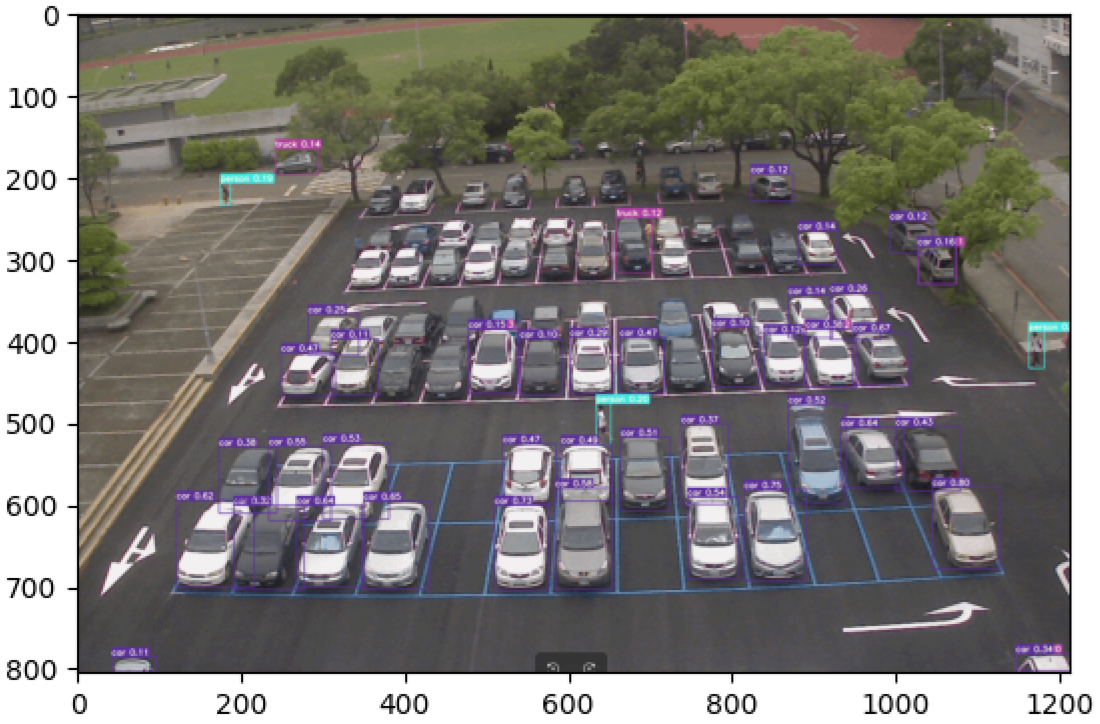
Questions

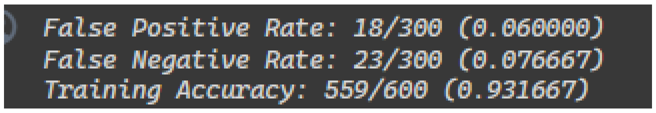
- Explain the difference between parametric and non-parametric models.
    - parametric models make hypothesis of relationship between input and output, but non-parametric models don't. Non-parametric models have flexible number of parameters, but parametric models don't.
- What is ensemble learning? Please explain the difference between bagging, boosting and stacking.
    - Ensemble learning is the method that tries to combine more than one model predictions into one to improve performance.
    - bagging is separating the sample into different subsets with bootstrapping usually for training weak learners.
    - boosting is training a sequence of model with weighted set, and the weight is assigned based on the training loss of the previous model in the sequence.
    - stacking has another element called meta-model, it takes the predictions of multiple models as input and output the final prediction.
- Explain the meaning of the "n_neighbors" parameter in KNeighborsClassifier, "n_estimators" in RandomForestClassifier and AdaBoostClassifier.
    - n_neighbors → number of nearest neighbors that puts into consideration when making predictions.
    - n_estimators
        - AdaBoostClassifier → number of weak learners
        - RandomForestClassifier → the number of trees to be used in the forest.
- Explain the meaning of four numbers in the confusion matrix.
    - True Negative → predicted Negative and actually Negative.
    - False Negative → predicted Negative but actually Positive.
    - False Positive → predicted Positive but actually Negative.

