1 Task A

1. Load and prepare your dataset

```python
import os
import cv2

def load_images(data_path):
    # Begin your code (Part 1)
    dataset = []
    ab_data_path = 'HW1/For-student/ML_Models/'

    for file in os.listdir(ab_data_path + data_path + '/car'):
        filepath = os.path.join(ab_data_path, data_path, 'car', file)
        img = cv2.imread(filepath)
        img = cv2.resize(img, (36, 16))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        dataset.append((img, 1))

    for file in os.listdir(ab_data_path + data_path + '/non-car'):
        filepath = os.path.join(ab_data_path, data_path, 'non-car', file)
        img = cv2.imread(filepath)
        img = cv2.resize(img, (36, 16))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        dataset.append((img, 0))
    # End your code (Part 1)
    return dataset
```

2. Build and Train Models

```python
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

class CarClassifier:
    def __init__(self, model_name, train_data, test_data):
        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.flatten() for i, _ in train_data])
        self.y_train = np.array([label for _, label in train_data])
        self.x_test = np.array([i.flatten() for i, _ in test_data])
        self.y_test = np.array([label for _, label in test_data])
        # End your code (Part 2-1)

        self.model = self.build_model(model_name)

    def build_model(self, model_name):
        # Begin your code (Part 2-2)
```

```
        if model_name == 'KNN':
            return KNeighborsClassifier()
        elif model_name == 'RF':
            return RandomForestClassifier()
        elif model_name == 'AB':
            return AdaBoostClassifier()
        else:
            raise ValueError(f"Unknown model name: {model_name}")
        # End your code (Part 2-2)

    def train(self):
        # Begin your code (Part 2-3)
        self.model.fit(self.x_train, self.y_train)
        # End your code (Part 2-3)

    def eval(self):
        y_pred = self.model.predict(self.x_test)
        print(f"Accuracy: {round(accuracy_score(y_pred, self.y_test), 4)}")
        print("Confusion Matrix: ")
        print(confusion_matrix(y_pred, self.y_test))

    def classify(self, input):
        return self.model.predict(input)[0]
```

Output:

```
Classifer: KNN      Classifer: Random Forest Classifer: AdaBoost
Accuracy: 0.845     Accuracy: 0.9767         Accuracy: 0.945
Confusion Matrix:   Confusion Matrix:        Confusion Matrix:
[[300  93]          [[288   2]               [[282  15]
 [  0 207]]          [ 12 298]]               [ 18 285]]
```

Performance: $Random\ Forest > AdaBoost > KNN$

(a) Explain the difference between parametric and non-parametric models.

i. parametric model: Parametric models are simpler and faster but less flexible, making strong assumptions about the data.

ii. non-parametric model: Non-parametric models are more flexible but computationally intensive and more prone to overfitting, especially with small datasets.

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

Ensemble learning is a machine learning technique that combines the predictions from multiple machine learning algorithms to make more accurate predictions than any individual model.

i. bagging: Reduce variance and avoid overfitting.

ii. boosting: Reduce both bias and variance.

iii. stacking: Explore the space of different models for the same task.

(c) Explain the meaning of the **n_neighbors** parameter in KNeighborsClassifier, **n_estimators** in RandomForestClassifier and AdaBoostClassifier.

i. n_neighbors parameter: The n_neighbors parameter specifies the number of nearest neighbors to consider when making a classification decision.

ii. n_estimators parameter: The n_estimators parameter defines the number of trees in the forest. Generally, a higher number of trees increases the performance and makes the predictions more stable, but also slows down the computation.

(d) Explain the meaning of four numbers in the confusion matrix.

i. True Positives (TP): Cases in which when predicted 1, actual output was 1.
ii. True Negatives (TN): Cases in which when predicted 0, actual output was 0.
iii. False Positives (FP): Cases in which when predicted 1, actual output was 0.
iv. False Negatives (FN): Cases in which when predicted 0, actual output was 1.

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

(e) 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**.

i. Precision: $Precision = \frac{TP}{TP+FP}$
ii. Recall: $Recall = \frac{TP}{TP+FN}$

When the classes are highly imbalanced, accuracy can be misleading. In such cases, precision and recall provide a better understanding of the model's performance.

