111550113 謝詠晴

Intro_to_AI HW1

Task A

Part 1: Load and prepare your dataset (10%)

Load images and resize them to 36 x 16 then convert to grayscale images. Label them with 1 and 0 and put them into 'dataset' list.

```python
    # Begin your code (Part 1)
    dataset = []
    labels = {"car": 1, "non-car": 0}
    for folder, label in labels.items():
        folder_path = os.path.join(data_path, folder)
        for file in os.listdir(folder_path):
            img = cv2.imread(os.path.join(folder_path, file))
            resized = cv2.resize(img,(36,16)) # resize to 36 x 16
            gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY)
            dataset.append((gray, label))
    #raise NotImplementedError("To be implemented")
    # End your code (Part 1)
    return dataset
```

Part 2: Build and Train Models (20%)

Using 'numpy.reshape()' to transpose images of training data and testing data from 2D arrays to 1D arrays. And assign them to self.

```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([data[0].reshape(-1) for data in train_data])
        self.y_train = np.array([data[1] for data in train_data])
        self.x_test = np.array([data[0].reshape(-1) for data in test_data])
        self.y_test = np.array([data[1] for data in test_data])
        #raise NotImplementedError("To be implemented")
        # End your code (Part 2-1)

        self.model = self.build_model(model_name)
```

Building KNN, RF, and AdaBoost models by KNeighborsClassifier(), RandomForestClassifier() and AdaBoostClassifier().

Then start training model in train() function.

```python
    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":
            model = KNeighborsClassifier()
        elif model_name == "RF":
            model = RandomForestClassifier()
        else: # "AB"
            model = AdaBoostClassifier()

        return model
        #raise NotImplementedError("To be implemented")
        # End your code (Part 2-2)
```

```python
    def train(self):
        '''
        Fit the model on training data (self.x_train and self.y_train).
        '''
        # Begin your code (Part 2-3)
        trained_model = self.model.fit(self.x_train, self.y_train)
        return trained_model
        #raise NotImplementedError("To be implemented")
        # End your code (Part 2-3)
```

Questions:

1. **Explain the difference between parametric and non-parametric models.**
   Parametric models make assumptions about the underlying distribution of the data and are defined by a finite set of parameters which are estimated from the data. One of the examples is gaussian distribution-based models. On the other hand, non-parametric models do not make assumptions about the distribution of the data. They are more flexible and can capture more complex patterns in the data.
2. **What is ensemble learning? Please explain the difference between bagging, boosting and stacking.**
   Ensemble learning is a technique used in machine learning which means the predictions of multiple models are merged to improve the overall performance. There are several methods, including bagging, boosting, and stacking.
   Bagging focuses on parallel training of multiple models on different subsets

of data; while boosting focuses on sequential training of models to correct errors from the previous models and each classifier has different weights, AdaBoost is one of the examples. Stacking involves training models of different types, and a meta-model is trained using the predictions made by the base models as features. This meta-model learns how to best combine the predictions to make the final prediction.

3. **Explain the meaning of the "n_neighbors" parameter in KNeighborsClassifier, "n_estimators" in RandomForestClassifier and AdaBoostClassifier.**

   "n_neighbors" means the number of nearest neighbors to consider when making classification prediction in KNN algorithm. "n_estimators" means the number of decision trees (Random Forest) or weak learners (AdaBoost) to be used in the ensemble.

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

Actual\predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

TP (True Positive): The answer should be "YES" (positive) and the prediction is also positive.

FN (False Negative): The answer should be "YES" (positive) but the prediction is negative.

FP (False Positive): The answer should be "NO" (negative) but the prediction is positive.

TN (True Negative): The answer should be "NO" (negative) and the prediction is also negative.

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 = TN + TP / TN + TP + FN +FP
   - Precision = TP / TP + FP
   - Recall = TP / TP + FN

   Accuracy means the overall correctness of predictions, while precision focuses on the accuracy of positive predictions and recall is also known as sensitivity, which focuses more on the ability to capture all positive instances. The are often used when the class is imbalance or when the costs associated with FP and FN differs significantly.

Part 3: Additional experiments (10%)

Adjust hyper-parameters to find the best model.