- True Positive → predicted Positive and actually Positive.

- 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 = (TP + TN) / (TP + TN + FP + FN)
- Precision = TP / (TP + FP)
    - When we want valid positive (cost of false positives is high), or we want our positive is really positive, such as Medical Diagnostics, we should use Precision rather than accuracy.
- Recall = TP / (TP + FN)
    - When positive cases are highly concerned (cost of false negatives is high), we want to catch all positive cases, we should use Recall rather than Accuracy, such as criminal detection.

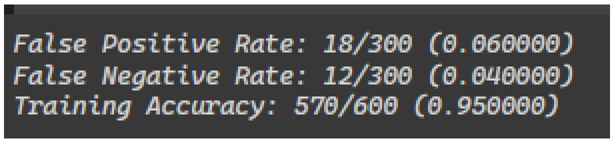


Task B part1



```
False Positive Rate: 18/300 (0.060000)
False Negative Rate: 23/300 (0.076667)
Training Accuracy: 559/600 (0.931667)
```

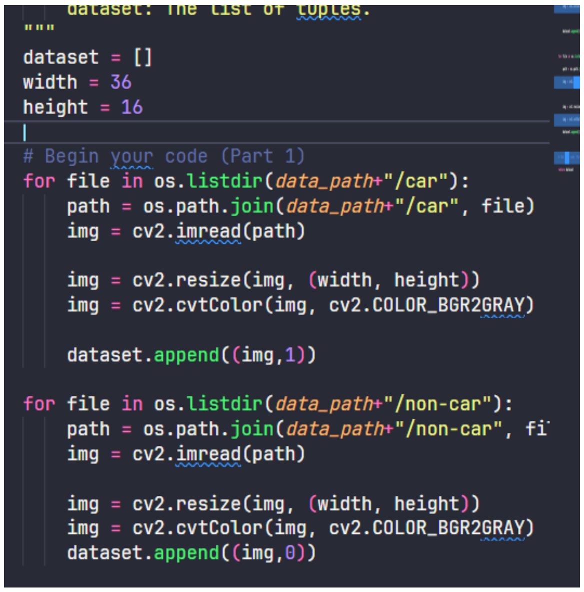
Train data Accuracy (TaskB part2)



```
False Positive Rate: 18/300 (0.060000)
False Negative Rate: 12/300 (0.040000)
Training Accuracy: 570/600 (0.950000)
```

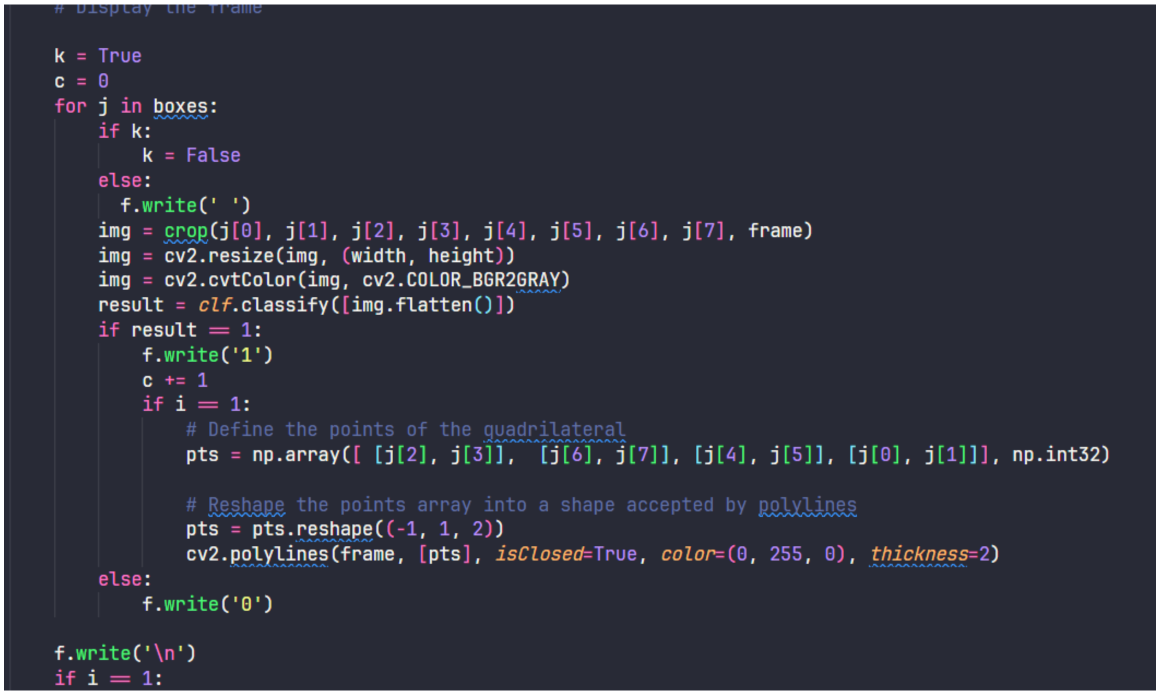
Test data Accuracy (TaskB part2)

