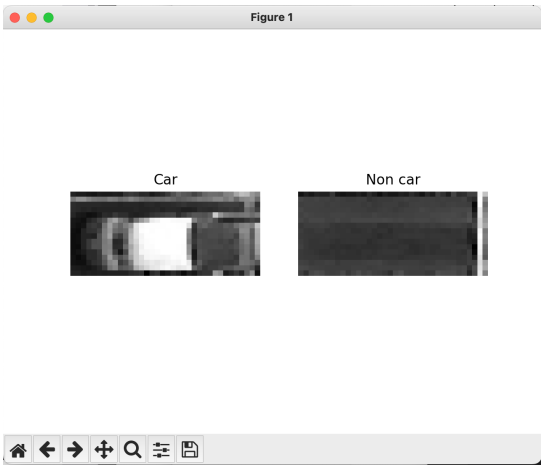
Task A

Part 1



確認能夠 load image

Part 2

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

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p.$$

Parametric models 為線性模型，他會決定像上述的參數來建造他的模型。
Non-parametric models 會找出各種近似值，盡可能地靠近實際的 data
Parametric models 比起 non-parametric 更容易會 fit 因為只要考量線性關係會產生的參數，但對於未知的資料很難去預測；而 non-parametric 可以比較好預測實際的情況，有更好的預測精準度，但是模型是很難去解釋理解的。

- What is ensemble learning? Please explain the difference between bagging, boosting and stacking.
  Ensemble learning 是指將監督式學習的模型整合在一起，來產生更強大、準確性更好的模型。
  Bagging 是我們把 training data 採樣產生不同組的訓練資料，根據不同組的訓練資料會使得用同一種演算法也會得到不一樣的模型（各自獨立而能平行化處理）。代表：Random Forest
  Boosting 則會根據每一筆訓練資料的困難度給予不同的權重。一開始會訓練 base learner 的結果來判斷資料是困難還是簡單，對於難的資料會加強權重再訓練一個新的模型，在新的訓練中希望模型能夠在困難的地方表現變好，把上一次做得不好的地方改善。代表：AdaBoost

Stacking 會先產出多個互不相關的模型，各自訓練完後合併在一起。根據各個模型最後的輸出為特徵，再訓練另一個模型來學習去預測結果（like 合併）。

– Bagging vs. Boosting vs. Stacking

Bagging 在同一種演算法多次訓練中所產生的資料跟結果是獨立且隨機的。

Boosting 在同一種演算法下會在更多的訓練中加強他的表現。其中的每一棵樹都是有關連的，前一棵樹會影響下一棵樹的生成。

Stacking 會結合不同的模型（might be 不同演算法），利用各個模型預測結果當特徵來訓練一個模型做最終預測。

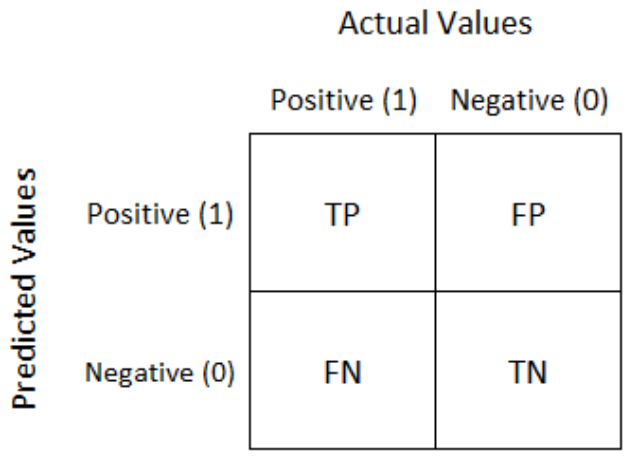
- Explain the meaning of the “n_neighbors” parameter in KNeighborsClassifier, “n_estimators” in RandomForestClassifier and AdaBoostClassifier.

  n_neighbors= k 代表了將會跟最靠近其新數據的 k 個樣本點為同一分類，其鄰居數量多寡會影響他分類的預測

  n_estimators = n in RandomForest 代表了在做 bagging 時會有 n 棵樹

  n_estimators = n in AdaBoost 代表有 n 個弱學習器

- Explain the meaning of four numbers in the confusion matrix.



True Positive (TP) 當預測結果是 Positive，實際結果也是 Positive

False Positive (FP) 當預測結果是 Positive，但實際結果是 Negative

False Negative (FN) 當預測結果是 Negative，但實際結果是 Positive

True Negative (TN) 當預測結果是 Negative，實際結果也是 Negative

- 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”.
Precision = tp/(tp+fp) ->預測為 Positive 的樣本中預測正確的
Recall = tp/(tp+fn) -> 實際事件為 positive 中預測正確的
當訓練資料的目標變數不平衡時，用 Accuracy 會有問題，因此要用 Precision 和 Recall 來計算。Ex: 1000 人中使用 A 牌快篩試劑，5 人實際確診，A 快篩檢查出 4 人確診，這時 Accuracy 的為 99.9%；而 B 牌快篩只查出 1 人確診，此時 Accuracy 依然是很高的 99.6%（不合理）。這種不平衡的情況用 Pecision 來算：A 牌為 80%；B 牌為 20% 便能發現 B 快篩是比較有問題的。

