AI_HW1_repo_111550076

notion link:

AI_HW1_repo_111550076

Task A

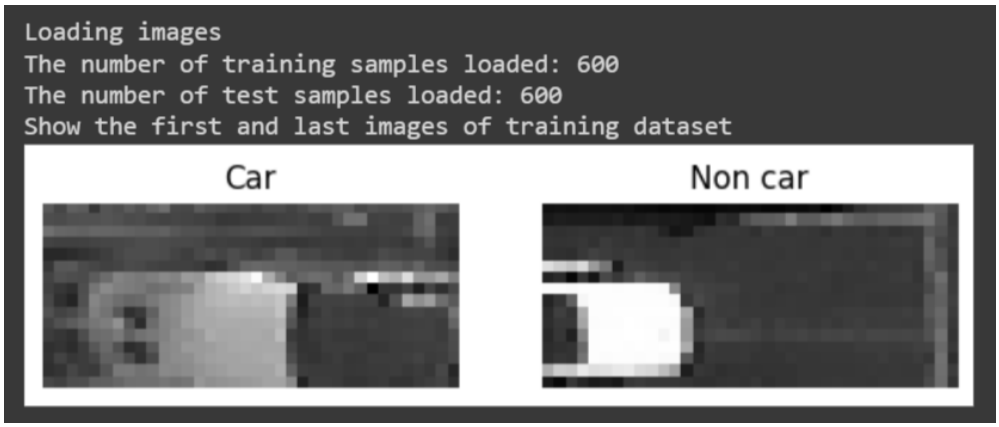
Part 1 (dataset.py):

Revise the following sentence with the cv2 package, such as cv2.imread and cv2.resize, to meet the data requirements. In the classification section, label the "car" folder as 1 and the "non-car" folder as 0.

```
import os
import cv2

def load_images(data_path):
    """
    Load all Images in the folder and transfer a list of tuples
    The first element is the numpy array of shape (m, n) represe
    (remember to resize and convert the parking space images to
    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 = []
    for category in ['car','non-car']:
        subfolder_path = os.path.join(data_path,category)
        classfication = 1 if category == 'car' else 0
```

```
        for filename in os.listdir(subfolder_path):
          image_path = os.path.join(subfolder_path, filename)
          img = cv2.imread(image_path,cv2.IMREAD_GRAYSCALE)
          if img is None:
            print(f"Warn")
            continue
          img_resized = cv2.resize(img,(36,16))
          dataset.append((img_resized,classfication))
    # End your code (Part 1)
    return dataset
```

result:



Part 2 (model.py):

Translate the data into x and y axis, create different models based on the model names, flatten the two-dimensional np array into a one-dimensional array, and feed it into the model.

result:

```python
class CarClassifier:
    def __init__(self, model_name, train_data, test_data):

        '''
        Convert the 'train_data' and 'test_data' into the format
        that can be used by scikit-learn models, and assign training images
        to self.x_train, training labels to self.y_train, testing images
        to self.x_test, and testing labels to self.y_test.These four
        attributes will be used in 'train' method and 'eval' method.
        '''

        self.x_train, self.y_train, self.x_test, self.y_test = None, None, None, None

        # Begin your code (Part 2-1)
        self.x_train = [x[0] for x in train_data]
        self.y_train = [x[1] for x in train_data]
        self.x_test = [x[0] for x in test_data]
        self.y_test = [x[1] for x in test_data]
        # End your code (Part 2-1)

        self.model = self.build_model(model_name)


    def build_model(self, model_name):
        '''
        According to the 'model_name', you have to build and return the
        correct model.
        '''
        # Begin your code (Part 2-2)
        if model_name == 'KNN':
            return KNeighborsClassifier(n_neighbors=5)
        elif model_name == 'RF':
            return RandomForestClassifier(n_estimators=100)
        elif model_name == 'AB':
            return AdaBoostClassifier(n_estimators= 50)
        # End your code (Part 2-2)

    def train(self):
        '''
        Fit the model on training data (self.x_train and self.y_train).
        '''
        # Begin your code (Part 2-3)
        x_train_flattened = np.array([img.flatten() for img in self.x_train])
        self.x_test= np.array([img.flatten() for img in self.x_test])
        self.model.fit(x_train_flattened,self.y_train)
        # End your code (Part 2-3)
```

```
Accuracy: 0.9733
Confusion Matrix:
[[285   1]
 [ 15 299]]
```

Question Explanation:

1. Explain the difference between parametric and non-parametric models:

`Parametric models` assume data follows a specific distribution, fully described by a fixed set of parameters. These models are computationally efficient, offer strong interpretability but may oversimplify reality, leading to poor fit if assumptions are incorrect.