1. Screenshot of your code if needed and a brief explanation.



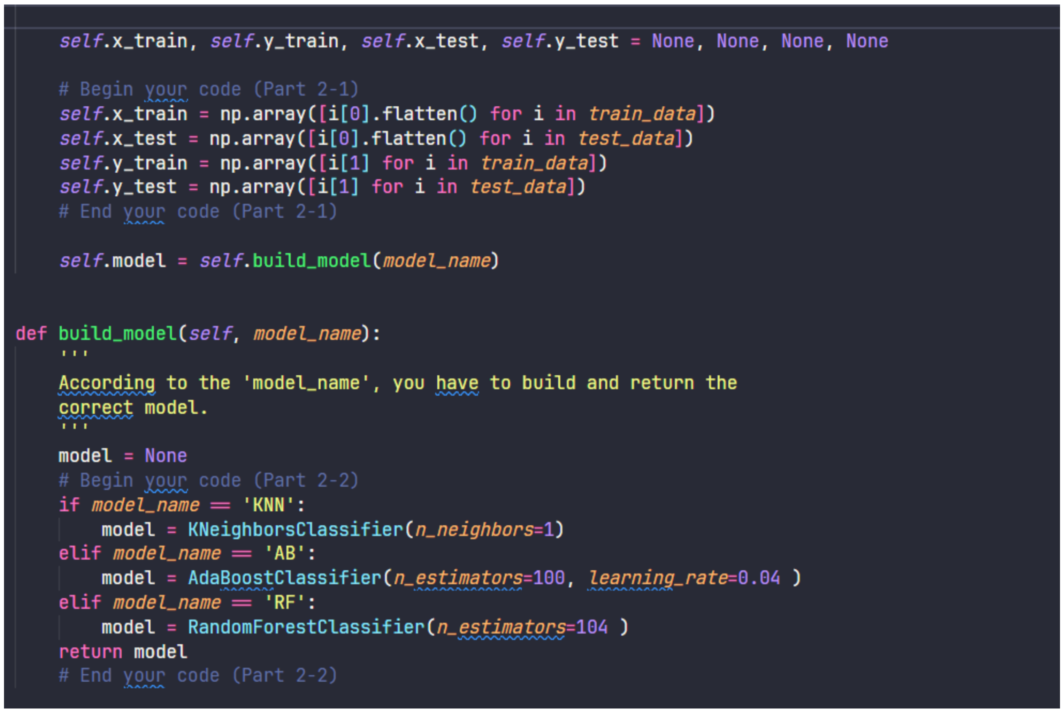
```python
    dataset: The list of tuples.
"""
dataset = []
width = 36
height = 16

# Begin your code (Part 1)
for file in os.listdir(data_path+"/car"):
    path = os.path.join(data_path+"/car", file)
    img = cv2.imread(path)

    img = cv2.resize(img, (width, height))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    dataset.append((img,1))

for file in os.listdir(data_path+"/non-car"):
    path = os.path.join(data_path+"/non-car", fi
    img = cv2.imread(path)

    img = cv2.resize(img, (width, height))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    dataset.append((img,0))
```

