report of AI HW1 109550027 紀竺均

a. Explanation of my code and my implementation.

Part 1: Load and prepare your dataset (10%)

(dataset.py- loadImages)

```
# Begin your code (Part 1)
#raise NotImplementedError("To be implemented")
dataset=[]
files = glob.glob(os.path.join(dataPath+"/car","*"))
for imgc in files:
    imc = cv2.imread(imgc,0)
    imc = cv2.resize(imc, (36, 16))
    tup = (imc,1)
    dataset.append(tup)
files = glob.glob(os.path.join(dataPath+"/non-car","*"))
for imgn in files:
    imn = cv2.imread(imgn,0)
    imn = cv2.resize(imn, (36, 16))
    tup = (imn,0)
    dataset.append(tup)
# End your code (Part 1)
```

首先，我用 glob 這個函示將路徑中的所有檔案存到 files 中，接著用 imread 讀取檔案，將其依照規定轉成灰階及 resize，最後將圖片資訊以及分類合併成 tuple 再把它存到 dataset 這個 list of tuples。

Part 2: Implement Adaboost algorithm (30%)

(adaboost.py- selectBest)

```
# Begin your code (Part 2)
#raise NotImplementedError("To be implemented")
bestClf, bestError = None, float('inf')
for feature in features:
    clf = WeakClassifier(feature)
    error = 0
    for image,label, w in zip(iis,labels, weights):
        if(clf.classify(image)!=label):
            error+=w
    error = error/len(iis)
    if error < bestError:
        bestError=error
        bestClf=clf

# End your code (Part 2)
```

Adaboost 概念簡述：Adaboost 全名 Adaptive Boosting，是靠一次一次的迭代更新權重，而得到效果更好的分類器。

更新權重的方式分為更新資料的權重以及更新分類器的權重，關於資料：若本次分類結果為正確，下一次降低權值；本次分類錯誤，則下一次增加權值。關於分類器：若此弱分類器的正確率高，則增加他在強分類器中佔的比重；若錯誤率高，則降低比重。本部分實作在 train()中。

buildFeatures():把 image shape 丟進去，對每種可能的大小建立一個 feature。

SelectBest(): //TO-DO
這部分要找出最好的 classifier，我利用前面 buildfeature()中建立好的 features，把他們逐一丟到 weakclassifier 建立一個一個弱分類器。接著，我把圖片資料(image)丟進 clf.classify()中，得到這個分類器判斷出來的分類(有車:1；沒車:0)，再和原本正確的分類(label)做比較，若兩者不相等，表示判斷錯誤，此處我用 error 來記錄錯誤的加總權重(w)，最後把 error 除以 len(iis)得到錯誤率。
藉由每個 feature 產生的 classifier，去計算不同的 error，不斷更新最小的 error 以及最佳的 classifier。
classify():判斷照片為有車或無車。

Part 3: Additional experiments (15%)

T=10:

```
Evaluate your classifier with training dataset
False Positive Rate: 75/300 (0.250000)
False Negative Rate: 141/300 (0.470000)
Accuracy: 384/600 (0.640000)

Evaluate your classifier with test dataset
False Positive Rate: 66/300 (0.220000)
False Negative Rate: 144/300 (0.480000)
Accuracy: 390/600 (0.650000)
```

T=9

```
Evaluate your classifier with training dataset
False Positive Rate: 92/300 (0.306667)
False Negative Rate: 86/300 (0.286667)
Accuracy: 422/600 (0.703333)

Evaluate your classifier with test dataset
False Positive Rate: 84/300 (0.280000)
False Negative Rate: 81/300 (0.270000)
Accuracy: 435/600 (0.725000)
```

T=8

```
Evaluate your classifier with training dataset
False Positive Rate: 33/300 (0.110000)
False Negative Rate: 146/300 (0.486667)
Accuracy: 421/600 (0.701667)

Evaluate your classifier with test dataset
False Positive Rate: 34/300 (0.113333)
False Negative Rate: 146/300 (0.486667)
Accuracy: 420/600 (0.700000)
```

T=7

```
Evaluate your classifier with training dataset
False Positive Rate: 65/300 (0.216667)
False Negative Rate: 76/300 (0.253333)
Accuracy: 459/600 (0.765000)

Evaluate your classifier with test dataset
False Positive Rate: 74/300 (0.246667)
False Negative Rate: 81/300 (0.270000)
Accuracy: 445/600 (0.741667)
```

