程序说明与注释

17343030 高镇

运行环境

Python 3.8，pytorch 1.5，django 3.0，cuda 10，opencv 3（可选）

运行方式

训练模型

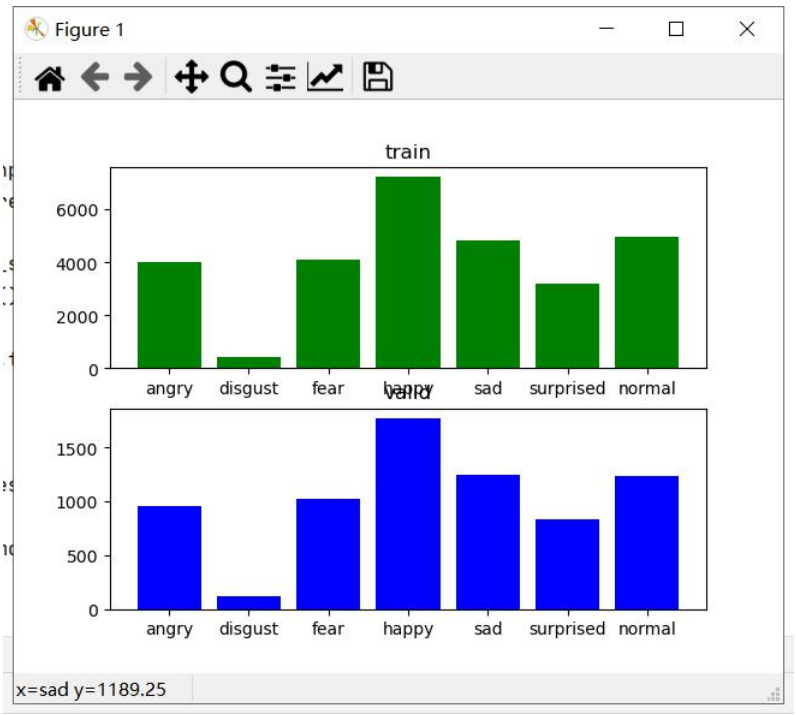
- 进入./ModelTraining；
- 下载 fer2013 数据集，将其中的 csv 文件命名为 fer2013.csv 放入 fer2013 文件夹（下载链接：链接：https://pan.baidu.com/s/1mOiGvRLAWEtYH8fyK_vKpg 提取码：d7hd）；
- 运行 save_image.py;
- 运行 transfer_learn.py;
- 目录下生成训练好的模型 resnet0.pth。

人机交互界面

- 进入./UserInterface；
- 将训练好的模型 pth 文件放入/hci/models（该目录下已经放入 resnet0.pth）；
- python manage.py runserver [IP 地址]:[端口号]；
- 使用浏览器登录对应的 ip 地址和端口；

