รายงาน

เรื่อง Ship Object Detection

จัดทำโดย

6530200151 นายณัฐพริษฐ์ สมบูรณ์

6530200266 นายปรวภัทร มุธะรพัฒน์

6530200304 นายพีระเมศร์ จุกกษัตริย์

6530200673 นายนนทวัฒน์ พันธ์เผือก

6530200819 นายสถาพร สัตยชิติ

เสนอ

อาจารย์ ชโลธร ชูทอง

รายงานนี้เป็นเป็นส่วนหนึ่งในรายวิชา Fundamental of Artificial Intelligence

มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตศรีราชา

คำนำ

รายงานเล่มนี้จัดทำขึ้นเพื่อเป็นส่วนหนึ่งในรายวิชา Fundamental of Artificial Intelligence เพื่อให้ได้ศึกษาหาความรู้ในเรื่อง Ship Object Detection ซึ่ง Model ที่นำมายังไม่เสร็จสมบูณร์ เป็นกรณีศึกษาในรายวิชา Fundamental of Artificial Intelligence โดยใช้ Model Faster R-CNN จากเริ่มต้นจนจบโดยใช้ TensorFlow ก่อนอื่นจะทำการแยกคุณลักษณะ (features) จากรูปภาพโดยใช้โมเดล CNN แล้วส่งผ่านเข้าสู่เครือข่าย Region Proposal Network (RPN) Model Faster R-CNN (Faster Region Convolutional Neural Network) เป็นโมเดลสำหรับการตรวจจับวัตถุในภาพที่มีประสิทธิภาพสูงและมีความแม่นยำ เป็นการปรับปรุงจากโมเดล R-CNN (Region Convolutional Neural Network) และ Fast R-CNN โดยเพิ่มโมดูลที่ชื่อว่า Region Proposal Network (RPN) ที่ช่วยในการสร้างพื้นที่เสนอ (region proposals) อัตโนมัติ และทำให้การตรวจจับวัตถุเป็นไปอย่างรวดเร็ว โดยมีเป้าหมายเพื่อให้สามารถทำงานในเวลาเป็นจริงได้ด้วยความเร็วสูงขึ้นโดยเฉพาะในงานที่ต้องการการตรวจจับวัตถุในเวลาเฉพาะเป้าหมาย (real-time object detection tasks) อย่างกว้างขวางทั้งในงานด้านวิจัยและงานประยุกต์ต่าง ๆ

Model จะประกอบเป็น 5 ส่วนคือ

1.สร้าง Model พื้นฐาน Selective Search หรือการใช้ Convolutional Neural Networks (CNN) เพื่อสร้างพื้นที่เสนอ (Region Proposals) ที่เป็นเวคเตอร์ที่เป็นไปได้ที่จะมีวัตถุที่สนใจอยู่

2.นำ Model ที่ได้ไปทำการสร้าง Feature Maps ส่งผ่านโครงข่าย CNN เพื่อสร้าง feature maps ที่มีลักษณะการแสดงของภาพที่ปรับเปลี่ยนเชิงลึก (deep feature representation)

3.จาก feature maps ที่ได้มา เราสร้าง Regions of Interest (RoIs) โดยใช้ Region Proposals จากขั้นตอนที่ 1 และปรับขนาดให้เข้ากับ feature maps

4.สร้างคุณลักษณะ (Features) จาก RoIs: แต่ละ RoI ถูกส่งผ่าน layers เพิ่มเติมของ CNN เพื่อสร้าง feature vectors ที่เกี่ยวข้องกับข้อมูลภายในแต่ละ RoI

5.ทำนายหมวดหมู่ของวัตถุและการกรองพื้นที่ที่เกิน: ใช้ layers สุดท้ายของโครงข่าย CNN เพื่อทำนายหมวดหมู่ของวัตถุที่อยู่ในแต่ละ RoI

Model จะแบ่งออกเป็น 2 Part

Part 1 ทำการสร้าง Model Train Model และบันทึกนำมาใช้ใน Part 2

Part 2 นำ Model CNN ทำนายหมวดหมู่ของวัตถุที่อยู่ในแต่ละ RoI

Step 1 Import and install สิ่งที่จำเป็นต่อการ Train Model

```
!pip install opencv-python
```

ทำการติดตั้งแพ็คเกจ opencv-python

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os, random, cv2, pickle, json, itertools
import imgaug.augmenters as iaa
import imgaug.imgaug

from IPython.display import SVG
from tensorflow.keras.utils import plot_model, model_to_dot
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from collections import Counter
from sklearn.utils import class_weight
from tqdm import tqdm
from sklearn.preprocessing import LabelBinarizer

from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import (Add, Input, Conv2D, Dropout, Activation, BatchNormalization, MaxPooling2D, ZeroPadding2D, AveragePooling2D, Flatten, Dense)
from tensorflow.keras.optimizers import Adam, SGD
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint, Callback
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.initializers import *
```

โมดูลทั้งหมดที่ Import เข้ามาจะถูกใช้ในการสร้างและฝึกโมเดลการเรียนรู้ของเครื่องใน Python โดยใช้ TensorFlow/Keras ในการดำเนินการต่อไป

```
def show_final_history(history):

    plt.style.use("ggplot")
    fig, ax = plt.subplots(1,2,figsize=(15,5))
    ax[0].set_title('Loss')
    ax[1].set_title('Accuracy')
    ax[0].plot(history.history['loss'],label='Train Loss')
    ax[0].plot(history.history['val_loss'],label='Validation Loss')
    ax[1].plot(history.history['accuracy'],label='Train Accuracy')
    ax[1].plot(history.history['val_accuracy'],label='Validation Accuracy')

    ax[0].legend(loc='upper right')
    ax[1].legend(loc='lower right')
    plt.show();
    pass
```

ฟังก์ชัน show_final_history(history) มีหน้าที่แสดงกราฟเพื่อสรุปประสิทธิภาพของโมเดลหลังจากการฝึก (training) โดยรับพารามิเตอร์ history ซึ่งเป็นข้อมูลประวัติการฝึกของโมเดลที่เก็บไว้ในรูปแบบของ dictionary

ซึ่งเก็บค่าความสูญเสีย (loss) และค่าความแม่นยำ (accuracy) ของชุดข้อมูลการฝึกและการทดสอบ (validation) ตลอดระยะเวลาการฝึกโมเดล ฟังก์ชันจะทำงานตามขั้นตอน ทั้งหมดนี้
เราจะได้กราฟที่แสดงประสิทธิภาพของโมเดลในเรื่องของ Loss และ Accuracy
ของการฝึกและการทดสอบในแต่ละรอบการฝึก
ทำให้เราสามารถประเมินประสิทธิภาพของโมเดลได้อย่างชัดเจน

```
def plot_confusion_matrix(cm,classes,title='Confusion Matrix',cmap=plt.cm.Blues):

#     np.seterr(divide='ignore',invalid='ignore')
    cm = cm.astype('float')/cm.sum(axis=1)[:,np.newaxis]
    plt.figure(figsize=(10,10))
    plt.imshow(cm,interpolation='nearest',cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes,rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f'
    thresh = cm.max()/2.
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
        plt.text(j,i,format(cm[i,j],fmt),
                 horizontalalignment="center",
                 color="white" if cm[i,j] > thresh else "black")
        pass

