Report 109550005 張可晴

Screenshot of your code and brief explanation

Part 1 (dataset.py) :

```
dataset=[]

fileRoute=os.getcwd()
fileRoute=os.path.join(fileRoute,dataPath,'car')


for imgName in os.listdir(fileRoute):
    imgName=os.path.join(fileRoute,imgName)
    img=cv2.imread(imgName)


    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (36, 16), interpolation=cv2.INTER_AREA)


    tupleImg=(img,1)
    dataset.append(tupleImg)

fileRoute=os.getcwd()
fileRoute=os.path.join(fileRoute,dataPath,'non-car')


for imgName in os.listdir(fileRoute):
    imgName=os.path.join(fileRoute,imgName)
    img=cv2.imread(imgName)


    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = cv2.resize(img, (36, 16), interpolation=cv2.INTER_AREA)


    tupleImg=(img,0)
    dataset.append(tupleImg)
```

I firstly use "*os.getcwd()*" to get the current working ,then applying "*os.path.join()*" to concatenate the route of the car/non-car file. After that, by calling "*os.listdir()*" , get a list containing the names in the directory and load image. In addition, convert the image to 36*16 grayscale one by using "*cv2.resize()*" and "*cv2.cvtColor() + cv2.COLOR_BEG2GRAY*", respectively. Finally, combining the image and its label (car=1, non-car=0) in a tuple and appending in a list named dataset returned after finishing processing all the image.

Part 2 (Adaboost.py):

Brief description of the sample code

For starter, using the function

$$ii(-1, y) = 0$$

$$s(x, -1) = 0$$

$$s(x, y) = s(x, y-1) + i(x, y)$$

$$ii(x, y) = ii(x-1, y) + s(x, y)$$

to compute the integral image.

Second of all, we need to build Haar feature depending on different image and select best of them by testing the return value of apply_function. After finishing these tasks, we can use the AdaBoost algorithm. Initializing the same label to be the same weight (1/2*n), then normalizing them. Using the weight and threshold that apply_function compute to select the best weak classifiers. After that we will recalculate the weight depending on the last iteration that each weak classifier's accuracy. If the accuracy is low, then its weight will be higher. In contrast, its weight will be lower. In this way, we can train the false classifier making the final strong classifier predict more better.

Detailed description of my code (select_best)

```
# Begin your code (Part 2)
bestClf, bestError = None, float('inf')
num=len(iis)

for f, val in features:
    error = 0
    clf = WeakClassifier(f)
    for img, label, w in zip(iis, labels, weights):
        correctness = abs(clf.classify(img) - label)
        error += w * correctness

    error = error / num
    if error < bestError:
        bestClf, bestError = clf, error

# End your code (Part 2)
return bestClf, bestError
```

I run a loop for each feature and build a classifier for it. After that, we put each image into the classifier to check its correctness. If the result of the classifier and the truth is different then plus one into correctness which count the number of the error. Last but not least, return the smallest error classifier and its error.

Part 4 (detection.py):

```
dataPathTXT=os.path.join('.',dataPath,'detectData.txt')

f=open(dataPathTXT,'r')

num=f.readline()

coordinate=[]

for i in range(int(num)):
    data=f.readline()
    list1=[]
    val=''
    for char in data:
        val+=char
        if char==chr(32):
            list1.append(int(val))
            val=''
        if char=='\n':
            list1.append(int(val))
            val=''
    coordinate.append(list1)

f.close()

dataPathVideo=os.path.join('.',dataPath,'video.gif')


cap = cv2.VideoCapture(dataPathVideo)

index=0
f=open('Adaboost_pred.txt','w')
```

First of all, I read the coordinate of each parking space in detectData.txt and record in a 2D list for latter cropping the image.

```
while(1):

    ret, frame = cap.read()

    if ret==False:
        break

    for i in range(int(num)):
        frameCroped = crop(coordinate[i][0],coordinate[i][1],coordinate[i][2],coordinate[i][3],coordinate[i][4]
        frameCroped = cv2.cvtColor(frameCroped, cv2.COLOR_BGR2GRAY)
        frameCroped = cv2.resize(frameCroped, (36, 16), interpolation=cv2.INTER_AREA)

        if clf.classify(frameCroped)==1:
            f.write('1 ')
            green_color = (0, 255, 0) # BGR
            pts = np.array([[coordinate[i][0], coordinate[i][1]], [coordinate[i][2], coordinate[i][3]], [coordi
            pts = pts.reshape((4, 1, 2))# 將座標轉為 (頂點數量, 1, 2) 的陣列
            cv2.polylines(frame, [pts], True, green_color, 1)# 繪製多邊形


        else:
            f.write('0 ')
    cv2.imshow('detectedImage',frame)

    f.write('\n')
    if index==0:
        cv2.imwrite('detectedImage.png',frame)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
        index=1

    if cv2.waitKey(30) == 27:
        break
```