3. Additional experiments

```
def __init__(self, model_name, train_data, test_data, n_neighbors=None,
    n_estimators=None):
    self.model_name = model_name
    self.train_data = train_data
    self.test_data = test_data
    self.n_neighbors = n_neighbors
    self.n_estimators = n_estimators

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

    self.x_train = np.array([i.flatten() for i, _ in train_data])
    self.y_train = np.array([label for _, label in train_data])
    self.x_test = np.array([i.flatten() for i, _ in test_data])
    self.y_test = np.array([label for _, label in test_data])

    if model_name == "KNN":
        self.model = KNeighborsClassifier(n_neighbors=n_neighbors if
            n_neighbors else 5)
    elif model_name == "RF":
        self.model = RandomForestClassifier(n_estimators=n_estimators if
            n_estimators else 100)
    elif model_name == "AB":
        self.model = AdaBoostClassifier(n_estimators=n_estimators if
            n_estimators else 50)
    else:
        raise ValueError("Invalid model name")
```

Output:

```
Classifer: KNN-3   Classifer: Random Forest-10   Classifer: AdaBoost-10
Accuracy: 0.855    Accuracy: 0.9567              Accuracy: 0.9233
Confusion Matrix:  Confusion Matrix:             Confusion Matrix:
[[299  86]         [[284  10]                    [[281  27]
 [  1 214]]         [ 16 290]]                    [ 19 273]]

Classifer: KNN-5   Classifer: Random Forest-50   Classifer: AdaBoost-50
Accuracy: 0.845    Accuracy: 0.975               Accuracy: 0.945
Confusion Matrix:  Confusion Matrix:             Confusion Matrix:
[[300  93]         [[287   2]                    [[282  15]
 [  0 207]]         [ 13 298]]                    [ 18 285]]

Classifer: KNN-7   Classifer: Random Forest-100  Classifer: AdaBoost-100
Accuracy: 0.8267   Accuracy: 0.9717              Accuracy: 0.9517
Confusion Matrix:  Confusion Matrix:             Confusion Matrix:
[[300 104]         [[285   2]                    [[283  12]
 [  0 196]]         [ 15 298]]                    [ 17 288]]

Classifer: KNN-10  Classifer: Random Forest-200  Classifer: AdaBoost-200
Accuracy: 0.7983   Accuracy: 0.9733              Accuracy: 0.955
Confusion Matrix:  Confusion Matrix:             Confusion Matrix:
[[300 121]         [[286   2]                    [[284  11]
 [  0 179]]         [ 14 298]]                    [ 16 289]]
```

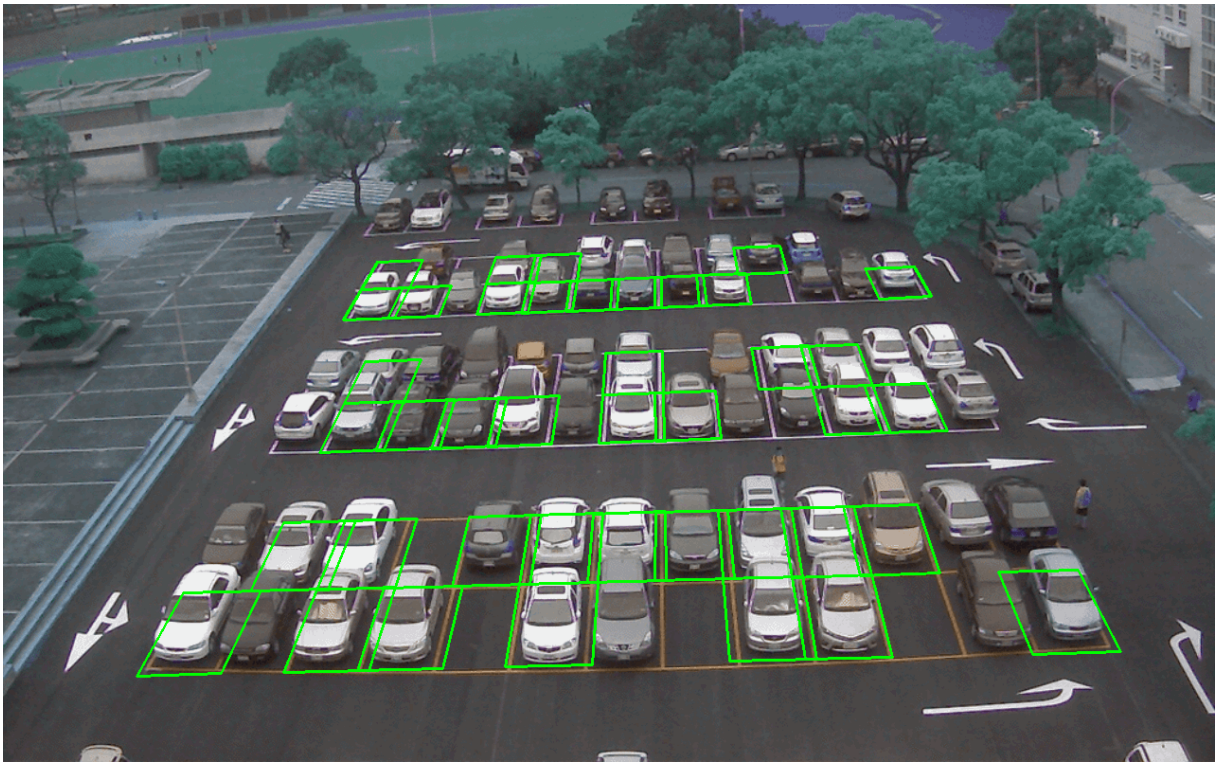
Best classifier: *Random Forest* with $n\_estimators = 10$

