STAT 377100 Homework 4

Lijing Wang

To implement the Viola-Jones face detection method, we need to calculate the Haar features of each picture. Here, I used four types of Haar features: 2 two-rectangular features and 2 three-rectangular features. To reduce the amount of calculation, the stride I use when getting the Haar features is 4 pixels. I also increase the size of rectangle features by 2 pixels each time. These significantly reduced the numbered of features from 295936 to around 52200.

We also need to record the index and location of each feature for future calculation.

Use Adaboost to select features that help classifiy the face and non-face images:

The steps are as follows:

- Assign initial weights D_0 to each image.
- Choose the classifier that minimizes the error$\epsilon_t = Pr_{D_t}(h_t x_i \neq y_i)$
- $\alpha_t = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$
- $Z_t = 2 * ((1-\epsilon_t)\epsilon_t)^{\frac{1}{2}}$
- Reassign weight to each image by $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_t h_t(x_i))}{Z_t}$
- Repeat previous steps

The final classifier is given by $h = sgn(\sum_t \alpha_t h_t)$

The Viola-Jones paper also proposed using cascaded classifier. The steps are described as below:

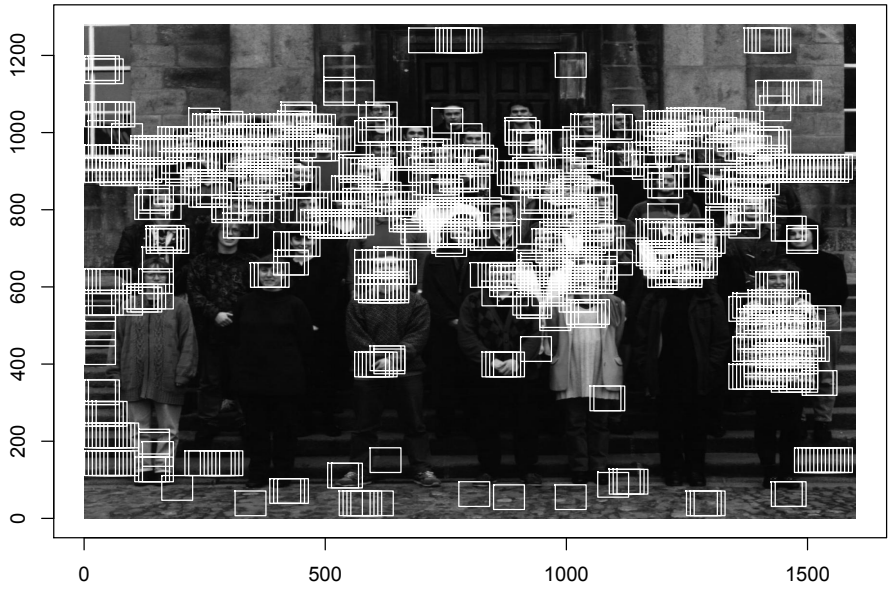
- Set the maximum acceptable false positive rate (FPR) and minimum acceptable detection rate (D)
- Set the target false positive rate (FPR_{target})
- Initialize FPR=1, D=1
- Iteration: If current FPR¿FPR_{target}, add a layer of adaboost classifier

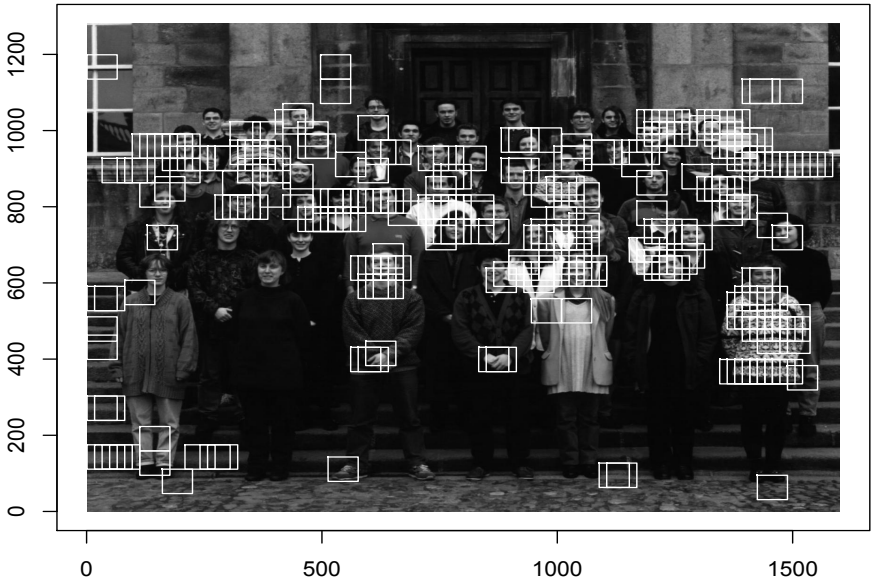
When training the cascaded classifier, if the current layer cannot satisfy the FPR and D requirement, then we should continue adding features to this classifier until the cascaded classifier meet the FPR and D requirement.