Secondly, I use the "*cv2.VideoCapture*" and "*cv2.read()*" to open and read the video.gif, respectively. Next, I send the image to crop function to get the each parking

space and send it to the "*clf.classify()*". if return 1, meaning the classifier detect there's a car, plot a bounding car on original image by using "cv2.polylines()", and write 1 in the Adaboost_pred.txt. If not, write 0 in the Adaboost_pred.txt.

Your implementation of the above requirements in detail

Part 3 :comparing the performance from T=1 to 10

TP = True Positive

TN = True Negative

FP = Falese Positive

FN = False Negative

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

T	FP rate (%)	FN rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
1	7.33	29.33	81.67	90.60	70.67	79.40
2	100.00	0.00	50.00	50.00	100.00	66.67
3	38.67	10.00	75.67	69.95	90.00	78.72
4	4.33	24.00	85.83	94.61	76.00	84.30
5	40.33	5.00	77.33	70.20	95.00	80.74
6	50.33	16.33	66.67	62.44	83.67	71.51
7	46.33	38.33	57.67	57.10	61.67	59.30
8	48.33	39.67	56.00	55.52	60.33	57.83
9	29.00	42.00	64.50	66.67	58.00	62.03
10	19.33	28.00	76.33	78.83	72.00	75.26

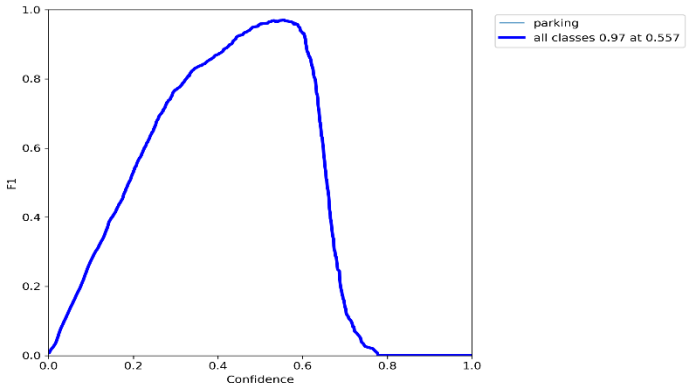
Training data

T	FP rate (%)	FN rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
1	8.33	34.00	78.83	88.79	66.00	75.72
2	100.00	0.00	50.00	50.00	100.00	66.67
3	46.33	17.00	68.33	64.18	83.00	72.38
4	7.00	26.00	83.50	91.36	74.00	81.77
5	49.00	9.33	70.83	64.92	90.67	75.66
6	47.00	18.67	67.17	63.38	81.33	71.24
7	51.00	40.00	54.50	54.05	60.00	56.87
8	51.00	41.00	54.00	53.63	59.00	56.19
9	35.33	46.00	59.33	60.45	54.00	57.04
10	28.33	35.00	68.33	69.64	65.00	67.24

Test data

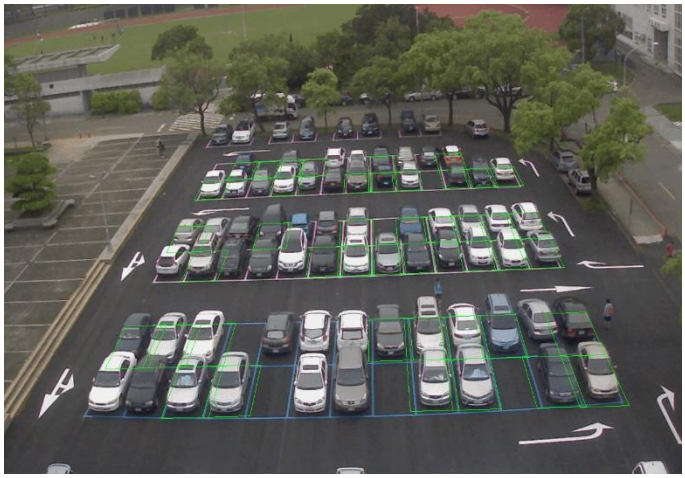
Part 5 (Yolov5):

