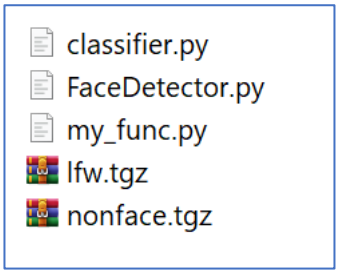
محمد حسین مسلمی / 97102463 / تمرین 4 computer vision

توضیحات اولیه درباره ساختار فایل ها و دیگر موارد :



ساختار فایلهای این تمرین به شکل روبروست :

که فایل lfw.tgz مربوط به داده های مثبت است که در صورت تمرین آمده بود و nonface.tgz مربوط به داده های منفی است که در ادامه توضیح داده شده و در این لینک موجود میباشد .

(مهم : نیازی به extract کردن این دو فایل نیست خود کد کارهای مربوطه را انجام میدهد)

همچنین فایل my_func.py مربوط به برخی توابع استفاده شده در فایل های دیگر است که توضیحات مربوط به هر کدام در ادامه آمده است . فایل classifier.py مربوط به train کردن SVM با داده های train و بررسی نتیجه بوسیله داده های test و سپس بدست آوردن دقت و نمودار های ROC و recall-precision میباشد . همچنین classifier بدست آمده در این قسمت ذخیره میشود و در فایل FaceDetector.py استفاده میشود .

در ابتدا لازم است که کد classifier.py ران شود که classifier بدست آید اما اگر FaceDetector.py هم اول ران شود خود کد میرود و فایل classifier.py را ران میکند و سپس ادامه کار را انجام میدهد.

در ابتدا به توضیح توابعی استفاده شده میپردازیم :(در my_func.py)

-1

```python
def saveVar(myvar,name):
    path=os.path.join(os.getcwd(),'variables')
    try:
        os.mkdir(path)
    except:
        pass
    name='variables/'+name+'.pckl'
    f = open(name, 'wb')
    pickle.dump(myvar, f)
    f.close()
    return
```

```python
def readvar(name):
    name='variables/'+name+'.pckl'
    f = open(name, 'rb')
    myvar = pickle.load(f)
    f.close()
    return myvar
```

```python
def show(im4,height=400):
    (h4,w4)=np.shape(im4)[0:2]
    scale=height/h4
    dim=(int(scale * w4) , height)
    im4_resize=cv2.resize(im4.copy(),dim)
    cv2.imshow('tmp', im4_resize)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

وظیفه دو تابع savevar , readvar این است که در صورت نیاز به ذخیره متغیری از روی ram بر روی disk این کار را انجام دهند ، به این دلیل که در بخش اول که classifier بدست آمد بعدا در تابع FaceDetector به آن نیاز است .برای اینکه دوباره نیاز نباشد آنرا محاسبه کنیم در فایل ها موجود است و بعدا import میکنیم در صورت نیاز. همچنین تابع show برای اینست که هر زمان نیاز شد بتوانیم یک تصویر را با اندازه معقول نشان دهیم .

-2

```python
def import_data():
    ##### import all data from face dataset ####
    f=tarfile.open('lfw.tgz', 'r')
    path = os.path.join(os.getcwd(), 'data')
    try:
        os.mkdir(path)
    except:
        pass
    for i,row in enumerate(f):
        f.extract(row,'data')

    path="data/lfw"
    names=os.listdir(path)
    img_face=[]
    for row in names:
        dir_=os.path.join(path,row)
        img_path=[]
        for j in os.listdir(dir_):
            img_path.append(os.path.join(path,row,j))
        for j in img_path:
            img_=cv2.imread(j)
            img_=cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
            img_face.append(img_)
    shutil.rmtree('data')
    f=tarfile.open('nonface.tgz', 'r')
    path = os.path.join(os.getcwd(), 'data')
    try:
        os.mkdir(path)
    except:
        pass
    for i,row in enumerate(f):
        f.extract(row,'data')
    path="data/nonface"
    names=os.listdir(path)
    img_nonface=[]
    for i,row in enumerate(names):
        dir_=os.path.join(path,row)
        img_=cv2.imread(dir_)
        img_=cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
        img_nonface.append(img_)
    shutil.rmtree('data')
    img_face=np.array(img_face,dtype=object)
    img_nonface=np.array(img_nonface,dtype=object)

    return [img_face,img_nonface]
```

دومین تابع به اسم import_data() است و وظیفه آن اینست که داده هایی که با آنها train,test,validation را انجام میدهیم ورودی بگیرد ، به این صورت که داده های مثبت که در فایل lfw.tgz هستند را extract میکند و سپس تمام عکس های داخل آنرا import میکند (grayscale) و در img_face میریزد ، سپس همینکار را برای فایل nonface.tgz انجام میدهد و در img_nonface میریزد . در ادامه درباره داده های منفی هم بحث میشود که چگونه و از کجا آمده اند .(نیازی به extract کردن فایل ها به صورت دستی نیست خود کد این کار را میکند و در نهایت فایلهای اضافی را پاک میکند) . همچنین درباره train,test , validation هم توضیح داده خواهد شد.

-3

