Introduction to Artificial Intelligence HW 1 Report

110550088 李杰穎

March 16, 2022

1 Code and Explanation

Below 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`)

```python
import os
import cv2
import numpy as np
def loadImages(dataPath):
    """
    Load all Images in the folder and transfer a list of tuples.
    The first element is the numpy array of shape (m, n) representing the image.
    (remember to resize and convert the parking space images to 36 x 16 grayscale images.)
    The second element is its classification (1 or 0)
        Parameters:
        dataPath: The folder path.
        Returns:
```

```
        dataset: The list of tuples.
    """
    # Begin your code (Part 1)
    dataset = [] # Declare an empty list to save the grayscale images

    # Process images in "car" directory
    for item in os.listdir(os.path.join(dataPath, "car")): # Use os.path.join to
    ↪ generate paths
        img = cv2.imread(os.path.join(dataPath, "car", item)) # Read image from files
        img = cv2.resize(img, (36, 16)) # Resize the image from (360, 160) to (36,
        ↪ 16)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert image to grayscale
        ↪ image
        data = (img, 1) # Create a tuple to store image and label and
        # because all images in "car" folder is the occupied parking space, the label
        ↪ is set to 1
        dataset.append(data) # Append the tuple to the dataset list

    for item in os.listdir(os.path.join(dataPath, "non-car")): # Do the same thing as
    ↪ above but this time is for "non-car" folder
        img = cv2.imread(os.path.join(dataPath, "non-car", item))
        img = cv2.resize(img, (36, 16))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        data = (img, 0) # Not occupied parking space, label is set to 0
        dataset.append(data)
    # End your code (Part 1)

    return dataset
```

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

```python
def selectBest(self, featureVals, iis, labels, features, weights):
    """
    Finds the appropriate weak classifier for each feature.
    Selects the best weak classifier for the given weights.
        Parameters:
        featureVals: A numpy array of shape (len(features), len(dataset)).
            Each row represents the values of a single feature for each training sample.
        iis: A list of numpy array with shape (m, n) representing the integral images.
        labels: A list of integer.
            The ith element is the classification of the ith training sample.
        features: A numpy array of HaarFeature class.
        weights: A numpy array with shape(len(dataset)).
            The ith element is the weight assigned to the ith training sample.
        Returns:
        bestClf: The best WeakClassifier Class
        bestError: The error of the best classifer
    """
    # Begin your code (Part 2)
    # Init. WeakClassifier by the feature in features list
    # And append them all into the clfs list
    clfs = [WeakClassifier(feature=feature) for feature in features]

    # Declare bestClf and bestError to store the currently best classifier and its error
    bestClf = None
    bestError = sum(weights) # The max error is the sum of weights

    for clf in clfs: # Iterate all classifer in clfs
        error = 0    # Declare a variable to track the error of the current clf
        for i in range(len(iis)): # Iterate all image sample
            if clf.classify(iis[i]) != labels[i]: # When the prediction of the model is
            ↪ different with the lable
                error += weights[i] # Add weights to error
```

```
    if error < bestError: # If the error is smaller than the best error, then classifier is the currently best classifer
        bestError = error # Change bestError to current error
        bestClf = clf     # Save this classifier as bestClf

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

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

```
print('Start training your classifier')
for t in range(1, 11): # Train with different T (1 ~ 10)
    print(f"Training with T={t}")
    clf = adaboost.Adaboost(T=t) # Init. clf using the given T
    clf.train(trainData) # Train of train data
    clf.save(f'clf_300_{t}') # Save the model file as clf_300_<t>
    clf = adaboost.Adaboost.load(f'clf_300_{t}') # Load the model after saving

    # Evaluate the model using already written utils.evaluate function
    print('\nEvaluate your classifier with training dataset')
    utils.evaluate(clf, trainData) # Use train data to evaluate model

    print('\nEvaluate your classifier with test dataset')
    utils.evaluate(clf, testData) # Use test data to evaluate model

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

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

```python
def detect(dataPath, clf, t=10):
    """
    Please read detectData.txt to understand the format.
    Use cv2.VideoCapture() to load the video.gif.
    Use crop() to crop each frame (frame size = 1280 x 800) of video to get parking space images. (image size = 360 x 160)
    Convert each parking space image into 36 x 16 and grayscale.