`Non-parametric models` do not assume a fixed distribution for data, allowing the model structure to adapt based on data diversity. They are more flexible and can handle complex data structures, but are computationally expensive and may lack interpretability.

In summary, `parametric models` are widely used for their simplicity and computational efficiency but may lack the ability to capture complex data structures. `Non-parametric models` provide greater flexibility to adapt to various data forms but are more complex and computationally intensive. The choice between them depends on the specific problem, the nature of the data, and available computational resources.

---

2. What is ensemble learning? Please explain the difference between bagging, boosting and stacking.

Ensemble Learning combines multiple algorithms to improve predictive performance. Main forms include:

- **Bagging**: Reduces variance by creating multiple independent predictors and combining their outputs. Example: Random Forest.
- **Boosting**: Reduces bias by sequentially training predictors to correct previous errors. Examples: AdaBoost, Gradient Boosting.
- **Stacking**: Enhances accuracy by combining different models' predictions and using a meta-model to synthesize these predictions.

The choice of method depends on the specific problem, dataset characteristics, and needs.

---

3. Explain the meaning of the `"n_neighbors"` paremeter in *KNeightborsClassifier*, `"n_estimators"` in *RandomForestClassifier* and *AdaBoostClassifier*.

The `n_neighbors` parameter in *K-Nearest Neighbors* (KNN) specifies the number of nearest neighbors to consider when classifying a new data point. A too small value may lead to overfitting due to noise sensitivity, while a too large value can cause underfitting by overlooking local information. Optimal `n_neighbors` is crucial and typically determined through cross-validation.

In *RandomForestClassifier* and *AdaBoostClassifier*, `n_estimators` represents the number of weak learners (usually decision trees) to use. In Random Forest, it indicates the number of trees in the forest, affecting stability and predictive power. In AdaBoost, it denotes the maximum number of sequential weak learners, influencing model complexity and performance. For both ensemble methods, an appropriate `n_estimators` value, balancing performance and overfitting, is best found via cross-validation.

---

4. Explain the meaning of four numbers in the `confusion matrix`.

The `confusion matrix` is a tool for evaluating the performance of classification models, showing the relationship between actual and predicted categories. In a binary classification problem, it consists of four components:

- True Positives (TP): Top-left corner, indicating the number of positive instances correctly predicted as positive.
- False Negatives (FN): Bottom-left corner, representing the number of positive instances incorrectly predicted as negative.
- False Positives (FP): Top-right corner, showing the number of negative instances incorrectly predicted as positive.
- True Negatives (TN): Bottom-right corner, indicating the number of negative instances correctly predicted as negative.

The confusion matrix provides crucial information for assessing a model's performance, including accuracy and types of errors. From these four values, various performance metrics like accuracy, recall, precision, and the `F1 score` can be calculated, offering a comprehensive evaluation of the model's effectiveness.

---

5. In addition to “Accuracy”, “Precision” and “Recall” are two common metrics in classification tasks. How to calculate them? And under what circumstances would you use them instead of “Accuracy”?

Accuracy

Definition: Measures the overall correctness of the model.

$$Accuracy = (TP + TN)/(TP + TN + TF + FN)$$

Interpretation: The proportion of true results (both true positives and true negatives) among the total number of cases examined.

Precision

Definition: Indicates the proportion of actual positives among items labeled as positive.

$$Precision = TP/(TP + FP)$$

Interpretation: The ratio of correctly predicted positive observations to the total predicted positives. High precision relates to a low rate of false positives.

Recall

Definition: Reflects the proportion of actual positives correctly identified by the model.

$$Recall = TP/(TP + FN)$$

Interpretation: The ratio of correctly predicted positive observations to all observations in actual class - high recall relates to a low rate of false negatives.