4. Detect car

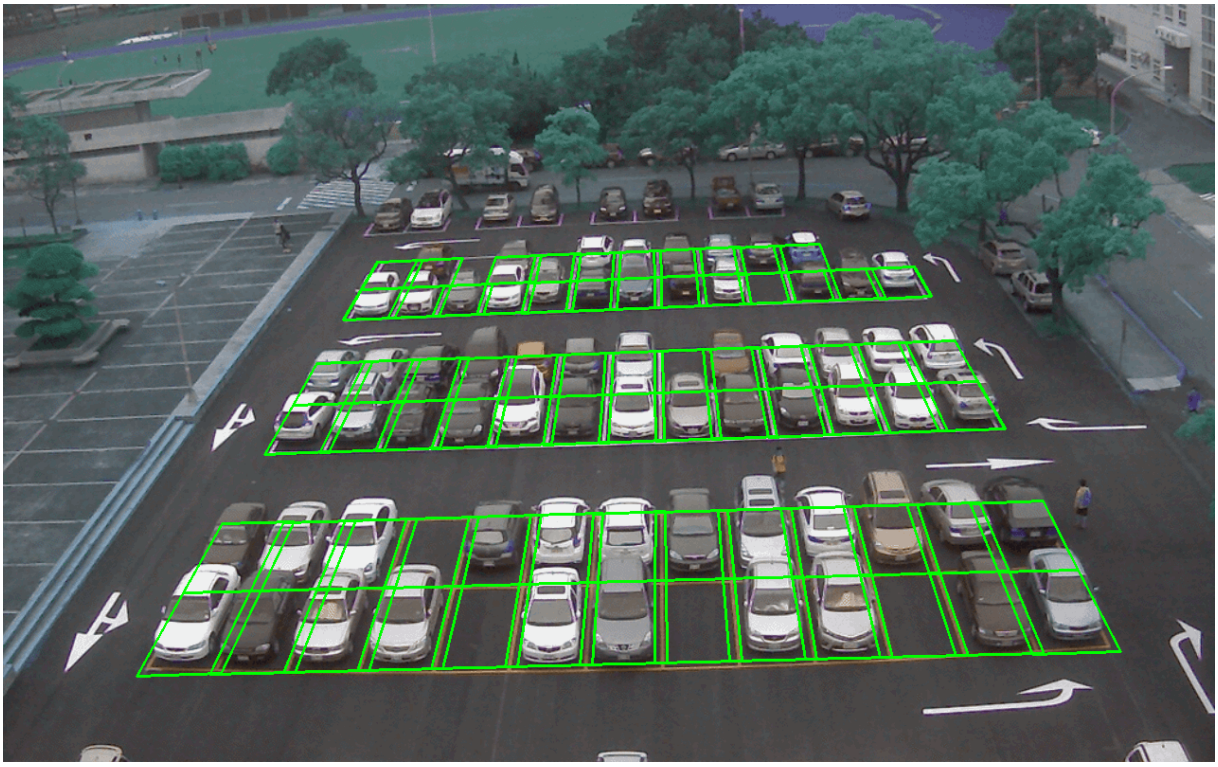
```python
import cv2
import matplotlib.pyplot as plt
import numpy as np

def crop(x1, y1, x2, y2, x3, y3, x4, y4, img) :
    left_front = (x1, y1)
    right_front = (x2, y2)
    left_bottom = (x3, y3)
    right_bottom = (x4, y4)
    src_pts = np.array([left_front, right_front, left_bottom, right_bottom]).astype(np.float32)
    dst_pts = np.array([[0, 0], [0, 160], [360, 0], [360, 160]]).astype(np.float32)
    projective_matrix = cv2.getPerspectiveTransform(src_pts, dst_pts)
    croped = cv2.warpPerspective(img, projective_matrix, (360,160))
    return croped


def detect(data_path, clf):
    # Begin your code (Part 4)
    with open(data_path, 'r') as file:
        parking_spaces = [list(map(int, line.strip().split())) for line in file.readlines()]
```

```
21
22      cap = cv2.VideoCapture('HW1/For-student/ML_Models/data/detect/video.gif')
23
24      predictions = []
25
26      ret, frame = cap.read()
27      first_frame = None
28      frame_count = 0
29      parking_spaces = parking_spaces[1:]
30      while ret:
31          frame_count += 1
32          frame_predictions = []
33          for space in parking_spaces:
34              x1, y1, x2, y2, x3, y3, x4, y4 = space
35              cropped_image = crop(x1, y1, x2, y2, x3, y3, x4, y4, frame)
36
37              processed_image = cv2.resize(cropped_image, (36, 16))
38              processed_image = cv2.cvtColor(processed_image, cv2.
                    COLOR_BGR2GRAY)
39              cls_img = np.array([processed_image.flatten()])
40
41              is_car = clf.classify(cls_img)
42              frame_predictions.append(is_car)
43
44              if is_car:
45                  cv2.polylines(frame, [np.array([(x1, y1), (x2, y2), (x4, y4),
                        (x3, y3)])], True, (0, 255, 0), 2)
46
47          predictions.append(frame_predictions)
48
49          if first_frame is None:
50              first_frame = frame
51
52          ret, frame = cap.read()
53
54      cv2.imwrite('HW1/For-student/pred.png', cv2.cvtColor(first_frame, cv2.
            COLOR_BGR2RGB))
55      with open('HW1/For-student/pred.txt', 'w') as file:
56          for i, frame_pred in enumerate(predictions):
57              file.write(f'{" ".join(map(str, frame_pred))}\n')
58
59      cap.release()
60      # End your code (Part 4)
```

