layers import Dense
from keras.layers import Input
from keras.layers import Lambda
from keras.layers import MaxPooling2D
from keras.layers import Reshape
from keras.layers import ZeroPadding2D
from keras.layers import concatenate
from keras.models import Model

from MLCore.Torch2KerasConverter.utils import lrn, sqrt, square, mulConstant, l2Normalize



def create_model():
    inp = Input(shape=(96, 96, 3))
    x = ZeroPadding2D(padding=(3, 3))(inp)
    [redacted]
    x = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(x)
    x = Activation('relu')(x)
    x = ZeroPadding2D(padding=(1, 1))(x)
    x = Convolution2D(192, (3, 3), strides=(1, 1))(x)
    x = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(x)
    x = Activation('relu')(x)
    x = Lambda(lrn, arguments={'size': 5, 'alpha': 0.0001, 'beta': 0.75})(x)
    x = ZeroPadding2D(padding=(1, 1))(x)
    x = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)


    # Inception module
```

```
inception_0 = Convolution2D(96, (1, 1), strides=(1, 1))(x)
```

[redacted]

```
inception_1 = Convolution2D(16, (1, 1), strides=(1, 1))(x)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
inception_1 = Activation('relu')(inception_1)
inception_1 = ZeroPadding2D(padding=(2, 2))(inception_1)
inception_1 = Convolution2D(32, (5, 5), strides=(1, 1))(inception_1)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
inception_1 = Activation('relu')(inception_1)


inception_2 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)
inception_2 = Convolution2D(32, (1, 1), strides=(1, 1))(inception_2)
inception_2 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_2)
inception_2 = Activation('relu')(inception_2)
inception_2 = ZeroPadding2D(padding=((3, 4), (3, 4)))(inception_2)


inception_3 = Convolution2D(64, (1, 1), strides=(1, 1))(x)
inception_3 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_3)
inception_3 = Activation('relu')(inception_3)


x = concatenate([inception_0, inception_1, inception_2, inception_3, ], axis=3)


# Inception module
inception_0 = Convolution2D(96, (1, 1), strides=(1, 1))(x)
```

```
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
inception_0 = ZeroPadding2D(padding=(1, 1))(inception_0)
inception_0 = Convolution2D(128, (3, 3), strides=(1, 1))(inception_0)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
```

[redacted]

```
inception_2 = Lambda(square)(x)
inception_2 = AveragePooling2D(pool_size=(3, 3), strides=(3, 3))(inception_2)
inception_2 = Lambda(mulConstant, arguments={'const': 9})(inception_2)
inception_2 = Lambda(sqrt)(inception_2)
inception_2 = Convolution2D(64, (1, 1), strides=(1, 1))(inception_2)
inception_2 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_2)
inception_2 = Activation('relu')(inception_2)
inception_2 = ZeroPadding2D(padding=((4, 4), (4, 4)))(inception_2)


inception_3 = Convolution2D(64, (1, 1), strides=(1, 1))(x)
inception_3 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_3)
inception_3 = Activation('relu')(inception_3)


x = concatenate([inception_0, inception_1, inception_2, inception_3, ], axis=3)
```

```
# Inception module
inception_0 = Convolution2D(128, (1, 1), strides=(1, 1))(x)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
inception_0 = ZeroPadding2D(padding=(1, 1))(inception_0)
inception_0 = Convolution2D(256, (3, 3), strides=(2, 2))(inception_0)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
```

[redacted]

```
inception_2 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)
inception_2 = ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_2)


x = concatenate([inception_0, inception_1, inception_2, ], axis=3)


# Inception module
inception_0 = Convolution2D(96, (1, 1), strides=(1, 1))(x)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
inception_0 = ZeroPadding2D(padding=(1, 1))(inception_0)
inception_0 = Convolution2D(192, (3, 3), strides=(1, 1))(inception_0)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
```