Usage Scenarios

Precision

- **When to use**: When the cost of falsely classifying negative items as positive is significant.

- **Example**: In spam detection, classifying a regular email as spam is a more serious error than failing to classify a spam email correctly.

Recall

- **When to use**: When missing positive items carries a higher cost.
- **Example**: In disease diagnosis, failing to detect a sick patient is much worse than incorrectly diagnosing a healthy person as sick.

Part3(additional experiments)

Adjust the `n_neighbors` parameter for KNN and the `n_estimators` parameter for Random Forest (RF) and AdaBoost (AB). Test ten parameters around the default values provided by Scikit-Learn. Utilize the precision and recall values mentioned in the previous question to calculate the F1-score, which will be used to determine the best parameters.

Definition of F-score:

The F-score is defined as:

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

In practical applications, we often use the "F1-score," which means β=1. Therefore, the "F1-score" can be expressed as:

$$F1\text{-score} = 2\cdot \frac{\text{precision}\cdot \text{recall}}{\text{precision} + \text{recall}}$$

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

- KNN:

We test the `n_neighbors` from 3 ~ 21, and find out that at `n_neighbors = 3` have the highest F1-score(0.815).

n_neighbors	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
3	72.76	68.90	99.74	81.50
5	68.42	63.89	99.76	77.89
7	65.00	59.96	99.75	74.90
9	61.74	56.06	100.00	71.84
11	58.68	52.58	99.94	68.91
13	56.05	49.56	99.94	66.26
15	55.45	48.87	99.94	65.64
17	54.79	48.11	99.94	64.95
19	53.84	47.02	99.94	63.95
21	52.84	45.88	99.93	62.88

- Random Forest:

We test the `n_estimaters` from 40 ~ 220, and find out that at `n_estimaters = 220` has the highest F1-score(0.9902).

n_estimators	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
40	97.95	99.28	98.38	98.83
60	95.82	98.76	96.52	97.62
80	96.53	98.40	97.63	98.01
100	96.45	97.73	98.18	97.95
120	97.84	98.76	98.76	98.76
140	97.68	98.79	98.55	98.67
160	97.68	99.52	97.86	98.68
180	96.95	98.37	98.13	98.25
200	97.95	98.70	98.94	98.82
220	98.29	99.21	98.83	99.02

- AdaBoost:

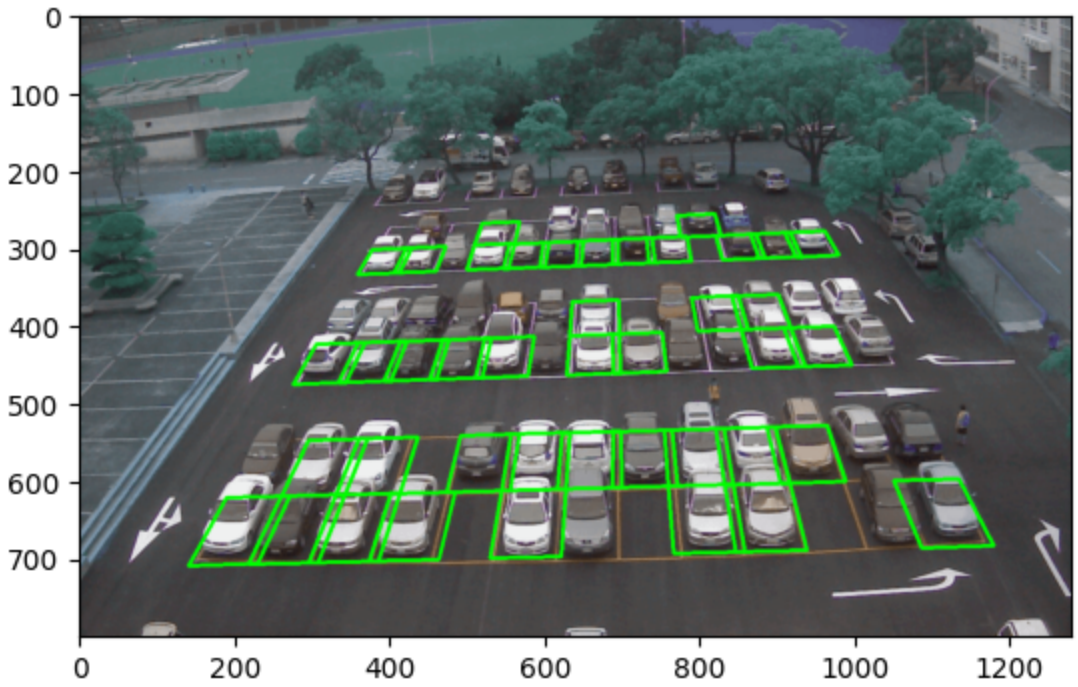
We test the `n_estimaters` from 10 ~ 100, and find out that at `n_estimaters = 80` has the highest F1-score(0.9554).