load 2 classes into np.array form, do 'car' and 'non-car' separately.(dataset.py)



```python
# Display the frame

k = True
c = 0
for j in boxes:
    if k:
        k = False
    else:
      f.write(' ')
    img = crop(j[0], j[1], j[2], j[3], j[4], j[5], j[6], j[7], frame)
    img = cv2.resize(img, (width, height))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    result = clf.classify([img.flatten()])
    if result == 1:
        f.write('1')
        c += 1
        if i == 1:
            # Define the points of the quadrilateral
            pts = np.array([ [j[2], j[3]],  [j[6], j[7]], [j[4], j[5]], [j[0], j[1]]], np.int32)

            # Reshape the points array into a shape accepted by polylines
            pts = pts.reshape((-1, 1, 2))
            cv2.polylines(frame, [pts], isClosed=True, color=(0, 255, 0), thickness=2)
    else:
        f.write('0')

f.write('\n')
if i == 1:
```

chop from the frame to img and used the trained classifier to detect if there is a car, I chop the image and detect it. (detection.py)



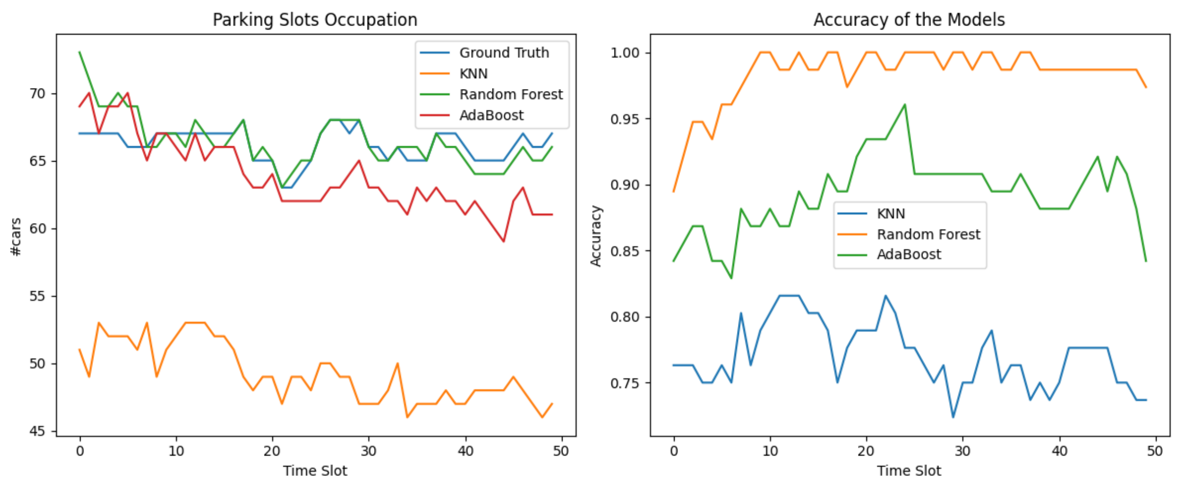
```python
    self.x_train, self.y_train, self.x_test, self.y_test = None, None, None, None

    # Begin your code (Part 2-1)
    self.x_train = np.array([i[0].flatten() for i in train_data])
    self.x_test = np.array([i[0].flatten() for i in test_data])
    self.y_train = np.array([i[1] for i in train_data])
    self.y_test = np.array([i[1] for i 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.
    '''
    model = None
    # Begin your code (Part 2-2)
    if model_name == 'KNN':
        model = KNeighborsClassifier(n_neighbors=1)
    elif model_name == 'AB':
        model = AdaBoostClassifier(n_estimators=100, learning_rate=0.04 )
    elif model_name == 'RF':
        model = RandomForestClassifier(n_estimators=104 )
    return model
    # End your code (Part 2-2)
```

init of the model for train and test data and choose what model to use. (model.py)

