

Introduction to Artificial Intelligence HW 1 Report

110550088 李杰穎

March 21, 2022

1 Code and Explanation

The following is the code that is required to be finished. Explanation of this code is presented in the comment of the program.

Code 1: Part 1 (`datasets.py`)

```
1 import os
2 import cv2
3 import numpy as np
4 def loadImages(dataPath):
5     """
6     Load all Images in the folder and transfer a list of tuples.
7     The first element is the numpy array of shape (m, n) representing the image.
8     (remember to resize and convert the parking space images to 36 x 16 grayscale images.)
9     The second element is its classification (1 or 0)
10    Parameters:
11        dataPath: The folder path.
12    Returns:
13        dataset: The list of tuples.
14    """
15    # Begin your code (Part 1)
16    dataset = [] # Declare an empty list to save the grayscale images
17
```

```

18 # Process images in "car" directory
19 for item in os.listdir(os.path.join(dataPath, "car")): # Use os.path.join to generate paths
20     img = cv2.imread(os.path.join(dataPath, "car", item)) # Read image from files
21     img = cv2.resize(img, (36, 16)) # Resize the image from (360, 160) to (36, 16)
22     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert image to grayscale image
23     data = (img, 1) # Create a tuple to store image and label and
24     # because all images in "car" folder is the occupied parking space, the label is set to 1
25     dataset.append(data) # Append the tuple to the dataset list
26
27 for item in os.listdir(os.path.join(dataPath, "non-car")): # Do the same thing as above but this time
28     ↪ is for "non-car" folder
29     img = cv2.imread(os.path.join(dataPath, "non-car", item))
30     img = cv2.resize(img, (36, 16))
31     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
32     data = (img, 0) # Not occupied parking space, Label is set to 0
33     dataset.append(data)
34
35 # End your code (Part 1)
36
37 return dataset

```

Code 2: Part 2 (`adaboost.py/selectBest`)

```

1 def selectBest(self, featureVals, iis, labels, features, weights):
2     """
3     Finds the appropriate weak classifier for each feature.
4     Selects the best weak classifier for the given weights.
5     Parameters:
6         featureVals: A numpy array of shape (Len(features), Len(dataset)).
7             Each row represents the values of a single feature for each training sample.
8         iis: A list of numpy array with shape (m, n) representing the integral images.
9         Labels: A list of integer.
10            The ith element is the classification of the ith training sample.
11         features: A numpy array of HaarFeature class.
12         weights: A numpy array with shape(Len(dataset)).
13            The ith element is the weight assigned to the ith training sample.
14     Returns:
15         bestClf: The best WeakClassifier Class
16         bestError: The error of the best classifier
17     """

```

```

18 # Begin your code (Part 2)
19 # Init. WeakClassifier by the feature in features list
20 # And append them all into the clfs list
21 clfs = [WeakClassifier(feature=feature) for feature in features]
22
23 # Declare bestClf and bestError to store the currently best classifier and its error
24 bestClf = None
25 bestError = sum(weights) # The max error is the sum of weights
26
27 for clf in clfs: # Iterate all classifier in clfs
28     error = 0      # Declare a variable to track the error of the current clf
29     for i in range(len(iis)): # Iterate all image sample
30         if clf.classify(iis[i]) != labels[i]: # When the prediction of the model is different with the
31             ↪   Label
32             error += weights[i] # Add weights to error
33     if error < bestError: # If the error is smaller than the best error, then classifier is the currently
34             ↪   best classifier
35             bestError = error # Change bestError to current error
36             bestClf = clf      # Save this classifier as bestClf
37
38 # End your code (Part 2)
39 return bestClf, bestError

```

Code 3: Part 3 (`main.py`)

```

1 print('Start training your classifier')
2 for t in range(1, 11): # Train with different T (1 ~ 10)
3     print(f"Training with T={t}")
4     clf = adaboost.Adaboost(T=t) # Init. clf using the given T
5     clf.train(trainData) # Train of train data
6     clf.save(f'clf_300_{t}') # Save the model file as clf_300_<t>
7     clf = adaboost.Adaboost.load(f'clf_300_{t}') # Load the model after saving
8
9     # Evaluate the model using already written utils.evaluate function
10    print('\nEvaluate your classifier with training dataset')
11    utils.evaluate(clf, trainData) # Use train data to evaluate model
12
13    print('\nEvaluate your classifier with test dataset')
14    utils.evaluate(clf, testData) # Use test data to evaluate model

```

```

15
16 # Part 4: Implement detect function in detection.py and test the following code.
17 print('\nUse your classifier with video.gif to get the predictions (one .txt and one .png)')
18 detection.detect('data/detect/detectData.txt', clf, t) # Save the detection results to detectData.txt

```

Code 4: Part 4 (`detection.py/detect`)

```

1 def detect(dataPath, clf, t=10):
2     """
3         Please read detectData.txt to understand the format.
4         Use cv2.VideoCapture() to Load the video.gif.
5         Use crop() to crop each frame (frame size = 1280 x 800) of video to get parking space images. (image
6             size = 360 x 160)
7         Convert each parking space image into 36 x 16 and grayscale.
8         Use clf.classify() function to detect car, If the result is True, draw the green box on the image Like
9             the example provided on the spec.
10            Then, you have to show the first frame with the bounding boxes in your report.
11            Save the predictions as .txt file (Adaboost_pred.txt), the format is the same as GroundTruth.txt.
12            (in order to draw the plot in Yolov5_sample_code.ipynb)
13
14    Parameters:
15        dataPath: the path of detectData.txt
16
17    Returns:
18        No returns.
19
20    """
21
22    # Begin your code (Part 4)
23    cords = [] # Declare a List that stores the coordinate of each parking slots
24    with open(dataPath) as file:
25        num_of_parking = int(file.readline()) # Read the number total parking slots from the first line of
26            files
27        for _ in range(num_of_parking): # Iterate all lines
28            tmp = file.readline() # Read a line in file
29            tmp = tmp.split(" ") # Split the line using " "
30            res = tuple(map(int, tmp)) # Convert the string type to int type using the built-in map
31            cords.append(res) # Append the coordinates to "cords" list
32
33    cap = cv2.VideoCapture(os.path.join(dataPath, "..", "video.gif")) # Use cv2.VideoCapture to read
34        video.gif

```

```

28     frame = 0 # Counter to track current frame number
29     output_gif = [] # Declare a List to store each processed frame of output.gif
30     first_frame = True # A flag that will help us store the processed first frame images
31
32     while True:
33         detect_label = [] # Declare a List to store the detect results of each parking slots
34         frame += 1 # Make frame number add 1
35         _, img = cap.read() # Read a frame of video.gif
36         if img is None: # If all frame are read, then img is None
37             break # If None, then break
38         for cord in cords: # Iterate all cords
39             pic = crop(*cord, img) # Use * to unpack cord e.g. (x1, y1, ..., x4, y4) -> x1, y1, ..., x4,
40             #<br> y4
41             pic = cv2.resize(pic, (36, 16)) # Resize image to (36, 16)
42             pic = cv2.cvtColor(pic, cv2.COLOR_RGB2GRAY) # Convert image to grayscale images
43             detect_label.append(clf.classify(pic)) # Use clf.classify to detect whether the parking slot
44             #<br> is occupied or not and append the result to "detect_label" list
45         for i, label in enumerate(detect_label): # Iterate all detect_label
46             if label: # If the model detects that this parking slot is occupied
47                 pos = [[cords[i][idx], cords[i][idx+1]] for idx in range(0, 8, 2)] # Add the four points
48                 #<br> of the rectangle to "pos" list
49                 pos[2], pos[3] = pos[3], pos[2] # swap pos[2] and pos[3]
50                 pos = np.array(pos, np.int32) # Convert python built-in List to numpy array
51                 cv2.polyline(img, [pos], color=(0, 255, 0), isClosed=True) # Use cv2.polyline to draw
52                 #<br> rectangle
53
54             output_gif.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # Append the result image of each frame to
55             #<br> output_gif list (Also convert the color)
56             if first_frame: # If this frame is the first frame in video.gif
57                 first_frame = False # After getting into this block, set first_frame to False
58                 cv2.imwrite(f"Adaboost_first_frame_{t}.png", img) # Save the results of first frame
59             with open(f"Adaboost_pred_{t}.txt", "a") as txt: # Open the Adaboost_pred_<t>.txt and use append
60                 #<br> mode to add new line to file
61                 res = "" # Declare an empty string
62                 for i, label in enumerate(detect_label): # Iterate all labels
63                     if label:
64                         res += "1" # If the parking slot is occupied, then write 1 to file
65                     else:
66                         res += "0" # Else write 1
67                     if i != len(detect_label) - 1:
68                         res += " " # If not the last label in a frame, then write a space to seperate each
69                         #<br> Label