    Use clf.classify() function to detect car, If the result is True, draw the green box on the image like the example provided on the spec.
    Then, you have to show the first frame with the bounding boxes in your report.
    Save the predictions as .txt file (Adaboost_pred.txt), the format is the same as GroundTruth.txt.
    (in order to draw the plot in Yolov5_sample_code.ipynb)

      Parameters:
        dataPath: the path of detectData.txt
      Returns:
        No returns.
    """
    # Begin your code (Part 4)
    cords = [] # Declare a list that stores the cordinate of each parking slots
    with open(dataPath) as file:
        num_of_parking = int(file.readline()) # Read the number total parking slots from the first line of files
        for _ in range(num_of_parking): # Iterate all lines
            tmp = file.readline() # Read a line in file
            tmp = tmp.split(" ") # Split the line using " "
            res = tuple(map(int, tmp)) # Convert the string type to int type using the built-in map function
            cords.append(res) # Append the cordinates to "cords" list

```

```
    cap = cv2.VideoCapture(os.path.join(dataPath, "..", "video.gif")) # Use cv2.VideoCapture to read video.gif
    frame = 0 # Counter to track current frame number
    output_gif = [] # Declare a list to store each processed frame of output.gif
    first_frame = True # A flag that will help us store the processed first frame images
    while True:
        detect_label = [] # Declare a list to store the detect results of each parking slots
        frame += 1 # Make frame number add 1
        _, img = cap.read() # Read a frame of video.gif