T=6

```
Evaluate your classifier with training dataset
False Positive Rate: 122/300 (0.406667)
False Negative Rate: 47/300 (0.156667)
Accuracy: 431/600 (0.718333)

Evaluate your classifier with test dataset
False Positive Rate: 130/300 (0.433333)
False Negative Rate: 59/300 (0.196667)
Accuracy: 411/600 (0.685000)
```

T=5

```
Evaluate your classifier with training dataset
False Positive Rate: 67/300 (0.223333)
False Negative Rate: 77/300 (0.256667)
Accuracy: 456/600 (0.760000)

Evaluate your classifier with test dataset
False Positive Rate: 77/300 (0.256667)
False Negative Rate: 89/300 (0.296667)
Accuracy: 434/600 (0.723333)
```

T=4

```
Evaluate your classifier with training dataset
False Positive Rate: 93/300 (0.310000)
False Negative Rate: 70/300 (0.233333)
Accuracy: 437/600 (0.728333)

Evaluate your classifier with test dataset
False Positive Rate: 108/300 (0.360000)
False Negative Rate: 90/300 (0.300000)
Accuracy: 402/600 (0.670000)
```

T=3

```
Evaluate your classifier with training dataset
False Positive Rate: 51/300 (0.170000)
False Negative Rate: 58/300 (0.193333)
Accuracy: 491/600 (0.818333)

Evaluate your classifier with test dataset
False Positive Rate: 45/300 (0.150000)
False Negative Rate: 66/300 (0.220000)
Accuracy: 489/600 (0.815000)
```

T=2

```
Evaluate your classifier with training dataset
False Positive Rate: 300/300 (1.000000)
False Negative Rate: 0/300 (0.000000)
Accuracy: 300/600 (0.500000)

Evaluate your classifier with test dataset
False Positive Rate: 300/300 (1.000000)
False Negative Rate: 1/300 (0.003333)
Accuracy: 299/600 (0.498333)
```

T=1 //best accuracy

```
Evaluate your classifier with training dataset
False Positive Rate: 25/300 (0.083333)
False Negative Rate: 88/300 (0.293333)
Accuracy: 487/600 (0.811667)

Evaluate your classifier with test dataset
False Positive Rate: 24/300 (0.080000)
False Negative Rate: 91/300 (0.303333)
Accuracy: 485/600 (0.808333)
```

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where $Precision = \dfrac{TP}{TP + FP}$ and $Recall = \dfrac{TP}{TP + FN}$

result: (run by Excel)

| T= | FP | FN | TP | PRECISION | RECALL | F1-SCORE |
|---|---|---|---|---|---|---|
| 10 TRAIN | 0.25 | 0.47 | 0.75 | 0.75 | 0.614754098 | 0.675675676 |
| 10 TEST | 0.22 | 0.48 | 0.78 | 0.78 | 0.619047619 | 0.690265487 |
| 9 TRAIN | 0.306 | 0.286 | 0.694 | 0.694 | 0.708163265 | 0.701010101 |
| 9 TEST | 0.28 | 0.27 | 0.72 | 0.72 | 0.727272727 | 0.72361809 |
| 8 TRAIN | 0.11 | 0.486 | 0.89 | 0.89 | 0.646802326 | 0.749158249 |
| 8 TEST | 0.113 | 0.486 | 0.887 | 0.887 | 0.64603059 | 0.747576907 |
| 7 TRAIN | 0.216 | 0.253 | 0.784 | 0.784 | 0.756027001 | 0.76975945 |
| 7 TEST | 0.246 | 0.27 | 0.754 | 0.754 | 0.736328125 | 0.745059289 |
| 6 TRAIN | 0.406 | 0.156 | 0.594 | 0.594 | 0.792 | 0.678857143 |
| 6 TEST | 0.433 | 0.196 | 0.567 | 0.567 | 0.743119266 | 0.643221781 |
| 5 TRAIN | 0.223 | 0.256 | 0.777 | 0.777 | 0.752178122 | 0.764387605 |
| 5 TEST | 0.256 | 0.296 | 0.744 | 0.744 | 0.715384615 | 0.729411765 |
| 4 TRAIN | 0.31 | 0.23 | 0.69 | 0.69 | 0.75 | 0.71875 |
| 4 TEST | 0.36 | 0.3 | 0.64 | 0.64 | 0.680851064 | 0.659793814 |
| 3 TRAIN | 0.17 | 0.19 | 0.83 | 0.83 | 0.81372549 | 0.821782178 |
| 3 TEST | 0.15 | 0.22 | 0.85 | 0.85 | 0.794392523 | 0.821256039 |
| 2 TRAIN | 1 | 0 | 0 | 0 | #DIV/0! | #DIV/0! |
| 2 TEST | 1 | 0.003 | 0 | 0 | 0 | #DIV/0! |
| 1 TRAIN | 0.83 | 0.293 | 0.17 | 0.17 | 0.367170626 | 0.23239918 |
| 1 TEST | 0.08 | 0.303 | 0.92 | 0.92 | 0.752248569 | 0.827710301 |