    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    pass
```

ฟังก์ชันนี้จะสร้างกราฟที่แสดงเมทริกซ์การสับเปลี่ยน (confusion matrix)
ที่แสดงความสัมพันธ์ระหว่างค่าที่ทำนายได้กับค่าจริงของข้อมูลที่ใช้ในการทดสอบโมเดลการเรียนรู้ของเครื่อง
ซึ่งเป็นเครื่องมือสำคัญในการประเมินประสิทธิภาพของโมเดล

Step 1 นายปรวภัทร มุธะรพัฒน์

Step 2 Upload Dataset (Upload Images) เตรียมข้อมูล

รูปภาพจากภาพถ่ายดาวเทียมของเรือถูกโหลดเป็นอาร์เรย์ของ numpy พร้อมกับ labels [0,1] ที่สอดคล้องกับคลาส no-ship และ ship ข้อมูลถูกโหลดเป็นอาร์เรย์ของ numpy เนื่องจากจะสามารถปรับปรุงข้อมูลและการเพิ่ม/ลดสัดส่วนข้อมูลสามารถทำได้ง่ายขึ้น

```
datasets = ['/content/drive/MyDrive/AI/Dataset']

class_names = ["no-ship","ship"]

class_name_labels = {class_name:i for i,class_name in
enumerate(class_names)}

num_classes = len(class_names)
class_name_labels
```

1. datasets: เป็นตัวแปรที่เก็บที่อยู่ของชุดข้อมูลภาพ

2. class_names: ชื่อคลาส มี "no-ship" และ "ship"

3. class_name_labels: จะมีการแมปชื่อคลาสกับตัวเลขรหัส โดยใช้ชื่อคลาสเป็นคีย์และตำแหน่งของคลาสในลิสต์ class_names เป็นค่า คือ "no-ship" จะเท่ากับ 0 และ "ship" จะเท่ากับ 1

4. num_classes: เป็นจำนวนของคลาสทั้งหมดในชุดข้อมูล

สรุปแล้ว เป็นการเตรียมข้อมูลสำหรับการทำงานกับโมเดลการเรียนรู้ของคอมพิวเตอร์เพื่อการจำแนกประเภทภาพระหว่าง "no-ship" และ "ship"

```
def load_data():
    images, labels = [], []

    for dataset in datasets:

        for folder in os.listdir(dataset):
            label = class_name_labels[folder]

            for file in tqdm(os.listdir(os.path.join(dataset,folder))):

                img_path = os.path.join(dataset,folder,file)

                img = cv2.imread(img_path)
                img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
                img = cv2.resize(img, (48,48))

                images.append(img)
                labels.append(label)
                pass
            pass

        images = np.array(images,dtype=np.float32)/255.0
        labels = np.array(labels,dtype=np.float32)
        pass

    return (images, labels)
    pass
```

ฟังก์ชัน load_data() คือเพื่อโหลดข้อมูลภาพจากชุดข้อมูลที่ระบุในตัวแปร datasets โดยทำการประมวลผลตามขั้นตอน

1. สร้างรายการว่างเพื่อเก็บภาพ (images) และป้ายชื่อ (labels)

2. นำเข้าโมดูล os เพื่อใช้ในการทำงานกับระบบไฟล์ และ tqdm เพื่อแสดงแถบความคืบหน้าในการวนลูป

3. ทำการวนลูปผ่านทุกๆชุดข้อมูลที่ระบุใน datasets:

- วนลูปผ่านทุกๆโฟลเดอร์ในชุดข้อมูล

- กำหนดป้ายชื่อ (label) โดยใช้รายการชื่อคลาสที่มีอยู่ใน class_name_labels

- วนลูปผ่านทุกไฟล์ภาพภายในโฟลเดอร์:

  - สร้างเส้นทางของภาพ (img_path) โดยรวมเส้นทางของโฟลเดอร์กับชื่อไฟล์

  - อ่านภาพจากเส้นทางที่ระบุ แปลงสีของภาพจาก BGR เป็น RGB (ที่ OpenCV จัดการแบบ BGR)

  - ปรับขนาดของภาพเป็นขนาด 48x48 pixels

  - เพิ่มภาพและป้ายชื่อลงในรายการที่เตรียมไว้

- ทำการแปลงรายการของภาพและป้ายชื่อเป็น numpy array และแบ่งด้วย 255.0 เพื่อทำให้ค่าพิกเซลอยู่ในช่วง 0 ถึง 1

4. ส่งคืนค่า images และ labels

สรุปฟังก์ชันนี้มีหน้าที่โหลดข้อมูลภาพและป้ายชื่อจากชุดข้อมูลที่ระบุ และทำการประมวลผลเบื้องต้นบางอย่าง เช่น การปรับขนาดและการแปลงสีของภาพ เพื่อให้เหมาะสมกับการนำไปใช้ในโมเดลของคอมพิวเตอร์เพื่อการจำแนกประเภท

Step 2 นายนนทวัฒน์ พันธ์เผือก

Step 3 Exploratory Data Analysis (EDA)

```
(images, labels) = load_data() # โหลดข้อมูลเข้าจากฟังชั่น load_data()
images.shape, labels.shape  #จำนวนแถวและคอลัมน์ของ  ข้อมูลimages และข้อมูลlabels
```

images.shape และ labels.shape จะให้ผลลัพธ์เป็นลักษณะของข้อมูลในรูปแบบ (จำนวนแถว, จำนวนคอลัมน์) โดยที่จำนวนแถวแทนจำนวนข้อมูล และจำนวนคอลัมน์แทนจำนวนคุณลักษณะหรือคุณสมบัติของแต่ละข้อมูลในเซ็ตข้อมูล

```
n_labels = labels.shape[0] # นับจำนวนแถวในตัวแปร labels

n_, count = np.unique(labels, return_counts=True)   #หาจำนวนของแต่ละคลาสที่มีใน labels เพื่อหาค่าที่ไม่ซ้ำกันใน labels และเก็บผลลัพธ์ไว้ในตัวแปร n_ และ count

df = pd.DataFrame(data = count) # สร้าง DataFrame โดยเก็บข้อมูลจำนวนของแต่ละคลาสที่ได้จากขั้นตอนก่อนหน้าไว้ในคอลัมน์ 'Count'
df['Class Label'] = class_names # เพิ่มคอลัมน์ 'Class Label' ที่มีค่าเป็น class_names
df.columns = ['Count','Class-Label'] #กำหนดชื่อคอลัมน์ใหม่เป็น ['Count','Class-Label']
df.set_index('Class-Label',inplace=True)# กำหนด 'Class-Label' เป็น index ของ DataFrame
df
```

แสดงข้อมูลทางสถิติเกี่ยวกับข้อมูลคลาสที่ปรากฏในตัวแปร labels ที่เก็บข้อมูลคลาสเพื่อให้ทราบถึงการกระจายของข้อมูลในแต่ละคลาส

