Introduction to Artificial Intelligence
Report

109550198 卜銳凱

Screenshot of your code and brief explanation:
TASK A
Part 1 (dataset.py)

```
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) representing the image.
    (remember to resize and convert the parking space images to 36 x 16 grayscale images.)
    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)
    #raise NotImplementedError("To be implemented")
    dataset=[]

    for image_name in os.listdir(os.path.join(data_path,"car")):

        img = cv2.imread(os.path.join(data_path,"car", image_name))
        img = cv2.resize(img, (36, 16))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        tuple=(img, 1)
        dataset.append(tuple)

    for image_name in os.listdir(os.path.join(data_path,"non-car")):
        img = cv2.imread(os.path.join(data_path,"non-car", image_name))
        img = cv2.resize(img, (36, 16))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        tuple=(img, 0)
        dataset.append(tuple)
    # End your code (Part 1)
    return dataset
```

To concatenate the route of the car/non-car files, I use "os.path.join(). Using "os.listdir()," to get a list of the names in the directory and load the image. In addition, use "cv2.resize()" and "cv2.cvtColor() + cv2.COLOR BGR2GRAY" to convert the image to 36*16 grayscale. Finally, after finishing processing all the images, combine the image and its label (car=1, non-car=0) in a tuple and append to a list named dataset.

Part 2(model.py)

```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)
        #raise NotImplementedError("To be implemented")
        self.x_train = np.array([data[0] for data in train_data])
        self.y_train = np.array([data[1] for data in train_data])
        self.x_test = np.array([data[0] for data in test_data])
        self.y_test = np.array([data[1] for data in test_data])

        self.x_train=np.reshape(self.x_train, (self.x_train.shape[0], -1))
       # print(self.x_train.shape)
        self.x_test=np.reshape(self.x_test, (self.x_test.shape[0], -1))
       #print(self.y_train.shape)
       # 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)
        #raise NotImplementedError("To be implemented")
        if model_name == 'KNN':
            model = KNeighborsClassifier(n_neighbors=5)
        elif model_name == 'RF':
            model = RandomForestClassifier(n_estimators=100, max_depth=5, random_state=0)
        elif model_name == 'AB':
            model = AdaBoostClassifier(n_estimators=100, random_state=0)
        else:
            raise ValueError('Invalid model name')
        return model
        # 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)
    #raise NotImplementedError("To be implemented")
    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]
```

I firstly convert train and test data into scikit-learn format and assign them to instance variables. Then I reshape training and test data to be in the correct format for scikit-learn models. After I create a KNN model, a Random Forest Model and Adaboost model, if an invalid model_name is passed, raise an error. Finally, I used self.model.fit(self.x_train, self.y_train) to train the model on training data.

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

  The main difference between parametric and non-parametric models is the number of parameters. Parametric models have a fixed number of parameters that are estimated from the data, whereas non-parametric models have a flexible number of parameters that depend on the volume and complexity of the data. Yet, they might not be able to capture the intricacy of the data because parametric models are typically easier and quicker to train. However, they may be slower and need more data to train than non-parametric models, which are more adaptable and can recognize complicated patterns in the data.

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

  - Ensemble learning is a machine learning technique that involves integrating various models to increase a prediction's overall performance and accuracy. It is predicated on the premise that integrating several models frequently yields more accurate forecasts than depending solely on one model**.**

  - Ensemble techniques like bagging, boosting, and stacking combine many models to enhance performance. Stacking can be more efficient when dealing with complicated situations or when combining models with varied strengths and weaknesses. Bagging and boosting are simpler and easier to execute than. stacking**.**

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

    - The "n neighbors" parameter in KNeighborsClassifier: KNeighborsClassifier is a machine learning algorithm used for classification tasks. It is a type of instance-based learning, which implies that it memorizes the training data rather than building a model from it in order to make predictions. The number of nearest neighbors to consider while creating a prediction is specified by the "n neighbors" option in the KNeighborsClassifier. For instance, if "n neighbors" is set to 5, the algorithm will base its prediction on the 5 test data points that are nearest to it.

    - n estimators" in RandomForestClassifier: A machine learning approach for classification tasks is called RandomForestClassifier. Several decision trees are combined in this ensemble learning technique to produce a prediction. The number of decision trees to utilize in the model is determined by the "n estimators" argument in RandomForestClassifier. As an example, if "n estimators" is set to 100, the algorithm will build 100 decision trees and then combine their predictions to form a final prediction.