效果展示

数据集统计

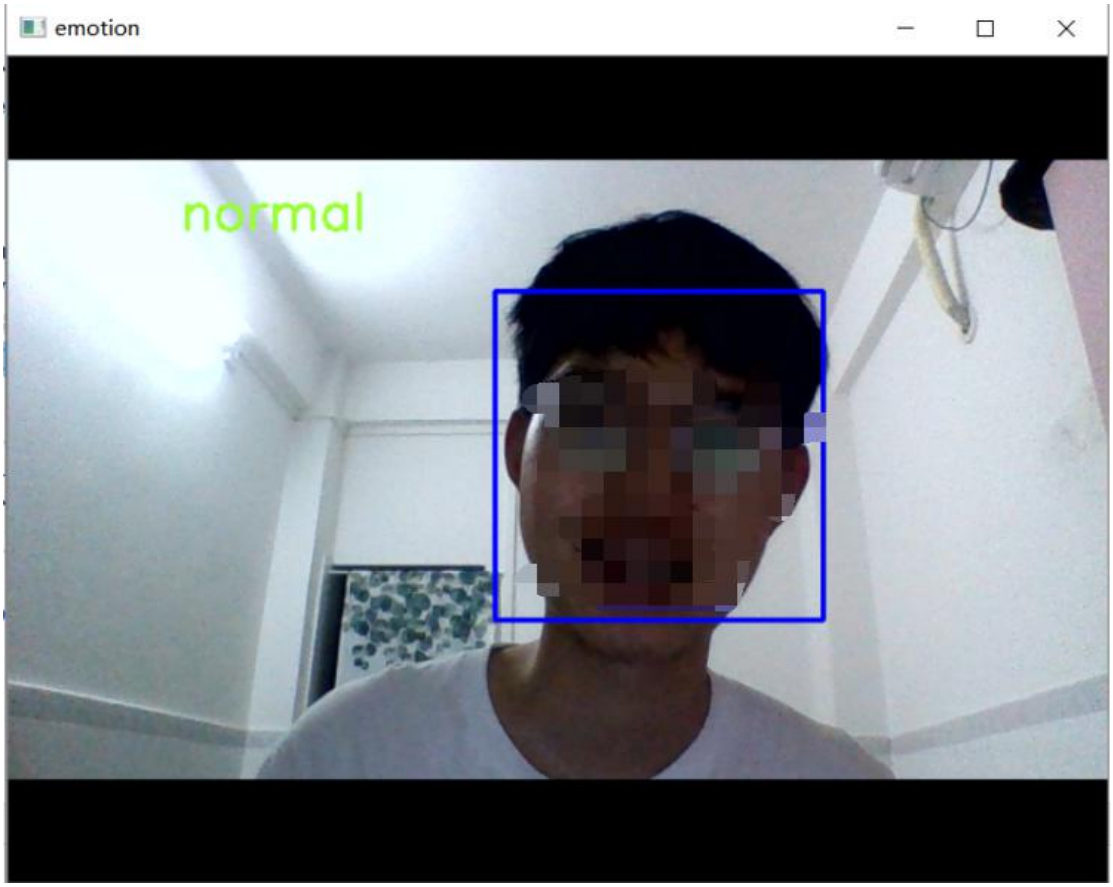


训练过程

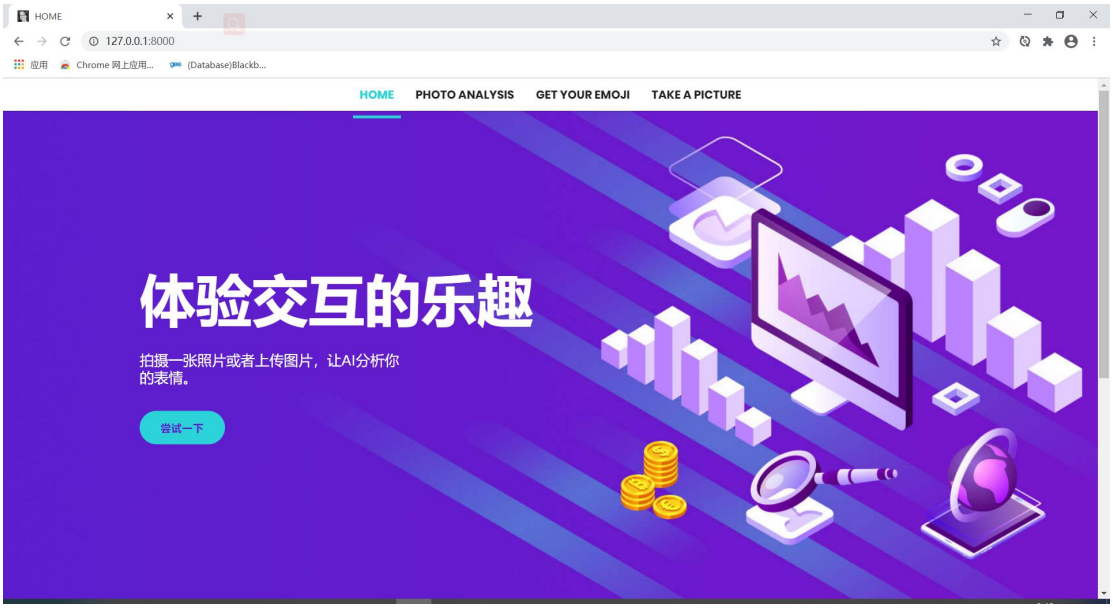
transfer_learn

```
----------
train Loss: 1.7758 Acc: 0.2848
val Loss: 1.7225 Acc: 0.3167

Epoch 2/14
----------
train Loss: 1.7004 Acc: 0.3222
val Loss: 1.6409 Acc: 0.3573

Epoch 3/14
----------
train Loss: 1.6361 Acc: 0.3526
val Loss: 1.5898 Acc: 0.3788

Epoch 4/14
----------
train Loss: 1.5853 Acc: 0.3809
val Loss: 1.5371 Acc: 0.4046

Epoch 5/14
----------
train Loss: 1.5396 Acc: 0.3999
val Loss: 1.5101 Acc: 0.4174

Epoch 6/14
----------
train Loss: 1.4977 Acc: 0.4193
val Loss: 1.4835 Acc: 0.4267

Epoch 7/14
----------
```

