口罩佩戴识别检测系统软件详细设计文档

目录

1. 引言

1.1. 编写目的

本系统介绍了在 Django 环境下采用“自上而下地总体规划，自下而上地应用开发”的策略开发的一个口罩佩戴识别检测系统的过程。

1.2. 总体背景

大面积新冠疫情的爆发，使得人群聚集密集的公共场所（例如机场、高铁站、商场等等）对人员口罩检测需要也越来越多，基于目前新冠疫情国内外的形式仍很严峻，故研发口罩佩戴识别检测系统非常必要，对比市面上的类似产品，我们计划开发一款基于神经网络模型、能够自动准确识别图片中人物口罩佩戴情况的跨平台系统（网站+App+客户端+小程序），来满足市场的迫切需求。

1.3. 业务背景

1. **基本功能**：使用者可以通过上传一系列图片或者视频，该系统可以在较短时间内抓捕到图片中的人脸以及识别人脸上是否佩戴口罩，然后反馈给用户结果。

2. **拓展功能**：应用在大型商场、超市、车站（地铁站和火车站）、小区入口等人流密集场景，流动人群在单人通过通道时，摄像头通过抓捕到人脸，拍照或者录像，传入系统内进行人脸口罩识别，将信息反馈到后台。若行人没有佩戴口罩，设备会进行提示。