I change the config threshold to observe its performance

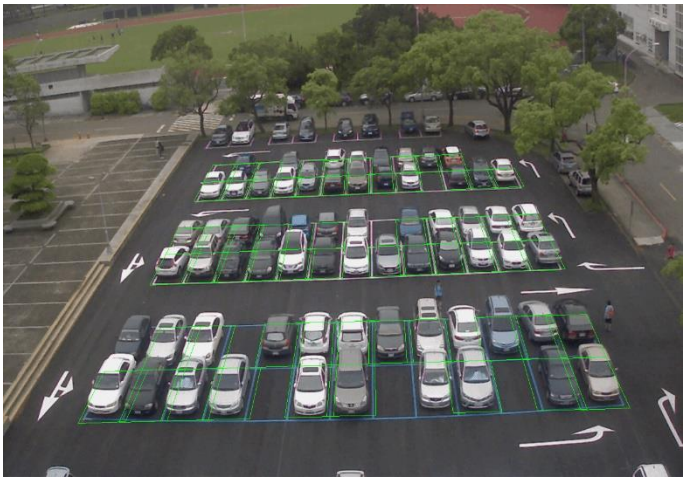


	Training Accuracy	Training F1-score	Testing Accuracy	Testing F1-score
Threshold=0.2	86.67	88.24	91.00	91.74
Threshold=0.3	88.67	89.82	92.33	92.88
Threshold=0.4	94.00	94.32	96.67	96.77

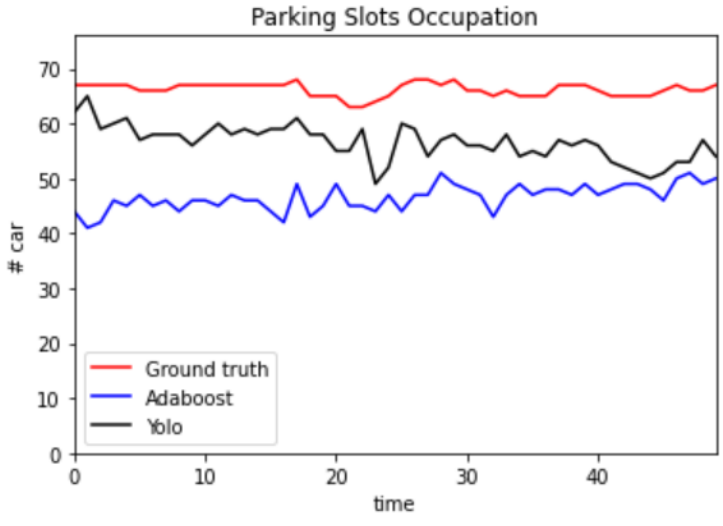
Discuss what you observed with accuracy, F1-score and parking slots occupation plot of different methods

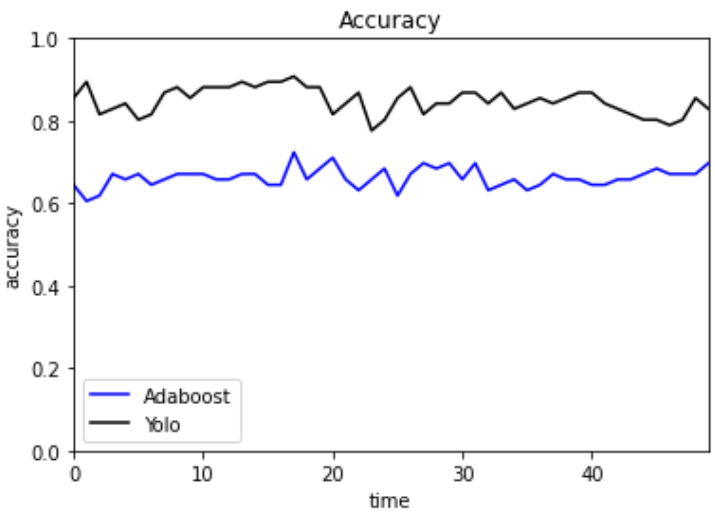


Adaboost



yolov5





From the parking slot occupation plot, we can tell that Yolov5 is closer to Ground truth than Adaboost, showing that the prediction of Yolov5 is more precise, confirmed by Accuracy plot.

Before discussing accuracy and F1-score, Why we need F1-score?

Since accuracy is (TP + FP)/(ALL EXAMPLE), it didn't tell us how the classify interested performs. As a result, we can use F1-score, which is the harmonic mean of precision and recall. In this way, we can **assess the performance of positive class**.