1. For KNeighborsClassifier( )

- When n_neighbors = 1 and p = 1 , KNN model has the best accuracy: 0.8933.
- The higher the value of n_neighbors, the lower the accuracy tends to be.
- There's isn't a significant difference between setting weight as 'distance' or 'uniform'.
- Setting p = 1 seems to yield better results.
- FN > FP (Since the positions of FN and FP in confusion matrix are reversed.)

```
n_neighbors = 1
weights = uniform
p = 1

Accuracy: 0.8933
Confusion Matrix:
[[298  62]
 [  2 238]]
```

```
n_neighbors = 1
weights = uniform
p = 2

Accuracy: 0.8867
Confusion Matrix:
[[297  65]
 [  3 235]]
```

```
n_neighbors = 1
weights = distance
p = 1

Accuracy: 0.8933
Confusion Matrix:
[[298  62]
 [  2 238]]
```

```
n_neighbors = 2
weights = uniform
p = 1

Accuracy: 0.86
Confusion Matrix:
[[300  84]
 [  0 216]]
```

```
n_neighbors = 2
weights = uniform
p = 2

Accuracy: 0.8433
Confusion Matrix:
[[300  94]
 [  0 206]]
```

```
n_neighbors = 2
weights = distance
p = 1

Accuracy: 0.8917
Confusion Matrix:
[[298  63]
 [  2 237]]
```

```
n_neighbors = 2
weights = distance
p = 2

Accuracy: 0.8867
Confusion Matrix:
[[297  65]
 [  3 235]]
```

```
n_neighbors = 5
weights = uniform
p = 1

Accuracy: 0.8717
Confusion Matrix:
[[300  77]
 [  0 223]]
```

```
n_neighbors = 5
weights = distance
p = 1

Accuracy: 0.8717
Confusion Matrix:
[[300  77]
 [  0 223]]
```

```
n_neighbors = 5
weights = distance
p = 2

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

```
n_neighbors = 8
weights = distance
p = 1

Accuracy: 0.85
Confusion Matrix:
[[300  90]
 [  0 210]]
```

2. For RandomForestClassifier( )

- When n_estimators = 300, criterion = log_loss, max_features = log2 and max_samples = 0.8 , RF model has the best accuracy: 0.9817.
- There's only a slight difference when adjusting the hyper-parameters.
- It seems that setting max_samples as 0.7 to 0.8 has a better result.
- Criterion accuracy results: log_loss > entropy > gini
- FP > FN (Since the positions of FN and FP in confusion matrix are reversed.)

```
n_estimators = 100
criterion = gini
max_features = sqrt
max_samples = 0.8

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

```
n_estimators = 100
criterion = gini
max_features = sqrt
max_samples = 0.7

Accuracy: 0.9783
Confusion Matrix:
[[288   1]
 [ 12 299]]
```

```
n_estimators = 100
criterion = gini
max_features = sqrt
max_samples = 0.5

Accuracy: 0.97
Confusion Matrix:
[[285   3]
 [ 15 297]]
```

```
n_estimators = 200
criterion = entropy
max_features = log2
max_samples = 0.8

Accuracy: 0.9783
Confusion Matrix:
[[288   1]
 [ 12 299]]
```

```
n_estimators = 300
criterion = entropy
max_features = log2
max_samples = 0.8

Accuracy: 0.9783
Confusion Matrix:
[[288   1]
 [ 12 299]]
```

```
n_estimators = 300
criterion = log_loss
max_features = log2
max_samples = 0.8

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 300
criterion = log_loss
max_features = log2
max_samples = 0.8

Accuracy: 0.9817
Confusion Matrix:
[[289   0]
 [ 11 300]]
```

```
n_estimators = 350
criterion = log_loss
max_features = log2
max_samples = 0.8

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 300
criterion = entropy
max_features = log2
max_samples = 0.7

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

3. For AdaBoostClassifier( )

- I use the best RF model as estimator of AdaBoostClassifier.
- The difference in effect between setting different n_estimators and learning_rate is not significant; primarily, the impact is more influenced by the estimator.
- Setting learning_rate to 0.4 will have a better result.
- When n_estimators = 300 and learning_rate = 0.4, AdaBoost model has the best accuracy: 0.9817, which is same as the best RF model.
- FP > FN (similar to RF model)