```
[9] df.plot.bar(rot=0) #ทำการพล็อตกราฟแท่ง (bar plot) ของข้อมูลจำนวนของแต่ละคลาสใน DataFrame
    plt.title("distribution of images per class"); #กำหนดชื่อกราฟเป็น "distribution of images per class"
```

การสร้างกราฟแท่ง (bar plot) เพื่อแสดงการกระจายของจำนวนข้อมูลในแต่ละคลาสภายใน DataFrame df ซึ่งเก็บข้อมูลจำนวนของแต่ละคลาสที่ได้จากขั้นตอนก่อนหน้าไว้ในคอลัมน์ 'Count'

```
#พล็อตกราฟแผนภูมิวงกลมของข้อมูลจำนวนของแต่ละคลาส
plt.pie(count,
        explode=(0,0),
        labels=class_names, #กำหนดให้ class_names เป็น labels
        autopct="%1.2f%%") #กำหนดรูปแบบของข้อความในแผนภูมิ เพื่อแสดงเป็นเปอร์เซ็นต์
plt.axis('equal');    #เพื่อให้กราฟแผนภูมิวงกลมมีสัดส่วนที่ถูกต้อง (หรือเท่ากัน) และไม่เบียดเบียน
```

พล็อตกราฟแผนภูมิวงกลม (pie chart) เพื่อแสดงการกระจายของจำนวนข้อมูลในแต่ละคลาสภายในตัวแปร count ซึ่งเก็บจำนวนข้อมูลของแต่ละคลาสที่ได้จากขั้นตอนก่อนหน้าไว้ในรูปแบบของ list

Step 3 นายสถาพร สัตยชิติ

Step 4 Augmenting Images of Minority Class

Minority Class เป็นขั้นตอนที่ใช้ในการเพิ่มจำนวนข้อมูลในคลาสที่มีจำนวนน้อยกว่าในชุดข้อมูล เพื่อให้มีการแจกแจงของคลาสที่สมดุลกันในชุดข้อมูล

```
[11] AUGMENTATION = True
```

+ โค้ด + ข้อความ

```
# สร้าง  ฟังก์ชัน augment_add() ที่รับพารามิเตอร์ images, seq (โมเดล augmentation), และ labels
def augment_add(images, seq, labels):

    augmented_images, augmented_labels = [],[] #สร้าง list ว่างสำหรับเก็บรูปภาพที่เพิ่มขึ้น
    for idx,img in tqdm(enumerate(images)):   #ลูป for เพื่อวนซ้ำผ่านทุกๆ รูปภาพใน images

        if labels[idx] == 1: # ตรวจสอบว่า labels ของรูปภาพนั้นเป็น 1 หรือไม่
            image_aug_1 = seq.augment_image(image=img)#ทำการ augmentation ภาพรอบที่ 1  โดยใช้โมเดล augmentation (seq)
            image_aug_2 = seq.augment_image(image=img)#ทำการ augmentation ภาพรอบที่ 2  โดยใช้โมเดล augmentation (seq)
            augmented_images.append(image_aug_1)# เพิ่มรูปภาพที่ augment แล้วเข้าไปใน augmented_images รอบที่ 1
            augmented_images.append(image_aug_2)# เพิ่มรูปภาพที่ augment แล้วเข้าไปใน augmented_images รอบที่ 2
            augmented_labels.append(labels[idx]#เพิ่ม label เดิมใน augmented_labels ครั้งที่ 1
            augmented_labels.append(labels[idx])#เพิ่ม label เดิมใน augmented_labels ครั้งที่ 2
        pass

    augmented_images = np.array(augmented_images, dtype=np.float32) #แปลง จาก list เป็น numpy array  numpy array ที่มี dtype เป็น np.float32
    augmented_labels = np.array(augmented_labels, dtype=np.float32) #แปลง จาก list เป็น numpy array  numpy array ที่มี dtype เป็น np.float32

    return (augmented_images, augmented_labels)# คืนค่า augmented_images และ augmented_labels

    pass
```

ฟังก์ชัน augment_add() นี้ถูกใช้เพื่อเพิ่มข้อมูลภาพที่ถูก augment ให้กับข้อมูลเดิมโดยการนำเข้า images, seq (โมเดล augmentation), และ labels และคืนค่า augmented_images และ augmented_labels ที่ได้หลังจากการ augment แล้ว

```
# สร้างโมเดล augmentation โดยใช้ Sequential()
seq = iaa.Sequential( [
    iaa.Fliplr(0.5),              #สลับภาพซ้าย-ขวาโดยมีโอกาส 50%
    iaa.Crop(percent=(0,0.1)),#ครอบภาพโดยตัดขอบข้างของภาพในอัตราส่วน 0-10%
    iaa.LinearContrast((0.75,1.5)),#เพิ่มความคมชัดของภาพโดยเปลี่ยนค่าความเข้มของภาพแบบเชิงเส้น
    iaa.Multiply((0.8,1.2), per_channel=0.2),#เพิ่มความเข้มของสีแต่ละช่องสีของภาพ (RGB) แบบสุ่ม
    iaa.Affine(
        scale={'x':(0.8,1.2), "y":(0.8,1.2)},#ปรับเปลี่ยนขนาดของภาพ
        translate_percent={"x":(-0.2,0.2),"y":(-0.2,0.2)},#ปรับเปลี่ยนตำแหน่งของภาพ
        rotate=(-25,25), #ปรับเปลี่ยนการหมุนของภาพ
        shear=(-8,8)#  ปรับเปลี่ยนการเอียงของภาพ
    )
], random_order=True) # ทำให้การ augmentation ถูกสุ่มลำดับ ซึ่งจะช่วยเพิ่มความหลากหลายให้กับข้อมูลที่ถูก augment อีกด้วย
```

การสร้างโมเดล augmentation โดยใช้ Sequential() จากแพ็กเกจ imgaug.augmenters โดยมีขั้นตอนการ augment แบบต่างๆนำมารวมกันใน Sequential() และกำหนด random_order=True เพื่อให้การ augment ถูกสุ่มลำดับ ซึ่งจะช่วยเพิ่มความหลากหลายให้กับข้อมูลที่ถูก augment อีกด้วย

```
if AUGMENTATION:
    (aug_images, aug_labels) = augment_add(images, seq, labels)# เรียกใช้ฟังก์ชัน augment_add และนำค่าจากฟังก์ชัน ให้กับตัวแปร aug_images และ aug_labels ตามลำดับ
    images = np.concatenate([images, aug_images])#เพื่อเพิ่มข้อมูลการ augmentation เข้าไปใน images เพื่อใช้ในการฝึกโมเดล
    labels = np.concatenate([labels, aug_labels])#เพื่อเพิ่มข้อมูลการ augmentation เข้าไปใน labels เพื่อใช้ในการฝึกโมเดล
```

เมื่อตัวแปร AUGMENTATION เป็น True จะมีการเพิ่มข้อมูลการ augmentation เข้าไปในข้อมูลภาพและป้ายกำกับของ images และ labels ซึ่งจะช่วยเพิ่มประสิทธิภาพในการฝึกโมเดล