We also need to notice that the negative images are the images that have been misclassified by previous classifiers.

To detect faces in an image, use subwindow to scan across the whole image and use the cascaded classifiers to eliminate possible negative images.

The final results are given as below:





The subwindow stride of first image is 8 pixels, while 16 pixels for the second image.

As we can see from the pictures, There're many overlaps in the images. In the second picture, quite a lot of faces have been misclassified. This might be due to the size and stride of subwindow scanning. Another possible

reason is the limited number of features used for face detection. Also, I only use 4 kinds of Haar features when training the data. Further improvement can be made to solve these problems.

Python code:

```
def computeFeature(I, row, col, numFeatures):
feature = np.zeros(numFeatures)
cnt = 0 # count the number of features
window_h = 1; window_w=2 #window/feature size
for h in xrange(8,int(row/window_h+1),2): #extend the size of the rectangular feature
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4): #stride size=4
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h]) #4x1
rect2=np.array([i,j+w,w,h])
feature [cnt]=sumRect(I, rect2)- sumRect(I, rect1)
cnt=cnt+1
window_h = 2; window_w=1
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4):
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i+h,j,w,h])
feature[cnt]=sumRect(I, rect1)- sumRect(I, rect2)
cnt=cnt+1
window_h = 1; window_w=2
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4):
for j in xrange(1,int(col+1-2*w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i,j+w,w,h])
rect3=np.array([i,j+2*w,w,h])
feature[cnt]=sumRect(I, rect1)+sumRect(I, rect3)- sumRect(I, rect2)
cnt=cnt+1
window_h = 2; window_w=1
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-2*h*window_h+1),4):
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i+h,j,w,h])
rect3=np.array([i+2*h,j,w,h])
feature[cnt]=sumRect(I, rect1)+sumRect(I, rect3)- sumRect(I, rect2)
cnt=cnt+1
return feature
row=64
col=64
n=0
rect2loc=[]
rect1loc=[]
rect3loc=[]
```

```
window_h = 1; window_w=2
for h in xrange(8,int(row/window_h+1),2): #extend the size of the rectangular feature
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4): #stride size=4
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h]) #4x1
rect2=np.array([i,j+w,w,h])
rect3=np.array([0,0,0,0])
rect1loc.append(rect1)
rect2loc.append(rect2)
rect3loc.append(rect3)
n=n+1
n
window_h = 2; window_w=1
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4):
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i+h,j,w,h])
rect3=np.array([0,0,0,0])
rect1loc.append(rect1)
rect2loc.append(rect2)
rect3loc.append(rect3)
n=n+1
n
window_h = 1; window_w=2
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-h*window_h+1),4):
for j in xrange(1,int(col+1-2*w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i,j+w,w,h])
rect3=np.array([i,j+2*w,w,h])
rect1loc.append(rect1)
rect2loc.append(rect2)
rect3loc.append(rect3)
n=n+1
n
window_h = 2; window_w=1
for h in xrange(8,int(row/window_h+1),2):
for w in xrange(8,int(col/window_w+1),2):