        if img is None: # If all frame are read, then img is None
            break # If None, then break
        for cord in cords: # Iterate all cords
            pic = crop(*cord, img) # Use * to unpack cord e.g. (x1, y1, ..., x4, y4) -> x1, y1, ..., x4, y4
            pic = cv2.resize(pic, (36, 16)) # Resize image to (36, 16)
            pic = cv2.cvtColor(pic, cv2.COLOR_RGB2GRAY) # Convert image to grayscale images
            detect_label.append(clf.classify(pic)) # Use clf.classify to detect whether the parking slot is occupied or not
                                                   # And append the result to "detect_label" list
        for i, label in enumerate(detect_label): # Iterate all detect_label
            if label: # If the model detects that this parking slot is occupied
                pos = [[cords[i][idx], cords[i][idx+1]] for idx in range(0, 8, 2)] # Add the four points of the rectangle to "pos" list
                pos[2], pos[3] = pos[3], pos[2] # swap pos[2] and pos[3]
                pos = np.array(pos, np.int32) # Convert python built-in list to numpy array
                cv2.polylines(img, [pos], color=(0, 255, 0), isClosed=True) # Use cv2.polylines to draw rectangle
```

```
49
50          output_gif.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # Append the result
            ↪ image of each frame to output_gif list (Also convert the color)
51          if first_frame: # If this frame is the first frame in video.gif
52              first_frame = False # After getting into this block, set first_frame to
                ↪ False
53              cv2.imwrite(f"Adaboost_first_frame_{t}.png", img) # Save the results of