```
n_estimators = 500
learning_rate = 0.4

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 300
learning_rate = 0.4

Accuracy: 0.9817
Confusion Matrix:
[[289   0]
 [ 11 300]]
```

```
n_estimators = 500
learning_rate = 0.4

Accuracy: 0.9817
Confusion Matrix:
[[289   0]
 [ 11 300]]
```

```
n_estimators = 500
learning_rate = 0.8

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 300
learning_rate = 1

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 500
learning_rate = 0.3

Accuracy: 0.98
Confusion Matrix:
[[289   1]
 [ 11 299]]
```

```
n_estimators = 50
learning_rate = 0.6

Accuracy: 0.9817
Confusion Matrix:
[[289   0]
 [ 11 300]]
```

4. Summary

- Accuracy: AB (0.9817) ≈ RF (0.9817) > KNN (0.8933)
- F1-Score: AB (0.9813) ≈ RF (0.9813) > KNN (0.9030)
  (F1-Score = 2 * Precision * Recall / Precision + Recall)
- Actually, not every execution of the same model yields identical results but they are very similar.
- In Part4, I use the best AdaBoost model to detect.

Part 4: Detect car (15%)

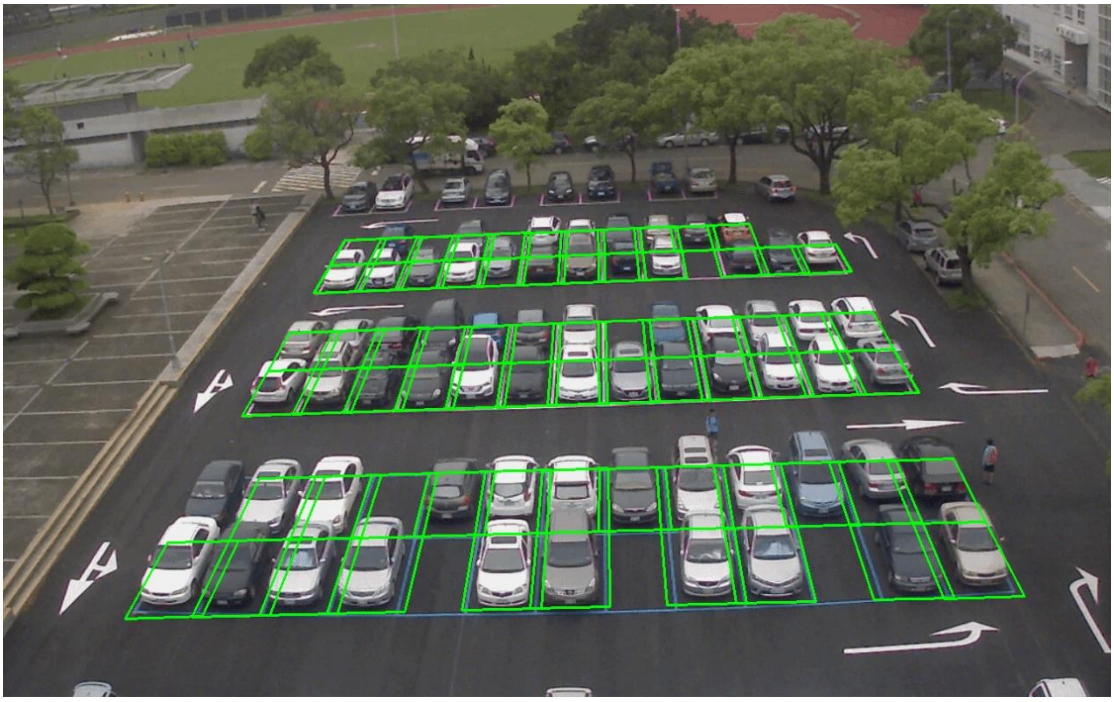
First, read ‘detectData.txt’ to access the coordinates of each parking lots and store them in ‘coordinates’ list. Next, use ‘cv2.VideoCapture()’ to separate the gif file into 50 frames. In each frame, use ‘crop’ function to crop each parking lots and turn them to 36 x 16 grayscale images. Finally, use clf.classify() function to make prediction and store the predicted result in ‘predicted_result‘ list.