```

```

62         else:
63             res += "\n" # Else write newline character
64         txt.write(res) # Write the res string to file
65     imageio.mimsave(f'results_{t}.gif', output_gif, fps=2) # Use imageio.imwrite to save result gif and set
66     ↪   fps to 2
# End your code (Part 4)

```

Code 5: Part 5 (yolov5_sample_code.ipynb)

```

1 dataPath = os.path.join("HW1_material", "detect", "detectData.txt") # Path to "detectData.txt"
2 cords = [] # Declare a List that stores the coordinate of each parking slots
3 with open(dataPath) as file: # Open "detectData.txt"
4     num_of_parking = int(file.readline()) # Read the number total parking slots from the first line of
5     ↪   files
6     for _ in range(num_of_parking): # Iterate all lines
7         tmp = file.readline() # Read a Line in file
8         tmp = tmp.split(" ") # Split the Line using " "
9         res = tuple(map(int, tmp)) # Convert the string type to int type using the built-in map function
10        cords.append(res) # Append the coordinates to "cords" list
11
12 cap = cv2.VideoCapture(os.path.join("HW1_material", "detect", "video.gif")) # Use cv2.VideoCapture to read
13 ↪   video.gif
14 frame = 0 # Counter to track current frame number
15 output_gif = [] # Declare a List to store each processed frame of output.gif
16 first_frame = True # A flag that will help us store the processed first frame images
17 while True:
18     detect_label = [] # Declare a List to store the detect results of each parking slots
19     frame += 1 # Make frame number add 1
20     _, img = cap.read() # Read a frame of video.gif
21     if img is None: # If all frame are read, then img is None
22         break # If None, then break
23     for cord in cords: # Iterate all cords
24         pic = crop(*cord, img) # Use * to unpack cord e.g. (x1, y1, ..., x4, y4) -> x1, y1, ..., x4, y4
25         pic = cv2.resize(pic, (36, 16)) # Resize image to (36, 16)
26         detect_label.append(yolov5_func.classify(pic, weight_path, confidence_threshold, (36, 16))) # Use
27         ↪   yolov5_func.classify to detect whether the parking slot is occupied or not, and append the
28         ↪   result to "detect_label" list
29     for i, label in enumerate(detect_label): # Iterate all detect_label
30         if label: # If the model detects that this parking slot is occupied

```

```

27     pos = [[cords[i][idx], cords[i][idx+1]] for idx in range(0, 8, 2)] # Add the four points of
28     ↵ the rectangle to "pos" list
29     pos[2], pos[3] = pos[3], pos[2] # swap pos[2] and pos[3]
30     pos = np.array(pos, np.int32) # Convert python built-in List to numpy array
31     cv2.polyline(img, [pos], color=(0, 255, 0), isClosed=True) # Use cv2.polyline to draw
32     ↵ rectangle
33
34     output_gif.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # Append the result image of each frame to
35     ↵ output_gif List (Also convert the color)
36     if first_frame: # If this frame is the first frame in video.gif
37         cv2.imwrite(img_save_path, img) # After getting into this block, set first_frame to False
38         first_frame = False # Save the results of first frame
39     with open(txt_save_path, "a") as txt: # Open txt file and use append mode to add new line to file
40         res = "" # Declare an empty string
41         for i, label in enumerate(detect_label): # Iterate all labels
42             if label:
43                 res += "1" # If the parking slot is occupied, then write 1 to file
44             else:
45                 res += "0" # Else write 1
46             if i != len(detect_label) - 1:
47                 res += " " # If not the last label in a frame, then write a space to separate each label
48             else:
49                 res += "\n" # Else write newline character
50         txt.write(res) # Write the res string to file
51     imageio.mimsave(gif_save_path, output_gif, fps=2) # Use imageio.imsave to save result gif and set fps to 2

```

2 Experiments

2.1 Hyperparameters Adjustment

2.1.1 Adaboost

In Adaboost algorithm, I only change the hyperparameter T , which indicates the number of classifiers that will be used in the training process.

I have trained 10 models, from $T = 1$ to $T = 10$. The performance of these 10 models will be shown in Section 3.2.

2.1.2 YOLOv5

In YOLOv5, the only hyperparameter I can change is the confidence threshold, which is a probability threshold. If the YOLOv5's output is greater than the threshold, then it considers that the parking slot is occupied.

I have tested three kinds of threshold, which are 0.3, 0.4 and 0.5. And the performance of these three value will be presented in Section 3.3.

3 Results

3.1 Measurements

3.1.1 Accuracy

In binary classification problem, we often define four kinds of accuracy, which are True Positive, True Negative, False Positive, and False Negative to show the performance of classifiers. Their definitions are written below:

- True Positive (TP): The real label is **true**, and our model predicts that it's **true**.
- True Negative (TN): The real label is **false**, and our model predicts that it's **false**.
- False Positive (FP): The real label is **true**, but our model predicts that it's **false**.
- False Negative (FN): The real label is **false**, but our model predicts that it's **true**.

3.1.2 F-Score

F-score is used to measure a test's accuracy. To define F-score, we first need to define two variables, “precision” and “recall”.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (1)$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2)$$

And the definition of F-score is:

$$\text{F-score} = \frac{(1 + \beta^2) \text{ precision} \times \text{recall}}{\beta^2 \text{ precision} + \text{recall}} \quad (3)$$

In reality, we often use “F1-score”, which means $\beta = 1$. Therefore, the “F1-score” can be written as:

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \quad (4)$$

The range of F1-score is $[0, 1]$. If F1-score is higher (closer to 1), then the performance of classifier is better.

3.2 Adaboost Comparision

Table 1: Performance of different Adaboost model from $T = 1$ to $T = 10$ on training datasets

T	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
1	8.33	29.00	81.33	91.67	75.97	83.08
2	8.33	29.00	81.33	91.67	75.97	83.08
3	4.33	23.33	81.50	95.67	80.39	87.37
4	9.00	17.00	87.00	91.00	84.26	87.50
5	7.00	13.33	89.83	92.33	88.50	90.38
6	8.00	12.00	90.00	92.00	88.46	90.20
7	7.00	13.33	89.83	93.00	87.46	90.15
8	7.00	12.00	90.50	93.00	88.57	90.73
9	8.00	10.67	90.67	92.00	89.61	90.79
10	7.33	12.67	90.00	92.67	87.97	90.26

Table 2: Performance of different Adaboost model from $T = 1$ to $T = 10$ on testing datasets

T	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
1	8.00	30.33	80.83	92.00	75.20	82.76
2	8.00	30.33	80.83	92.00	75.20	82.76
3	6.33	30.67	81.50	93.67	75.34	83.51
4	9.00	22.67	84.17	91.00	80.06	85.18
5	8.67	21.00	85.17	91.33	81.31	86.03
6	8.33	19.33	86.17	91.67	82.58	86.89
7	7.67	23.67	84.33	92.33	79.60	85.49
8	7.67	21.67	85.33	92.33	80.99	86.29
9	10.00	20.00	85.00	90.00	81.82	85.71
10	8.33	20.67	85.50	91.67	81.60	86.34

From Table 2, we can observe that when $T = 6$, the F1-score is the highest.