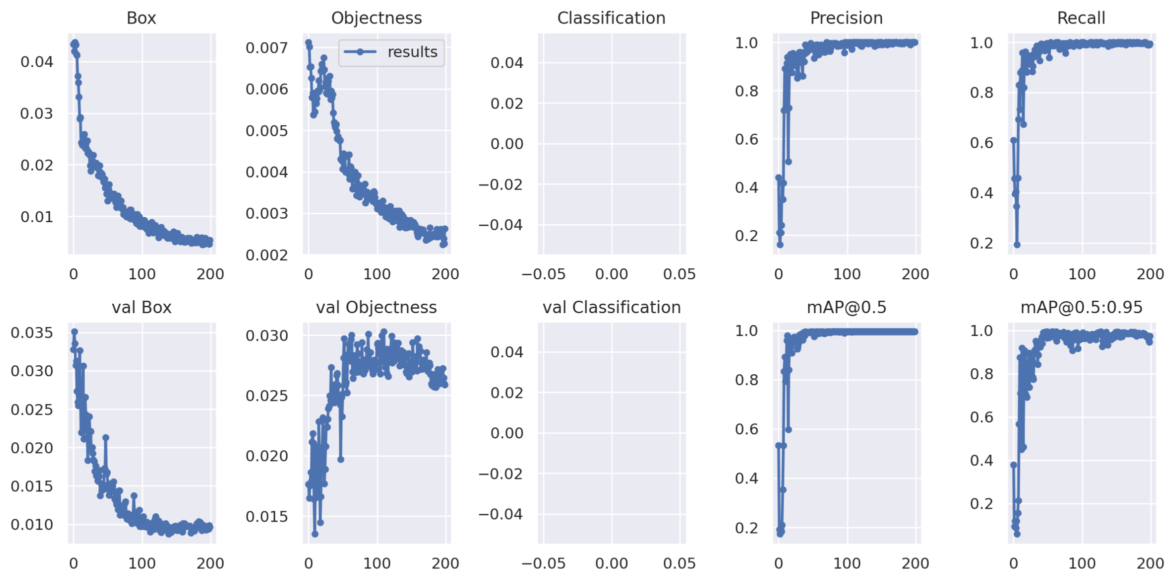
2. Your implementation of the above requirements in detail.

PartA code detail has already displayed above, no much detail except Task A part3 which is about tuning hyperparameters, I additionally create a for in range(a,b,step) and pass it into model class constructor to find the best hyperparameter for those three model separately, the step should be tuned smaller and smaller when finding the best parameter and I need to adjust (a,b,step) gradually by my instinct and based on my experience, which is quite interesting. Sorry I erased the code segment and I'm too busy to redo and screenshot it and, but this is the best detail I can show, Part B has no detail because I was just following those instructions, the problem I met can be the detail and I'll show it later.

3. The line graph and the training curve



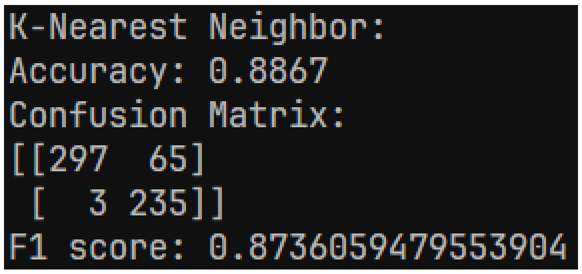
Task A line graph (Code can be seen in main.py)



TaskB training curve

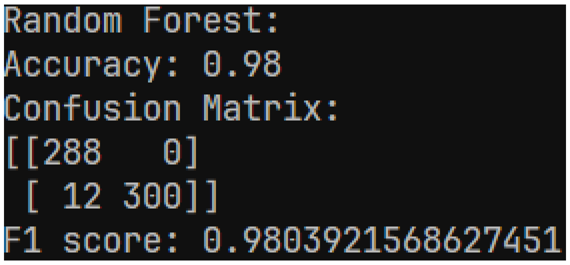
4. Discuss what you observed with accuracy, F1-score, and parking slots occupation plots of different methods in the report.