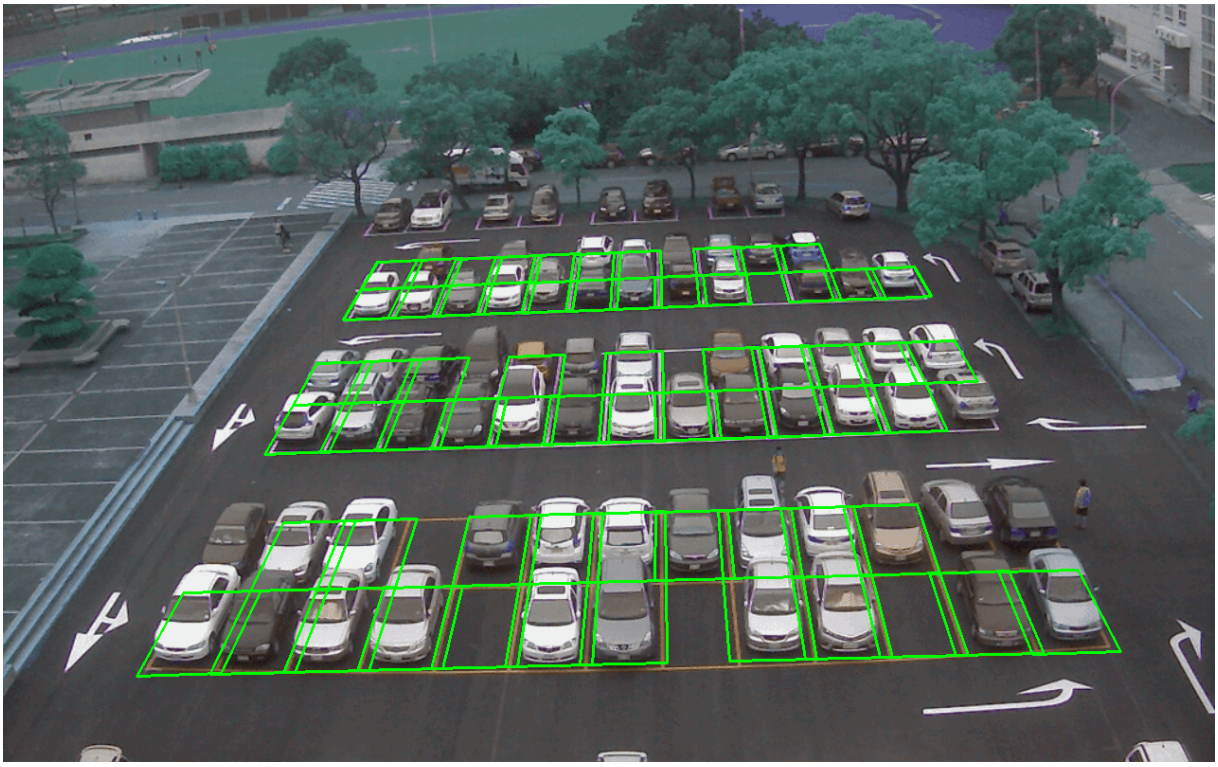
(a) KNN



(b) Random Forest



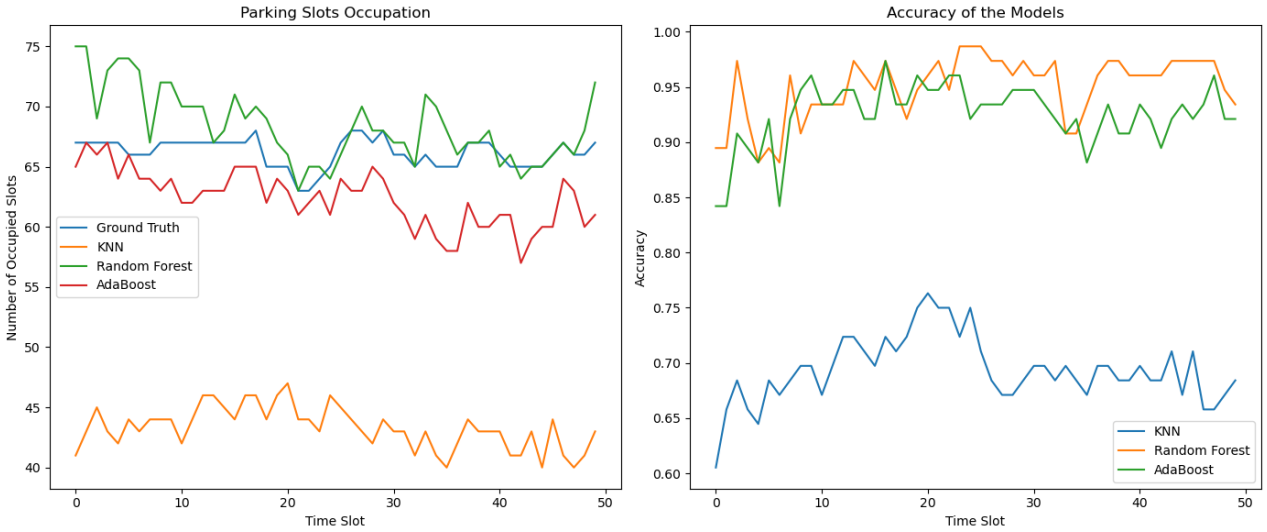
(c) AdaBoost



5. Draw a line graph

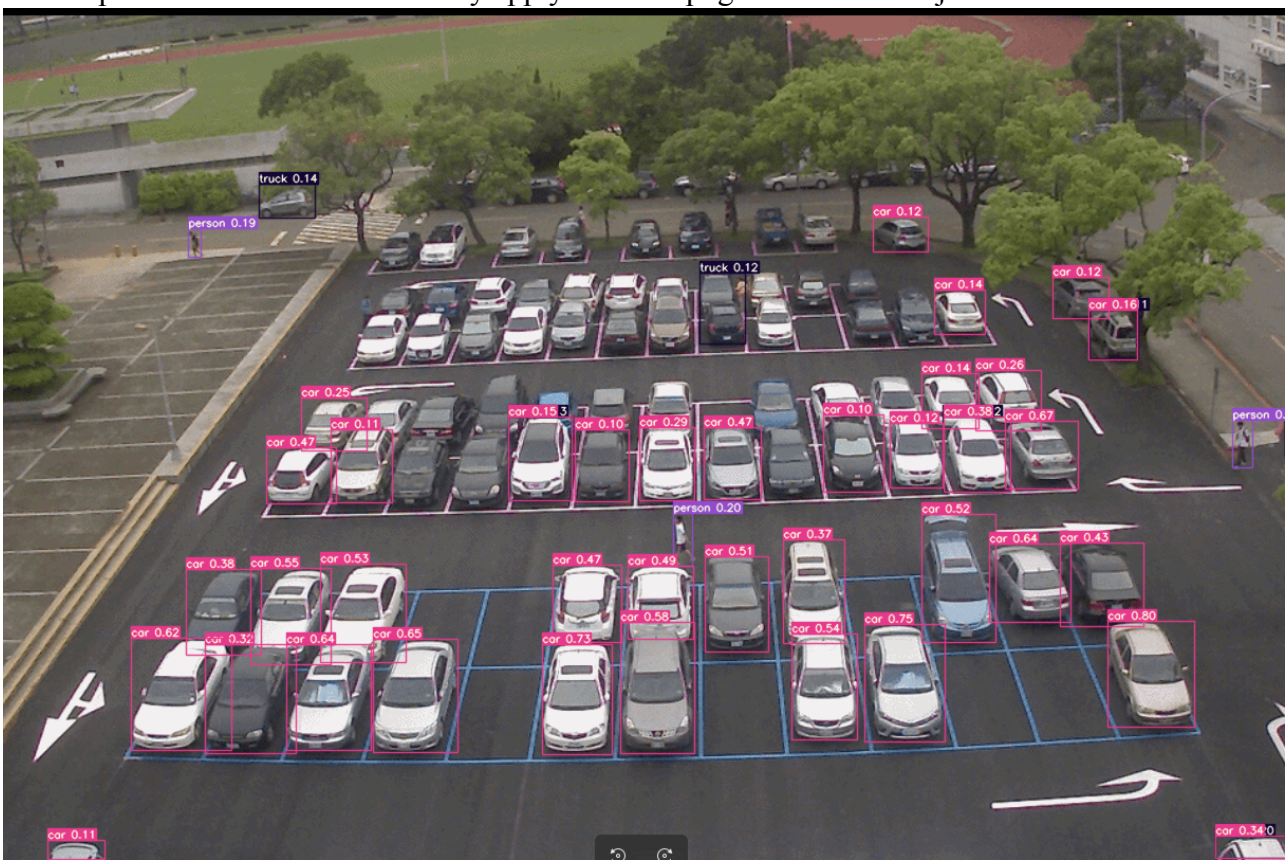
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

tableau_palette = sns.color_palette("tab10")

ground_truth = pd.read_csv('HW1/For-Student/GroundTruth.txt', sep=' ', header
    =None)
knn_pred = pd.read_csv('HW1/For-Student/KNNpred.txt', sep=' ', header=None)
rf_pred = pd.read_csv('HW1/For-Student/RFpred.txt', sep=' ', header=None)