This means the best value for hyperparameter T is 6. Therefore, in the comparision of Adaboost and YOLOv5, I will set $T = 6$ to compare with YOLOv5.

3.3 YOLOv5 Comparison

Table 3: Performance of different confidence threshold on training datasets

Threshold	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
0.1	32.00	0.00	84.00	68.00	100.00	80.95
0.2	24.67	0.00	87.67	75.33	100.00	85.93
0.3	19.67	0.00	90.17	80.33	100.00	89.09
0.4	12.00	0.33	93.83	88.00	99.62	93.45
0.5	2.67	7.00	95.17	97.33	93.29	95.27
0.6	0.67	67.33	66.00	99.33	59.60	74.50

Table 4: Performance of different confidence threshold on testing datasets

Threshold	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
0.1	30.00	0.00	85.00	70.00	100.00	82.35
0.2	19.67	0.00	90.17	80.33	100.00	89.09
0.3	13.00	0.00	93.50	87.00	100.00	93.05
0.4	5.00	0.00	97.50	95.00	100.00	97.44
0.5	1.67	1.00	98.67	98.33	98.99	98.66
0.6	0.00	59.00	70.50	100.00	62.89	77.22

From Table 3 and Table 4, we can observe that when the confidence threshold equals to 0.5, the F1-score on both training and testing is the highest.

This means the best value for the hyperparameter “confidence threshold” is 0.5. Therefore, in the comparison of Adaboost and YOLOv5, I will set the confidence threshold to 0.5 when comparing with Adaboost.

3.4 Comparision between Adaboost and YOLOv5

Table 5: Performance of Adaboost and YOLOv5 on testing datasets

Model	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
Adaboost	8.33	19.33	86.17	91.67	82.58	86.89
YOLOv5	1.67	1.00	98.67	98.33	98.99	98.66

From Table 5, we can observe that YOLOv5 outperforms the traditional Adaboost method.

3.5 The detection result of a real parking lot

The figures and plots below are the detection result of the parking lot. I use different hyperparameters and models to detect the occupied parking slots. I will also compare the performance of each model.

3.5.1 Plots

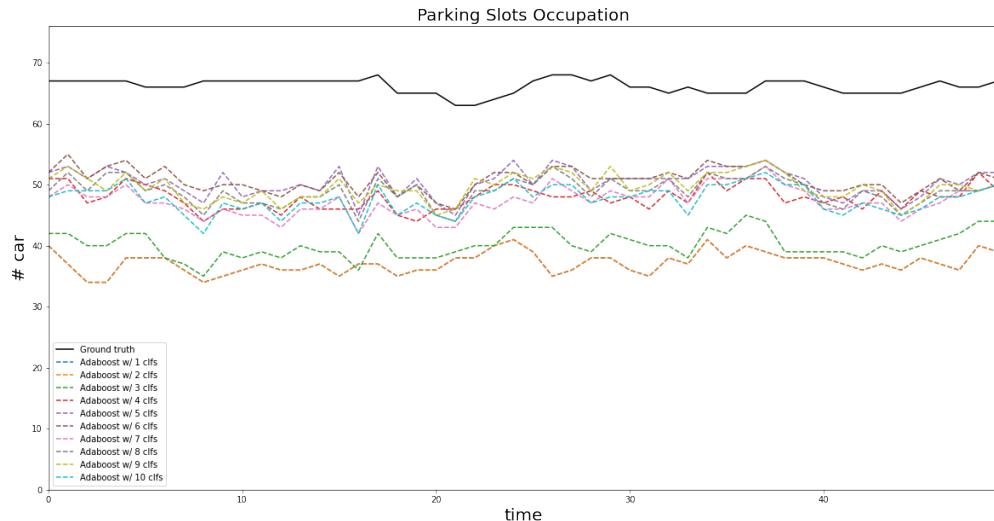


Figure 1: The number of occupied parking slots detected by Adaboost ($T = 1$ to $T = 10$) and the ground truth

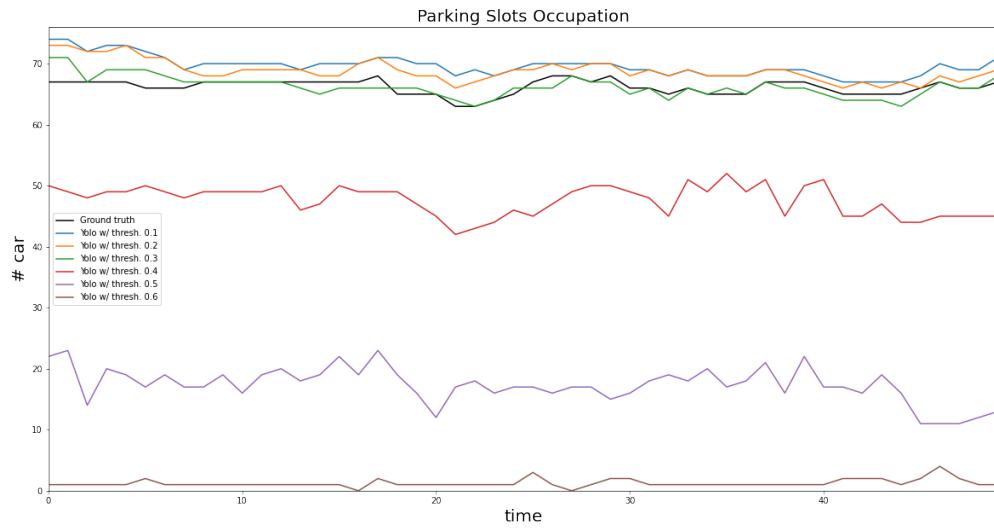


Figure 2: The number of occupied parking slots detected by YOLOv5 (threshold = 0.1 to threshold = 0.6) and the ground truth