```
inception_1 = Convolution2D(32, (1, 1), strides=(1, 1))(x)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
inception_1 = Activation('relu')(inception_1)
inception_1 = ZeroPadding2D(padding=(2, 2))(inception_1)
inception_1 = Convolution2D(64, (5, 5), strides=(1, 1))(inception_1)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
inception_1 = Activation('relu')(inception_1)


[redacted]
inception_2 = ZeroPadding2D(padding=((2, 2), (2, 2)))(inception_2)


inception_3 = Convolution2D(256, (1, 1), strides=(1, 1))(x)
inception_3 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_3)
inception_3 = Activation('relu')(inception_3)


x = concatenate([inception_0, inception_1, inception_2, inception_3, ], axis=3)


# Inception module
inception_0 = Convolution2D(160, (1, 1), strides=(1, 1))(x)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
inception_0 = ZeroPadding2D(padding=(1, 1))(inception_0)
inception_0 = Convolution2D(256, (3, 3), strides=(2, 2))(inception_0)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
```

```
inception_0 = Activation('relu')(inception_0)

[redacted]
inception_1 = Activation('relu')(inception_1)


inception_2 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)
inception_2 = ZeroPadding2D(padding=((0, 1), (0, 1)))(inception_2)


x = concatenate([inception_0, inception_1, inception_2, ], axis=3)


# Inception module
inception_0 = Convolution2D(96, (1, 1), strides=(1, 1))(x)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)
inception_0 = ZeroPadding2D(padding=(1, 1))(inception_0)
inception_0 = Convolution2D(384, (3, 3), strides=(1, 1))(inception_0)
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)


inception_1 = Lambda(square)(x)
inception_1 = AveragePooling2D(pool_size=(3, 3), strides=(3, 3))(inception_1)
inception_1 = Lambda(mulConstant, arguments={'const': 9})(inception_1)
inception_1 = Lambda(sqrt)(inception_1)
inception_1 = Convolution2D(96, (1, 1), strides=(1, 1))(inception_1)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
```

```
inception_1 = Activation('relu')(inception_1)
inception_1 = ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_1)


inception_2 = Convolution2D(256, (1, 1), strides=(1, 1))(x)
inception_2 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_2)
inception_2 = Activation('relu')(inception_2)


x = concatenate([inception_0, inception_1, inception_2, ], axis=3)
x = Reshape(target_shape=(3, 3, 736, ))(x)
```

[redacted]

```
inception_0 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_0)
inception_0 = Activation('relu')(inception_0)


inception_1 = MaxPooling2D(pool_size=(3, 3), strides=(2, 2))(x)
inception_1 = Convolution2D(96, (1, 1), strides=(1, 1))(inception_1)
inception_1 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_1)
inception_1 = Activation('relu')(inception_1)
inception_1 = ZeroPadding2D(padding=((1, 1), (1, 1)))(inception_1)


inception_2 = Convolution2D(256, (1, 1), strides=(1, 1))(x)
inception_2 = BatchNormalization(axis=3, momentum=0.1, epsilon=1e-05)(inception_2)
inception_2 = Activation('relu')(inception_2)
```

```
    x = concatenate([inception_0, inception_1, inception_2, ], axis=3)
    x = AveragePooling2D(pool_size=(3, 3), strides=(1, 1))(x)
    x = Reshape(target_shape=(736, ))(x)
    x = Reshape(target_shape=(736, ))(x)
    x = Dense(128)(x)
    x = Lambda(l2Normalize, arguments={'axis': 1})(x)

    MindFaceKerasSmall2V1 = Model(inputs=[inp], outputs=x)
    MindFaceKerasSmall2V1.summary()

    return MindFaceKerasSmall2V1
```

1.8. face_detect.py

[redacted]