n_estimators	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
10	85.11	84.44	98.21	90.80
20	89.11	89.12	98.20	93.44
30	89.03	90.81	96.38	93.51
40	90.29	91.69	97.00	94.27
50	89.47	91.14	96.57	93.78
60	90.92	91.21	98.24	94.59
70	91.37	91.93	98.03	94.88
80	92.45	92.81	98.43	95.54
90	91.90	91.84	98.77	95.18
100	91.32	91.60	98.31	94.84

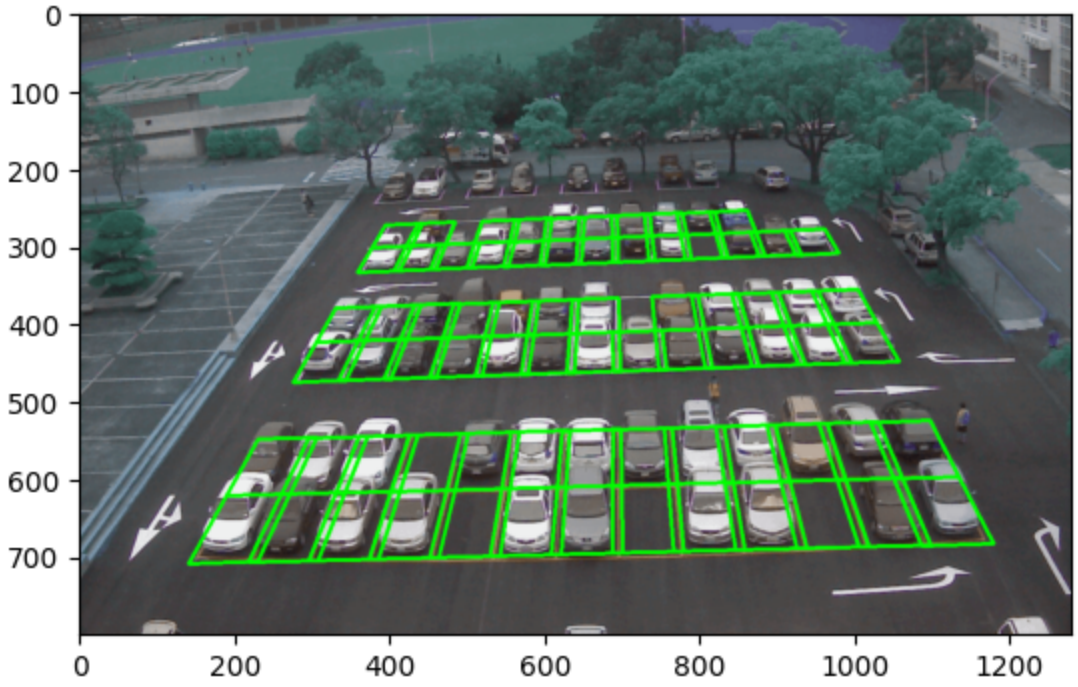
Part4 (detection.py) :

Use cv2.VideoCapture() to capture 50 frames. For each frame, perform the following operations: read all coordinate data, apply cropping with crop() to each image, then perform grayscale conversion, resize, and flattening. Call classify to determine if there is a car in the image. If there is, outline it with a green line, and save the results as a boolean array in a text file for later comparison with groundTruth.txt.