   KNN has the worst accuracy overall, and Random Forest got the best accuracy, Adaboost in the middle. The Random Forest occupation curve is almost fitting on ground truth, and the occupation detection of KNN is too conservative which the numbers are much lower than ground truth, numbers of occupation Adaboost detected is quite reasonable with respect of ground truth but not as accurate as Random Forest. F1 Score is the overall score of recall and precision which can also be explained by above statements about those models.



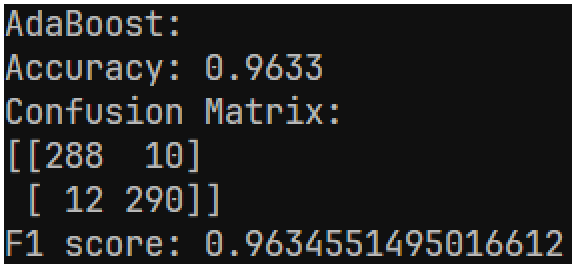
```
K-Nearest Neighbor:
Accuracy: 0.8867
Confusion Matrix:
[[297  65]
 [  3 235]]
F1 score: 0.8736059479553904
```

k nearest neighbor



```
Random Forest:
Accuracy: 0.98
Confusion Matrix:
[[288   0]
 [ 12 300]]
F1 score: 0.9803921568627451
```

Random Forest

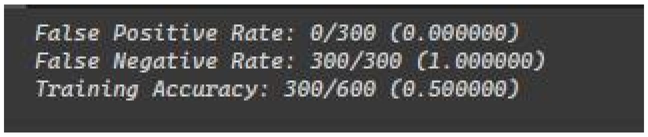


```
AdaBoost:
Accuracy: 0.9633
Confusion Matrix:
[[288  10]
 [ 12 290]]
F1 score: 0.9634551495016612
```

Adaboost

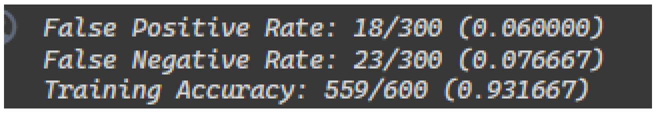
5. Describe problems you meet and how you solve them.

   the biggest problem I met is the one when I was evaluating the model I trained in Task B part2, I always got very bad accuracy no matter how I changed my batch size, I later realized where the problem came from by checking my result.png and asking others, it is caused by the low iteration, epoch wasn't enough for training, I thought it should not take too much time for training since this is not a complicated problem, so I tuned epoch no higher than 30 at first. After struggling, I tuned epoch to 90 with small batch size resulting 72% accuracy with the improvement of 22%. Finally, I tuned epoch to 200 and batch size 5 hopping to get 90% accuracy, and I get 93% and 95% accuracy in the end.



```
False Positive Rate: 0/300 (0.000000)
False Negative Rate: 300/300 (1.000000)
Training Accuracy: 300/600 (0.500000)
```

epoch under 50

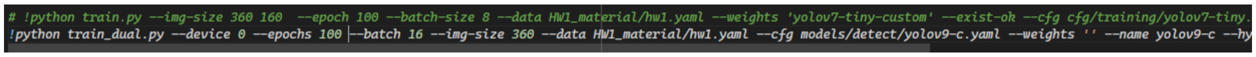


```
False Positive Rate: 18/300 (0.060000)
False Negative Rate: 23/300 (0.076667)
Training Accuracy: 559/600 (0.931667)
```

epoch = 200

TaskB part 3

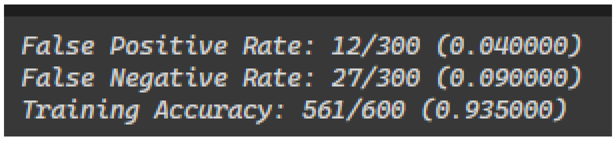
I try to implement the newest Yolo version, which is YoloV9 with google colab, which just updated few days ago. And the code is in TaskB_part3.ipynb



```
# !python train.py --img-size 360 160  --epoch 100 --batch-size 8 --data HW1_material/hw1.yaml --weights 'yolov7-tiny-custom' --exist-ok --cfg cfg/training/yolov7-tiny.
!python train_dual.py --device 0 --epochs 100 --batch 16 --img-size 360 --data HW1_material/hw1.yaml --cfg models/detect/yolov9-c.yaml --weights '' --name yolov9-c --hy
```