```
def extract_features(data ,feature, label , cell_size , block_size):
    for row in data:
        row=np.uint8(row)
        resized_img = resize(row.copy(), (64,64))
        fd = hog(resized_img, orientations=9, pixels_per_cell=(cell_size, cell_size), cells_per_block=(block_size, block_size))
        feature.append([label,fd])
    return feature
```

تابع extract_features برای اینست که از داده های train , test , validation با استفاده از skimage.feature.hog بردار feature را استخراج کند ، به اینصورت که به ازای هر عکس که در هر سطر از آرایه data ورودی موجود است ابتدا انرا با استفاده از skimage.transform.resize به اندازه 64 در 64 کرده (البته قبلش آنرا به صورت int درآورده ایم) و سپس با cell size , blocksize که در ورودی تابع داده شده و با تعداد bin های 9 در منحنی histogram بردار ویژگی هر عکس را بدست می آوریم و سپس به انتهای آرایه feature که ورودی تابع است ؛ به همراه label مربوطه آن عکس اضافه می کنیم و feature را خروجی میدهیم . اندازه 64 در 64 بنابر آزمون و خطا بدست آمده که هم دقت خراب نشود و هم کند نشود بیش از اندازه .

-4

تابع روبرو برای عمل non max suppression به کار میرود به این صورت که box هایی از تصویر که در آنها face پیدا شده است را به همراه یک threshold میگیرد و عمل NMS را انجام میدهد به اینصورت که ابتدا مختصات گوشه بالا چپ و پایین راست و مساحت تمام box ها را بدست میاورد و آنها را به ترتیب ارتفاع گوشه راست پایین مرتب میکند و در index میریزد سپس از آخر index یکی یکی برمیدارد و بقیه را با آن مقایسه میکند که مساحت یکسان چقدر دارند ، اگر این مساحت یکسان نسبت به مساحت پنجره انتخابی از threshold بزرگتر بود ، پنجره ثانویه را حذف میکند .

```
def NMS(boxes, th,method=1):
    result = []
    (x1,y1)=(boxes[:,0],boxes[:,1])
    (x2,y2)=(boxes[:,2]+x1,boxes[:,2]+y1)
    area = (x2 - x1 ) * (y2 - y1)
    index = np.argsort(y2)
    while (True):
        if len(index) <=0:
            break
        i = index[len(index)-1]
        result.append(i)
        tmp = [len(index)-1]
        for row in range(len(index)):
            j = index[row]
            (x1_,y1_) =( max(x1[i], x1[j]),max(y1[i], y1[j]))
            (x2_,y2_) =( min(x2[i], x2[j]),min(y2[i], y2[j]))
            (w ,h) = (max(0, x2_ - x1_ ),max(0, y2_ - y1_ ))
            area_mutual = (w * h)
            m= area[i]
            if method==2:
                m=area[j]
            if (area_mutual/m) > th:
                tmp.append(row)
        index = np.delete(index, tmp)
    return boxes[result]
```

```python
def sliding_window(img1,repeat=12):
    img2=img1.copy()
    scale=1.1
    win_size,step=64,11
    windows,windows2,cnt=[],[],0
    while(cnt<repeat):
        (h,w)=np.shape(img2)
        x,y=0,0
        while(True):
            if x+win_size>=w:
                x=0
                y+=step
            if y+win_size>=h:
                break
            tmp=np.uint8(img2[y:y+win_size,x:x+win_size])
            x+=step
            windows.append([tmp,(x,y),cnt])
        cnt+=1
        win_size=int(64*scale**cnt)