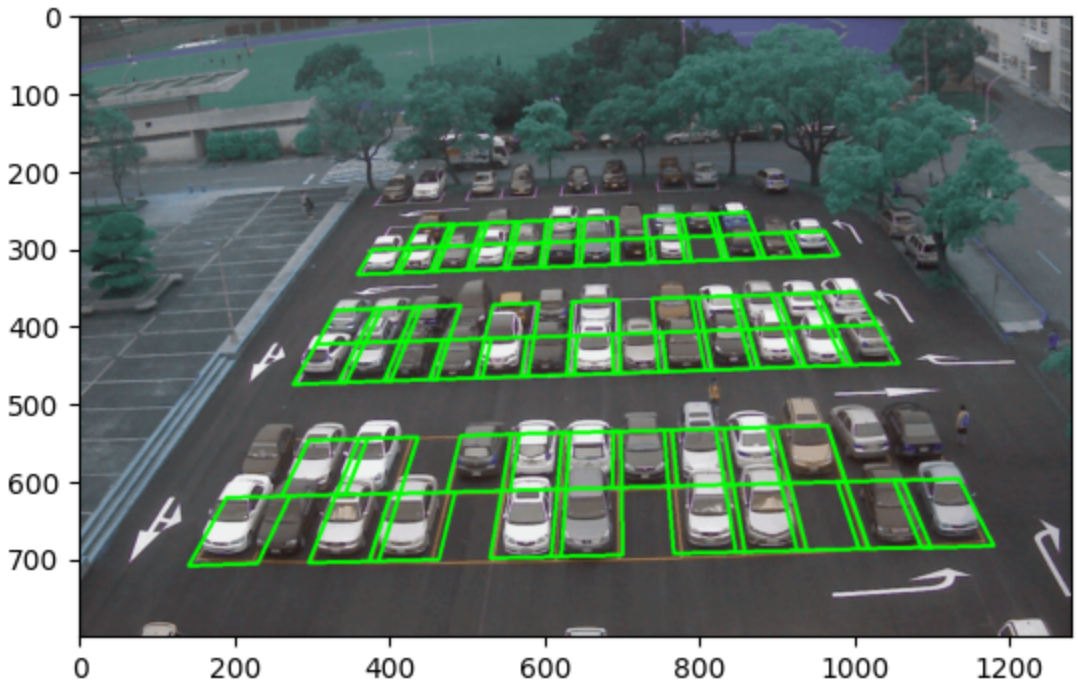
```python
# Begin your code (Part 4)
#read detectdata

with open(data_path,'r')as file:
  lines = file.readlines()

num_frames = int(lines[0])
coordinates = []
for line in lines[1:]:
  coordinates.append(list(map(int,line.strip().split())))
# load gif
gif = cv2.VideoCapture('data/detect/video.gif')
frames = []
prediction = []
success, image = gif.read()
i = 0
while success and i < 50:
  frame_prediction = []
  for coor in coordinates:
    x1,y1,x2,y2,x3,y3,x4,y4 = coor
    cropped_image = crop(x1,y1,x2,y2,x3,y3,x4,y4,image)
    gray_image = cv2.cvtColor(cropped_image,cv2.COLOR_BGR2GRAY)
    resized_image = cv2.resize(gray_image,(36,16))
    flattened_image = resized_image.flatten().reshape(1,-1)
    if clf.classify(flattened_image):
      frame_prediction.append(1)
      # Define the points
      point_list = [[x3, y3], [x4, y4], [x2, y2], [x1, y1]]

      points = np.array(point_list, dtype=np.int32)
      points = points.reshape((-1, 1, 2))
      cv2.polylines(image, [points], isClosed=True, color=(0, 255, 0), thickness=3)
    else:
      frame_prediction.append(0)
  frames.append(image)
  success,image = gif.read()
  i += 1
  prediction.append(frame_prediction)
gif.release()
plt.imshow(frames[0])
plt.show()
# End your code (Part 4)
```