```python
        # Begin your code (Part 4)
        # read in coordinates
        coordinates = []
        with open(data_path, 'r') as file:
            for line in file.readlines():
                coordinates.append(line.strip().split())
        coordinates = coordinates[1:]
        # deal with each frame
        video = cv2.VideoCapture('data/detect/video.gif')
        predicted_results = [] # for all frames
        frames = []

        while True:
            retval, frame = video.read()
            predicted_result = [] # for this frame's all parking lots
            if not retval:
                break

            # prediction for each parking lots
            for coordinate in coordinates:
                x1, y1, x2, y2, x3, y3, x4, y4 = (coordinate[0],coordinate[1],coordinate[2],coordinate[3],coordinate[4],
                                                  coordinate[5],coordinate[6],coordinate[7])
                cropped_img = crop(x1, y1, x2, y2, x3, y3, x4, y4, frame)
                cropped_img = cv2.resize(cropped_img,(36,16))
                gray_img = cv2.cvtColor(cropped_img, cv2.COLOR_BGR2GRAY)
                label = str(clf.classify([gray_img.reshape(-1)]))
                predicted_result.append(label+' ')
```

Using cv2.polylines() to draw the green bounding box if predicted_result of that parking lots is 1. Finally, save the results of all parking lots of that frames in ‘predicted_results’ list, which is for all frames and is used to output the ML_Models_pred.txt file.

```
    # draw the green box if predicted_result is 1
    # because not all the box is parallel to the image axis, using polylines()
    # for each parking lots
    for i , coordinate in enumerate(coordinates):
        if predicted_result[i] == '1 ':
            x1, y1, x2, y2, x3, y3, x4, y4 = (int(coordinate[0]),int(coordinate[1]),int(coordinate[2]),
              int(coordinate[3]),int(coordinate[4]),int(coordinate[5]),int(coordinate[6]),int(coordinate[7]))
            pts = np.array([[x1,y1],[x2,y2],[x4,y4],[x3,y3],[x1,y1]], np.int32)
            frame = cv2.polylines(frame, [pts], False, (0,255,0), 2)

    # save the predict result and frame
    predicted_result.append('\n')
    predicted_results.append(predicted_result)
    frames.append(frame)

cv2.imwrite('firstframe.png', frames[0])
cv2.destroyAllWindows()

# output predictions as .txt file
with open ("ML_Models_pred.txt", 'w') as file:
    for predicted_result in predicted_results:
        file.writelines(predicted_result)

#raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

The first frame with the bounding boxes (Using AdaBoost model) :



Part 5: Draw a line graph (5%)