ab_pred = pd.read_csv('HW1/For-Student/ABpred.txt', sep=' ', header=None)

ground_truth_sum = ground_truth.sum(axis=1)
knn_sum = knn_pred.sum(axis=1)
rf_sum = rf_pred.sum(axis=1)
ab_sum = ab_pred.sum(axis=1)

knn_accuracy = (knn_pred == ground_truth).mean(axis=1)
rf_accuracy = (rf_pred == ground_truth).mean(axis=1)
ab_accuracy = (ab_pred == ground_truth).mean(axis=1)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.plot(ground_truth_sum, label='Ground Truth', color=tableau_palette[0])
plt.plot(knn_sum, label='KNN', color=tableau_palette[1])
plt.plot(rf_sum, label='Random Forest', color=tableau_palette[2])
plt.plot(ab_sum, label='AdaBoost', color=tableau_palette[3])
```

```
plt.title('Parking Slots Occupation')
plt.xlabel('Time Slot')
plt.ylabel('Number of Occupied Slots')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(knn_accuracy, label='KNN', color=tableau_palette[0])
plt.plot(rf_accuracy, label='Random Forest', color=tableau_palette[1])
plt.plot(ab_accuracy, label='AdaBoost', color=tableau_palette[2])
plt.title('Accuracy of the Models')
plt.xlabel('Time Slot')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