Part 3

Knn

n_neighbors default(==5)

```
Accuracy: 0.845
Confusion Matrix:
[[300   0]
 [ 93 207]]
```

n_neighbors == 3 (best)

```
Accuracy: 0.8583
Confusion Matrix:
[[299   1]
 [ 84 216]]
```

n_neighbors == 4

```
Accuracy: 0.83
Confusion Matrix:
[[300   0]
 [102 198]]
```

n_neighbors = 10

```
Accuracy: 0.7967
Confusion Matrix:
[[300   0]
 [122 178]]
```

n_neighbors =15(越多越低 因此不再往上測試)

```
Accuracy: 0.7733
Confusion Matrix:
[[300   0]
 [136 164]]
```

Random Forest (random state=42)

n_estimators = 10

```
Accuracy: 0.9633
Confusion Matrix:
[[293   7]
 [ 15 285]]
```

n_estimators = 30

```
Accuracy: 0.9733
Confusion Matrix:
[[287  13]
 [  3 297]]
```

n_estimators = 100 (再上去沒有什麼差別 暫定為 best)

```
Accuracy: 0.9767
Confusion Matrix:
[[287  13]
 [  1 299]]
```

AdaBoost (random state 42)

n_estimators = 10

```
Accuracy: 0.9383
Confusion Matrix:
[[282  18]
 [ 19 281]]
```

n_estimators = 30

```
Accuracy: 0.945
Confusion Matrix:
[[281  19]
 [ 14 286]]
```

n_estimators = 100

```
Accuracy: 0.945
Confusion Matrix:
[[283  17]
 [ 16 284]]
```

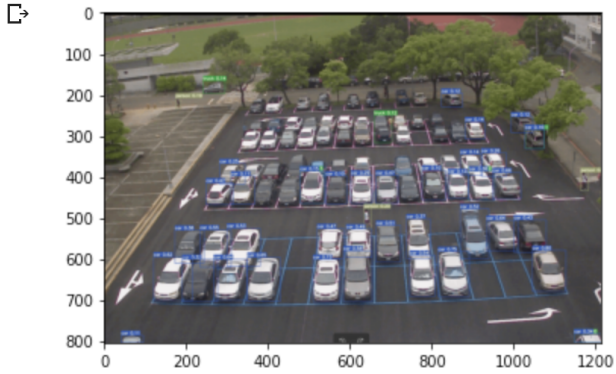
n_estimators = 150 (再上去沒有差太多 就當它是 best)

```
Accuracy: 0.9517
Confusion Matrix:
[[285  15]
 [ 14 286]]
```

Task B

Part 1

```
# Show image in /content/yolov7/runs/detect/parking_area/parking_area.png
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('/content/yolov7/runs/detect/parking_area/parking_area.png')
imgplot = plt.imshow(img)
plt.show()
```



Show result

Part 2

Epoch=25 Batch=10

```
      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      18/24    0.625G   0.03365  0.006399         0   0.04005        12       384: 100% 60/60 [00:12<00:00,  4.76it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300      0.647       0.69       0.603       0.431

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      19/24    0.625G   0.03045  0.006379         0   0.03683        12       384: 100% 60/60 [00:12<00:00,  4.69it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300      0.275      0.773       0.268       0.164

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      20/24    0.625G   0.03219  0.006334         0   0.03852        17       384: 100% 60/60 [00:12<00:00,  4.76it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300      0.522      0.709       0.502       0.274

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      21/24    0.625G   0.02728  0.006216         0   0.03349         9       384: 100% 60/60 [00:12<00:00,  4.80it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300        0.8      0.786       0.849       0.593

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      22/24    0.625G   0.02848  0.006638         0   0.03511        13       384: 100% 60/60 [00:12<00:00,  4.77it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300      0.799        0.8        0.85       0.566

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      23/24    0.625G   0.02875  0.006468         0   0.03522        14       384: 100% 60/60 [00:12<00:00,  4.84it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:02<00:00,
                   all         600         300      0.873       0.82       0.897       0.621

      Epoch   gpu_mem       box       obj       cls     total    labels  img_size
      24/24    0.625G    0.0249  0.006196         0    0.0311         6       384: 100% 60/60 [00:12<00:00,  4.91it/s]
                 Class      Images      Labels          P          R      mAP@.5  mAP@.5:.95: 100% 30/30 [00:03<00:00,
                   all         600         300      0.851       0.88       0.911       0.702
25 epochs completed in 0.113 hours.
```

最終 mAP 有達到 0.911 Precision 有 0.851 Recall 有 0.88 因此用此次來測試

用 best.pt 有 FN 問題 可能參數沒有清乾淨->用 last.pt

```
[16] # Calculate yolov7 performance
     # You have to adjust the parameters to get more than 90% accuracy
     # Warning: make sure that txtpath is currect because you may get many train{n} file when training more than one time.

     Calculate('/content/yolov7/HW1_material/train/','/content/yolov7/runs/detect/train/labels/')
```

```
False Positive Rate: 17/300 (0.056667)
False Negative Rate: 38/300 (0.126667)
Training Accuracy: 545/600 (0.908333)
```

Train 的 Accuracy 為 0.908 (>90)

Test 的 Accuracy 為 0.928 (>90)

Problem I met:

測試時用 best.pt 會有問題(FN 一直=1)，可能是參數沒清乾淨，因此選用 last.pt

Colab GPU 不夠用，需要要換帳號來 finetune 參數