1.4. 定义

一方面通过对用户上传的图片和视频进行人脸抓捕、口罩识别以及反馈结果；另一方面通过外部设备摄像头进行行人人脸抓捕、拍照或者摄像、口罩识别、结果反馈后台。

2. 总体设计

2.1. 需求设定

1. 用户可以在界面进行注册和登录功能，拥有属于自己的空间，所有进行的图片和视频口罩检测结果均存储在自己的账户内。

2. 系统自动通过摄像头识别人脸，对人脸图像进行拍摄或者录影，传递到后端进行口罩识别，将结果反馈给用户以及后台

3. 可以将检测的图片保存起来，将数据部分解码成图片后存储到服务器，同时将图片的访问路径存储到数据库。

4. 用户可以在事后进行历史图片查询，该系统通过获取用户 id 后到数据库查询，将多个图片存入字典中，再打包成 json 返回前端。

2.2. 运行环境

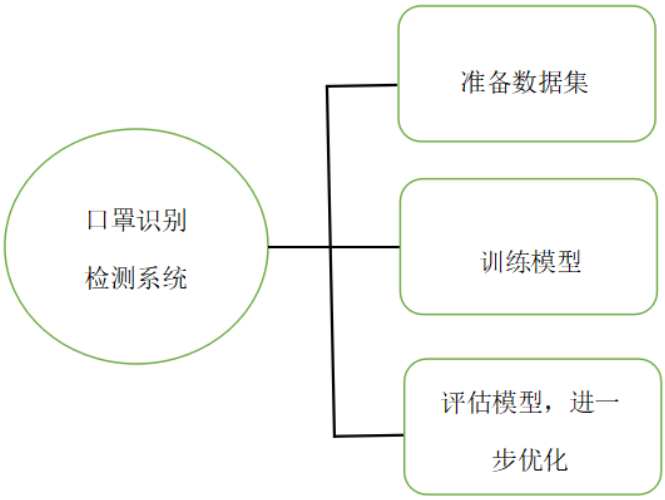
1. 浏览器
2. Android
3. 微信小程序

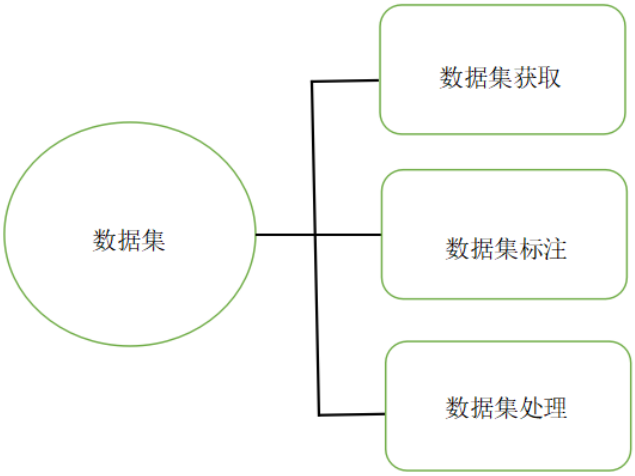
2.3. 基本设计概要和处理流程

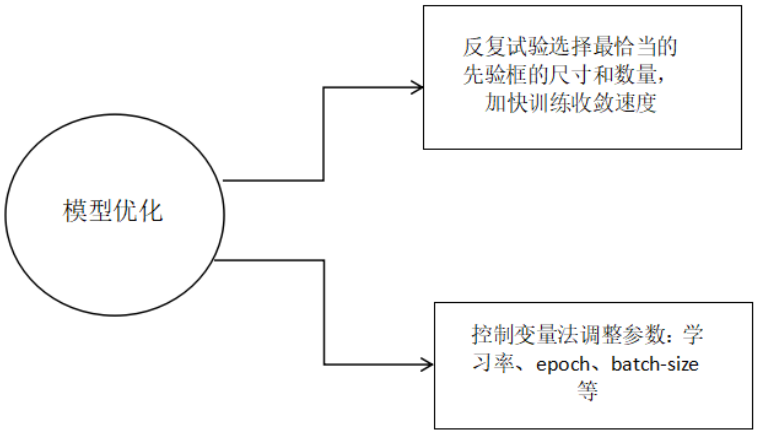
本系统是基于神经网络模型、能够自动准确识别图片和视频中人物口罩佩戴情况的跨平台系统（网站+App+客户端+小程序）。项目前端部分，设计网页 UI ，构建项目原型，学习和使用 uni-app 框架，构建跨平台应用；项目后端部分，应用 Django 框架和 mysql 数据库，部署服务器端，接收并响应前端请求；识别算法，在 pytorch 环境下，运用 YOLOV3 框架，进行数据集准备工作，使用 colab 进行模型训练，并在服务器端加载模型进行图片检测。