Opencv 实时检测表情



HCI 页面



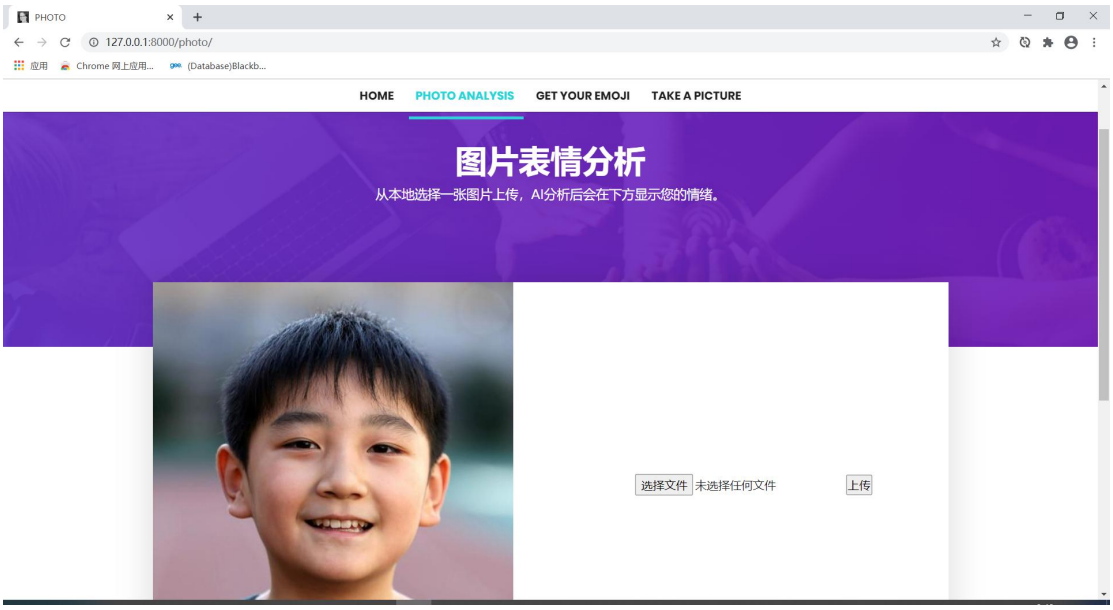