for i in xrange (1,int(row+1-2*h*window_h+1),4):
for j in xrange(1,int(col+1-w*window_w+1),4):
rect1=np.array([i,j,w,h])
rect2=np.array([i+h,j,w,h])
rect3=np.array([i+2*h,j,w,h])
rect1loc.append(rect1)
rect2loc.append(rect2)
rect3loc.append(rect3)
n=n+1
##ADABOOST##
numh=200
```

```
Nimg=features.shape[1]
iniweight = np.zeros((Nimg,1))
iniweight[:,0] = 1/Nimg


#label face as 1, nonface as 0.
label = np.zeros((4000,1))
label[:2000] = 1

weight=iniweight
alpha=np.zeros(numh)
thres=np.zeros(numh)
polarity=np.zeros(numh)
idx=np.zeros(numh)
index=np.arange(features.shape[0])
result=[]
for l in xrange(numh):
a=getWeakClassifier(features[index], weight, label, Npos)
currentMin=a[0]
alpha[l]=np.log((1-currentMin)/currentMin)/2
result.append(a[4])
idx[l]=index[a[3]]
index=np.delete(index,a[3])
print idx[l]
print a[3]
thres[l]=a[1]
polarity[l]=a[2]
weight=weight*np.exp(-alpha[l]*(-1)**(1-(a[4]==label)))
weight=weight/np.sum(weight)
print l


##CACSCADED CLASSIFIER##
feature=np.copy(features[idx.astype(int)])
n=1
cascaded=[]
theta=[]
#labela=np.copy(label)
#resulta=np.copy(result)
alphah=np.zeros(feature.shape[1])
for i in xrange(feature.shape[0]):
for j in xrange(feature.shape[1]):
alphah[j]=alphah[j]-(-1)**result[i][j][0]*alpha[i]
print alphah
theta1=np.percentile(alphah[0:200],0.5)
delindex=np.where(alphah>=theta1)[0]
count=0
for k in xrange(feature.shape[1]):
if(label[k]==0 and alphah[k]>=theta1):
count=count+1
fpr=count/feature.shape[1]
print fpr
```

```
print np.mean(alphah[Npos:])
print np.mean(alphah[0:Npos])
delindex=delindex[delindex>=200]
if fpr<=0.3:
n=n+1
cascaded.append(i)
theta.append(theta1)
feature=np.delete(feature,delindex,1)
alphah=np.delete(alphah,delindex)
result=np.delete(result,delindex,1)
print feature.shape
print alphah.shape
if np.power(0.3,n)<1-0.999:
break
def classifier(Img,k):
intImg=np.zeros((row+1,col+1))
intImg[1:row+1,1:col+1]=np.cumsum(cumsum(Img,axis=0),axis=1)
xfeature=np.zeros(cascaded[k]+1)
for i in xrange(cascaded[k]+1):
rec1=rect1loc[int(idx[i])]
rec2=rect2loc[int(idx[i])]
rec3=rect3loc[int(idx[i])]
if rec3[2]==0:
xfeature[i]=sumRect(intImg,rec1)-sumRect(intImg,rec2)
else:
xfeature[i]=sumRect(intImg,rec1)+sumRect(intImg,rec3)-sumRect(intImg,rec2)
ah=0
for i in xrange(200):
ah=ah+np.sign((xfeature[i]-thres[i])*polarity[i])*alpha[i]
if ah>0:
return 1
else:
return 0
patch=[]
windloc=[]
n=0
for i in xrange(1,testpic.shape[0]-64,16):
for j in xrange(1,testpic.shape[1]-64,16):
patch.append(testpic[i:i+64,j:j+64])
windloc.append([i,j])
n=n+1
indct1=np.zeros(n)
indct1=indct1+1
i=0
m=0
indc=np.zeros(n)
for i in xrange(len(patch)):
for k in xrange(len(cascaded)):
indct1[i]=classifier(patch[n],k)
if indct1[i]==0:
break
np.savetxt('windloc1.txt', windloc1)
np.savetxt('indct1.txt', indct1)
```

R code:

```
im<-readJPEG("class.jpeg")
indct1<-read.table("~/indct1.txt")
windloc1<-read.table("~/windloc1.txt")
h_1=windloc1[indct1==1,1]
w_1=windloc1[indct1==1,2]
h_1=1280-h_1
plot(1,1,xlim=c(1,res[2]),ylim=c(1,res[1]),type='n')
rasterImage(im,1,1,res[2],res[1])
points(w_1,h_1,pch=".",col="white")
for (i in 1:length(w_1))
{
x<-c(w_1[i],w_1[i]+64,w_1[i]+64,w_1[i],w_1[i])
y<-c(h_1[i],h_1[i],h_1[i]-64,h_1[i]-64,h_1[i])
lines(x,y,lwd=1,col="white")
}
```