總結來說，T=1 的 training data 有最好的 accuracy；T=1 的 testing data 有最好的 F1-score。而 T=2 的時候表現最差。

Part 4: Detect car (15%)

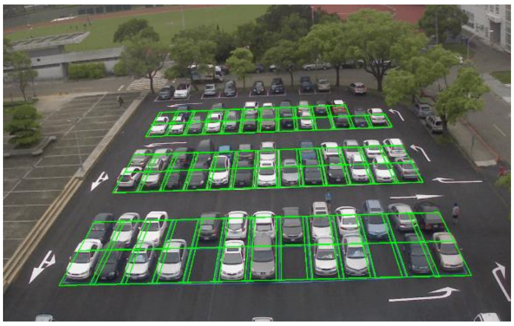
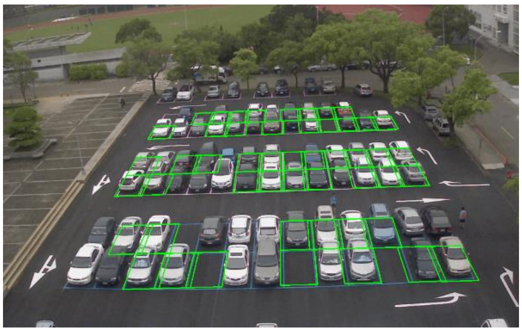
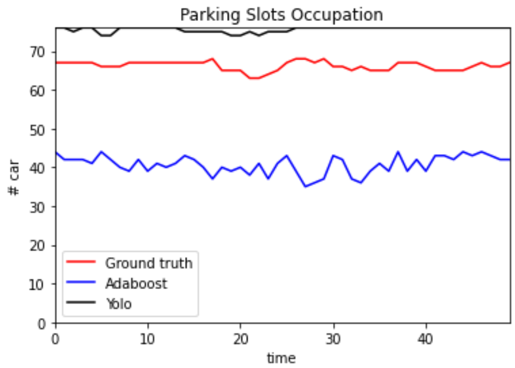
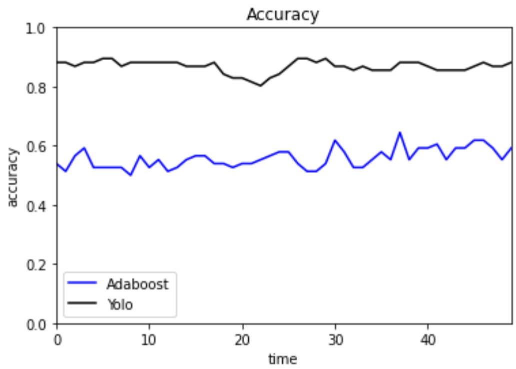
```
# Begin your code (Part 4)
#raise NotImplementedError("To be implemented")
file = open(dataPath,'r')
lines = file.readlines()
file.close()
f = open("Adaboost_pred.txt", "w")
cap = cv2.VideoCapture('data/detect/video.gif')
first=1
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        print("cant recieve frame anymore")
        break
    for i in range(1,int(lines[0])+1):
        line = lines[i].split(' ')
        xy=[0]*8
        for ind,s in enumerate(line):
            xy[ind]=int(s)
        image = crop(xy[0],xy[1],xy[2],xy[3],xy[4],xy[5],xy[6],xy[7],frame)
        image=cv2.resize(image,(36,16))
        image=cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        result=clf.classify(image)

        if result==1:
            pts = np.array([[xy[0],xy[1]],[xy[2],xy[3]],[xy[6],xy[7]],[xy[4],xy[5]]], np.int32)
            pts = pts.reshape((-1,1,2))
            cv2.polylines(frame,[pts],True,(0,255,0),2)
            f.write('1 ')
        else:
            f.write('0 ')
    f.write('\n')
    cv2.imshow('frame',frame)
    cv2.waitKey(1)
    if(first==1):
        cv2.imwrite('frist_frame.png',frame)
    first=0

f.close()
cap.release()
cv2.destroyAllWindows()

# End your code (Part 4)
```