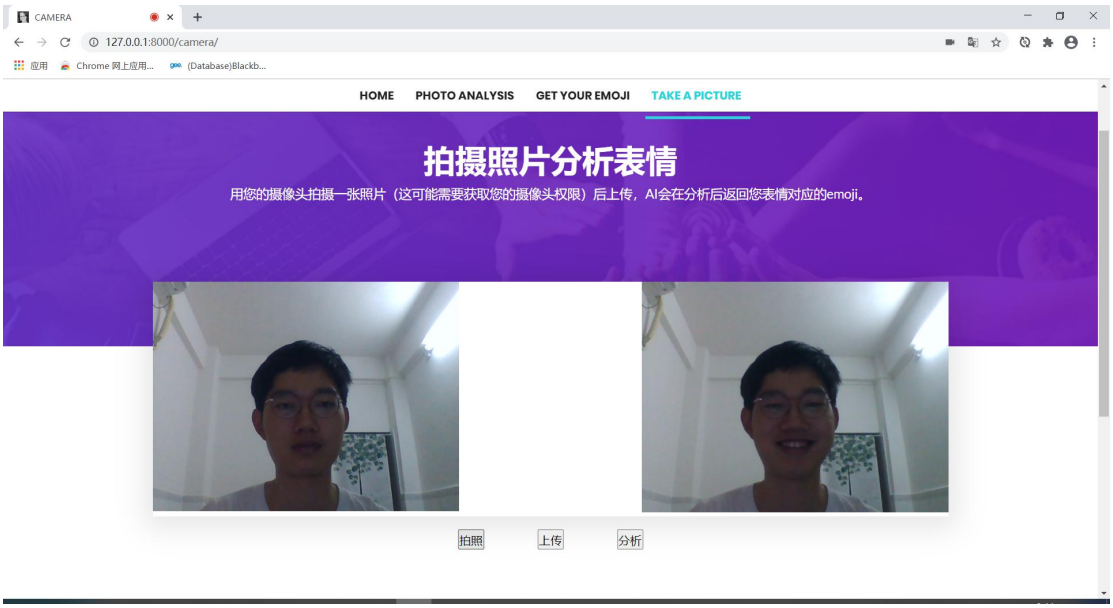
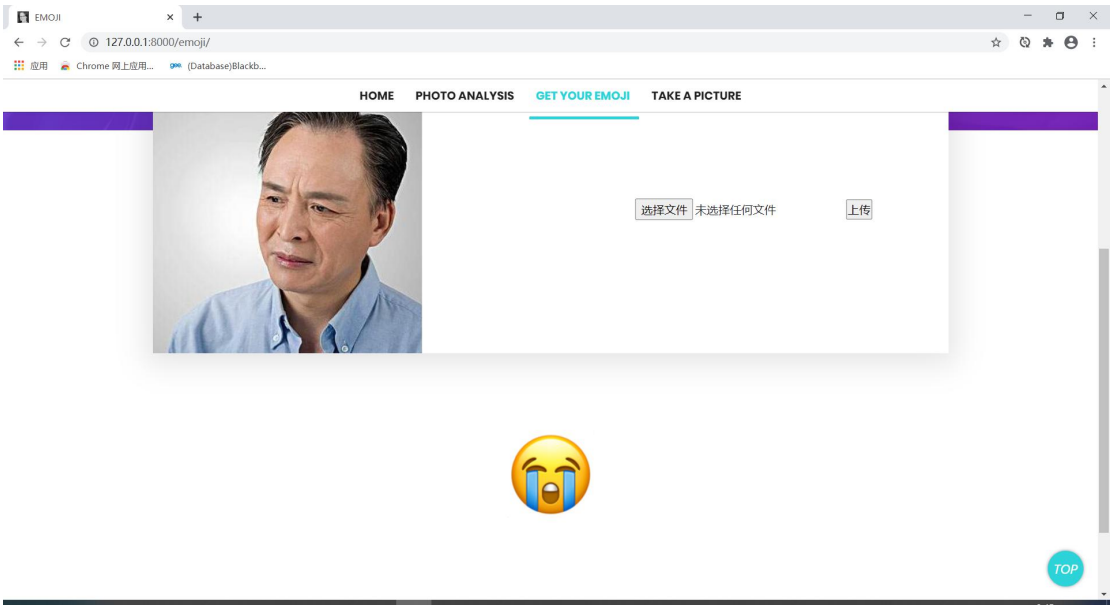