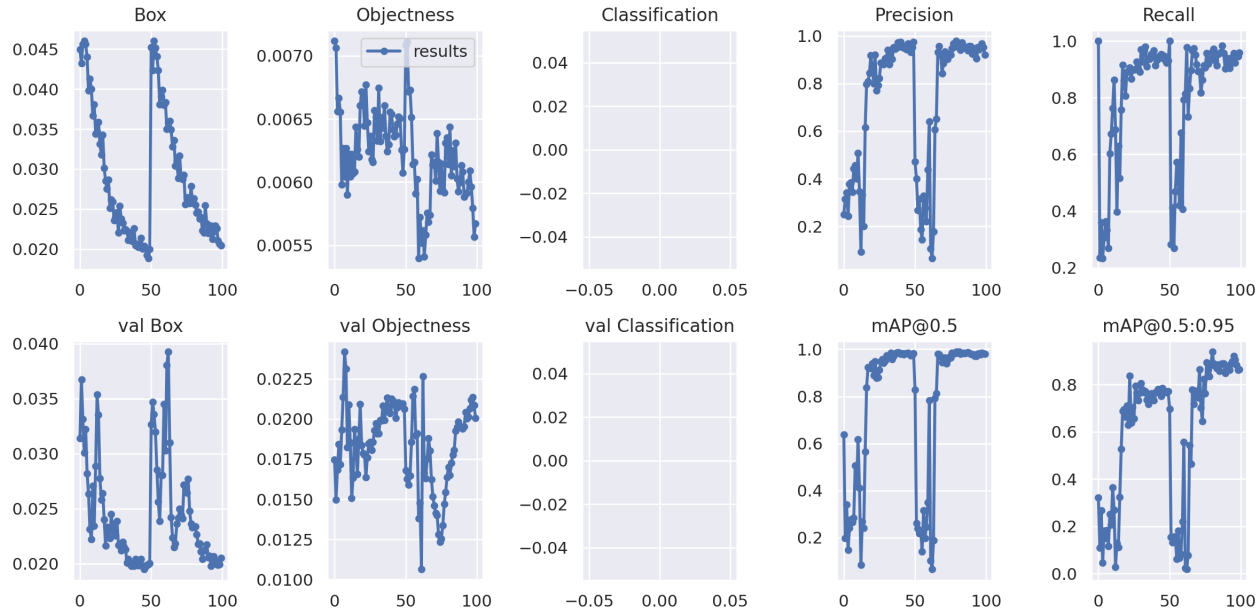
2 Task B

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



2. Learn to fine-tune yolov7 model

```
# Finetune the Yolov7 model
!python train.py --img-size 360 160 --data HW1_material/hw1.yaml
--weights 'yolov7-tiny-custom' --exist-ok --cfg cfg/training/yolov7-tiny.yaml
    --name yolov7-tiny --hyp data/hyp.scratch.custom.yaml --device 0 --epoch
    50 --batch-size 8

# Apply on training data
!python detect.py --conf 0.1 --source /content/yolov7/HW1_material/train/car
   --weights $savePath --name train --save-txt --exist-ok
!python detect.py --conf 0.1 --source /content/yolov7/HW1_material/train/non-
   car --weights $savePath --name train --save-txt --exist-ok

# Apply on testing data
!python detect.py --conf 0.1 --source /content/yolov7/HW1_material/test/car
   --name test --save-txt --exist-ok --weights $savePath
!python detect.py --conf 0.1 --source /content/yolov7/HW1_material/test/non-
   car --name test --save-txt --exist-ok --weights $savePath
```