    - AdaBoostClassifier "n estimators": A machine learning algorithm called AdaBoostClassifier is employed for classification tasks. Additionally, it uses an ensemble learning technique to integrate the predictions of several weak learners. AdaBoostClassifier's "n estimators" parameter sets the maximum number of weak learners (usually decision trees) that can be used in the model. The method will generate a maximum of 100 decision trees and combine their predictions to get a final prediction, for example, if "n estimators" is set to 100.

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

    The confusion matrix is a table that is frequently used to assess the performance of a classification model. It summarizes the predicted and actual class labels for a set of data points. The confusion matrix has four numbers, which are as follows:

    - True Positive (TP): The number of data points correctly classified as positive. In other words, the model predicted the positive class, and the actual class was also positive.
    - False Positive (FP): The number of data points that were incorrectly classified as positive. In other words, the model predicted a positive class, but the actual class was negative.

- True Negative (TN): The number of data points classified correctly as negative. In other words, the model predicted the negative class, and it was also correct.
- False Negative (FN): The number of data points classified incorrectly as negative. In other words, the model predicted a negative class but found a positive class.

- **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 and recall are two often used metrics to assess a classification model's performance, particularly when the classes are imbalanced.

  Precision reveals how frequently a model predicts a positive case correctly.

  Precision = TP / (TP + FP)

  Recall reveals how frequently, out of all the real positive cases, the model correctly detects positive cases.

  Recall = TP / (TP + FN)

  Precision is used when the goal is to minimize false positives, whereas recall is used when the goal is to minimize false negatives. If the class distribution is imbalanced, accuracy may not be a good metric to use to assess the model's performance because it is biased towards the majority class. In such cases, precision and recall are useful metrics for evaluating the classifier's performance.

Part 3:

KNeighborsClassifier:

n_neighbors=1	**Accuracy=0.8867**
n_neighbors=3	**Accuracy=0.855**
n_neighbors=5	**Accuracy=0.845**
n_neighbors=10	**Accuracy=0.7983**
n_neighbors=100	**Accuracy=0.6367**

RandomForestClassifier:

n_estimators=1	**Accuracy=0.9067**
n_estimators=3	**Accuracy=0,9667**
n_estimators=5	**Accuracy=0.9667**
n_estimators=10	**Accuracy=0.985**
n_estimators=100	**Accuracy=0.9767**

AdaBoostClassifier:

n_estimators=1	**Accuracy=0.8117**
n_estimators=3	**Accuracy=0,925**
n_estimators=5	**Accuracy =0.93**
n_estimators=10	**Accuracy=0.923**
n_estimators=100	**Accuracy=0.9517**

After changing , the parameter “n_neighbors” in the KNeighborsClassifier and “n_estimators” in RandomForestClassifier and AdaBoostClassifier, I found the best classifier with its optimal hyperarameters :KneighborsClassifier with n_neighbors=1, RandomForestClassifier with n_estimators=10 , AdaBoostClassifier with n_estimators=100.

Part 4 (detection.py):

```
def detect(data_path, clf):
    """
    Please read detectData.txt to understand the format.
    Use cv2.VideoCapture() to load the video.gif.
    Use crop() to crop each frame (frame size = 1280 x 800) of video to get parking space images. (image size = 360 x 160)
    Convert each parking space image into 36 x 16 and grayscale.
    Use clf.classify() function to detect car, If the result is True, draw the green box on the image like the example provided on the spec.
    Then, you have to show the first frame with the bounding boxes in your report.
    Save the predictions as .txt file (ML_Models_pred.txt), the format is the same as Sample.txt.

      Parameters:
        dataPath: the path of detectData.txt
      Returns:
        No returns.
    """
    # Begin your code (Part 4)
    #raise NotImplementedError("To be implemented")
    # Load data from the specified file path
    with open(data_path, "r") as f:
        nums = int(f.readline().strip())
        coordinates = []
        for i in range(nums):
            num_list = f.readline().strip().split(' ') # ['101', '102']

            x = []
            for num in num_list:
                x.append(int(num))

            coordinates.append(tuple(x))
```

I first read each parking spaces coordinate from detectData.txt and store them in a 2D list for later cropping the image.