PHOTO
127.0.0.1:8000/photo/
应用
Chrome 网上应用...
(Database)Blackb...
HOME
PHOTO ANALYSIS
GET YOUR EMOJI
TAKE A PICTURE
图片表情分析
从本地选择一张图片上传，AI分析后会在下方显示您的情绪。
选择文件
未选择任何文件
上传


CAMERA
127.0.0.1:8000/camera/
应用
Chrome 网上应用...
(Database)Blackb...
HOME
PHOTO ANALYSIS
GET YOUR EMOJI
TAKE A PICTURE
拍摄照片分析表情
用您的摄像头拍摄一张照片（这可能需要获取您的摄像头权限）后上传，AI会在分析后返回您表情对应的emoji。
拍照
上传
分析


EMOJI
127.0.0.1:8000/emoji/
应用
Chrome 网上应用...
(Database)Blackb...
HOME
PHOTO ANALYSIS
GET YOUR EMOJI
TAKE A PICTURE
选择文件
未选择任何文件
上传
TOP

程序结构

./ModelTraining

./data

/train

/val

./fer2013

fer2013.csv

save_image.py

transfer_lean.py

camera.py

resnet0.pth

./UserInterface

./hci

/face_rec

/models

resnet0.pth

/uploadfiles

settings.py

urls.py

views.py

……

./statics

/css

/js

/images

/plugins

./templates

manage.py

红色部分为需要下载或运行后生成部分。

Modeltraining 为模型训练部分，使用 fer2013 数据集，save_image.py 会读取目录下的数据集 csv 文件，分类保存为训练数据和验证数据两个文件夹/train，/val，在每个目录下，按照数据对应的标签为文件存储（/0，/1……）。transfer_learn.py 加载 torchvision 中的 resnet，更改为单通道（fer2013 是灰度图），更改全连接层进行训练，将训练好的模型保存在目录下。

UserInterface 为 HCI 页面部分，使用 django 搭建服务框架，/templates 存放 django 模板，即各个页面的 html 文件，/statics 存放模板加载的静态文件，包括图片，css 文件，JavaScript 文件。/hci 为后台的主体部分，views.py 处理请求，urls.py 将 views 中的方法与 url，/face_rec 下为 views 中方法调用的本地 python 包，用于存储用户拍摄或上传的图片，加载模型分析表情。

程序详细说明

训练模型部分

save_image.py

```
def save_images():
    df = pd.read_csv(data_path)
    t_n = [1 for i in range(0,7)]
    v_n = [1 for i in range(0,7)]
    for index in range(len(df)):
        emotion = df.loc[index][0] #int represent the emotion
        image = df.loc[index][1] #a string of number represent the pixel of the image
        usage = df.loc[index][2] # is training or valid
        data_array = list(map(float, image.split())) #list
        data_array = np.asarray(data_array)
        image = data_array.reshape(48, 48)
        im = Image.fromarray(image).convert('L') #8bit grey picture
        if(usage=='Training'):
            t_p = os.path.join(train_path,str(emotion),'{}.jpg'.format(t_n[emotion]))
            im.save(t_p)
            t_n[emotion] += 1
        else:
            v_p = os.path.join(vaild_path,str(emotion),'{}.jpg'.format(v_n[emotion]))
            im.save(v_p)
            v_n[emotion] += 1
    return t_n, v_n
```

读取 csv 文件，按照每个数据对应的标签（0-6 共 7 个数字，代表 7 种表情）存储到对应的训练或验证目录下。另外两个函数功能为创建目录和可视化数据集统计情况。

transfer_learn.py

```
BATCH_SIZE=256
EPOCH=15
LR=0.001
DAMP_STEP=20 #after these step, the learning rate damping the amount
DAMP_RATE=0.1
data_dir = './data'
```