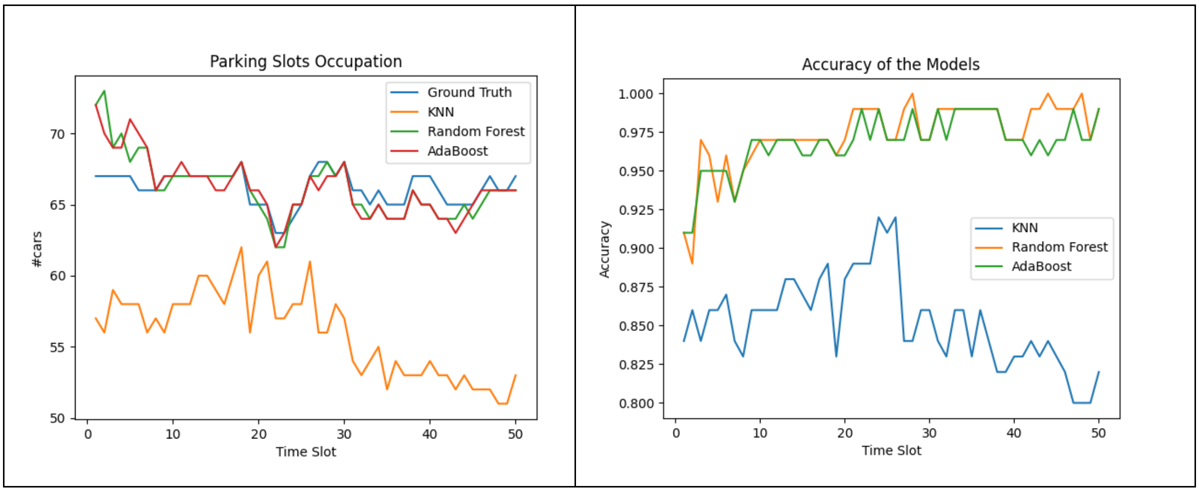
The code of this part is written in main.py. And I first generated three txt file of the detection result of KNN, RF, AB model. Then store the data in each list like the code below. And then using matplotlib to draw the two graph.

```python
# Part 5: Draw line graph
# open txt file and store value
groundtruth = [[0 for _ in range(76)] for _ in range(51)]
knn_pred, rf_pred, ab_pred = [0]*51, [0]*51, [0]*51

with open('GroundTruth.txt', 'r') as file:
        ground_count = [0]*51
        for line_num, line  in enumerate(file.readlines(),1):
            values = line.strip().split()
            for i, value in enumerate(values,0):
                groundtruth[line_num][i] = value
                if value == '1':
                    ground_count[line_num]+=1


with open('KNN.txt', 'r') as file:
        knn_count = [0]*51
        for line_num, line  in enumerate(file.readlines(),1):
            values = line.strip().split()
            for i, value in enumerate(values,0):
                if value == '1':
                    knn_count[line_num]+=1
                if value == groundtruth[line_num][i]:
                    knn_pred[line_num] +=1
```

```python
# 1. Parking Slots Occupation

x_values = list(range(1, 51))
y_values_1 = ground_count[1:51]
y_values_2 = knn_count[1:51]
y_values_3 = rf_count[1:51]
y_values_4 = ab_count[1:51]

plt.plot(x_values, y_values_1, label='Ground Truth')
plt.plot(x_values, y_values_2, label='KNN')
plt.plot(x_values, y_values_3, label='Random Forest')
plt.plot(x_values, y_values_4, label='AdaBoost')

plt.xlabel("Time Slot")
plt.ylabel("#cars")
plt.title("Parking Slots Occupation")

plt.legend()
plt.savefig("Occupation2.png")
plt.close()
```

```python
# 2. Accuracy of the models

x = list(range(1, 51))
for i in range(1,51):
    knn_pred[i] = knn_pred[i]/76
    rf_pred[i] = rf_pred[i]/76
    ab_pred[i] = ab_pred[i]/76

plt.plot(x, knn_pred[1:51], label='KNN')
plt.plot(x, rf_pred[1:51], label='Random Forest')
plt.plot(x, ab_pred[1:51], label='AdaBoost')

plt.xlabel("Time Slot")
plt.ylabel("Accuracy")
plt.title("Accuracy of the Models")
plt.legend()
plt.savefig("Accuracy2.png")
plt.close()
```

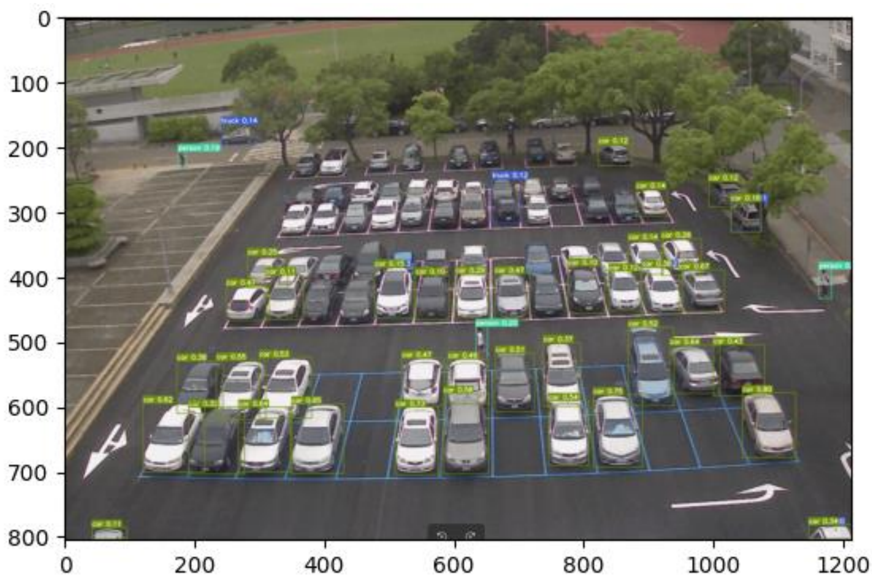
According to the left graph (Parking Slots Occupation), it is obvious to notice that KNN has the most significant disparity with the Ground Truth. While the trends of RF and Adaboost are closely similar to Ground Truth. In the right graph (Accuracy of the Models) , KNN shows the lowest accuracy around 0.85, declining in later frames. RF and AdaBoost, however, maintain accuracy above 0.9, initially slightly lower but improving notably afterward, exceeding 0.95.