```
face_detector = dlib.get_frontal_face_detector()
face_aligner = AlignDlib(predictor_face_landmark_model)



def face_num(file_name):
    image = io.imread(file_name)
    detected_faces = face_detector(image, 1)
    num = len(detected_faces)
    return num



def rewrite_alignet_resize96_face(file_name):
    image = io.imread(file_name)
    detected_faces = face_detector(image, 1)
    face_rect = detected_faces[0]
    alignedFace = face_aligner.align(534, image, face_rect,
landmarkIndices=AlignDlib.OUTER_EYES_AND_NOSE)
    alignedFace = cv2.resize(alignedFace, (96, 96))
    cv2.imwrite(file_name, cv2.cvtColor(alignedFace, cv2.COLOR_RGB2BGR))



def rewrite_alignet_resize96_face_image(image1):
[redacted]
        alignedFace = cv2.resize(alignedFace, (96, 96))
        return alignedFace
    else:
        return None
```

1.9. classifier_controller.py

```
from DBEngine import db
from DBEngine.dbmodel import Classifier
from DBEngine.dbmodel import Photo
from DBEngine.dbmodel import Person
from MLCore.face_detect import rewrite_alignet_resize96_face
import cv2
import numpy as np
from MLCore.MindFaceKerasSmall2V1 import create_model
from config import model_path
from MLCore.classifier import create_classifier
from config import clf_path, clf_path_local, reload_clf, reload_clf_local, retrain_bin_classifier
# from config import main_clf
import pickle
from MLCore.SVMBinaryClassificator.SVMBinaryClassificator import SVMBinaryClassificator
```

```
def retrain_classifier(user_id):
    global main_clf
    global main_bin_clf
    photos = get_unprocess_photos(user_id)
    process_faces(photos)

    photos = get_all_photos(user_id)
    code_faces(photos)

    vecs = get_marked_vecs(user_id)
    codes = []
    labels = []
    for vec in vecs:
        codes.append(vec[0])
        labels.append(vec[1])
```

[redacted]

```
    retrain_bin_classifier()

    for person in Person.query.filter_by(user_id=user_id):
        person.is_train = True
        db.session.commit()

    # vecs = get_marked_vecs_names(user_id)
    # codes = []
    # labels = []
    # for vec in vecs:
    #     codes.append(vec[0])
    #     labels.append(vec[1])
    # clf = create_classifier(codes, labels)
    # pickle.dump(clf, open(clf_path_local, 'wb'))
    # reload_clf_local()

    query = Classifier.query.filter_by(user_id=user_id)
    result = query.first()
    result.version = result.version + 1
    result.is_ready = True
    db.session.commit()
```

[redacted]

```
def get_all_photos(user_id):
    query = Photo.query.join(Photo.persons_list).filter_by(user_id=user_id)
    result = query.all()
    return result



def get_marked_vecs(user_id):
    query = Photo.query.join(Photo.persons_list).filter_by(user_id=user_id)\
        .with_entities(Photo.features, Photo.person_id)
    result = query.all()
    return result



def get_marked_vecs_names(user_id):
    query = Photo.query.join(Photo.persons_list).filter_by(user_id=user_id)\
        .with_entities(Photo.features, Person.surname)
    result = query.all()
    return result
```

```
def code_faces(photos):
    MindFaceKerasSmall2V1 = create_model()
    MindFaceKerasSmall2V1.load_weights(model_path)

    for photo in photos:
        path = photo.get_absolute_path()

        image = np.zeros((1, 96, 96, 3))
        img = cv2.imread(path, 1)
        img = img[..., ::-1]
        img = np.around(img / 255.0, decimals=12)
        image[0, :, :, :] = img
        vec = MindFaceKerasSmall2V1.predict_on_batch(image)

        photo.features = vec[0].tolist()
        db.session.commit()
```

1.10. classifier.py

```
from sklearn import svm



def create_classifier(vec, labels):
    clf = svm.LinearSVC()
    # clf = svm.SVC(decision_function_shape='ovo')
    clf.fit(vec, labels)
    return clf
```

1.11. AlighnDlib.py