```
if AUGMENTATION:
    _, count = np.unique(labels, return_counts=True) #หาจำนวนของแต่ละคลาสที่มีใน labels เพื่อหาค่าที่ไม่ซ้ำกันใน labels และเก็บผลลัพธ์ไว้ในตัวแปร _ และ count
 #พล็อตกราฟแผนภูมิวงกลมของข้อมูลจำนวนของแต่ละคลาส
    plt.pie(count,
            explode=(0,0),
            labels=class_names,#กำหนดให้ class_names เป็น labels
            autopct="%1.2f%%")#กำหนดรูปแบบของข้อความในแผนภูมิ เพื่อแสดงเป็นเปอร์เซ็นต์

    plt.axis('equal');  #เพื่อให้กราฟแผนภูมิวงกลมมีสัดส่วนที่ถูกต้อง (หรือเท่ากัน) และไม่เบียดเบียน
```

แสดงสัดส่วนของข้อมูลแต่ละคลาสในชุดข้อมูล ซึ่งแบ่งออกเป็นชื่อคลาสที่แสดงบนแผนภูมิ และเปอร์เซ็นต์ของจำนวนข้อมูลในแต่ละคลาสต่อจำนวนข้อมูลทั้งหมดที่มีในชุดข้อมูล การใช้แผนภูมิวงกลมช่วยให้เราเห็นภาพรวมของการกระจายของข้อมูลและสัดส่วนของแต่ละคลาสในชุดข้อมูลได้อย่างชัดเจนและกระชับ

```
labels = to_categorical(labels)# แปลงข้อมูล (labels) จากรูปแบบของตัวเลขเป็นรูปแบบ One-Hot Encoding
# การทำ One Hot Encoding บนตัวแปร labels numpy array เป็นกระบวนการที่ใช้ในการแปลงข้อมูลที่มีลักษณะแบบหมวดหมู่
```

รแปลงข้อมูล labels จากรูปแบบของตัวเลขเป็นรูปแบบ One-Hot Encoding โดยทำการแปลงข้อมูล labels จากรูปแบบของตัวเลข (ที่แทนคลาสแต่ละคลาสด้วยตัวเลขเพียงอย่างเดียว) เป็นรูปแบบ One-Hot Encoding ที่เป็นเวกเตอร์แบบ binary ที่มีค่า 0 หรือ 1 โดยแทนแต่ละคลาสด้วยเวกเตอร์ที่มีค่า 1 ที่ตำแหน่งที่เป็นคลาสนั้น และค่า 0 ที่ตำแหน่งที่เป็นคลาสอื่น ๆ

Step 4 นายสถาพร สัตยชิติ

Step 5 Training Validation and Testing

การพัฒนาแบบจำลอง การฝึก (Training), การตรวจสอบความแม่นยำ (Validation), และการทดสอบ (Testing)

```
[18] np.random.seed(42) #กำหนดค่า seed ให้กับเจเนอเรเตอร์ของตัวสร้างเลขสุ่มของ NumPy
     np.random.shuffle(images) # เพื่อสับเรียงข้อมูลในตัวแปร images และ labels

     np.random.seed(42)
     np.random.shuffle(labels)
     # เพื่อเปรียบเทียบผลลัพธ์
```

การเตรียมข้อมูลก่อนการฝึกโมเดล โดยการสับเรียงข้อมูลชุดฝึก (images) และป้ายชื่อ (labels) เป็นขั้นตอนที่สำคัญในการเตรียมข้อมูลสำหรับการฝึกโมเดลแบบจำลอง

```
total_count = len(images) #กำหนดค่าของตัวแปร total_count ให้เท่ากับจำนวนข้อมูลทั้งหมดใน images โดยใช้ฟังก์ชัน len() เพื่อนับจำนวนสมาชิกใน images.
total_count

train = int(0.7*total_count) #คำนวณขนาดของชุดฝึก (train), ชุดตรวจสอบ (val), และชุดทดสอบ (test) โดยใช้สัดส่วนที่กำหนดไว้ (70%, 20%, 10%).
val = int(0.2*total_count)
test = int(0.1*total_count)

train_images, train_labels = images[:train], labels[:train] #แบ่งข้อมูลเป็นชุดต่างๆ โดยใช้การ slice ข้อมูลจาก images และ labels
val_images, val_labels = images[train:(val+train)], labels[train:(val+train)]
test_images, test_labels = images[-test:], labels[-test:]

train_images.shape, val_images.shape, test_images.shape
```

เตรียมข้อมูลสำหรับการฝึกและทดสอบโมเดลการเรียนรู้ของเครื่อง (Machine Learning Model)

```
if not AUGMENTATION:
    count_labels = train_labels.sum(axis=0) # นับจำนวนข้อมูลที่มีในแต่ละคลาสของชุดข้อมูลฝึก (train_labels.sum(axis=0)).

    classTotals = train_labels.sum(axis=0) # การคำนวณค่าน้ำหนัก (class weights) สำหรับแต่ละคลาสในชุดข้อมูลฝึก (train_labels) เพื่อใช้ในการปรับความสำคัญของแต่ละคลาสในกระบวนการฝึกโมเดล
    classWeight = {}

    for i in range(0,len(classTotals)):
        classWeight[i] = classTotals.max()/classTotals[i] # คำนวณค่าน้ำหนักของแต่ละคลาสโดยหารจำนวนมากสุดของข้อมูลในคลาสด้วยจำนวนข้อมูลในแต่ละคลาส classTotals.max()/classTotals[i]
        pass
    print(classWeight)
```

การปรับความสำคัญของแต่ละคลาสในกระบวนการฝึกโมเดล ซึ่งเป็นสิ่งที่สำคัญในกรณีที่ข้อมูลในแต่ละคลาสมีจำนวนไม่สมดุลกัน

```
# ฟังก์ชัน conv_block ด้านบนใช้สำหรับสร้างบล็อกการคอนโวลูชัน (convolutional block) ในโมเดล CNN
def conv_block(X,k,filters,stage,block,s=2): # stage: หมายเลขขั้นของบล็อก (stage) # block: หมายเลขของบล็อก (block) # s: ค่า stride ในการทำคอนโวลูชัน (default เป็น 2)
    # k: ขนาดของ kernel ในการทำคอนโวลูชัน
    conv_base_name = 'conv_' + str(stage)+block+'_branch' #ปรับปรุงค่าในชั้นปรับค่าแบตช์ (BatchNormalization) ด้วย ReLU
    bn_base_name = 'bn_'+str(stage)+block+"_branch" # การเปิดเผย (Activation) ด้วย ReLU

    F1 = filters # filters: จำนวนฟิลเตอร์ (filters) ที่จะใช้ในการคอนโวลูชัน
    # X: ข้อมูลนำเข้า (input data) หรือภาพที่ผ่านการแปลงแล้วจากบล็อกก่อนหน้า
    X = Conv2D(filters=F1, kernel_size=(k,k), strides=(s,s),
               padding='same',name=conv_base_name+'2a')(X)
    X = BatchNormalization(name=bn_base_name+'2a')(X)
    X = Activation('relu')(X)

    return X
    pass
    # ส่วนที่สำคัญของฟังก์ชันนี้คือการสร้างชั้นของคอนโวลูชัน (Conv2D), การปรับปรุงค่าในชั้นปรับค่าแบตช์ (BatchNormalization), และการเปิดเผย (Activation) ด้วย ReLU
```

สร้างชั้นของคอนโวลูชัน (Conv2D), ชั้นปรับค่าแบตช์ (BatchNormalization), และชั้นเปิดเผย (Activation) ด้วย ReLU ซึ่งเป็นชุดการทำงานหลักของบล็อกการคอนโวลูชันในโมเดล CNN โดยส่วนที่สำคัญของฟังก์ชันคือการสร้างชั้นของคอนโวลูชัน (Conv2D), การปรับปรุงค่าในชั้นปรับค่าแบตช์ (BatchNormalization), และการเปิดเผย (Activation) ด้วย ReLU