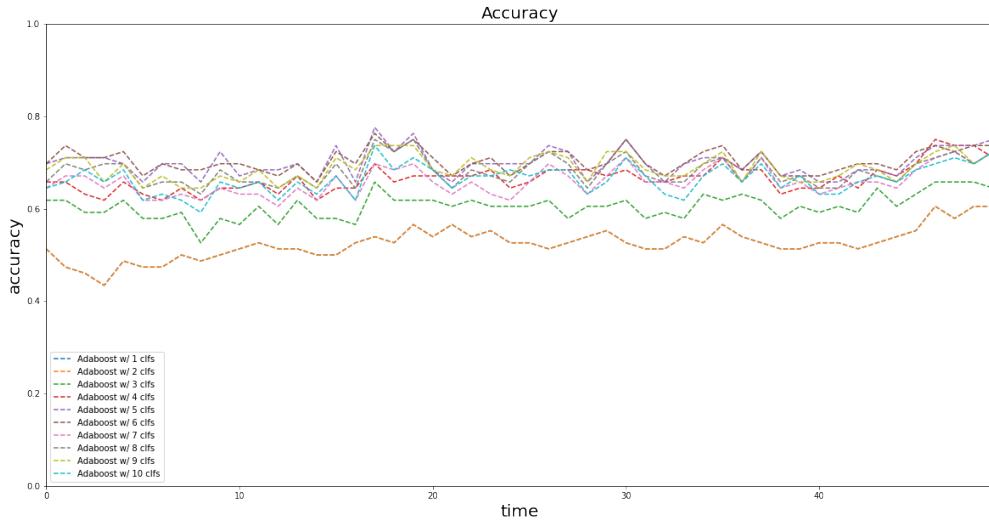


Figure 3: Accuracy of Adaboost ($T = 1$ to $T = 10$) over time

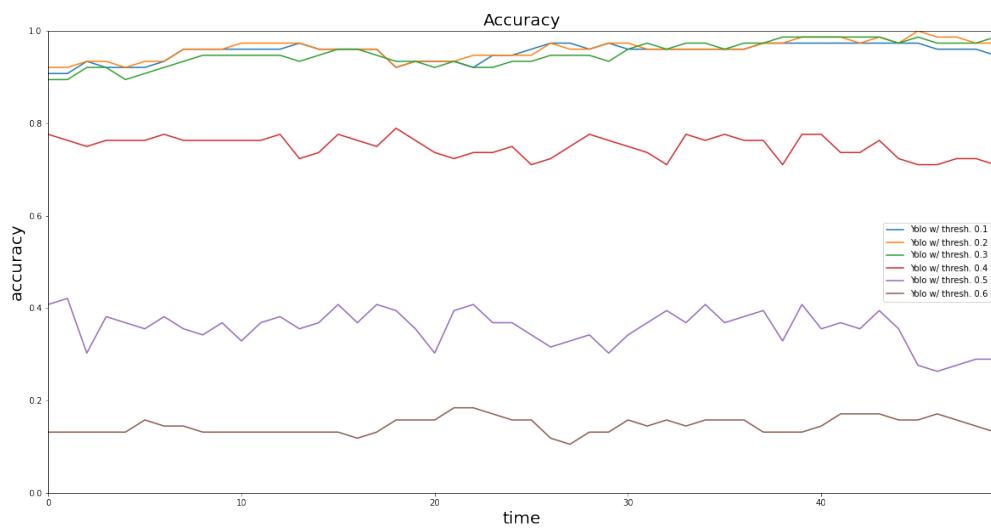


Figure 4: Accuracy of YOLOv5 (threshold = 0.1 to threshold = 0.6) over time

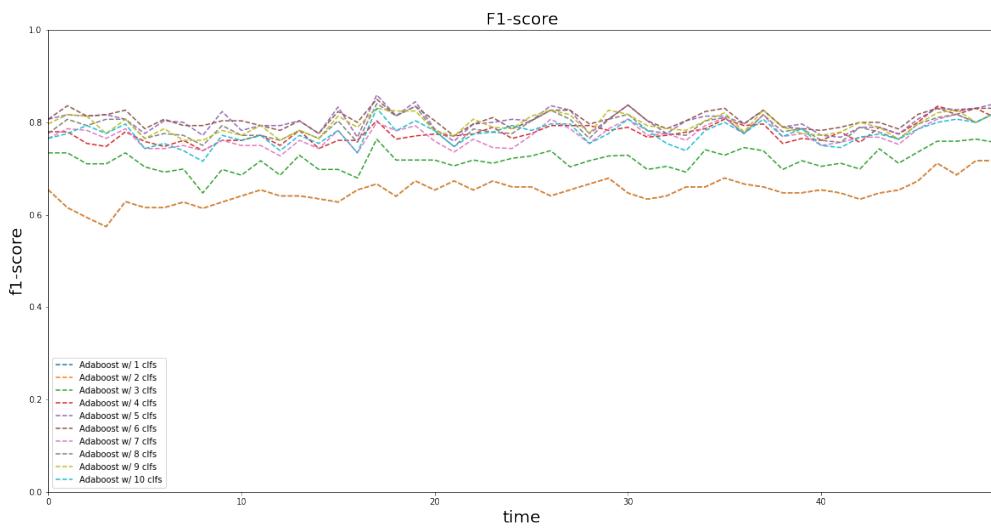


Figure 5: F1-score of Adaboost ($T = 1$ to $T = 10$) over time

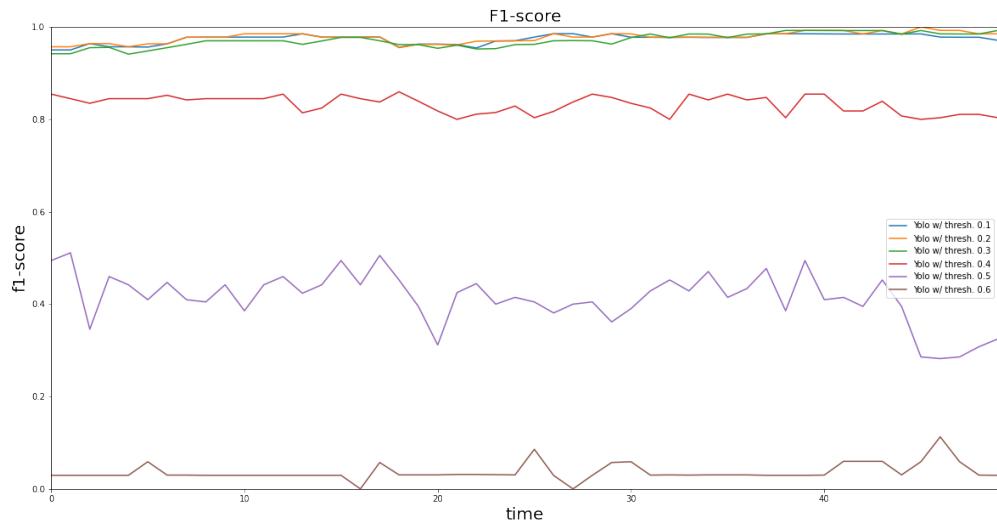


Figure 6: F1-score of YOLOv5 (threshold = 0.1 to threshold = 0.6) over time

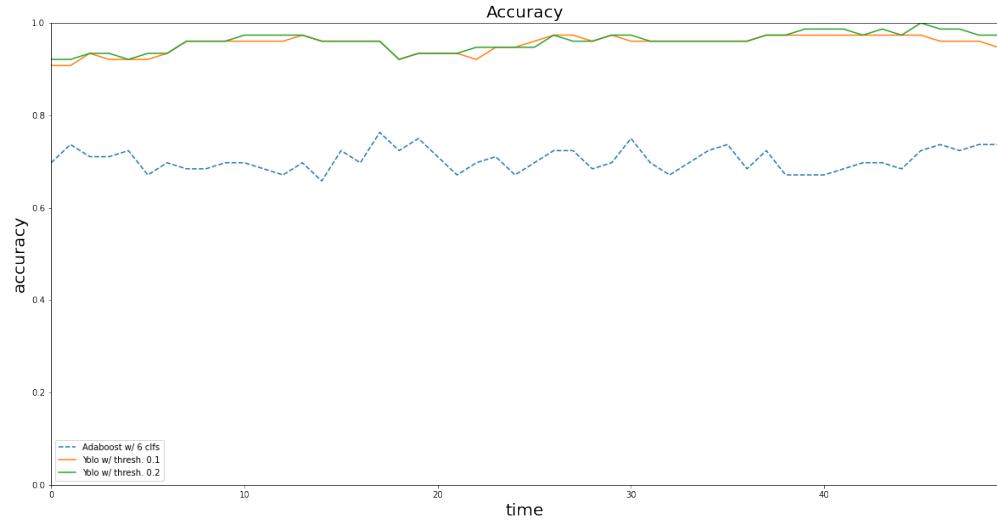


Figure 7: Accuracy of Adaboost ($T = 6$) and YOLOv5 (threshold = 0.1, 0.2) over time

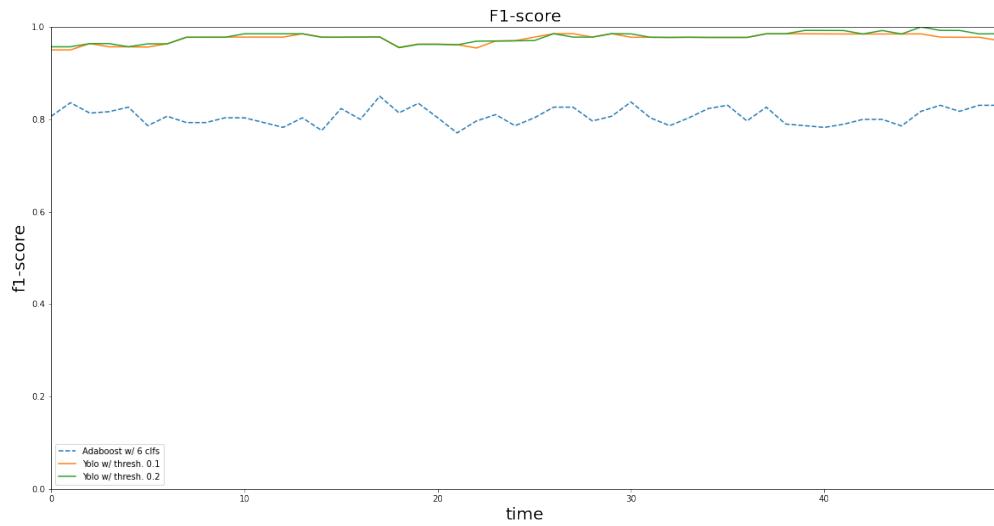


Figure 8: F1-score of Adaboost ($T = 6$) and YOLOv5 (threshold = 0.1, 0.2) over time