设置 Batch size，epoch 数，学习率，设置优化器隔些步学习率下降减少过拟合。

```
image_datasets = {x: datasets.ImageFolder(os.path.join(
    data_dir, x), data_transforms[x]) for x in ['train','val']}
#num_workers is the number of thread using to load data
dataloaders = {x: torch.utils.data.DataLoader(
    image_datasets[x], batch_size=BATCH_SIZE, shuffle=True, num_workers=4) for x in ['train', 'val']}
```

读入图片用来 PIL 的 ImageFolder，但这样读出来的是三通道。

```
data_transforms = {
    'train': transforms.Compose([
        transforms.Grayscale(),  # 使用ImageFolder存图片时默认扩展为了三通道，现在变成一通道
        transforms.RandomHorizontalFlip(),  # 随机翻转
        transforms.ColorJitter(brightness=0.5, contrast=0.5),  # 随机调整亮度和对比度
        transforms.ToTensor(),
        transforms.Normalize([0.485,], [0.229,])
    ]),
    'val': transforms.Compose([
        transforms.Grayscale(),
        transforms.ToTensor(),
        transforms.Normalize([0.485,], [0.229,])
    ])
}
```

用用 torchvision 的变换功能预处理图片，在输入模型训练前，先改成一通道的灰度图，然后随机翻转，调整亮度对比度，增加数据多样性，normalize 是所以预训练模型都要加的，具体的数值来源于 LeNet，这里就直接使用。

```
model_ft = models.resnet18(pretrained=True) #use resnet
#将resnet输入改成单通道
model_ft.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 7) #修改全连接层
```

训练 resnet 之前，更改输入，成为对应 fer2013 的接口，将全连接层输出改为 7，对应 7 个表情类别。

camera.py

```
while(cap.isOpened()):
    ret, frame = cap.read()
    frame = frame[:,::-1,:]#水平翻转，符合自拍习惯
    frame= frame.copy()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    face = face_cascade.detectMultiScale(gray,1.1,3)
    img = frame
    if(len(face)>=1):
        (x,y,w,h)= face[0]
        cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)
        img = frame[:][y:y+h,x:x+w]
    # 如果分类器能捕捉到人脸，就对其进行剪裁送入网络，否则就将整张图片送入
    img = Image.fromarray(img)
    img = transforms(img)
    img = img.reshape(1,1,48,48).cuda()
    pre = model(img).max(1)[1].item()
    frame = cv2.putText(frame, emotion[pre], (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (55,255,155), 2)
    #显示窗口第一个参数是窗口名，第二个参数是内容
    cv2.imshow('emotion', frame)
    if cv2.waitKey(1) == ord('q'):#按q退出
        break
```

opencv 调用电脑摄像头，这里用到了一个 opencv 自带的面部识别（我复制到了目录下面，安装了 opencv 的话应该在[your python environment]\Library\etc\haarcascades），摄像头翻转，然后把捕捉到的人脸裁剪成 48*48，如果没捕捉到人脸就整张图片送入。开着摄像头就不断输入模型预测表情，达到实时的效果。

人机交互界面部分

views.py

```
def home(request):
    return render(request, 'home.html')
def photo(request):
    context = {}
    context['subtitle'] = 'PHOTO EMOTION RECOGNITION'
    context['showimg'] = 'images/smile.jpg'
    if request.method == 'GET':
        return render(request, 'photo.html',context)
    if request.POST:
        myfile = request.FILES.get('face')
        showpath,filepath = photo_rec.save_img(myfile)
        context['showimg'] = showpath
        context['rlt'] = photo_rec.rec(filepath)
    return render(request, "photo.html", context)
```

这个是页面 photo 的处理函数，当服务器接到 request 请求时，如果是 get，直接渲染页面，context 是一个字典，里面存放着页面的 django 变量，render 函数可以不带这个参数。如果接到的请求是 post，说明用户上传了图片，这时调用同目录下包里 save_img 的把图片存储到服务器，再调用 rec 把图片输入训练好的模型，返回的是一个代表表情的字符串变量。