```
# ฟังก์ชัน basic_model ที่ใช้สร้างโมเดล neural network สำหรับงานประมวลผลภาพ
def basic_model(input_shape,classes):
    # รับขนาดของข้อมูลนำเข้า (input_shape) และจำนวนคลาส (classes) ที่จะแยกแยะ
    X_input = Input(input_shape)
    # สร้างชั้นนำเข้า (Input layer) โดยใช้ค่า input_shape ที่รับเข้ามา
    X = ZeroPadding2D((5,5))(X_input)
    # ใส่ Zero Padding เพื่อเพิ่มพื้นที่ให้กับข้อมูล
    X = Conv2D(16,(3,3),strides=(2,2),name='conv1',padding="same")(X) # เพิ่มชั้นคอนโวลูชัน (Convolutional Layer) และชั้นปรับปรุงค่าในชั้นปรับค่าแบตช์ (Batch Normalization)
    X = BatchNormalization(name='bn_conv1')(X)

    # stage 2
    X = conv_block(X,3,32,2,block='A',s=1)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X) # สร้างชั้นโดยใช้ฟังก์ชัน conv_block ที่กำหนดไว้ โดยมีการใช้ Dropout เพื่อลด Overfitting
```

```
#     Stage 3
    X = conv_block(X,5,32,3,block='A',s=2)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X)

#     Stage 4
    X = conv_block(X,3,64,4,block='A',s=1)
    X = MaxPooling2D((2,2))(X)
    X = Dropout(0.25)(X)

#   Output Layer
    X = Flatten()(X) # สร้างชั้น Flatten เพื่อแปลงข้อมูลให้อยู่ในรูปแบบแถวเดียวกัน
    X = Dense(64)(X) # เพิ่มชั้น Dense และ Dropout ในการป้องกันการเกิด Overfitting
    X = Dropout(0.5)(X)

    X = Dense(128)(X)
    X = Activation("relu")(X) # สร้างชั้น Dense สุดท้ายที่มีจำนวนเท่ากับจำนวนคลาสและใช้ Activation
Function เป็น softmax เพื่อให้ได้ผลลัพธ์ในรูปแบบความน่าจะเป็นสำหรับแต่ละคลาส

    X = Dense(classes,activation="softmax",name="fc"+str(classes))(X)

    model =
Model(inputs=X_input,outputs=X,name='Feature_Extraction_and_FC') # สร้างและคืนค่า
โมเดล neural network ที่ได้ที่มีชื่อว่า 'Feature_Extraction_and_FC'

    return model
    pass
```

การสร้างโมเดล CNN ซึ่งมีการใช้งานชั้นต่าง ๆ เช่น Convolutional Layer, Batch Normalization, MaxPooling Layer, Dropout, และชั้น Dense ในการสร้างโมเดลที่เหมาะสมสำหรับงานประมวลผลภาพ เนื่องจากการประมวลผลภาพมักจะมีการตรวจจับลักษณะเฉพาะของภาพและการจำแนกแยกคลาสต่าง ๆ ดังนั้นโมเดลนี้ถูกออกแบบมาเพื่อให้สามารถจำแนกแยกคลาสของภาพได้อย่างแม่นยำและมีประสิทธิภาพสูงสุด

```
model = basic_model(input_shape=(48,48,3),classes=2) # เรียกใช้ฟังก์ชัน basic_model
สร้างโมเดล
# input_shape=(48,48,3): รูปร่างของข้อมูลนำเข้า ซึ่งเป็นภาพขนาด 48x48 พิกเซลและมี 3 ช่องสี (RGB)
# classes=2: จำนวนคลาสที่จะแยกแยะ ซึ่งในที่นี้เป็นแบบ binary classification ซึ่งมีคลาสทั้งหมด 2 คลาส
```

รสร้างโมเดล neural network ที่มีโครงสร้างพื้นฐานสำหรับการจำแนกแยกคลาสของภาพในโปรแกรมคลาส 0 และคลาส 1 โดยใช้ภาพขนาด 48x48 พิกเซลและข้อมูลสี RGB และมีจำนวนชั้นและชั้นย่อยต่าง ๆ ที่ถูกออกแบบมาเพื่อให้โมเดลมีประสิทธิภาพในการทำนายผลลัพธ์ของภาพในแต่ละคลาส

```
plot_model(model,to_file='basic_model.png') # ใช้สร้างภาพของโมเดลที่ถูกสร้างขึ้น และบันทึกภาพเอาไว้ในไฟล์ 'basic_model.png
SVG(model_to_dot(model).create(prog='dot',format='svg')) # ใช้สร้างกราฟของโมเดลในรูปแบบของ DOT (Graphviz DOT language)

model.summary()
# เรียกใช้ฟังก์ชัน plot_model และ model_to_dot เพื่อสร้างภาพและ SVG ของโมเดลและแสดงผลสรุปของโมเดลที่สร้างด้วย model.summary()
```

สร้างกราฟของโมเดลในรูปแบบของ DOT (Graphviz DOT language) และแสดงผลสรุปของโมเดลที่สร้างด้วย model.summary() ซึ่งจะแสดงรายละเอียดเกี่ยวกับโครงสร้างของโมเดลรวมถึงจำนวนของพารามิเตอร์และการสรุปของการส่งผ่านข้อมูลผ่านชั้นต่าง ๆ ในโมเดลนั้น ๆ อย่างชัดเจน

Graphviz DOT language เป็นภาษาที่ใช้ในการอธิบายโครงสร้างของกราฟในรูปแบบของข้อความ (text-based format) ซึ่งสามารถนำมาใช้ในการสร้างและแสดงผลกราฟต่าง ๆ ได้อย่างมีประสิทธิภาพ

```
opt = Adam(lr=1e-3) # lr=1e-3 คือการกำหนดอัตราการเรียนรู้ (learning rate) ให้เท่ากับ 0.001
model.compile(optimizer=opt,loss='binary_crossentropy',metrics=['accuracy'
]) # ฟังก์ชัน compile() ซึ่งทำการกำหนดการเรียนรู้ของโมเดล
# จะทำให้โมเดลได้รับการเตรียมพร้อมสำหรับการฝึกและการทดสอบ โดยกำหนดการเรียนรู้, ฟังก์ชันสูญเสียและวัตถุประสงค์ในการวัดประสิทธิภาพ
```

optimizer: กำหนดอัลกอริทึมในการปรับค่าพารามิเตอร์ของโมเดลในระหว่างกระบวนการฝึก (training) optimizer เป็น Adam ซึ่งเป็นวิธีการปรับค่าพารามิเตอร์ที่มีประสิทธิภาพมากในการฝึกโมเดล

loss: กำหนดฟังก์ชันสูญเสีย (loss function) ที่ใช้ในการปรับค่าพารามิเตอร์ของโมเดลในระหว่างการฝึก

metrics: กำหนดวัตถุประสงค์ในการวัดประสิทธิภาพของโมเดล ซึ่งในที่นี้ใช้ 'accuracy' เพื่อให้สามารถติดตามความถูกต้องของการทำนายในขณะทดสอบ (testing) โมเดลได้