3.5.2 The first frame of video.gif



Figure 9: The detection result of Adaboost with 6 classifiers

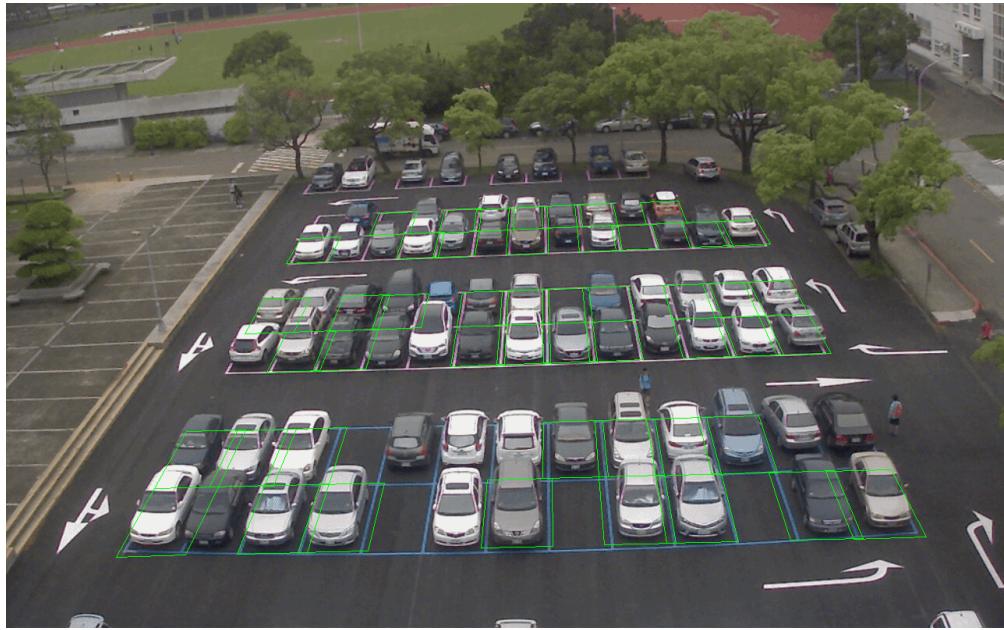


Figure 10: The detection result of Adaboost with 8 classifiers

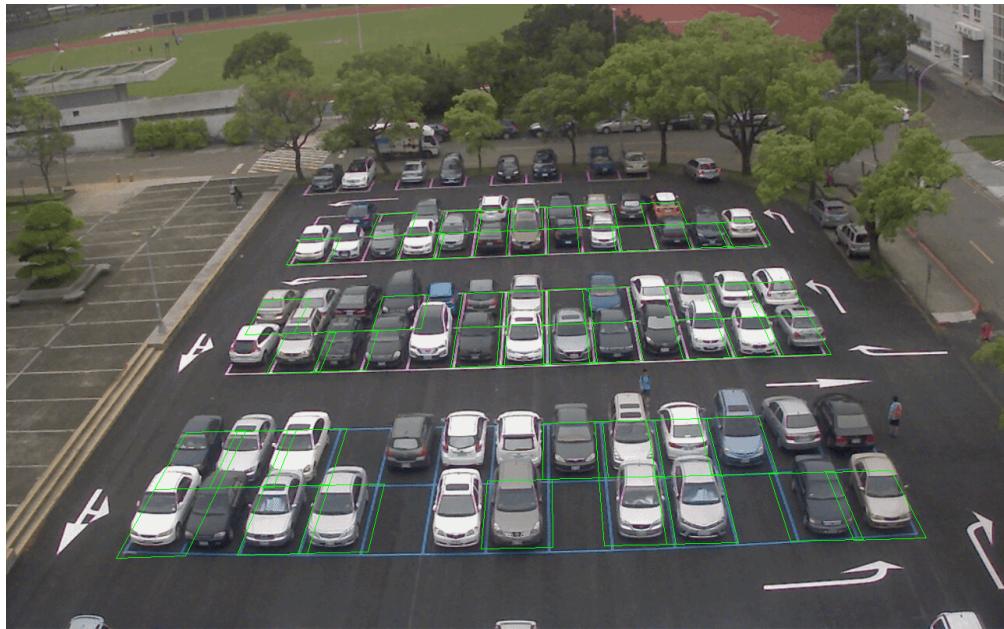


Figure 11: The detection result of Adaboost with 10 classifiers

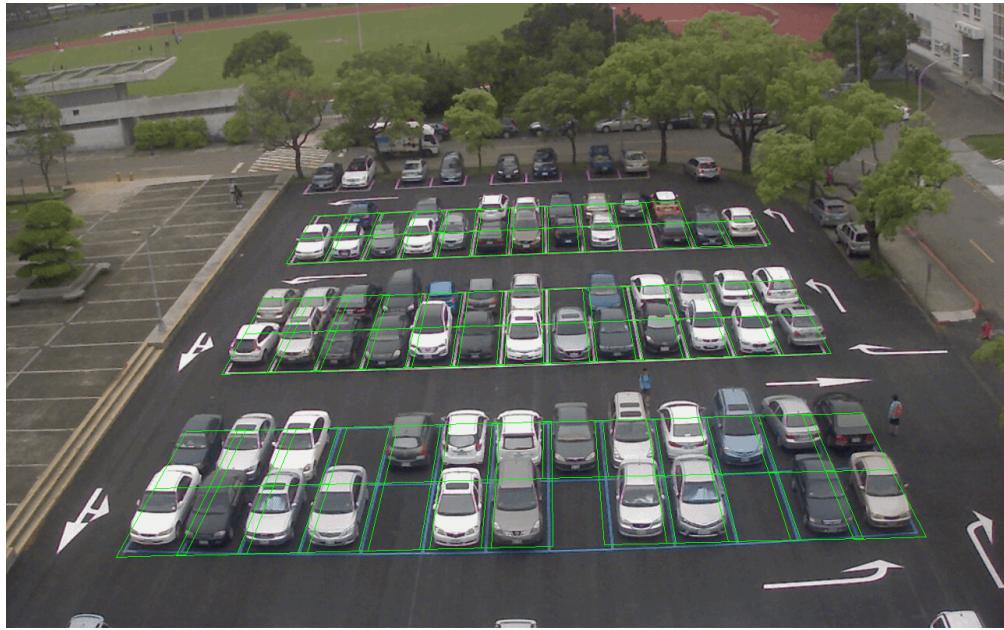


Figure 12: The detection result of YOLOv5 with threshold set to 0.1

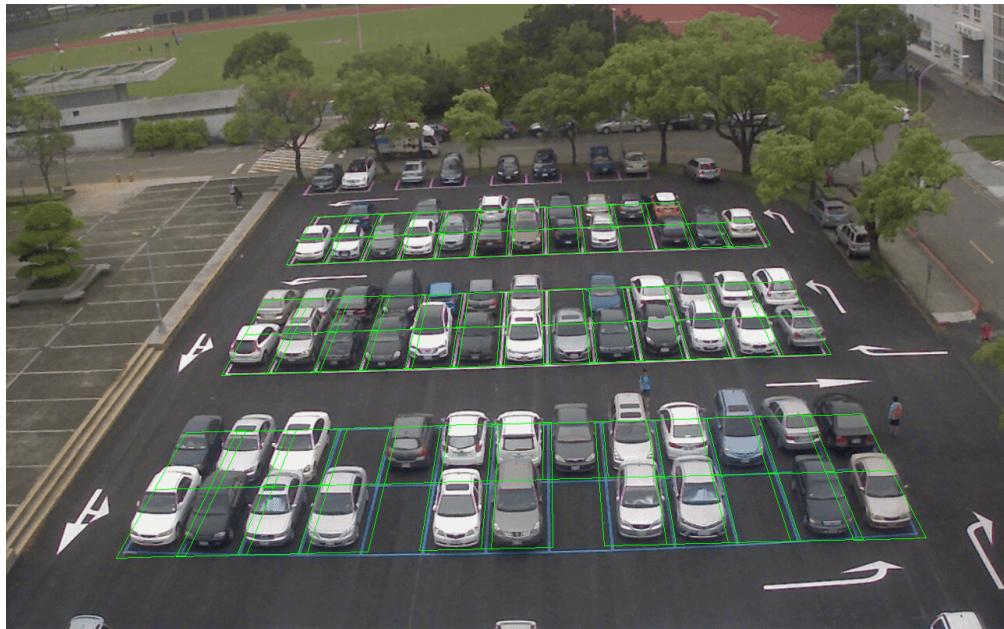


Figure 13: The detection result of YOLOv5 with threshold set to 0.2