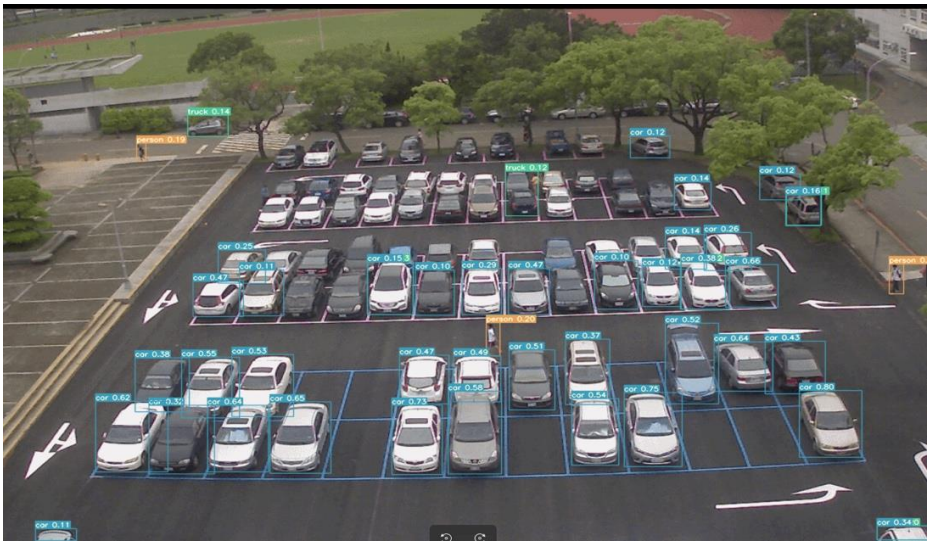
```
# Open a video file for processing
cap = cv2.VideoCapture('data/detect/video.gif')
idx=0
with open('ML_Models_pred.txt', 'a') as f:
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        for i in range (int(nums)):
            cropped_frame = crop(coordinates[i][0],coordinates[i][1],coordinates[i][2],coordinates[i][3],coordinates[i][4],coordinates[i][5],coordinates[i][6],coor
            cropped_frame = cv2.cvtColor(cropped_frame, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
            cropped_frame = cv2.resize(cropped_frame, (36, 16), interpolation=cv2.INTER_AREA)  # Resize the cropped frame to a fixed size

            # print(cropped_frame.shape)
            cropped_frame=np.reshape(cropped_frame,(1,-1))
            # print(cropped_frame.shape)
            if clf.classify(cropped_frame) == 1:
                f.write('1 ')
                green_color = (0, 255, 0) # BGR
                pts = np.array([[coordinates[i][0], coordinates[i][1]], [coordinates[i][2], coordinates[i][3]], [coordinates[i][6], coordinates[i][7]], [coordinate
                pts = pts.reshape((4, 1, 2))# Reshape the coordinates array
                cv2.polylines(frame, [pts], True, green_color, 1)# Draw a polygon on the frame
            else:
                f.write('0 ')
        cv2.imshow('Image', frame)
        f.write('\n')
        if idx==0:
            cv2.imwrite('Image.png',frame)
            cv2.waitKey(0)
            cv2.destroyAllWindows()
            index=1
        if cv2.waitKey(30) == 13:
            break


cv2.destroyAllWindows()
cap.release()


# End your code (Part 4)
```

Finally, I open and read the video.gif using "cv2.VideoCapture" and "cv2.read()," respectively. I then use the crop function to take each parking space from the image before sending it to the "clf.classify()" function. If the classifier returns 1, indicating that a car was detected, then display a bounding car on the original picture using "cv2.polylines()," and enter 1 in the 'ML_Models_pred.txt' file. Write 0 in the 'ML_Models_pred.txt' if not.

TASK B:

Part 1:

Here is the result:



Part 2:

Here is the accuracy screenshot:

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

False Positive Rate: 16/300 (0.053333)
False Negative Rate: 26/300 (0.086667)
Training Accuracy: 558/600 (0.930000)
```

Discuss what you observed with accuracy, F1-score and parking slots occupation plot of different methods .

Adaboost

yolov7

Why do we need F1-score before discussing accuracy and F1-score? Because accuracy is (TP + FP)/(ALL EXAMPLE), we don't know how the classify in question performs. As a result, the F1-score, which is the harmonic mean of precision and recall, can be used. In this manner, we can evaluate the performance of the positive class.

	Accuracy	F1-score
Adaboost	0.95117	0.95
Yolov7	0.93	0.96

Furthermore, in light of Yolov7, the value of F1-score is greater than the value of accuracy, indicating that Yolov7 predicts even better on our interesting class.

Describe problems you meet and how you solve them.

In part 1, when I want to convert an image to grayscale, I get an error, so I use "img = cv2.cvtColor(img, cv2.COLOR BGR2GRAY)" to solve the problem.

Since I'm not familiar with os, I naively typed all the route by for loop to load image at first, and there was an error. Finally, I employ the "os.listdir()" function to solve the problem.