```
import cv2
import dlib
import numpy as np


TEMPLATE = np.float32([
    (0.0792396913815, 0.339223741112), (0.0829219487236, 0.456955367943),
    (0.0967927109165, 0.575648016728), (0.122141515615, 0.691921601066),
    (0.168687863544, 0.800341263616), (0.239789390707, 0.895732504778),
    (0.325662452515, 0.977068762493), (0.422318282013, 1.04329000149),
    (0.531777802068, 1.06080371126), (0.641296298053, 1.03981924107),
    (0.738105872266, 0.972268833998), (0.824444363295, 0.889624082279),
    (0.894792677532, 0.792494155836), (0.939395486253, 0.681546643421),
    (0.96111933829, 0.562238253072), (0.970579841181, 0.441758925744),
    (0.971193274221, 0.322118743967), (0.163846223133, 0.249151738053),
    (0.21780354657, 0.204255863861), (0.291299351124, 0.192367318323),
    (0.367460241458, 0.203582210627), (0.4392945113, 0.233135599851),
[redacted]
    (0.565874114041, 0.618796581487), (0.607054002672, 0.60157671656),
    (0.252418718401, 0.331052263829), (0.298663015648, 0.302646354002),
    (0.355749724218, 0.303020650651), (0.403718978315, 0.33867711083),
    (0.352507175597, 0.349987615384), (0.296791759886, 0.350478978225),
    (0.631326076346, 0.334136672344), (0.679073381078, 0.29645404267),
    (0.73597236153, 0.294721285802), (0.782865376271, 0.321305281656),
    (0.740312274764, 0.341849376713), (0.68499850091, 0.343734332172),
    (0.353167761422, 0.746189164237), (0.414587777921, 0.719053835073),
    (0.477677654595, 0.706835892494), (0.522732900812, 0.717092275768),
    (0.569832064287, 0.705414478982), (0.635195811927, 0.71565572516),
```

[redacted]

```
TPL_MIN, TPL_MAX = np.min(TEMPLATE, axis=0), np.max(TEMPLATE, axis=0)
MINMAX_TEMPLATE = (TEMPLATE - TPL_MIN) / (TPL_MAX - TPL_MIN)



class AlignDlib:
    #: Landmark indices.
    INNER_EYES_AND_BOTTOM_LIP = [39, 42, 57]
    OUTER_EYES_AND_NOSE = [36, 45, 33]

    def __init__(self, facePredictor):
        assert facePredictor is not None

        self.detector = dlib.get_frontal_face_detector()
        self.predictor = dlib.shape_predictor(facePredictor)

    def getAllFaceBoundingBoxes(self, rgbImg):
        assert rgbImg is not None

        try:
            return self.detector(rgbImg, 1)
        except Exception as e:
            print("Warning: {}".format(e))
            # In rare cases, exceptions are thrown.
            return []

    def getLargestFaceBoundingBox(self, rgbImg, skipMulti=False):
        assert rgbImg is not None
        faces = self.getAllFaceBoundingBoxes(rgbImg)
        if (not skipMulti and len(faces) > 0) or len(faces) == 1:
            return max(faces, key=lambda rect: rect.width() * rect.height())
        else:
            return None

    def findLandmarks(self, rgbImg, bb):
        assert rgbImg is not None
        assert bb is not None

        points = self.predictor(rgbImg, bb)
        return list(map(lambda p: (p.x, p.y), points.parts()))
```

[redacted]

```
        if bb is None:
            bb = self.getLargestFaceBoundingBox(rgbImg, skipMulti)
            if bb is None:
                return

        if landmarks is None:
            landmarks = self.findLandmarks(rgbImg, bb)

        npLandmarks = np.float32(landmarks)
        npLandmarkIndices = np.array(landmarkIndices)

        a = npLandmarks[npLandmarkIndices]
        b = imgDim * MINMAX_TEMPLATE[npLandmarkIndices]
        c = MINMAX_TEMPLATE[npLandmarkIndices]
        H = cv2.getAffineTransform(npLandmarks[npLandmarkIndices],
                                   imgDim * MINMAX_TEMPLATE[npLandmarkIndices])
        thumbnail = cv2.warpAffine(rgbImg, H, (imgDim, imgDim))

        return thumbnail
```

1.12. SVMBinaryClassificator.py

```
from sklearn.cluster import KMeans
from MLCore.classifier import create_classifier
import numpy as np