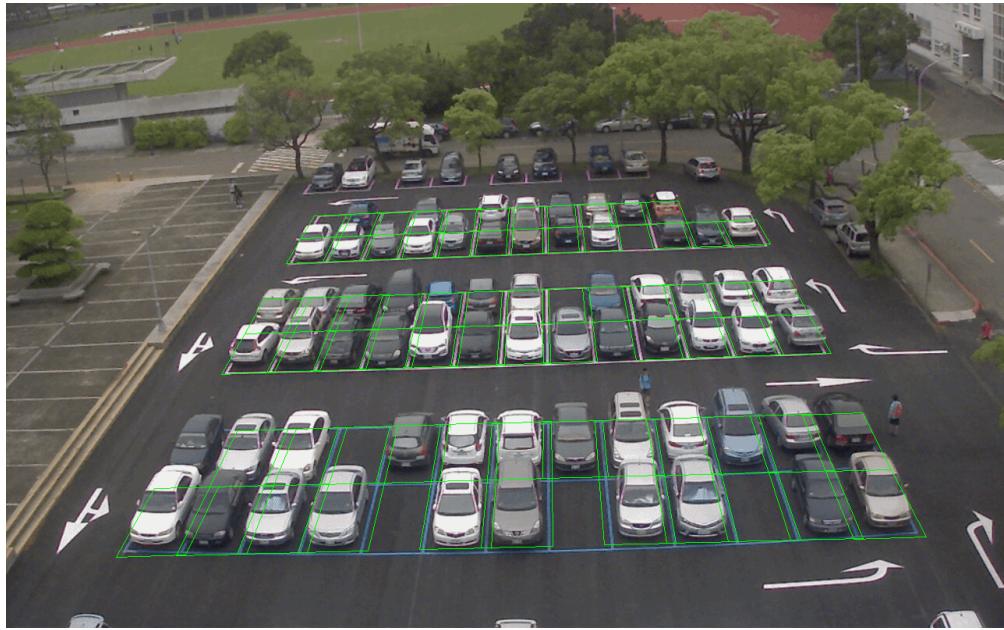


Figure 14: The detection result of YOLOv5 with threshold set to 0.3

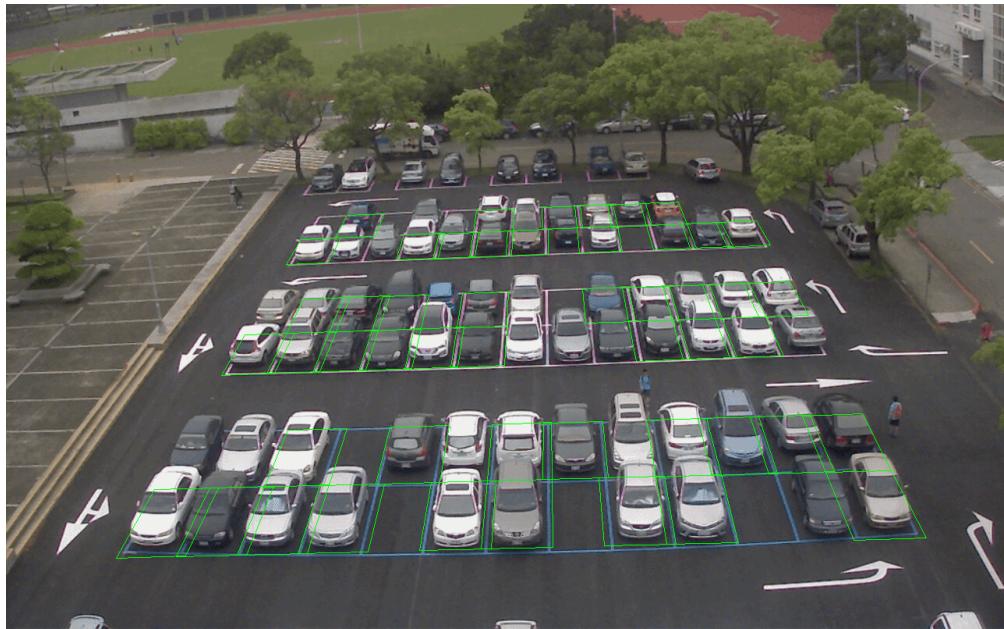


Figure 15: The detection result of YOLOv5 with threshold set to 0.4

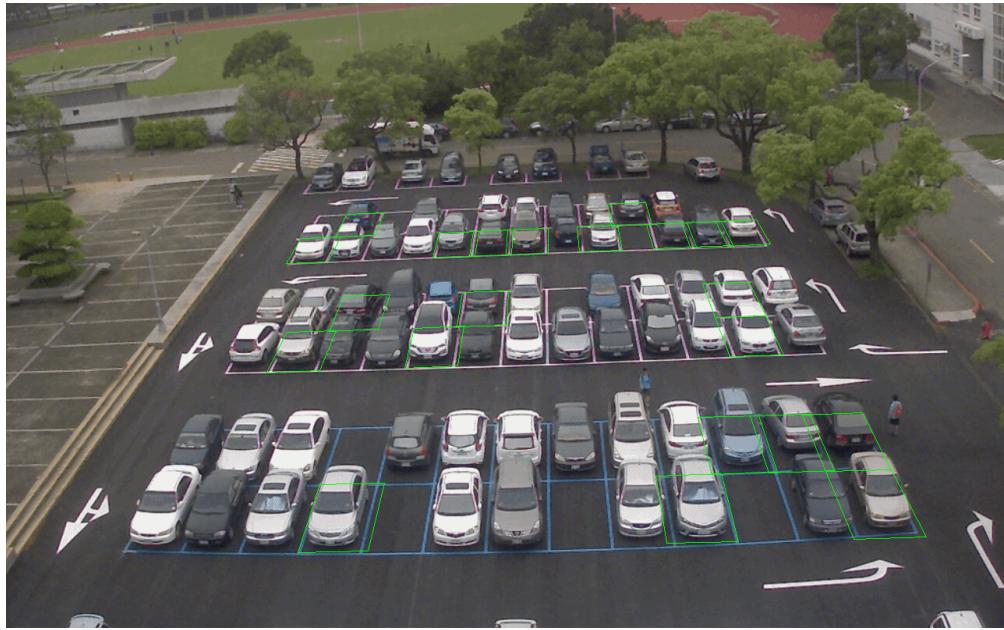


Figure 16: The detection result of YOLOv5 with threshold set to 0.5



Figure 17: The detection result of YOLOv5 with threshold set to 0.6

3.5.3 Discussion

As we can see in the plots and figures, the best model in this task is the YOLOv5 whose confidence thresholds are set to 0.1.