result with KNN, n_neighbors = 3



result with RandomForest, n_estimators = 220

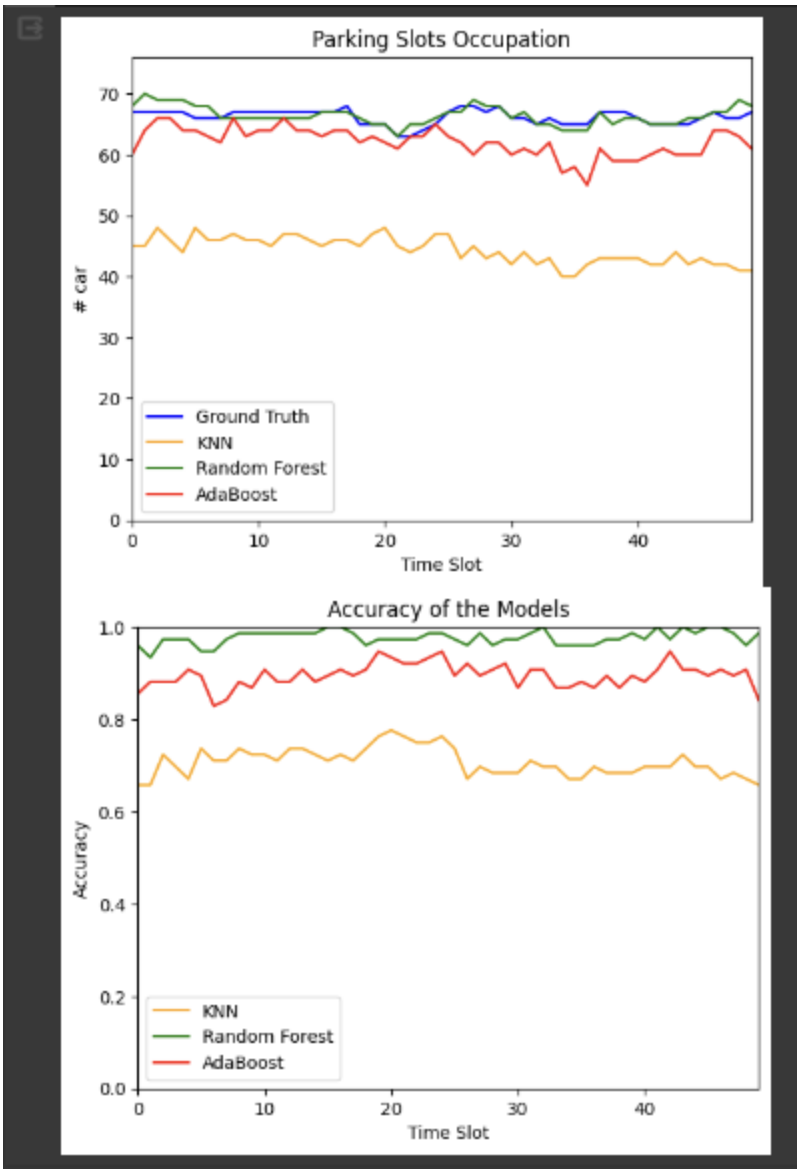


result with AdaBoost, n_estimators = 80

Part 5 (draw a line graph):

I've added it to main.py and obtained the graph. Run the three algorithms in advance, and save the results separately as knn_pred.txt, rf_pred.txt, and ab_pred.txt. Then plot these three results against the contents of groundtruth.txt for comparison. Simply compare each point with the ground truth, calculate the accuracy, and plot to compare the three algorithms.

result



Task B

Part 1 : Load a pre-trained model and directly apply it to the .png to detect the object.

Unzip 'HW1_material', move it to 'content/yolov7', run the evaluation, and find the photo at '/content/yolov7/runs/detect/parking_area/parking_area.png'.

Part 2: Learn to fine-tune yolov7 model.

Run 'Finetune the yolov7 model', wait 1.5 hours, then conduct train and test evaluations using 'best.pt'. Adjust the threshold using '--conf 0.1' to confirm 0.1 as the optimal choice.

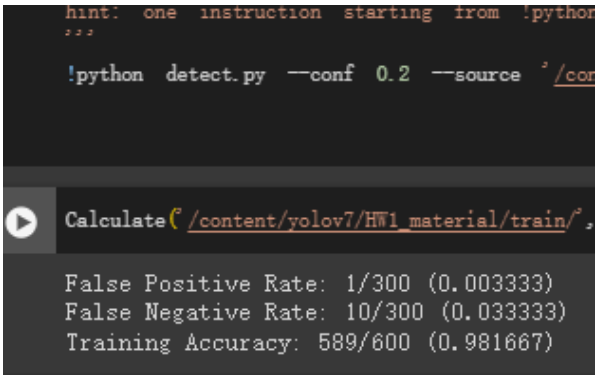
```
!python detect.py --conf 0.1 --source '/co

# Calculate yolov7 performance
# You have to adjust the parameters to ge
# Warning: make sure that txtpath is curre

Calculate('/content/yolov7/HW1_material/train/','
```