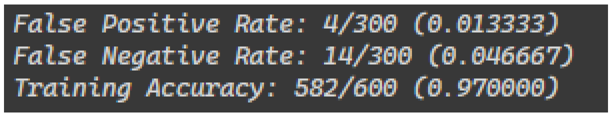
epoch 100 batch size 16

As we can seem from the result, The accuracy of Yolov9 surpass Yolov7 with 0.34% and 2% in training and testing set respectively, and the epoch of Yolov9 is even h



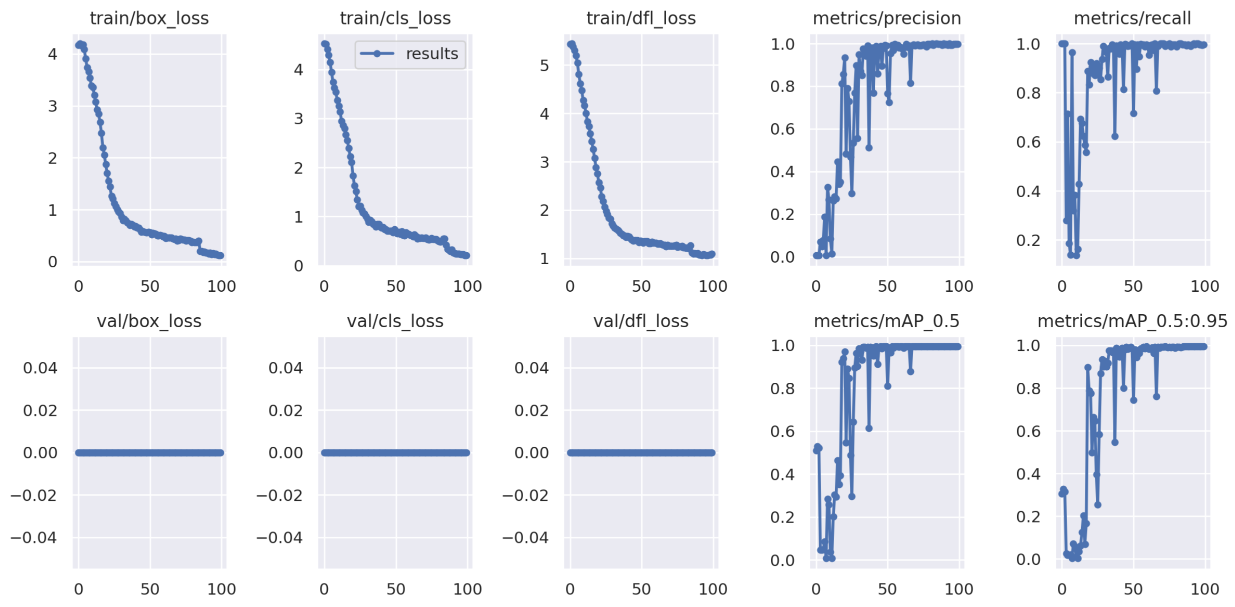
```
False Positive Rate: 12/300 (0.040000)
False Negative Rate: 27/300 (0.090000)
Training Accuracy: 561/600 (0.935000)
```

training set acc 93.5%



```
False Positive Rate: 4/300 (0.013333)
False Negative Rate: 14/300 (0.046667)
Training Accuracy: 582/600 (0.970000)
```

testing set acc 97%



It seems that the loss can still be improve with more than 100 epoch. And there is a big jump at epoch 80. And the precision is quite noisy and unstable until epoch 70.