This result is inconsistent with the results present in Section 3.3.

I guess that this situation happened because in the real parking lot, most of the parking slots are occupied. Therefore, if we set the confidence threshold to a lower value (like 0.1), then the model will more likely to detect an occupied parking slot. This will make the model's performance better. However, if the parking lot we are going to detect has less occupied parking slots, setting a lower confidence threshold might not be a good idea. This may cause the detection results has lots of false positive samples.

In conclusion, I think the selection of the confidence threshold is hard. I suggest set the confidence threshold to 0.5. Because if the parking lot is half-occupied (like in training and testing datasets), setting the confidence threshold to 0.5 will have the best performance.

4 Problems

4.1 `alpha = math.log(1.0/beta)` ValueError: math domain error

The original code in `adaboost.py` line 59 is `weights = weights / np.linalg.norm(weights)`, but the default normalization method used in `np.linalg.norm` is L2-Norm, which may lead some divide by zero problem in the calculation of alpha. Therefore, I changed this line to `weights = weights / np.linalg.norm(weights, ord=1)`, which used the L1-Norm. After this change, the error in the title never shows up again.

5 Bonus: Use HSV channel to detect occupied parkint slots

5.1 Plot the histogram of HSV channel of “car” and “non-car” datasets

Out of curious, I plot the histogram of car and non-car training datasets HSV channels after converting the image from RGB to HSV.

The result is shown as Figure 18, 19 and 20.

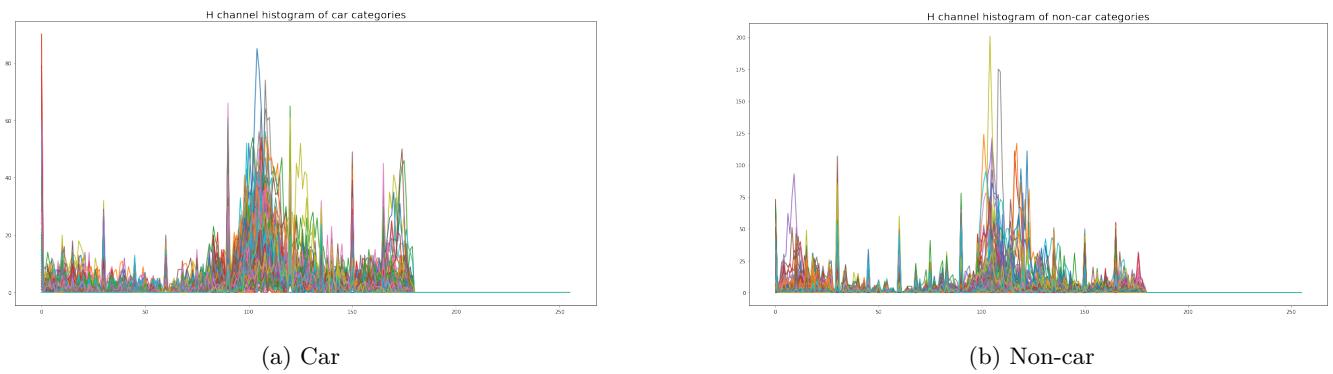


Figure 18: The H channel histogram of “car” and “non-car” images in training datasets.

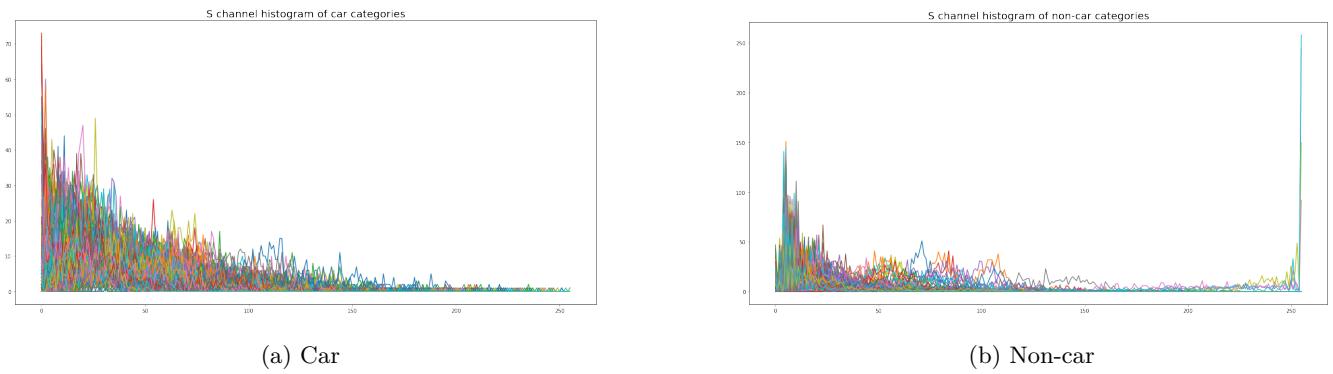


Figure 19: The S channel histogram of “car” and “non-car” images in training datasets.

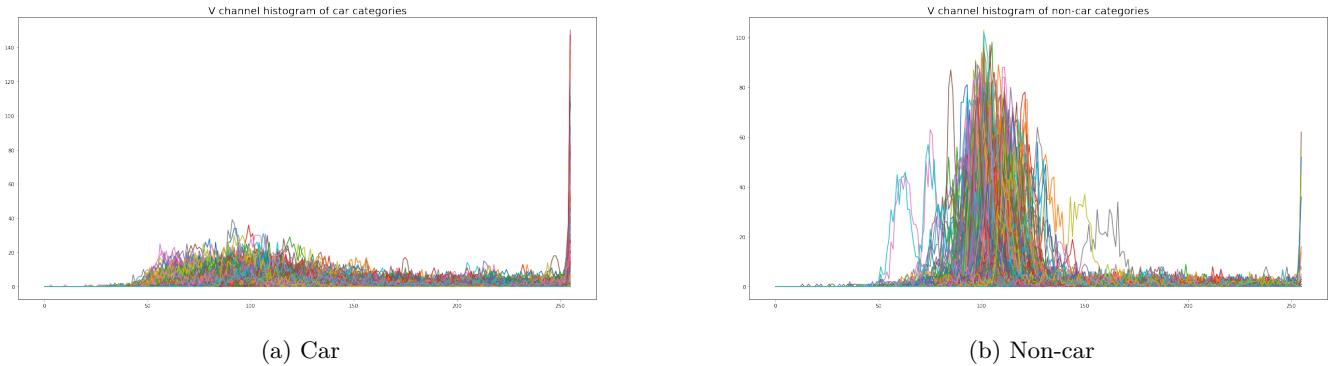


Figure 20: The V channel histogram of “car” and “non-car” images in training datasets.

As we can see, the V channel of “car” and “non-car” images have significant differences.

Most of the “non-car” images have a peak intensity is in the range of [40, 200] and their maximum y-value (number of pixels) are greater than 40.

On the other hand, most of the “car” images’ maximum y-values are less than 35.

Therefore, we can use this feature to classify whether this parking slot is occupied or not. Next section will discuss how to write the actual code to make this method work.

5.2 The detection code of the “V-channel histogram” method

After conducting some experiments, I set the highest peak range to [40, 200] and the threshold to 27. Below is the code I implemented to detect the parking slot using the “V-channel histogram” method.

Code 6: classify function(bonus classify.py)

```
1 import cv2  
2 import numpy as np  
3  
4 def classify(img):
```

```

5     v = img[:, :, 2] # Extract the V channel from an HSV images
6     hist_v = cv2.calcHist([v], [0], None, [256], [0, 256]) # Use cv2.calcHist to calculate the histogram.
7     # If hist_v's highest peak occurred in [40, 200] and its y-value is greater than 27, then we consider
8     # this parking slot isn't occupied
9     if 40 <= np.argmax(hist_v) and np.argmax(hist_v) <= 200 and 27 <= np.max(hist_v):
10        return False
11    else:
12        return True

```

Because we need to convert images to HSV, I also edit `datasets.py` to convert images in advance.