```
checkpoint =
ModelCheckpoint("model_weights.h5",monitor='val_accuracy',verbose=1,save_b
est_only=True,mode='max')
# บันทึกโมเดลที่มีประสิทธิภาพที่สุดของเวอร์ชันก่อนหน้า (best model) ตามค่าที่กำหนด (ในที่นี้คือ val_accuracy) ลงในไฟล์ "model_weights.h5"
logs = TensorBoard("logs") # ใช้สร้าง TensorBoard callback ซึ่งเป็นตัวช่วยในการเก็บข้อมูลการฝึกของโมเดล โดยกำหนดให้ข้อมูลนี้ถูกบันทึกไว้ในไดเรกทอรีที่ชื่อว่า "logs"
```

เป็นการกำหนด checkpoint และ TensorBoard callback สำหรับใช้ในการฝึกโมเดล neural network

ใช้สร้าง callbacks สำหรับการฝึกโมเดล TensorBoard callback ทำหน้าที่เก็บข้อมูลการฝึกของโมเดลทุก ๆ เพิ่มเข้าไปในไดเรกทอรีที่กำหนดไว้ ซึ่งข้อมูลเหล่านี้รวมถึงข้อมูลการฝึก

```
epochs = 50
batch_size = 16

history = model.fit(train_images,train_labels,
                    steps_per_epoch=len(train_images)//batch_size,  # กำหนดจำนวนขั้นตอนที่ต้องทำในแต่ละรอบการฝึก
                    epochs=epochs, # กำหนดจำนวนรอบการฝึกโมเดล
                    verbose=1, # ระบุให้แสดงผลลัพธ์ของการฝึกโมเดลทุกๆ รอบ
                    validation_data=(val_images,val_labels), # กำหนดข้อมูลการทดสอบเพื่อให้ความแม่นยำของโมเดลในแต่ละรอบการฝึก
                    validation_steps=len(val_images)//batch_size, #  กำหนดจำนวนขั้นตอนที่ต้องทำในแต่ละรอบการทดสอบ
                    callbacks=[checkpoint, logs] # ระบุ callbacks ที่ใช้ในการบันทึกโมเดลที่ดีที่สุด (checkpoint) และเก็บข้อมูลการฝึกของโมเดลสำหรับ TensorBoard (logs)
#                     class_weight=classWeight # Uncomment if AUGMENTATION is set to FALSE
                    )
# เริ่มต้นการฝึกโมเดล neural network โดยใช้ข้อมูลการฝึก (train_images และ train_labels) และข้อมูลการทดสอบ (val_images และ val_labels)
```

model.fit() เพื่อเริ่มต้นการฝึกโมเดล neural network ด้วยข้อมูลการฝึก (train_images และ train_labels) และข้อมูลการทดสอบ (val_images และ val_labels) ฝึกโมเดลจะทำงานเป็นจำนวนรอบตามที่กำหนดใน epochs โดยจะทำการปรับค่าของพารามิเตอร์ของโมเดลให้มีประสิทธิภาพตามข้อมูลการฝึกและการทดสอบที่ให้มา ในแต่ละรอบการฝึกโมเดล โมเดลจะถูกทดสอบด้วยข้อมูลการทดสอบเพื่อประเมินประสิทธิภาพของมัน และผลลัพธ์จะถูกบันทึกไว้ใน TensorBoard และโมเดลที่มีประสิทธิภาพที่สูงที่สุดในการทดสอบจะถูกบันทึกลงในไฟล์ "model_weights.h5"

```
show_final_history(history) # สร้างกราฟแสดงค่าความแม่นยำของการฝึกและการทดสอบของโมเดล โดยใช้ข้อมูลที่เก็บไว้ในตัวแปร history
```

สร้างกราฟที่แสดงค่าความแม่นยำของการฝึกและการทดสอบของโมเดลในแต่ละรอบ ที่ปรากฏบนกราฟหน้าต่างใหม่ โดยมักจะมีแกน x เป็นรอบการฝึกและแกน y เป็นค่าความแม่นยำ จากนั้นจะแสดงกราฟนี้ให้ผู้ใช้ดูเพื่อประเมินผลลัพธ์ของการฝึกและการทดสอบของโมเดลที่สร้างขึ้น

```
val_pred = model.predict(val_images) # เพื่อทำการทำนายผลลัพธ์ของชุดข้อมูลการทดสอบ val_images จากโมเดลที่ฝึก โดยผลลัพธ์ที่ได้จะเป็นค่าความน่าจะเป็น (probability)
val_pred = np.argmax(val_pred,axis=1) # เพื่อให้ได้เฉพาะค่าดัชนีที่มีความน่าจะเป็นสูงสุดของแต่ละภาพ
val_pred.shape # (จำนวนภาพที่ใช้ในการทดสอบ,) ซึ่งแสดงถึงจำนวนของการทำนายสำหรับแต่ละภาพที่ใช้ในการทดสอบ
```

ทำนายผลลัพธ์ของชุดข้อมูลการทดสอบ val_images จากโมเดลที่ฝึกแล้ว โดยผลลัพธ์ที่ได้จะเป็นค่าความน่าจะเป็น (probability) สำหรับแต่ละคลาสที่มีทั้งหมด 2 คลาส

```
val_actual = np.argmax(val_labels,axis=1) # หาค่าดัชนีที่มีค่าสูงสุดในแต่ละแถวของ val_labels เพื่อให้ได้ค่าคลาสที่ถูกต้องที่สุดสำหรับแต่ละภาพในชุดข้อมูลการทดสอบ val_images.

cnf_mat = confusion_matrix(val_actual, val_pred) # สร้างเมทริกซ์การสับเรียงโดยใช้ค่าคลาสจริง (val_actual) และค่าคลาสที่ทำนายได้ (val_pred) จากโมเดล โดยใช้ฟังก์ชัน confusion_matrix
np.set_printoptions(precision=2) # กำหนดค่า precision ของเลขทศนิยมในเมทริกซ์การสับเรียงเป็น 2 หลักทศนิยม

plt.figure() # สร้างภาพใหม่สำหรับแสดงผล
plot_confusion_matrix(cnf_mat,classes=class_names) # เรียกใช้ฟังก์ชัน plot_confusion_matrix เพื่อแสดงผลของเมทริกซ์การสับเรียง โดยส่งค่าเมทริกซ์ cnf_mat และชื่อคลาส class_names เข้าไป
plt.grid(None) # ไม่แสดงเส้นกริดในกราฟ
plt.show(); # แสดงกราฟ confusion matrix บนหน้าต่างแสดงผลลัพธ์
```

แสดงผลของเมทริกซ์การสับเรียงผ่านกราฟที่เรียกว่า confusion matrix จะทำให้ผู้ใช้สามารถวิเคราะห์และทำความเข้าใจได้ง่ายขึ้นถึงความแม่นยำและประสิทธิภาพของโมเดลในการทำนายแต่ละคลาส