                ↪ first frame
54          with open(f"Adaboost_pred_{t}.txt", "a") as txt: # Open the
            ↪ Adaboost_pred_<t>.txt and use append mode to add new line to file
55              res = "" # Declare an empty string
56              for i, label in enumerate(detect_label): # Iterate all labels
57                  if label:
58                      res += "1" # If the parking slot is occupied, then write 1 to
                        ↪ file
59                  else:
60                      res += "0" # Else write 1
61                  if i != len(detect_label) - 1:
62                      res += " " # If not the last label in a frame, then write a space
                        ↪ to seperate each label
63                  else:
64                      res += "\n" # Else write newline character
65              txt.write(res) # Write the res string to file
66      imageio.mimsave(f'results_{t}.gif', output_gif, fps=2) # Use imageio.imsave to
        ↪ save result gif and set fps to 2
67      # 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 cordinate of each parking slots
```

```
with open(dataPath) as file: # Open ”detectData.txt”
    num_of_parking = int(file.readline()) # Read the number total parking slots from
    ↪ the first line of files
    for _ in range(num_of_parking): # Iterate all lines
        tmp = file.readline() # Read a line in file
        tmp = tmp.split(” ”) # Split the line using ” ”
        res = tuple(map(int, tmp)) # Convert the string type to int type using the
        ↪ built-in map function
        cords.append(res) # Append the cordinates to ”cords” list