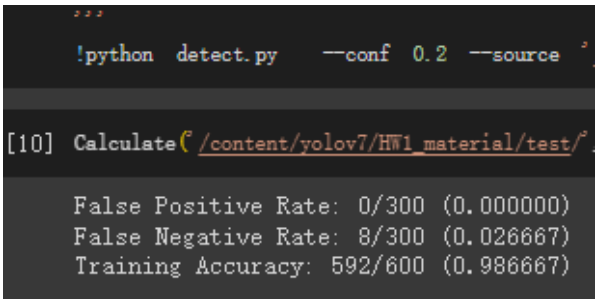
```
False Positive Rate: 1/300 (0.003333)
False Negative Rate: 8/300 (0.026667)
Training Accuracy: 591/600 (0.985000)
```

```
'''
modelPath = os.path.join(savePath,'best.pt')
!python detect.py --conf 0.1 --source '/content/yolov7/HW1_material/t

'''
Run evaluation like cell above
you must use parameter: --source /content/yolov7/HW1_material/test/non-
hint: one instruction starting from !python
'''
!python detect.py --conf 0.1 --source '/content/yolov7/HW1_material/t

Calculate('/content/yolov7/HW1_material/test/','/content/yolov7/runs/detect/
```

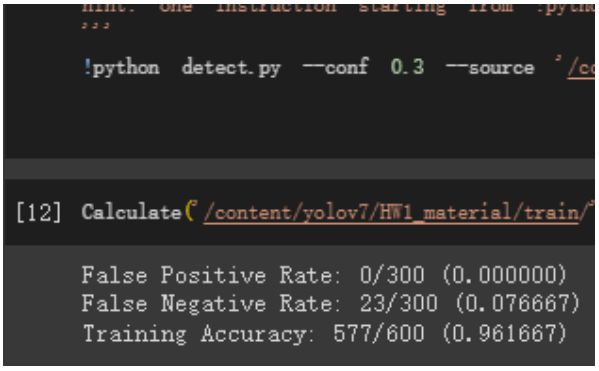
```
False Positive Rate: 4/300 (0.013333)
False Negative Rate: 4/300 (0.013333)
Training Accuracy: 592/600 (0.986667)
```

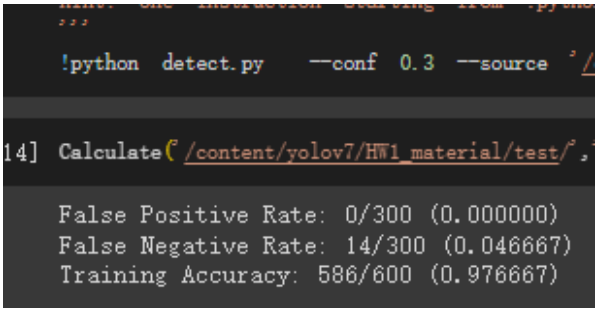
train and test accuracy with config = 0.1 (highest)



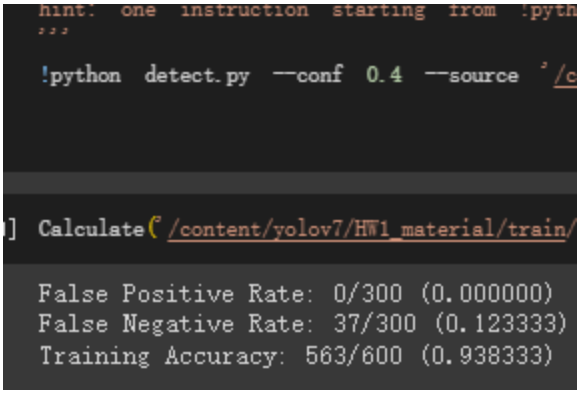


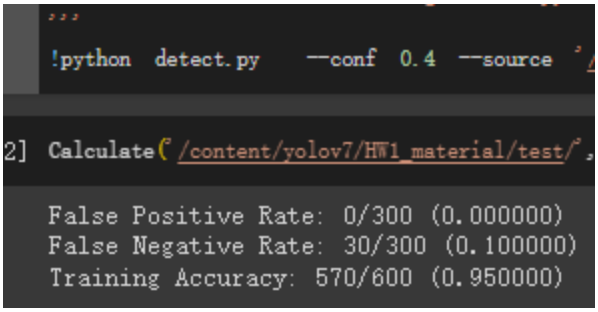
train and test accuracy with config = 0.2