```
test_pred = model.predict(test_images)
test_pred = np.argmax(test_pred,axis=1)
test_pred.shape
# หาค่าที่มีความน่าจะเป็นสูงสุดในแต่ละแถวของเมทริกซ์ โดยที่แต่ละแถวนั้นแทนด้วยการทำนายของโมเดลสำหรับแต่ละภาพในชุดข้อมูลทดสอบ
```

การใช้โมเดลที่ฝึกมาเพื่อทำนายผลลัพธ์สำหรับชุดข้อมูลทดสอบ test_images ซึ่งส่งออกมาเป็นค่าความน่าจะเป็นสูงสุดในแต่ละแถวของเมทริกซ์ผลลัพธ์ test_pred โดยใช้ฟังก์ชัน argmax ซึ่งจะคืนค่าดัชนีที่มีค่าสูงสุดในแต่ละแถวของเมทริกซ์

```
test_actual = np.argmax(test_labels,axis=1) # เพื่อหาค่าคลาสที่เป็นไปได้ที่สูงสุด

cnf_mat_test = confusion_matrix(test_actual, test_pred) # สร้างเมทริกซ์การสับเรียงโดยใช้ค่าคลาสจริง (test_actual) และค่าคลาสที่ทำนายได้ (test_pred) จากโมเดล โดยใช้ฟังก์ชัน confusion_matrix
np.set_printoptions(precision=2) # กำหนดค่า precision ของเลขทศนิยมในเมทริกซ์การสับเรียงเป็น 2 หลักทศนิยม

plt.figure() # เริ่มต้นสร้างภาพใหม่สำหรับแสดงผล
plot_confusion_matrix(cnf_mat_test,classes=class_names) # เรียกใช้ฟังก์ชัน plot_confusion_matrix เพื่อแสดงผลของเมทริกซ์การสับเรียง โดยส่งค่าเมทริกซ์ cnf_mat_test และชื่อคลาส class_names เข้าไป
plt.grid(None) # ไม่แสดงเส้นกริดในกราฟ
plt.show() # แสดงกราฟ confusion matrix บนหน้าต่างแสดงผลลัพธ์
```

กราฟ confusion matrix ที่แสดงการเปรียบเทียบระหว่างคลาสที่ทำนายได้และคลาสที่เป็นจริง โดยแกน x แทนคลาสที่ทำนายได้ และแกน y แทนคลาสที่เป็นจริง ค่าที่ปรากฏในเซลล์ของแต่ละแถวและคอลัมน์แสดงจำนวนของข้อมูลที่มีการจำแนกให้ถูกต้องและไม่ถูกต้อง โดยหากโมเดลทำงานอย่างเหมาะสม เมทริกซ์นี้ควรจะมีค่าสูงในเส้นทแยงมุม (diagonal) ของเมทริกซ์ เนื่องจากเป็นการทำนายคลาสที่ตรงกับคลาสที่เป็นจริงที่สุด

```
rnd_idx = random.sample(range(len(test_images)), 10)
# สร้างลิสต์ของดัชนีที่ถูกสุ่มจากชุดข้อมูลการทดสอบ test_images จำนวน 10 ตัว โดยใช้ฟังก์ชัน random.sample() โดยระบุรายการที่สุ่มได้จาก range(len(test_images))

class_labels = {i:class_name for (class_name,i) in
class_name_labels.items()}
class_labels # เพื่อทำการวนลูปผ่านคู่ของคลาสและดัชนีของคลาสแต่ละคู่

# fig, ax = plt.subplots(2,5,figsize=(5,5))

for i,idx in enumerate(rnd_idx): #วนลูปผ่านดัชนีที่ถูกสุ่มไว้ โดยใช้ enumerate() เพื่อเก็บค่าดัชนีและค่าสุ่มที่ได้

    plt.imshow(test_images[idx]) # แสดงรูปภาพที่อยู่ในดัชนีที่ถูกสุ่มโดยใช้ plt.imshow()
    plt.title("Actual: {}\nPredicted:
{}".format(class_labels[test_actual[idx]],class_labels[test_pred[idx]])) # กำหนดหัวข้อของรูปภาพด้วยข้อความที่ระบุคลาสจริงและคลาสที่ทำนายได้
    plt.grid(None) # ไม่แสดงเส้นกริดบนรูปภาพ
    plt.show() # แสดงรูปภาพที่มีคำอธิบายบนหัวข้อ
    pass
```

ทำการสุ่มดัชนีจากชุดข้อมูลการทดสอบ test_images จำนวน 10 ตัวโดยใช้ฟังก์ชัน random.sample() ซึ่งจะสุ่มดัชนีออกมาจากช่วงตัวเลขตั้งแต่ 0 ถึงความยาวของ test_images แล้วเก็บไว้ในตัวแปร rnd_idx.

```
model.save("ship-model.h5")
#  ใช้เพื่อบันทึกโมเดลที่ฝึกสอนไว้ในไฟล์ชื่อ "ship-model.h5"  โดยใช้เมธอด save()
```

Step 5 นายพีระเมศร์ จุกกษัตริย์

Step 2 นำ Model จาก Part1 มาใช้

```
!pip install opencv-python-headless
!pip install opencv-contrib-python-headless
```

```
!pip install opencv-python
```

```
!pip3 install imutils
```

```
import numpy as np
import pandas as pd
import cv2, os, re
import matplotlib.pyplot as plt
import imutils

from tqdm import tqdm
from imutils.object_detection import non_max_suppression
```

```
from tensorflow.keras.models import load_model
```

```
def find_regions(image, method):

    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    ss.setBaseImage(image)

    if method == 'fast':
        ss.switchToSelectiveSearchFast()
    else:
        ss.switchToSelectiveSearchQuality()

    rects = ss.process()
    boxes = []
    for (x,y,w,h) in rects:

        boxes.append([x,y,w,h])
        pass

    return boxes
    pass
```

ฟังก์ชัน find_regions ใช้เทคนิคการตรวจจับวัตถุแบบสร้างป้ายอัตโนมัติ (selective search) เพื่อค้นหาพื้นที่ที่เป็นไปได้ที่มีวัตถุในภาพที่กำหนด (image) โดยมีการใช้คลาส cv2.ximgproc.segmentation.createSelectiveSearchSegmentation() เพื่อสร้างวัตถุในภาพ และกำหนดภาพฐานด้วย ss.setBaseImage(image) ซึ่งจะเป็นการตั้งค่าภาพที่จะนำไปใช้ใน selective search ต่อไป

หลังจากนั้นฟังก์ชันจะเลือกวิธีการของ selective search ตามที่กำหนดในพารามิเตอร์ method ซึ่งสามารถเป็น 'fast' หรือวิธีการอื่น ๆ ที่กำหนดไว้ในฟังก์ชัน และทำการประมวลผล selective search เพื่อค้นหาพื้นที่ที่พบวัตถุ ซึ่งจะถูกจัดเก็บไว้ในรายการ boxes ซึ่งจะถูกส่งคืนเป็นผลลัพธ์ของฟังก์ชัน

ในส่วนท้ายของฟังก์ชัน จะมีการส่งคืนรายการของพิกัดของสี่เหลี่ยมที่พบวัตถุในภาพออกไปให้กับโปรแกรมที่เรียกใช้งาน