此外，开发本地桌面应用。使用 pyqt 设计 GUI，结合 YOLOV3 实现了实时视频检测功能。

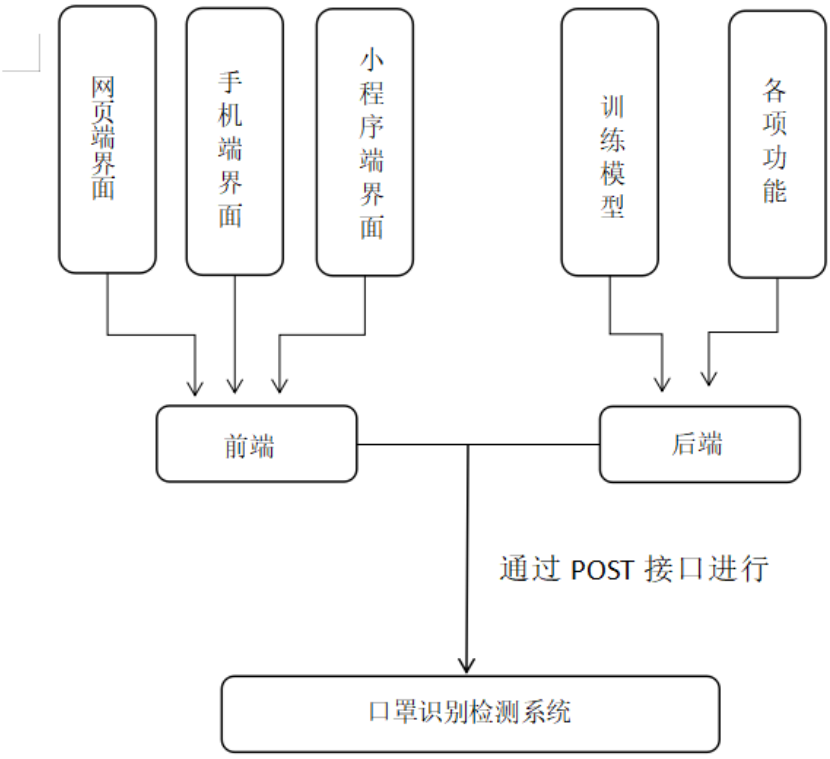
处理流程图：







2.4. 主模块结构图



2.5. 功能需求与程序代码的联系

1. 注册功能（核心代码）

```
# 用户注册
userName = request.POST.get('username')
user = models.User.objects.filter(username=userName).first()
#密码加密，数据库存储安全
password = hashlib.md5(request.POST.get('password').encode(encoding='UTF-8')).hexdigest()
#判断用户是否存在
if user != None:
    result = {'error_code': 1, 'msg': '该用户已存在'}
else:
    user = models.User(username=userName, password=password)
    user.save()
    User = models.User.objects.filter(username=userName).first()
    result = {'error_code': 0, 'msg': '注册成功', 'data': {'userid': User.userid, 'username': User.username}}
return HttpResponse(json.dumps(result), content_type="application/json")
```

2. 登录功能（核心代码）

```
# 用户登录
userName = request.POST.get('username')
user = models.User.objects.filter(username=userName).first()
if user == None:
    result = {'error_code': 1, 'msg': '该账号不存在，请注册'}
else:
    password = hashlib.md5(request.POST.get('password').encode(encoding='UTF-8')).hexdigest()
    if user.password == password:
        result = {'error_code': 0, 'msg': '登录成功', 'data': {'userid': user.userid, 'username': user.username}}
    else:
        result = {'error_code': 2, 'msg': '密码错误，请重新输入'}
return HttpResponse(json.dumps(result), content_type="application/json")
```

3. 图片接收功能（核心代码）

```
# 根据base64格式，只截取数据部分
piece = 0
for i in range(50):
    if fileFromPost[i] == ',':
        piece = i
        break
file = fileFromPost[piece:len(fileFromPost):1]
# 命名文件的方式
filename = str(hash(file[2000:2010:1]))
imageurl = "yolov3-archive\\source\\" + filename + ".jpg"
imgdata = base64.b64decode(file)  # 解码成图片并存储
with open(imageurl, "wb") as f:
    f.write(imgdata)
    f.close()
result = detect_image(userid, imageurl)
```

4. 图片检测功能（核心代码）

```
# 读取存储图片
image = cv2.imdecode(np.fromfile(os.path.join(imageurl), dtype=np.uint8), cv2.IMREAD_COLOR)
with torch.no_grad():
    outImage = yolo.detect_image(image)
# 图片压缩
imSize = outImage.shape
rate = float(imSize[0]) / (imSize[0] + imSize[1])
outImage = cv2.resize(outImage, (int(1000 - 1000 * rate), int(rate * 1000)))
# 保存图片
save_path = os.path.join("result", imageurl.split("\\")[-1])
cv2.imwrite(save_path, outImage)
# 写入数据库
if userid != -1:
    add_image(save_path, userid, datetime.datetime.now())
# 读取结果并转换为base64格式进行传输
with open(save_path, "rb") as f:
    image_data = f.read()
    image_data = base64.b64encode(image_data).decode()
    f.close()
```

5. 用户历史图片查询功能（核心代码）

```
# 获取用户
user_id = int(request.POST.get('userid'))
# 根据用户获取图片
_images = models.Image.objects.filter(userid=user_id)
images = []
length = 0
# 遍历插入图片到字典中
for _image in _images:
    length += 1
    f = open(_image.imageurl, 'rb')
    image_pure_data = base64.b64encode(f.read()).decode()  # 强转成图片格式
    image_data = '' + image_pure_data
    image = {'url': _image.imageurl, "image": image_data}
    images.append(image)
    f.close()
result = {"user_id": user_id, "images": images, "length": length}
# 封装成json传输
return HttpResponse(json.dumps(result), content_type="application/json")
```