這部分要把 video 裡每一張 frame 裡的每個停車格拿去做分類並框出判斷為有車的格子。實作如下：

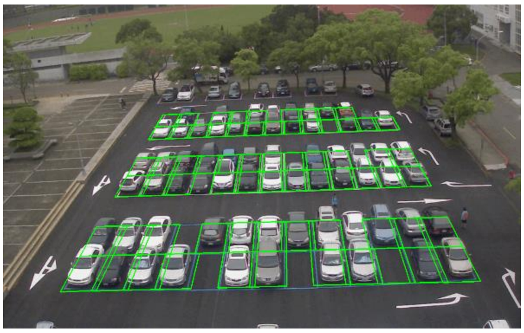
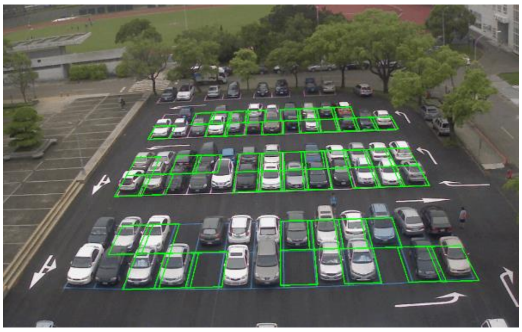
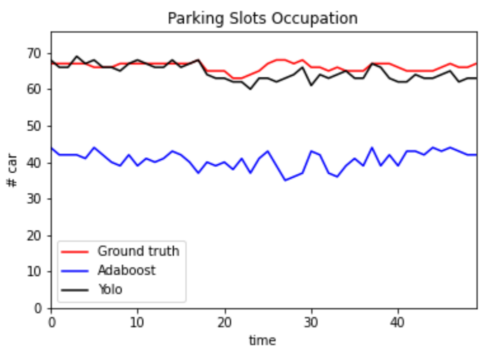
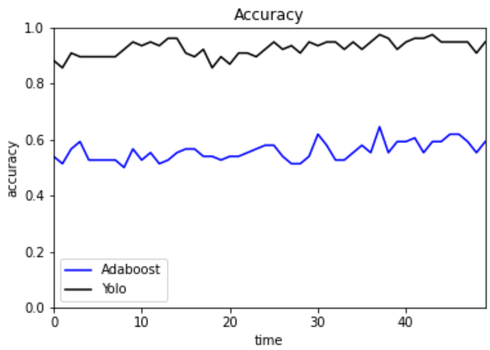
先讀取照片以及停車格座標，利用 crop()把照片裁切成一塊一塊的停車格，接著 resize 和轉灰階之後，就可以丟到 classify()中判斷分類了。如果判斷為 1(有車)，就會畫出四個座標點構成的綠色四邊形。同時也用 Adaboost_pred.txt 紀錄結果並且輸出框完車子的第一張 frame。

Part 5: Discuss the difference between Adaboost and Yolov5 and draw a scatter plot/line graph to show the temporal parking slots occupation (10%)

此部分我將會分別探討 Yolov5 中 confidence_threshold=0.4 以及 confidence_threshold=0.3 和 Adaboost T=10 的差異，以結果來說，Yolov5 中 confidence_threshold=0.3 的表現最好，Adaboost 中 T=10 的表現最差。

| Yolov5 (confidence_threshold=0.4) | Adaboost (T=10) |
| --- | --- |
|  |  |
| Occupation | Accuracy |
|  |  |

一開始尚未改變任何參數時，我發現我的 Yolov5 的結果幾乎都是 1，但準確度還是比 adaboost 還要高，我覺得和**停車場中大部分車位都是有車的**這個原因有關。為了改善，我想我的 confidence_threshold 設定的太高，所以將它下降為 0.3，也的確得到了更高的準確度。

| Yolov5 (confidence_threshold=0.3) | Adaboost (T=10) |
| --- | --- |
|  |  |
| Occupation | Accuracy |
|  |  |

b. problems encountered

一開始覺得最困難的是 part 2 的部分，覺得沒有方向不知如何實作，後來在網路上看了很多 adaboost 的範例，才漸漸知道每個 function 到底是要幹嘛的、為什麼需要他。

接著遇到的問題是我要 write file 時，明明 print 出來都是對的，寫到檔案裡卻變成一堆%%%%%%%，我原本的作法是全部寫到一個 string 然後 write，後來改成直接 f.write()然後就可以了。

而 Yolov5 的執行也花了我蠻多時間的，一方面對 google colab 不熟悉，還有 yolo 用了很多我沒有使用過的函示庫，讀起來蠻吃力地，最後是它一直報一堆奇怪的錯誤給我，有些重開幾次就好了，還有其中一個錯是在畫圖表的地方，因為我在 Adaboost_pred.txt 的每一行最後多了一個空白鍵，導致他沒有辦法和 ground_truth.txt 比較…，而最後也都順利解決了。