```
scene_path = '/content/drive/MyDrive/AI/scenes/scenes'

def load_scenes():

    scenes = []
```

```
    dirfiles = os.listdir(scene_path)
    dirfiles = sorted(dirfiles)
    for file in dirfiles:

        print(os.path.join(scene_path, file))
        scene = cv2.imread(os.path.join(scene_path, file))
        scene = cv2.cvtColor(scene, cv2.COLOR_BGR2RGB)
        scenes.append(scene)
        pass

    return scenes
    pass
```

โค้ดด้านบนเป็นฟังก์ชัน `load_scenes` ที่ใช้ในการโหลดภาพจากโฟลเดอร์ที่กำหนดไว้ใน `scene_path` และนำมาเก็บไว้ในรายการ `scenes` โดยดำเนินการดังนี้:

1. สร้างรายการ `scenes` เปล่าขึ้นมา
2. ดึงรายชื่อของไฟล์ทั้งหมดในโฟลเดอร์ `scene_path` และเรียงลำดับตามตัวอักษร
3. วนลูปผ่านรายชื่อของไฟล์แต่ละตัว
    - โหลดภาพด้วย OpenCV (`cv2.imread`)
    - แปลงภาพให้อยู่ในรูปแบบของสี RGB โดยใช้ `cv2.cvtColor`
    - เพิ่มภาพลงในรายการ `scenes`
4. ส่งคืนรายการ `scenes` ที่มีภาพทั้งหมดที่โหลดมา

```
scenes = load_scenes()
```

โค้ด `scenes = load_scenes()` โหลดรูปภาพจากโฟลเดอร์ที่ระบุไว้ใน `scene_path` และเก็บภาพในตัวแปร `scenes` เพื่อใช้งานต่อไปได้ โดยการทำงานของฟังก์ชัน `load_scenes()` คือ:

1. สร้างรายการเปล่าสำหรับเก็บรูปภาพที่โหลด
2. ดึงรายชื่อของไฟล์ทั้งหมดในโฟลเดอร์ที่กำหนด
3. โหลดและแปลงภาพในรูปแบบ RGB จากที่ได้รับมา
4. เพิ่มรูปภาพลงในรายการ
5. ส่งคืนรายการรูปภาพทั้งหมดที่โหลดมาในฟังก์ชัน

ดังนั้นตัวแปร `scenes` จะมีรูปภาพทั้งหมดที่โหลดมาจากโฟลเดอร์ที่กำหนดไว้ใน `scene_path` ที่ใช้ในการประมวลผลต่อไปได้

```
%%time
method = "fast"

boxes = []

for scene in scenes:

    box_in_scene = find_regions(scene, method)
    boxes.append(box_in_scene)
    pass
```

โค้ดด้านบนใช้วิธีการ selective search เพื่อค้นหาพื้นที่ที่พบวัตถุในภาพทั้งหมดที่โหลดมา โดยใช้วิธีการ fast selective search และเก็บพื้นที่ที่พบวัตถุลงในรายการ boxes สำหรับแต่ละภาพใน scenes โดยใช้วิธีการ fast selective search และส่งคืนรายการ boxes ที่พบในแต่ละภาพ

```
model = load_model('/content/drive/MyDrive/AI/model_weights.h5')

model.summary()
```

```
%%time

rois, locs = [], []
images = []
for i,scene in tqdm(enumerate(scenes)):

    (H, W) = scene.shape[:2]
    region, loc = [],[]
    for (x,y,w,h) in boxes[i]:

        if w/float(W) > 0.10 and h/float(H) > 0.10:
            continue

        roi = scene[y:y+h,x:x+w]
        roi = cv2.cvtColor(roi,cv2.COLOR_BGR2RGB)
        roi = cv2.resize(roi, (48,48))

        rois.append(roi)
        locs.append((x,y,x+w,y+h))
        pass

    preds = model.predict(np.array(rois,dtype=np.float32))
    preds = np.argmax(preds, axis=1)
```

```
    img = scene.copy()
    for (i,label) in enumerate(preds):

        if label == 1:
            (startX,startY,endX,endY) = locs[i]
            cv2.rectangle(img,(startX,startY),(endX,endY),(0,255,0),2)
        pass

    images.append(img)
    del rois[:]
    del locs[:]
    pass
```

โค้ดด้านบนทำการประมวลผลทุกภาพใน `scenes` โดยดำเนินการดังนี้:

1. วนลูปผ่านทุกภาพใน `scenes` โดยใช้ `tqdm` เพื่อแสดงแถบความคืบหน้า
2. สร้างตัวแปร `H` และ `W` เพื่อเก็บความสูงและความกว้างของภาพ
3. วนลูปผ่านพื้นที่ที่พบวัตถุในแต่ละภาพ (ที่ถูกเก็บไว้ใน `boxes[i]`)
    - ตรวจสอบว่าขนาดของพื้นที่ที่พบวัตถุมีขนาดเล็กกว่า 10% ของความสูงหรือความกว้างของภาพหรือไม่ ถ้าใช่ก็ข้ามขั้นตอนนี้
    - คัดลอกพื้นที่ที่พบวัตถุ (ROI) จากภาพเพื่อนำมาใช้ในการทำนาย
    - เพิ่ม ROI และพื้นที่ที่พบวัตถุ (ในรูปแบบของ bounding box) เข้าไปในรายการ `rois` และ `locs` ตามลำดับ
4. ทำการทำนายบน ROIs โดยใช้โมเดล
5. สร้างภาพที่มีการวาดกรอบรอบพื้นที่ที่พบวัตถุบนภาพด้วย OpenCV
6. เพิ่มภาพที่ผ่านการประมวลผลลงในรายการ `images`

โดยสรุปคือโค้ดนี้ทำการตรวจจับวัตถุในแต่ละภาพที่โหลดมา และสร้างภาพที่มีการวาดกรอบรอบวัตถุที่ตรวจจับได้ เพื่อให้เห็นการตรวจจับวัตถุบนภาพได้ชัดเจน

```
for image in images:

    plt.imshow(image)
    plt.show();
```

โค้ดนี้ใช้วนลูปผ่านภาพที่ผ่านการประมวลผลแล้ว (ภาพที่มีกรอบรอบวัตถุที่ตรวจจับได้) และแสดงภาพในหน้าต่างกราฟิกโดยใช้ `plt.imshow(image)` และ `plt.show()` โดยทำให้สามารถดูภาพที่ผ่านการประมวลผลแล้วได้แบบต่อเนื่องโดยไม่ต้องเปิดภาพเข้ามาดูทีละภาพในโปรแกรมต่างๆ โดยใช้งานไลบรารี `matplotlib.pyplot`

Part 2 นายณัฐพริษฐ์ สมบูรณ์

Reference

Ship Detection using Faster R-CNN: Part 1

- https://www.kaggle.com/code/apollo2506/ship-detection-using-faster-r-cnn-part-1

Ship Detection using Faster R-CNN: Part 2

- https://www.kaggle.com/code/apollo2506/ship-detection-using-faster-r-cnn-part-2