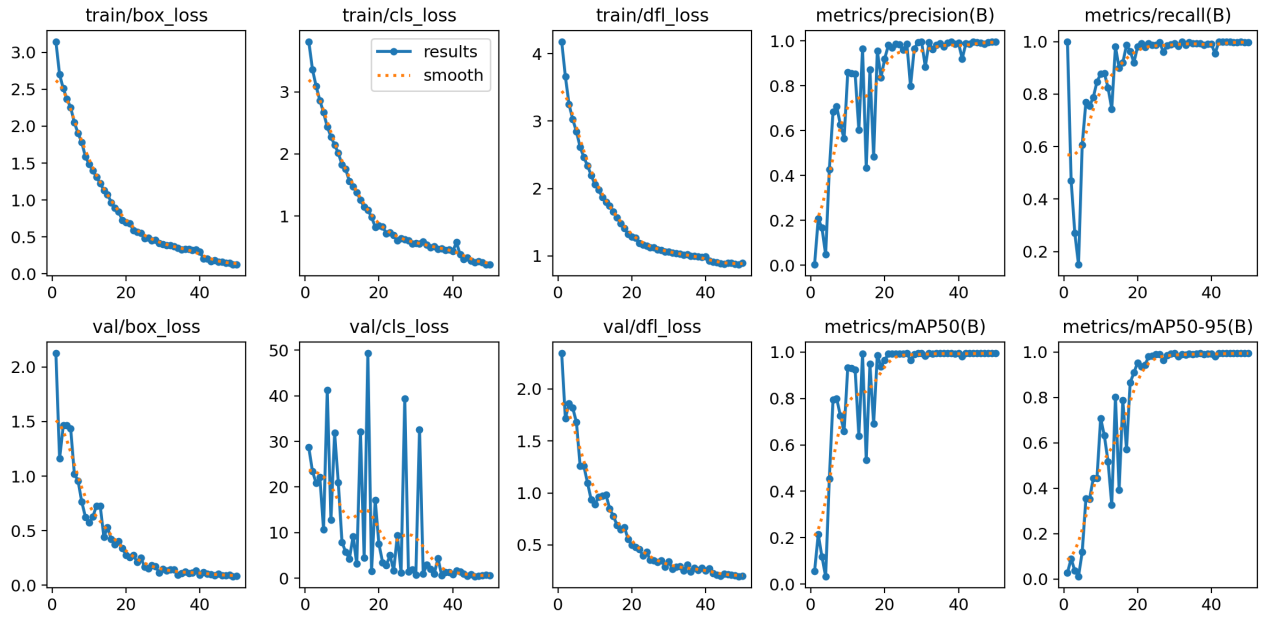


Output:

```
# Training Performance
False Positive Rate: 33/300 (0.110000)
False Negative Rate: 10/300 (0.033333)
Training Accuracy: 557/600 (0.928333)

# Testing Performance
False Positive Rate: 30/300 (0.100000)
False Negative Rate: 1/300 (0.003333)
Training Accuracy: 569/600 (0.948333)
```

3. Try other SOTA methods (Bonus)

```
# Install the Yolov8 model
!pip install ultralytics

# Finetune the Yolov8 model
!yolo task=detect mode=train model=yolov8n.yaml data=HW1_material/hw1.yaml
    epochs=50 batch=8

# Apply on training data
!yolo task=detect mode=predict model=$savePath conf=0.1 source='/content/
    HW1_material/train/car' save_txt=True name=train exist_ok=True
!yolo task=detect mode=predict model=$savePath conf=0.1 source='/content/
    HW1_material/train/non-car' save_txt=True name=train exist_ok=True

# Apply on testing data
!yolo task=detect mode=predict model=$savePath conf=0.1 source='/content/
    HW1_material/test/car' save_txt=True name=test exist_ok=True
!yolo task=detect mode=predict model=$savePath conf=0.1 source='/content/
    HW1_material/test/non-car' save_txt=True name=test exist_ok=True
```



Output:

```
# Training Performance
False Positive Rate: 11/300 (0.036667)
False Negative Rate: 0/300 (0.000000)
Training Accuracy: 589/600 (0.981667)

# Testing Performance
False Positive Rate: 10/300 (0.033333)
False Negative Rate: 0/300 (0.000000)
Training Accuracy: 590/600 (0.983333)
```

Comparison:

	Training Performance	**Testing Performance**
AdaBoost	×	0.945
Yolov7	0.928333	0.948333
Yolov8	0.981667	0.983333

Performance: $Yolov8 > Yolov7 > AdaBoost$

3 Appendix

1. Problems:
   (a) File not found error

```
Traceback (most recent call last):
  File "/Users/chou.ting.wei/Coding/NYCU/AI/HW1/For-student/ML_Models/main.py", line 8, in <module>
    train_data = dataset.load_images('data/train')
  File "/Users/chou.ting.wei/Coding/NYCU/AI/HW1/For-student/ML_Models/dataset.py", line 19, in load_images
    for file in os.listdir(ab_data_path + data_path + '/car'):
FileNotFoundError: [Errno 2] No such file or directory: 'data/train/car'
```

   Solution: Change the file path into an absolute file path.

```
ab_data_path = 'HW1/For-student/ML_Models/'

for file in os.listdir(ab_data_path + data_path + '/car'):
    filepath = os.path.join(ab_data_path, data_path, 'car', file)
    ...
```

   (b) You cannot currently connect to a GPU due to usage limits in Colab.
   Solution: Add parameters to modify the epoch and batch size.

```
# Add parameters
--device 0 --epoch 50 --batch-size 8
```

   (c) Accuracy is lower than 90%.
   Solution: Add parameter to modify the confidence threshold.

```
# Add parameter
--conf 0.1
```

2. Reference
   (a) OpenAI. (2024). ChatGPT (4) [Large language model]. https://chat.openai.com
   (b) Official YOLOv7. https://github.com/WongKinYiu/yolov7
   (c) Ultralytics YOLOv8. https://github.com/ultralytics/ultralytics