	Training Accuracy	Training F1-score	Testing Accuracy	Testing F1-score
Adaboost	76.33	75.26	68.33	67.24
Yolov5	94.00	94.32	96.67	96.77

From the above table, we can observe that no matter it is accuracy or F1-score. Yolov5 always performs better than Adaboost. Additionally, in the light of Yolov5, the value of F1-score is higher than accuracy, indicating that Yolov5 predict even better on our interesting class. By contrast, the performance of Adaboost is worse on our interesting model.

Bonus : SOTA method

```
total_pos, total_neg = 0, 0
for w, label in zip(weights, y):
    if label == 1:
        total_pos += w
    else:
        total_neg += w

classifiers = []
total_features = X.shape[0]
for index, feature in enumerate(X):
    if len(classifiers) % 1000 == 0 and len(classifiers) != 0:
        print("Trained %d classifiers out of %d" % (len(classifiers), total_features))

    applied_feature = sorted(zip(weights, feature, y), key=lambda x: x[1])

    pos_seen, neg_seen = 0, 0
    pos_weights, neg_weights = 0, 0
    min_error, best_feature, best_threshold, best_polarity = float('inf'), None, None, None
    for w, f, label in applied_feature:
        error = min(neg_weights + total_pos - pos_weights, pos_weights + total_neg - neg_weights)
        if error < min_error:
            min_error = error
            best_feature = features[index]
            best_threshold = f
            best_polarity = 1 if pos_seen > neg_seen else -1

        if label == 1:
            pos_seen += 1
            pos_weights += w
        else:
            neg_seen += 1
            neg_weights += w

    clf = WeakClassifier(best_feature, best_threshold, best_polarity)
    classifiers.append(clf)
return classifiers
```

train_weak function

I add a function called "train_weak" to firstly train the weak classifier. To do so, it

firstly reorder the weights depend on the feature value. Run in a for loop selecting the threshold and calculating its error depending on the difference of real number of positive or negative and the predicted number of positive or negative. Then, choose the smallest error as the best classifier and return corresponding feature, feature value (threshold), and if there's more positive case let polarity to be 1; otherwise, 0.

```
# Begin your code (Part 2)
bestClf, bestError = None, float('inf')
num=len(iis)

classifiers = self.train_weak(featureVals,labels,features,weights)

for clf in classifiers:
    error = 0
    for img, label, w in zip(iis, labels, weights):
        correctness = abs(clf.classify(img) - label)
        error += w * correctness

    error = error / num
    if error < bestError:
        bestClf, bestError = clf, error

# End your code (Part 2)
return bestClf, bestError
```

The select_best function is similar to the original one, except for prebuild the classifier in train_weak to promote its performance.

Compare to Adaboost and do some discussions in your report

	Training Accuracy	Training F1-score	Testing Accuracy	Testing F1-score
Adaboost	76.33	75.26	68.33	67.24
SOTA	97.33	97.36	95.00	94.90

From above table, we can clearly observe that the SOTA method performance is much higher than Adaboost in both accuracy and F1-score. It's because the SOTA method find the best feature, threshold, and evaluate its polarity in advance, while Adaboost just find the best weak classifier given feature. As a result, it is no surprise that Adaboost perform worse.

Describe problems you meet and how you solve them

1. In part1, when I want to convert the image to grayscale one, I firstly use *"img=np.array(Image.open(fileRoute).convert('L'))"* and it shows error, then I found it is a PIL image, so I use "img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)" to cope with the problem.
2. Since I'm not familiar with os, at the beginning, I naively type all the route by for loop to load image, and somehow there's error. Finally, I use the *"os.listdir()"* to

solve the problem

3. In part 2, I was confused that to train a classifier needs to use the value of both feature and featureVal, resulting the same performance whether the T is. To resolve this problem, I build a classifier given a feature. Since Adaboost use decision stump to build a weak classifier. Another solution is to run each featureVal and if it is negative keep it as error. Finally, the smaller error will be selected to build the best classifier. However, I don't use the latter solution, since the former is more intuitive for me.
4. In part 4, when I read the frame from the video.gif, there is no frame anymore at the end but it doesn't break out of the while loop, so the crop function get a empty image and report error. To address this problem, I set plus a if to decide whether there's frame that be read. If not, then break.
5. I draw a weird bounding box at first because of using "*cv2.rectangle()*" to draw it, which is corrected by using "*cv2.polyline()*".
6. In part 5, there's error about warmup function, and I find the solution from the forum, changing the branch to get previous version.
7. And I also meet some trivial problem in part 5, such as missing "import pandas as pd", the size of Adaboost.txt being too big, and so on.