Django 中可以使用变量动态渲染模板，变量的形式是外套两个大括号。

```
{% get_static_prefix as STATIC_PREFIX %}
<img id="upload" src="{{ STATIC_PREFIX }}{{ emoji }}" />
```

Django 中加载静态文件需要使用{% static [加载静态文件的语句] %}，一开始因为这个我怎么也不能动态渲染图片，后来查手册知道可以加个静态的前缀，然后把变量放图片名字里，每次改图片名字。

```
def rec(filepath):
    modelpath = os.path.join(MODEL_DIR, 'resnet0.pth')
    model = torch.load(modelpath)
    use_cuda = torch.cuda.is_available()
    if use_cuda:
        model = model.cuda()
    else:
        print("cuda is not avaliable")
    img = pre_process(filepath).cuda()

    emotion = ["angry", "disgust", "fear", "happy", "sad", "surprised", "normal"]  #
    res = model(img).max(1)[1].item()
    emo = emotion[res]
    return emo
def rec_and_emoji(filepath):
    emo = rec(filepath)
    emo += 'emo.png'
    showpath = os.path.join(SHOW_DIR, emo)
    return showpath
```

rec 函数把服务器存储的图片（在 uploadfiles 目录下），输入模型，把结果对应成表情字符串，如果要返回 emoji，就把名字扩展成[表情]emo.png，比如 happyemo.png，emoji 被按照这种命名格式存储在 statics/images 目录下，动态渲染之后会从这个目录下加载 emoji 表情以显示。

Javascripts

```
function showImg(input) {
        var file = input.files[0];
        var reader = new FileReader()
        // 图片读取成功回调函数
        reader.onload = function(e) {
            document.getElementById('upload').src=e.target.result
        }
        reader.readAsDataURL(file)
}
```

在上传图片的时候，给 input 按钮加一个功能，让用户在选择好图片的时候就能立即看到这张图片，再决定要不要发送给后台。

```
function getUserMedia(constraints, success, error) {
    if (navigator.mediaDevices.getUserMedia) {
        //最新的标准API
        navigator.mediaDevices.getUserMedia(constraints).then(success).catch(error);
    } else if (navigator.webkitGetUserMedia) {
        //webkit核心浏览器
        navigator.webkitGetUserMedia(constraints, success, error)
    } else if (navigator.mozGetUserMedia) {
        //firfox浏览器
        navigator.mozGetUserMedia(constraints, success, error);
    } else if (navigator.getUserMedia) {
        //旧版API
        navigator.getUserMedia(constraints, success, error);
    }
}

let video = document.getElementById('video');
let canvas = document.getElementById('canvas');
let context = canvas.getContext('2d');
```

在浏览器上显示视频用到了 getUserMedia 的 API，页面上再创建一个 canvas，按下拍照以后，在 canvas 上绘制此刻的 video。

```
document.getElementById('capture').addEventListener('click', function () {
    context.drawImage(video, 0, 0, 425, 320);
})

function send(){
    var imgData = canvas.toDataURL("image/jpg",0.95);
    var formdata = new FormData();
    formdata.append("face", imgData);
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '/camera/', true);
    //xhr.setRequestHeader("Content-type", "application/json");
    xhr.send(formdata);
}
```

调用 canvas 的 toDataURL，会得到一个图片编码成的 base64 字符串，发送给后台，python 可以转成二进制再写入文件，就完成了拍照的存储。这里用来 xmlhttprequest 发送 post 请求，django 的 settings 里面默认是有 csrf 的中间件的，这是为了防止恶意请求的攻击，如果是在模板里面写一个表单来 post，前面可以加{% csrf_token %}，发送的字段里会带上，但是用 xmlhttprequest 写的要自己加一个 csrf token，我试了一下没通过，所以在 settings 里面取消了 csrf 中间件。