class SVMBinaryClassificator:
    bin_clf = None
    next0 = None
    next1 = None
    cls0 = None
    cls1 = None
[redacted]
        print(labels)

        self.bin_clf = create_classifier(vec=vecs, labels=kmeans.labels_)

        mask0 = kmeans.labels_ == 0
        mask1 = kmeans.labels_ == 1

        uniq0 = np.unique(labels[mask0])
        if len(uniq0) > 1:
            self.next0 = SVMBinaryClassificator(vecs[mask0], labels[mask0])
        else:
            self.cls0 = uniq0[0]

        uniq1 = np.unique(labels[mask1])
        if len(uniq1) > 1:
            self.next1 = SVMBinaryClassificator(vecs[mask1], labels[mask1])
        else:
            self.cls1 = uniq1[0]

    def predict(self, vec):
        if self.bin_clf is not None:
            res = int(self.bin_clf.predict([vec]))

            if res == 0:
                if self.next0 is None:
```

[redacted]

```
                else:
                    return self.next1.predict(vec)
        else:
            return None
```

1.13. dbmodel.py

```
from DBEngine import db
from config import app



class User(db.Model):
    __tablename__ = 'users'

    user_id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String)
    password = db.Column(db.String)

    def __init__(self, username, password):
        self.username = username
        self.password = password


class Person(db.Model):
    __tablename__ = 'persons'
```

[redacted]

```
    def __init__(self, name, surname, user_id):
        self.name = name
        self.surname = surname
        self.is_train = False
        self.user_id = user_id


class Photo(db.Model):
    __tablename__ = 'photos'

    photo_id = db.Column(db.Integer, primary_key=True)
    path = db.Column(db.String)
    is_train = db.Column(db.Boolean)
    features = db.Column(db.ARRAY(db.Float))
    person_id = db.Column(db.Integer, db.ForeignKey('persons.person_id'))

    persons_list = db.relationship("Person", foreign_keys=[person_id])
    def __init__(self, path, person_id):
        self.path = path
        self.is_train = False
        self.features = None
        self.person_id = person_id

    def get_absolute_path(self):
        return app.config['UPLOAD_FOLDER'] + str(self.person_id) + '/' + self.path + '.png'



class Classifier(db.Model):
    __tablename__ = 'classifiers'
```

[redacted]

```
        self.user_id = user_id
        self.is_ready = True
```

1.14. vector.py

```
# coding=utf-8


from flask import flash, redirect, render_template, request, session, abort
from DBEngine.dbmodel import User
from flask import Blueprint
import json
from config import pred_clf_vec


vector_blueprint = Blueprint('vector', __name__)



@vector_blueprint.route("/api/vector", methods=['POST'])
```

```
def vector():
    if request.json.get('vector'):
        vec = request.json.get('vector')

        pers = pred_clf_vec(vec)

        res = {"person": str(pers)}
        res = json.dumps(res)

        return res
    else:
        abort(400)
```

1.15. load.py

```
# coding=utf-8
```

[redacted]

```
load_blueprint = Blueprint('load', __name__)



@load_blueprint.route("/api/load", methods=['POST'])
def load():
    if request.json.get('load'):
        return redirect('static/classifier_local.sav')
    else:
        abort(400)
```

1.16. image.py

```
# coding=utf-8


from flask import flash, redirect, render_template, request, session, abort, escape
[redacted]
from config import model_path



# image_blueprint = Blueprint('getad', __name__)
#
#
#
#
# @image_blueprint.route("/api/image", methods=['POST'])
# def image():
#     global MindFaceKerasSmall2V1
#     image_data = request.get_data()
#     image = Image.open(io.BytesIO(image_data)).convert('RGB')
#
#     img = rewrite_alignet_resize96_face_image(np.array(image))
#
#     image = np.zeros((1, 96, 96, 3))
#     img = img[..., ::-1]
#     img = np.around(img / 255.0, decimals=12)
#     image[0, :, :, :] = img
#     vec = MindFaceKerasSmall2V1.predict_on_batch(image)
#
#     print("1")
```

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					