2.6. 过程中碰到的问题

1. 将跨平台版完全配置在服务器上，并且发布微信小程序、网页版和安卓版；

2. 研究如何将代码打包；

3. 进行桌面版（实时视频检测）的开发以及图片检测功能的性能优化；

4. 如何进行实时视频的解析与处理（人脸检测+口罩佩戴识别）。

3. 接口设计

3.1. 服务端接口

序号	接口名称	接口地址
1	注册	http://www.oucwechat.com:8000/user_registered/
2	登录	http://www.oucwechat.com:8000/user_getin/
3	上传图片	http://www.oucwechat.com:8000/receive/
4	查询历史记录	http://www.oucwechat.com:8000/user_images/

3.2. 详细说明

接口名称（1）	注册
接口地址	http://www.oucwechat.com:8000/user_registered/
输入方式	POST/json
输入数据	username：用户名 password：密码 confirmPassword：确认密码
返回方式	Ajax/json

返回数据	注册成功：
	{
	"error_code": 0,
	"msg": "注册成功",
	"data": {
	"userid": "3",
	"username": "name",
	}
	}

	注册失败：
	{
	"error_code": 1,
	"msg": "该用户已存在",
	}

接口名称（2）	登录
接口地址	http://www.oucwechat.com:8000/user_getin/
输入方式	POST/json
输入数据	phone：手机号 password：密码
返回方式	Ajax/json

返回数据	该账号不存在
	{
	"error_code": 1,
	"msg": "账号不存在"
	}

	密码错误
	{
	"error_code": 2,
	"msg": "密码错误"
	}

	登录成功：
	{
	"error_code": 0,
	"msg": "登录成功",
	"data": {
	"userid": null,
	"username": "name",
	}
	}

接口名称（3）	上传图片接口
接口地址	http://www.oucwechat.com:8000/receive/
输入方式	POST/json
输入数据	userid：用户 id filePath：图片文件
返回方式	Ajax/json

返回数据	{
	"resimage":处理后的图片文件
	}

接口名称（4）	查询历史记录

接口地址	http://www.oucwechat.com:8000/user_images/
输入方式	POST/json
输入数据	userid：用户 id
返回方式	Ajax/json

返回数据	参数不足:
	{
	"error_code": 1,
	"msg": "参数不足：userid"
	}
	获取成功：
	{
	"images": [
	{
	"image": 处理后的图片文件
	}
	]
	}