cap = cv2.VideoCapture(os.path.join(”HW1_material”, ”detect”, ”video.gif”)) # Use
↪ cv2.VideoCapture to read video.gif
frame = 0 # Counter to track current frame number
output_gif = [] # Declare a list to store each processed frame of output.gif
first_frame = True # A flag that will help us store the processed first frame images
while True:
    detect_label = [] # Declare a list to store the detect results of each parking
    ↪ slots
    frame += 1 # Make frame number add 1
    _, img = cap.read() # Read a frame of video.gif
    if img is None: # If all frame are read, then img is None
        break # If None, then break
    for cord in cords: # Iterate all cords
        pic = crop(*cord, img) # Use * to unpack cord e.g. (x1, y1, ..., x4, y4) ->
        ↪ x1, y1, ..., x4, y4
        pic = cv2.resize(pic, (36, 16)) # Resize image to (36, 16)
        detect_label.append(yolov5_func.classify(pic, weight_path,
        ↪ confidence_threshold, (36, 16))) # Use yolov5_func.classify to detect
        ↪ whether the parking slot is occupied or not
```

```
25
26      for i, label in enumerate(detect_label): # Iterate all detect_label
27          if label: # If the model detects that this parking slot is occupied
28              pos = [[cords[i][idx], cords[i][idx+1]] for idx in range(0, 8, 2)] # Add
                ↪  the four points of 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.polylines(img, [pos], color=(0, 255, 0), isClosed=True) # Use
                ↪  cv2.polylines to draw rectangle
32
33      output_gif.append(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)) # Append the result image
        ↪  of each frame to output_gif list (Also convert the color)