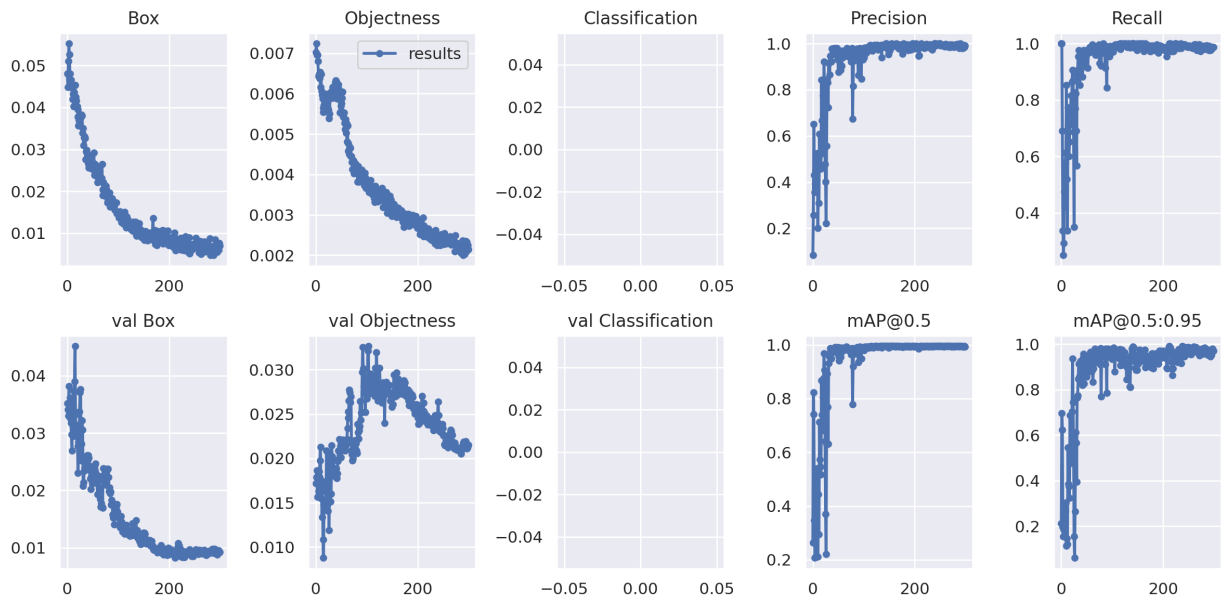


train and test accuracy with config = 0.3





train and test accuracy with config = 0.4



training curve

The problem I meet and how I solve them

1. Data type problem

At the start of the homework, I encountered numerous errors when inputting image arrays into the sklearn model. To resolve this, I learned that the images must be in grayscale and the numpy arrays need to be flattened into one-dimensional arrays for compatibility with the sklearn model.

2. Don’t know how to determine which parameter is better

In Part 3, I wasn't sure how to determine which parameter was better, so I looked up some online resources and found that the F1 score is a good choice. I then researched how to calculate it and used it as the standard for this part.

3. Testing time expense

In Part 3, instead of manually testing each model with 10 different parameters – which would require 30 separate starts and waste time – I modified the model and detect file to include the parameter and model name. I then ran all 30 iterations using a loop with parameter control, saving the results into 'modelname_parameter_pred.txt'. After running, I read each text file, calculated the F1 score, and outputted the results to a text file. Finally, I was able to compare each set of data effectively.

4. Uncertain how to plot the graph in part 5

In Part 5, I invested considerable time learning how to plot graphs, then I compared each '1' and '0' between 'groundtruth.txt' and 'model_pred.txt', calculating quantities and accuracies to create the charts.