模块间接口采用数据耦合方式，通过参数表达传送数据，交换信息。

4. 运行设计

4.1. 运行模块结合

系统的运行模块组合为程序多窗口的运行环境，各部分模块在软件运行过程中能快速地交换各自信息以及处理从外部接收的数据。

4.2. 运行控制

系统运行时有良好交互界面，能基本满足用户的数据处理要求。

界面展示：

1. 网页端：（网页版演示地址：http://amoniaa.gitee.io/facemask-identify）

2. Android 端


Android端
主页
疫情信息页
个人中心页
主页
历史记录页
注册登录页

3. 小程序端


小程序端
主页
疫情信息页
个人中心页
主页
历史记录页
注册登录页

5. 系统数据结构设计

5.1. 逻辑结构

根据系统需求，把系统分为登录模块、用户模块

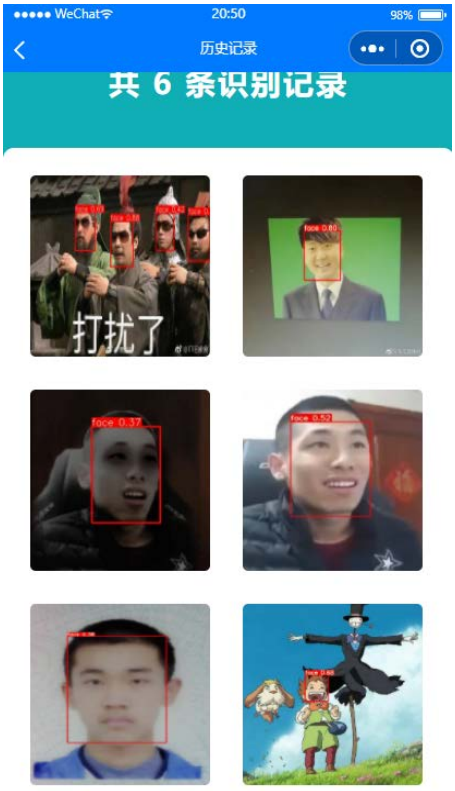
1. 登录模块：

2. 用户模块：

2.1 用户图片检测界面：


图片检测
口罩佩戴图片检测平台
选择图片
开始识别
图片检测
疫情信息
我的

2.2 用户历史记录查询


WeChat
20:50
98%
历史记录
共 6 条识别记录
打扰了

5.2. 物理结构

系统的物理结构具体由数据库来设计与生成。

user 表，用于存储用户信息

序号	字段名	类型	备注
1	userid	int(11)	用户id
2	username	varchar(50)	用户名
3	phone	varchar(11)	手机号
4	password	varchar(32)	密码（md5加密）
5	face_url	varchar(250)	用户头像地址

image 表，用于存储图片信息

序号	字段名	类型	备注
1	imageid	int(11)	图片id
2	imageurl	varchar(500)	图片链接
3	userid	int(11)	图片上传者id
4	send_timestamp	int(11)	图片上传的时间戳

5.3. 数据结构与程序关系

1. 前端：

调用后端图片检测接口，传入用户选择的图片，服务器经过计算识别后返回一张带有标记的图片。

代码如图所示：

```
post: function() {
    var that = this
    if (!this.image) {
        uni.showModal({
            title: '提示',
            content: '请选择一张图片！',
            success: function(res) {
                if (res.confirm) {
                    console.log('用户点击确定');
                    that.chooseImage()
                } else if (res.cancel) {
                    console.log('用户点击取消');
                }
            }
        });
    } else {
        uni.showLoading({
            title: '识别中，请稍后',
            icon: 'loading',
            mask: true
        });
        uni.request({
            url: 'http://www.oucsrdpwechat.com:8000/receive/',
            method: 'POST',
            header: {
                'content-type': 'application/x-www-form-urlencoded',
            },
            data: {
                filePath: this.base64,
                userid: getApp().globalData.userid
            },
            success: (res) => {
                this.resimage = '' + res.data.image;
                this.imageList.unshift(this.resimage);
                uni.hideLoading();
                uni.showToast({
                    title: '识别成功',
                    duration: 1000
                });
            }
        });
    }
},
```

2. **后端：**

图片检测（代码如图所示）

```
# 读取存储图片
image = cv2.imdecode(np.fromfile(os.path.join(imageurl), dtype=np.uint8), cv2.IMREAD_COLOR)
with torch.no_grad():
    outImage = yolo.detect_image(image)
# 图片压缩
imSize = outImage.shape
rate = float(imSize[0]) / (imSize[0] + imSize[1])
outImage = cv2.resize(outImage, (int(1000 - 1000 * rate), int(rate * 1000)))
# 保存图片
save_path = os.path.join("result", imageurl.split("\\")[-1])
cv2.imwrite(save_path, outImage)
# 写入数据库
if userid != -1:
    add_image(save_path, userid, datetime.datetime.now())
# 读取结果并转换为base64格式进行传输
with open(save_path, "rb") as f:
    image_data = f.read()
    image_data = base64.b64encode(image_data).decode()
    f.close()
```

用户历史图片查询（代码如图所示）

```
# 获取用户
user_id = int(request.POST.get('userid'))
# 根据用户获取图片
_images = models.Image.objects.filter(userid=user_id)
images = []
length = 0
# 遍历插入图片到字典中
for _image in _images:
    length += 1
    f = open(_image.imageurl, 'rb')
    image_pure_data = base64.b64encode(f.read()).decode()  # 强转成图片格式
    image_data = '' + image_pure_data
    image = {'url': _image.imageurl, "image": image_data}
    images.append(image)
    f.close()
result = {"user_id": user_id, "images": images, "length": length}
# 封装成json传输
return HttpResponse(json.dumps(result), content_type="application/json")
```