34      if first_frame: # If this frame is the first frame in video.gif
35          cv2.imwrite(img_save_path, img) # After getting into this block, set
            ↪  first_frame to False
36          first_frame = False # Save the results of first frame
37      with open(txt_save_path, "a") as txt: # Open txt file and use append mode to add
        ↪  new line to file
38          res = "" # Declare an empty string
39          for i, label in enumerate(detect_label): # Iterate all labels
40              if label:
41                  res += "1" # If the parking slot is occupied, then write 1 to file
42              else:
```

```
                res += "0" # Else write 1
            if i != len(detect_label) - 1:
                res += " " # If not the last label in a frame, then write a space to
                 ↪  seperate each label
            else:
                res += "\n" # Else write newline character
        txt.write(res) # Write the res string to file
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, we only change the hyperparameter `T`, which indicates the number of classifers that will be used in the training process.

We've 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 we can change is the confidence threshold, which is a probability threshold. If the YOLOv5's output is greater than the threshold, then we consider that the parking slot is occupied.

We've 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".

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

And the definition of F-score is:

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

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

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

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

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

3.2 Adaboost Comparision

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

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

3.3 YOLOv5 Comparison

3.4 Visual Results

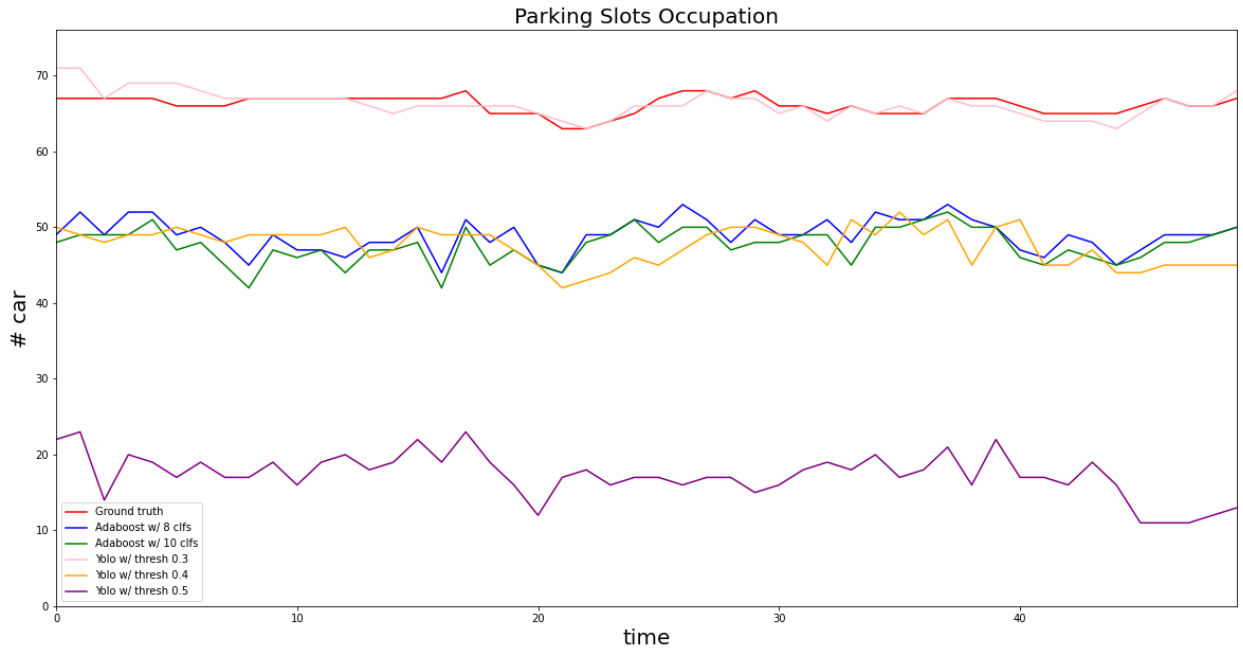

Figure 1: The number of occupied parking slots detected by Adaboost(with 8 and 10 classifers) , YOLOv5(with threshold 0.3, 0.4 and 0.5) and the ground truth

We use equation 5 to calculate the accuracy in figure **??**.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{Total}} \tag{5}$$

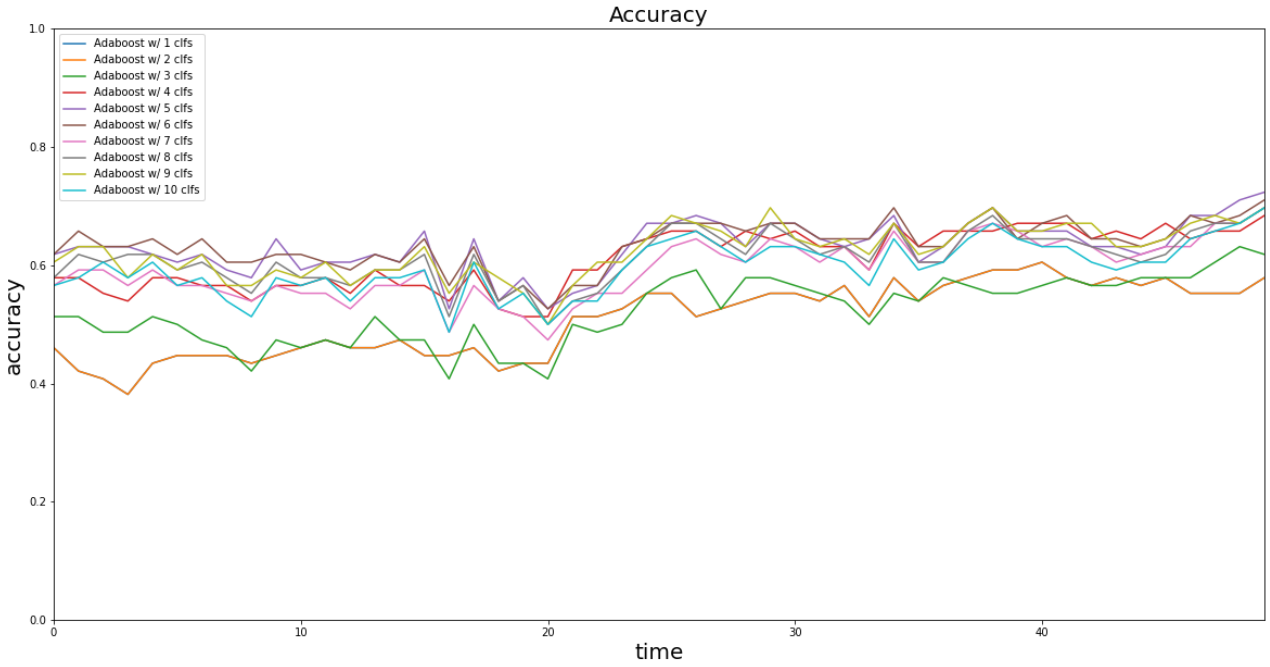

Figure 2: Accuracy of Adaboost ($T = 1$ to $T = 10$) over time

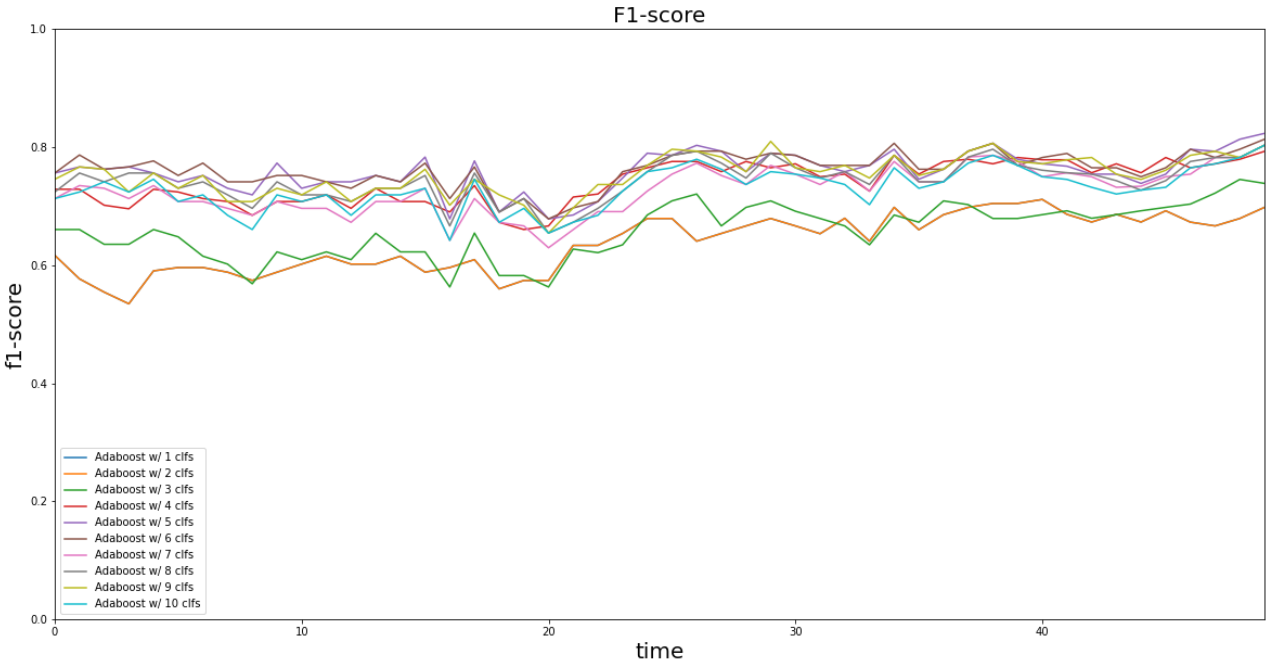

Figure 3: F1-score of Adaboost ($T = 1$ to $T = 10$) over time

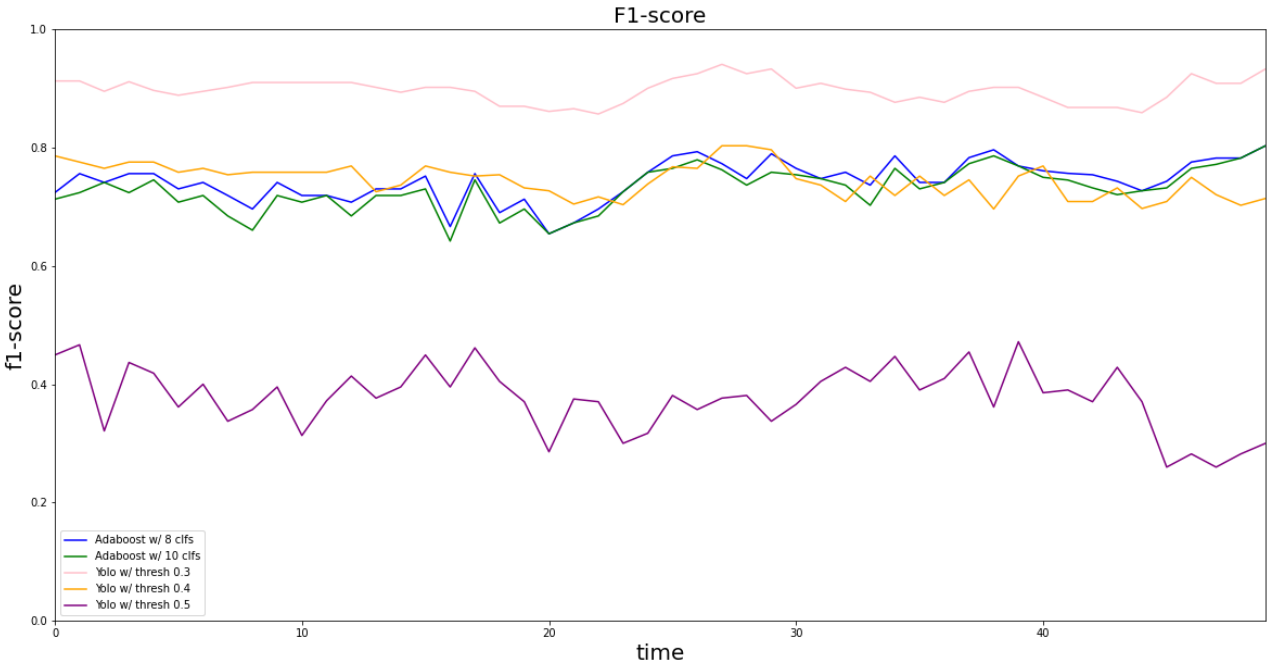

Figure 4: F1-score of Adaboost (w/ 8 and 10 classifers) and YOLOv5 (w/ threshold 0.3, 0.4 and 0.5) over time

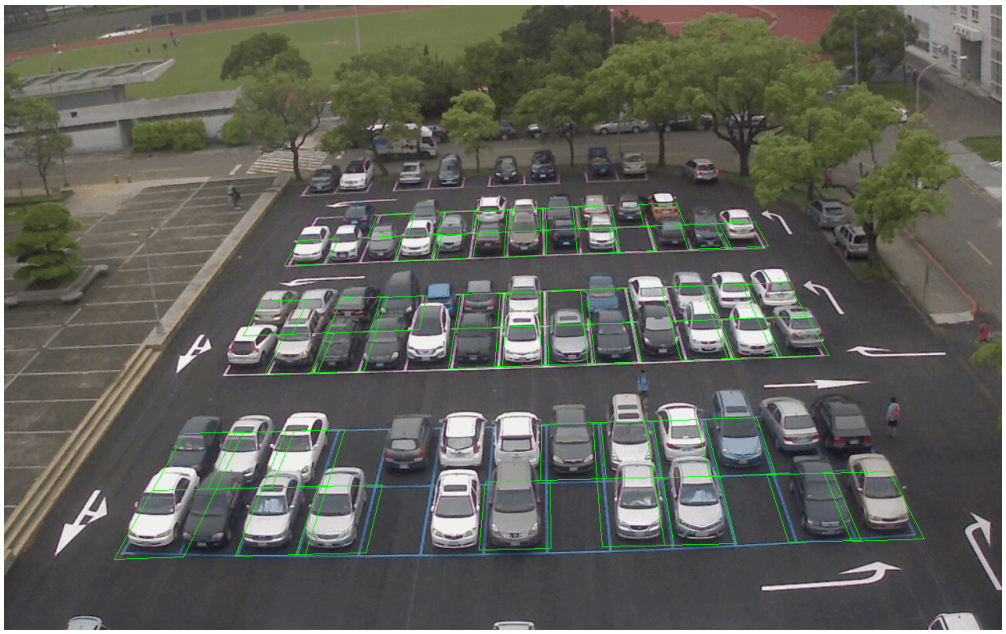

Figure 5: The detection result of Adaboost with 8 classfiers

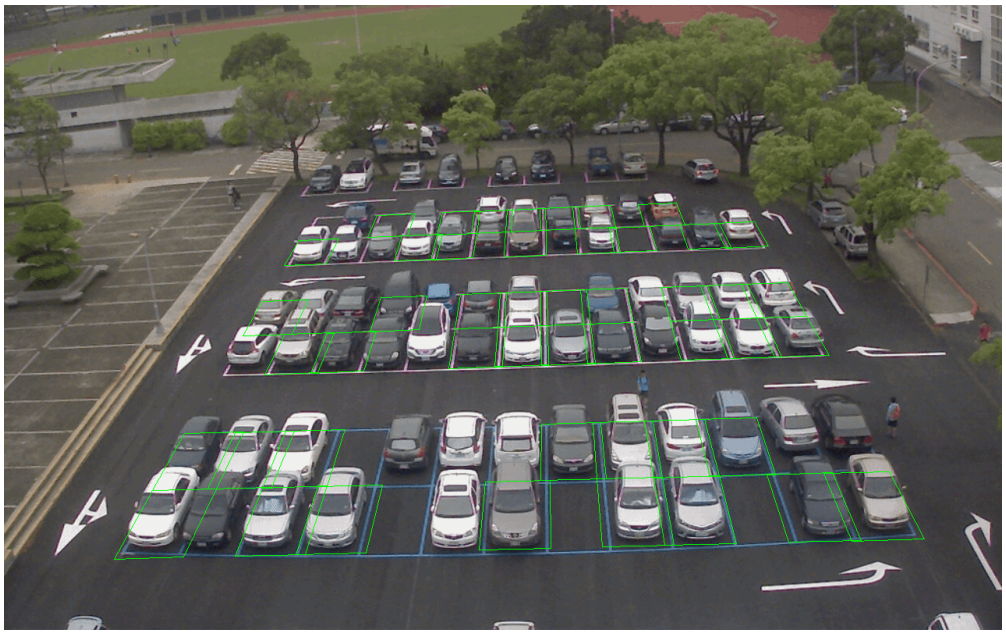

Figure 6: The detection result of Adaboost with 10 classfiers

Figure 7: The detection result of YOLOv5 with threshold set to 0.3

Figure 8: The detection result of YOLOv5 with threshold set to 0.4

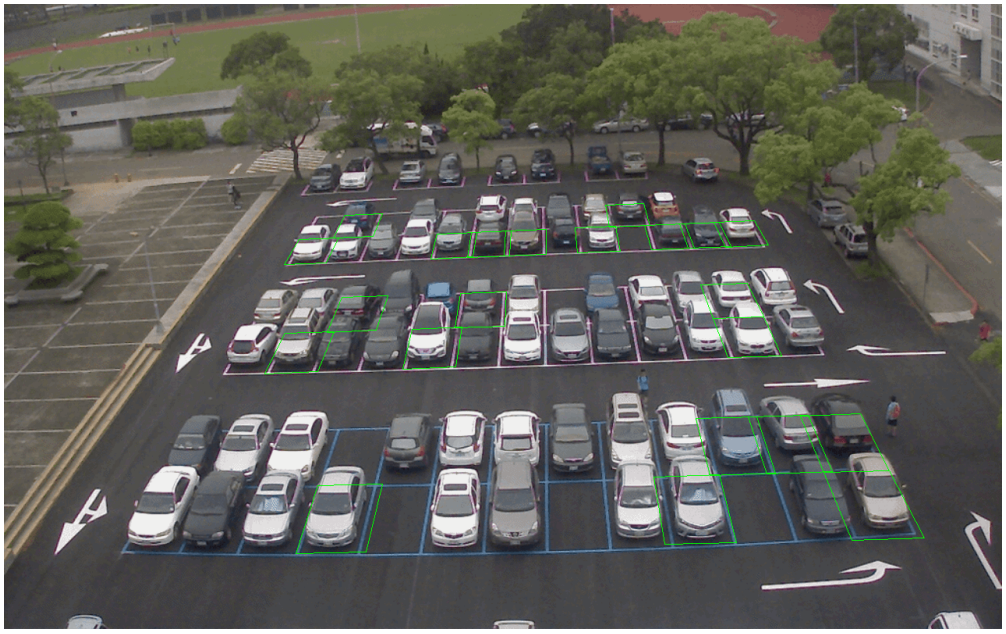

Figure 9: The detection result of YOLOv5 with threshold set to 0.5

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 divides by zero problem in the calculation of alpha. Therefore, we 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.

4.2