Task B

Part 1: Load a pre-trained model and directly apply it to the .png to detect the object. (10%)

Run part 1 of Yolov7_sample_code.ipynb in Colab and get the image below.



Part 2 : Learn to fine-tune yolov7 model (10%)

First, since it takes around 90 minutes to run the default setting model, I set the epoch to 200 and set confidence threshold in testing data to 0.1, trying to see how it works. And the resulting accuracy of training data is 96% and testing data is 91.7%.

```
[11] # 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: 19/300 (0.063333)
     False Negative Rate: 5/300 (0.016667)
     Training Accuracy: 576/600 (0.960000)
```

```
[13] Calculate('/content/yolov7/HW1_material/test/','/content/yolov7/runs/detect/test/labels/')

     False Positive Rate: 50/300 (0.166667)
     False Negative Rate: 0/300 (0.000000)
     Training Accuracy: 550/600 (0.916667)
```

I want to get a better predicting result, thus I set epoch to 300, batch-size to 8 and still set confidence threshold in testing data to 0.1. The resulting training data's accuracy is 92.8% and testing data with 93.3% accuracy.

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

```
[ ] Calculate('/content/yolov7/HW1_material/test/','/content/yolov7/runs/detect/test/labels/')

    False Positive Rate: 31/300 (0.103333)
    False Negative Rate: 9/300 (0.030000)
    Training Accuracy: 560/600 (0.933333)
```

In conclusion, the higher the epoch size is, the longer the time is taken to train the model. And there's only a slightly different between setting different epoch size. But I only give these two tries, maybe setting a different batch-size will have a better result. According to the references, the larger the batch size, the shorter the time required, while smaller batch sizes introduce higher stochasticity in gradients and generally lead to better generalization.

Bonus:

(The detailed code is in bonus.py)

Using GradientBoostingClassifier() to train the model and get the result of 0.975 accuracy by setting n_estimators=100, learning_rate=0.1, max_depth=3. Then I increase the n_estimators to 300, it gets a better result of 0.9817 accuracy but takes longer time to complete.

Comparing with AdaBoost model, GradientBoosting exhibits a more balanced distribution between FP and FN. Unlike AdaBoost, where FP tend to be larger than FN. The accuracy is actually quite similar with AdaBoost, however, GradientBoostingClassifier takes more time to complete the detection.

```
    def build_model(self):
        model = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3)
        return model
```

```
Using GradientBoostingClassifier:

Accuracy: 0.975
Confusion Matrix:
[[293   8]
 [  7 292]]
```

```
Using GradientBoostingClassifier:

Accuracy: 0.9817
Confusion Matrix:
[[294   5]
 [  6 295]]
```

Trying to get a higher accuracy, I set the n_estimators to 200 and increase the learning_rate to 0.4, and get a result of 0.9833 accuracy, which is higher than the AdaBoost model's performance. I keep increasing the learning rate to 0.7, but it turned out worse. So, setting learning_rate around 0.4-0.5 may be more suitable.

```
Using GradientBoostingClassifier:

Accuracy: 0.9833
Confusion Matrix:
[[294   4]
 [  6 296]]
```

```
Using GradientBoostingClassifier:

Accuracy: 0.9783
Confusion Matrix:
[[294   7]
[[294   7]
 [  6 293]]
```

Problems and Solutions:

1. In part 4, when drawing the green bounding box of the first frame of predicting image, I use cv2.rectangle() to draw the box first. But quickly found out that it doesn't work well since the image is not totally parallel. Therefore, I shifted to use cv2.polylines() to draw the box and it works successfully.
2. In part B, when using Colab, it takes over an hour to train the model, which is time-consuming. Additionally, if there is a long interval between steps without executing the next run, the connection is prone to disconnection, requiring the entire process to be restarted. However, this may be a drawback of using Colab, despite the availability of free GPU resources for computation.