Code 7: `bonus_dataset.py`

```

1 import os
2 import cv2
3
4 def loadImages(dataPath):
5     """
6         Load all Images in the folder and transfer a list of tuples.
7         The first element is the numpy array of shape (m, n) representing the image.
8         (remember to resize and convert the parking space images to 36 x 16 grayscale images.)
9         The second element is its classification (1 or 0)
10        Parameters:
11            dataPath: The folder path.
12        Returns:
13            dataset: The list of tuples.
14        """
15    # Begin your code (Part 1)
16    dataset = [] # Declare an empty list to save the grayscale images
17
18    # Process images in "car" directory
19    for item in os.listdir(os.path.join(dataPath, "car")): # Use os.path.join to generate paths
20        img = cv2.imread(os.path.join(dataPath, "car", item)) # Read image from files
21        img = cv2.resize(img, (36, 16)) # Resize the image from (360, 160) to (36, 16)
22        img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # Convert image to HSV image
23        data = (img, 1) # Create a tuple to store image and Label and
24        # because all images in "car" folder is the occupied parking space, the label is set to 1
25        dataset.append(data) # Append the tuple to the dataset list
26

```

```

27     for item in os.listdir(os.path.join(dataPath, "non-car")): # Do the same thing as above but this time
28         ↵ is for "non-car" folder
29         img = cv2.imread(os.path.join(dataPath, "non-car", item))
30         img = cv2.resize(img, (36, 16))
31         img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
32         data = (img, 0) # Not occupied parking space, label is set to 0
33         dataset.append(data)
34
35     # End your code (Part 1)

36
37
38     return dataset

```

5.3 Results

5.3.1 Comparison on testing datasets

Table 6: Performance of Adaboost, YOLOv5 and V-Channel Method on testing datasets

Model	FP Rate (%)	FN Rate (%)	Accuracy (%)	Precision (%)	Recall (%)	F1-score
Adaboost	8.33	19.33	86.17	91.67	82.58	86.89
YOLOv5	1.67	1.00	98.67	98.33	98.99	98.66
V-Channel	3.33	1.33	97.6	96.67	98.64	97.64

As we can see in Table 6, though our method's performance is slightly worse than YOLOv5 , the advantage of V-channel method is that the detection speed is much faster than YOLOv5, and running the algorithm does not require an expensive GPU or chip.

5.3.2 Comparison on an real parking lot

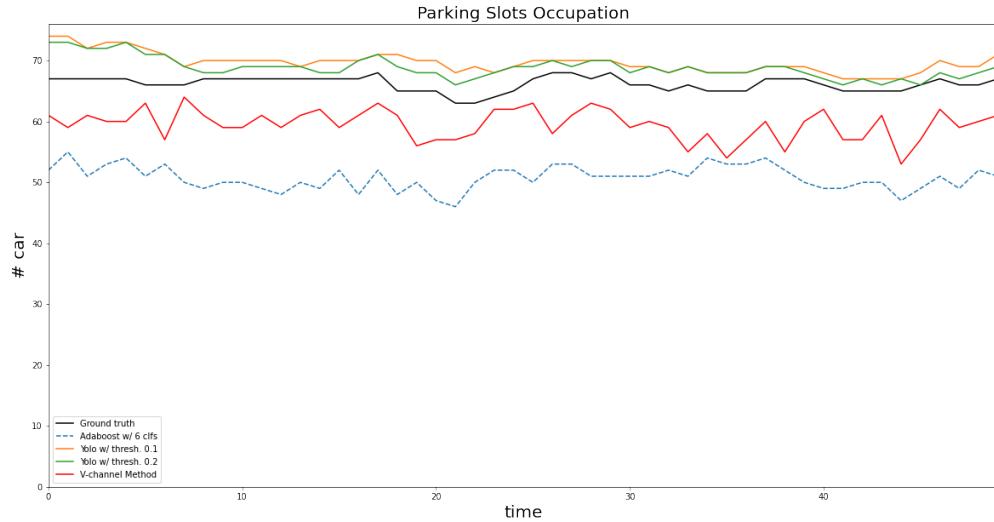


Figure 21: The number of occupied parking slots detected by Adaboost(with 6 classifiers) , YOLOv5(with threshold 0.1 and 0.2), V-channel method and the ground truth

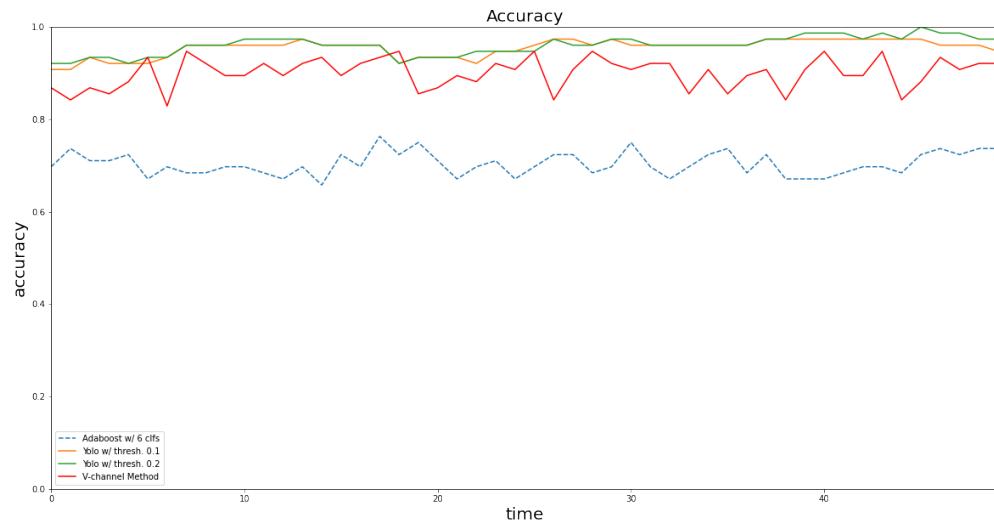


Figure 22: The Accuracy of Adaboost(with 6 classifiers) , YOLOv5(with threshold 0.1 and 0.2) and V-channel method

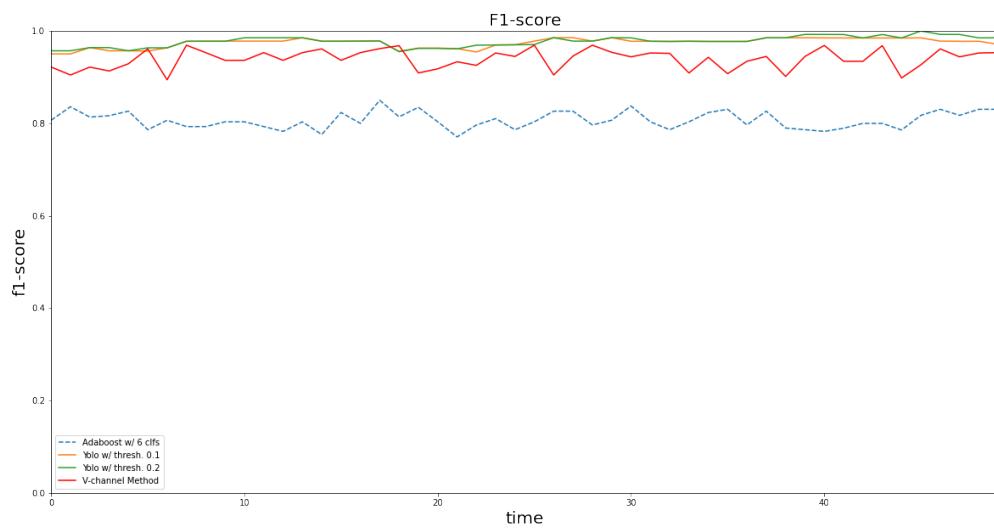


Figure 23: The F1-score of Adaboost(with 6 classifiers) , YOLOv5(with threshold 0.1 and 0.2) and V-channel method



Figure 24: The detection result of V-channel method