    for row in windows:
        if np.mean(row[0])<=55 or np.sum(row[0])<180000 or np.mean(row[0])>=200:
            continue
        windows2.append(row)
    return windows2
```

5 – این تابع برای عمل sliding window به کار میرود به این صورت که یک پنجره اولیه دارد به اندازه 64 در 64 و به جای این که تصویر را کوچک کنیم و پنجره ثابت باشد در عمل اینگونه بهتر بود که تصویر اندازه ثابت داشته باشد و پنجره را بزرگ کنیم ، همچنین طول step های حرکت پنجره نیز برابر 11 پیکسل است به اینصورت ابتدا هر سطر را تا آخر پیمایش میکند و سپس به سطر بعد میرویم و تا انتهای تصویر پیش میرویم .و پنجره ها را فقط 10 بار بزرگتر میکنیم که این اعداد بنابر تجربه بدست آمده .

همچنین پنجره هایی که تقریبا سفید و یا تقریبا سیاه اند میتوان گفت صورت در آنها نیست لذا چون میانگین پیکسل های این تصاویر خیلی کم (برای سیاه ها) و خیلی زیاد (برای سفید ها) میباشد لذا میتوان آنها را با این تکنیک حذف کرد.

توضیح درباره داده های منفی :

این داده ها از dataset caltech256 در این لینک بدست آمده اند که حدود 27000 تا عکس بود که تعدادی مشخصا با label انسان بودند ؛ آنها را پاک شدند اما در بین عکس های دیگر هم مقداری تصویر انسان موجود بود ، این تصاویر را با استفاده از classifier های آماده در اینترنت تصاویری که صورت در آنها بود حذف کردیم که classifier ها در این لینک هستند ، پس از استفاده از classifier های مختلف صورت ها حذف شدند که به صورت دستی هم بررسی شد و در نهایت عرض تصاویر را روی 180 پیکسل قرار دادیم و ارتفاع scale شد و سپس در فایل nonface.tgz ذخیره شدند که لینک دانلود آنها در ابتدای گزارش آمده بود .

بدست آوردن Hyperparameters :

برای اینکه مشاهده شود کدام پارامتر ها عملکرد بهتری دارند مقدار cell size برای مقادیر 5 تا 12 و block size برای مقادیر 2 تا 5 و همینطور کرنل SVM برای سه مقدار linear , poly , rbf باید بر روی داده های validation بررسی میشدند اما train کردن آنها با 20000 داده train و 2000 داده validation زمان و محاسبات قابل توجهی نیاز داشت برای همین ابتدا از 3000 داده train و 300 داده test استفاده شد و از بین پارامتر ها آنهایی که عملکرد بهتری داشتند به تعداد 10 تای برتر انتخاب شدند و سپس برای کل داده ها دو باره train شدند و در نهایت این مقادیر از نظر سرعت و دقت بهینه شدند :

Kernel	poly
Cell size	6
Block size	3

توضیحات مربوط به classifier.py :

در ابتدا همه تصاویر بوسیله تابع import_data() ورودی گرفته شده اند و در مرحله بعد آنها را به صورت رندم shuffle کرده و از هر کدام 10000 تا برای train ؛ 1000 تا validation و 1000 تا نیز test گرفته شده .

```
np.random.shuffle(img_face)
train_pos=img_face[0:10000]
validation_pos=img_face[10000:11000]
test_pos=img_face[11000:12000]

np.random.shuffle(img_nonface)
train_neg=img_nonface[0:10000]
validation_neg=img_nonface[10000:11000]
test_neg=img_nonface[11000:12000]
```

سپس به صورت زیر feature ها برای آنها استخراج میشود و با بردار های X_train,Y_train مدل را بدست میاوریم :

```
feature_train=extract_features(train_pos, [] ,label=1, cell_size=cell_, block_size=block_)
feature_train=extract_features(train_neg, feature_train ,label=0, cell_size=cell_, block_size=block_)
X_train=[]
Y_train=[]
for row in feature_train:
    X_train.append(row[1])
    Y_train.append(row[0])
X_train=np.array(X_train)
```

پس از آنکه SVM_classifier بدست آمد آنرا در فایل کد ها ذخیره کرده و feature vector را برای داده های test بدست میاوریم و از آنها دقت را میسنجیم که برابر :

```
final precision is :  99.75  %
```

سپس به اینصورت مقادیر AP, ROC , precission , recall , true_pos, false_pos را بدست میاوریم و از آنها نمودار ها ی مربوطه را میکشیم :

```
y_score = classifier.decision_function(X_test)

AP = sklearn.metrics.average_precision_score(Y_test, y_score)
precision, recall, _ = sklearn.metrics.precision_recall_curve(Y_test,y_score)

false_pos, true_pos, _ = sklearn.metrics.roc_curve(Y_test, y_score)
ROC = sklearn.metrics.auc(false_pos, true_pos)
```

برای فایل FaceDetector.py به این صورت است که یک تابع FaceDetector دارد که نام فایلی تصویر ورودی را در ورودی میگیرد سپس با استفاده از تابع get_classifier اگر قبلا classifier.py ران شده باشد فایل classifier را از دیسک بر میدارد و گرنه فایل classifier.py را اجرا میکند و سپس فایل classifier را میخواند . سپس بوسیله تابع sliding_window پنجره های تصویر را جدا میکنیم و برای هر کدام مثل قبل feature vector بدست میاوریم .سپس برای این feature vector های مربوط به هر عکس با استفاده از تابع classifier.decision_function برای هر تصویر یک score بدست میاوریم که اگر از مقدار th1 در ورودی تابع بزرگتر بود حاوی صورت و گرنه بدون face است . حال قبلا در windows هر پنجره را با مختصات بالا چپ و یک عدد ذخیره کرده بودیم که این عدد نشان میدهد window چند مرتبه scale شده است لذا طول ضلع آن برابر $64\times scale^{cnt}$ میباشد که 64 برابر اندازه اولیه پنجره و cnt تعداد scale شدن آن است همچنین مقدار scale نیز 1.1 در نظر گرفتیم

سپس این پنجره ها را به تابع NMS میدهیم و خروجی آنرا رسم میکنیم در تصویر مربوطه و خروجی میدهیم .