后端训练模型：

yolov3 github地址: https://github.com/ultralytics/yolov3

装载谷歌云端硬盘，数据集存放在谷歌云盘，另外数据集还需要进行处理，数据集处理代码可见于getSize.py,makeText.py,voc_label.py，其中getSize主要是用于获取图片的size的，因为我发现许多没有任何object存在的xml标注文件就连size都不给出了，其他部分文件太大了，等待后续更新

```
[ ]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

切换文件夹

```
[ ]: import os
     path = r'/content/drive/MyDrive'
     os.chdir(path)
```

```
[ ]: !ls
     !nvidia-smi
```

```
[ ]: !unzip yolov3-master.zip
```

```
[ ]: path = r'/content/drive/MyDrive/yolov3-archive'
     os.chdir(path)
```

```
[ ]: !ls
```

```
cfg          getSize.py  __pycache__       runs           utils
data         LICENSE     python            test.py        voc_label.py
detect.py    makeTxt.py  README.md         train.py       weights
Dockerfile   models.py   requirements.txt  tutorial.ipynb yolov3-tiny.pt
```

```
[ ]: !pip install -r requirements.txt
```

训练数据集，数据集是处理完后上传云端的

```
[ ]: !python train.py --data data/rbc.data --cfg cfg/yolov3-tiny.cfg --weights weights/yolov3-tiny.pt --epochs 20
```

Training

Start Training: `python3 train.py` to begin training after downloading COCO data with `data/get_coco2017.sh`. Each epoch trains on 117,263 images from the train and validate COCO sets, and tests on 5000 images from the COCO validate set.

Resume Training: `python3 train.py --resume` to resume training from `weights/last.pt`.

Plot Training: `from utils import utils; utils.plot_results()`

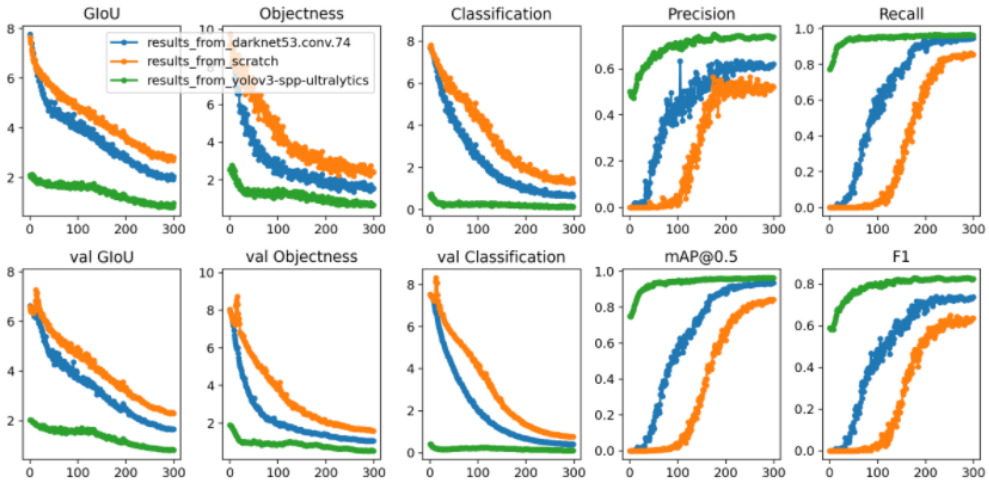


Image Augmentation

`datasets.py` applies OpenCV-powered (https://opencv.org/) augmentation to the input image. We use a **mosaic dataloader** to increase image variability during training.

6. 项目优势总结

1. 系统界面精简，操作简单，具有良好的交互性。
2. 受众群体广泛，在不同时间段、不同环境中都能方便使用。
3. 具有跨平台性，免安装，敏捷高效。
4. 搭配先进的算法，识别结果快速精准。
5. 能够节约大量的人力、物力和财力，为客户节省相应的成本。