交通號誌辨識

林庠葳
資訊工程系

摘要

1.前言

在目前有許多智能車會使用到交通號誌的自動辨試, 雖然此技術相對成熟, 速度與準確度都已經可以商業化了, 我想以此做練習並找出其中的技術重點。

2.背景

使用R-CNN技術進行物體檢測, 首先通過selective search選擇2000個候選區域, 這些候選區域中有我們所需要對應物體的bounding-box, 對於每一個region proposal都wrap到固定大小的 scale(以AlexNet input size為例: 227x227), 對於每一個處理過後的圖片, 都把它放到CNN模型進行特徵提取, 得到每個region proposal的feature map, 這些特徵用固定長度的特徵集合feature vector來表示。

對於每一個類別, 都會得到很多的feature vector, 把這些特徵向量直接放到svm分類器去判斷當前region所對應的實物是background還是所對應的物體類別, 每個region都會給出所對應的score, 針對這些score, 選出數值較大的score, 再用非極大值抑制(canny NMS)來進行邊緣檢測, 最後就會得到所對應的bounding-box。

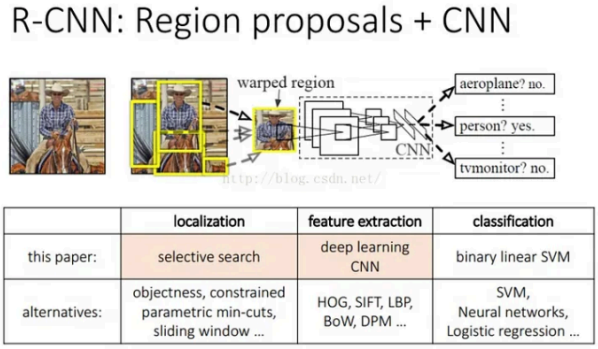
算法分為4個步驟:

1.候選區域生成:一張圖像生成多個候選區域(採用Selective Search方法)

2. 特徵提取:對每個候選區域, 使用深度捲積網路提取特徵(CNN)

3. 類別判斷:特徵送入每一類的SVM分類器, 判別是否屬於該類

4. 位置精修:使用回歸器精細修正候選框位置



	localization	feature extraction	classification
this paper:	selective search	deep learning CNN	binary linear SVM
alternatives:	objectness, constrained parametric min-cuts, sliding window ...	HOG, SIFT, LBP, BoW, DPM ...	SVM, Neural networks, Logistic regression ...

圖1. R-CNN架構圖

候選區域生成:

使用了Selective Search方法從一張圖像生成多個候選區域。基本思路如下:

輸入:彩色圖片

輸出:物體可能位置(bounding box)

首先, 將圖片初始化為很多區域

初始化一個相似集合為空集:

計算所有相鄰區域之間的相似度, 放入集合S中, 集合S保存的其實是一 個區域對以及他們之間的相似度。

找出S中相似度最高的區域對將它們合併, 並從S中刪除與它們相關的所有相似度和區域對。重新計算這個新區域與周圍區域的相似度, 放入集合S中, 並將這個新合併的區域放入集合R中, 重複這個步驟直到S為空。

從R中找出所有區域的bounding box, 這個box可能就是物體可能的區域。

相似度計算方法:

相似度計算方法將直接影響合併區域的順序, 進而影響到檢設結果的好壞。在Selective Search的論文中比較了八種顏色空間的特點, 在實際操作中, 只選擇一個顏色空間(ex:RGB空間)進行計算。作者將相似度度量公式分為四個公式, 稱為互補相似度測量(Complementary Similarity Measures)。這四個公式的值都會被歸一化到區間[0–1]內。

1.顏色相似度S{color} (r_i, r□):

顏色是一個很重要區分物體的因素, 論文中將每個region的像素按不同顏色通道統計成直方圖, 其中每個顏色通道的直方圖為25bins(ex:0~255的顏色通道來說, 就是每隔9(255/25=9)個數值統計像素數量)。這樣三個通道可以得到一個75維的直方圖向量 □ᵢ=□ᵢ¹···□ᵢⁿ (其中n=75)。之後我們用L1正規化(絕對值之和)。由直方圖就可以計算兩個區域的顏色相似度:

$$s_{color}(r_i, r_j) = \sum_{k=1}^{n} min(c_i^k, c_j^k)$$

圖2. 兩個區域的顏色相似度公式

這個相似度其實就是計算兩個區域直方圖的交集。這個顏色直方圖可以在合併區域的時候, 很方便地傳遞給下一級區域。即它們合併後的區域的直方圖向量為

$$C_t = \frac{size(r_i) * C_i + size(r_j) * C_j}{size(r_i) + size(r_j)}$$

圖3. 顏色相似度直方圖向量

2.紋理相似度S□ₑₓ□ᵤᵣₑ (r_i, r□):

另一個需要考慮的因素是紋理, 即圖像的梯度訊息。論文中對紋理的計算採用了SIFT-like特徵, 該特徵

借鑒了SIFT的計算思路，對每個顏色通道的像素點，沿周圍8個方向計算高斯一階倒數(σ=1)，每個方向統計一個直方圖(bin=10)，這樣一個顏色通道統計得到的直方圖向量為80維，三個通道就是240維:$T_i=t_i^1 \cdots t_i^n(n=240)$。注意這個直方圖要用L1正規化。然後我們按照顏色相似度的計算思路計算兩個區域的紋理相似度:

$$s_{texture}(r_i,r_j)=\sum_{k=1}^{n} min(t_i^k,t_j^k)$$

圖4. 兩個區域的紋理相似度公式

同理，合併區域後紋理直方圖可以很方便地傳遞到下一級區域，計算方法和顏色直方圖一樣。

3.尺寸相似度Size (r_i, $r_□$):

在合併區域的時候，論文優先考慮小區域的合併，這種做法可以在一定程度上保證每次合併的區域面積都比較相似，防止大區域對小區域的逐步蠶食。理由很簡單，我們要均勻地在圖片的每個角落生成不同尺寸的區域，作用相當於exhaustive search中用不同尺寸的矩形掃描圖片。具體的相似度計算公式為:

$$s_{size}(r_i,r_j)=1-\frac{size(r_i)+size(r_j)}{size(im)}$$

圖5. 相似度計算公式

其中size(im)表示原圖片的像素數量。

4.填充相似度S{fill} (r_i, $r_□$):

填充相似度主要用來測量兩個區域之間fit的程度。在給出填充相似度的公式前，我們需要定義一個矩形區域$BB_{i□}$，它表示包含r_i和$r_□$的最小的bounding box。基於此，填充相似度計算公式為:

$$s_{fill}(r_i,r_j)=1-\frac{size(BB_{ij})-size(r_i)-size(r_j)}{size(im)}$$

圖6. 填充相似度計算公式

為了高效的計算$BB_{i□}$，我們可以在計算每個region的時候都保存它們的bounding box的位置，這樣$BB_{i□}$就可以很快地由兩個區域的bounding box 推出來。

5.相似度計算公式:

綜合上面四個子公式，可以得到計算相似度的最終公式:

$$s(r_i,r_j)=a_1 s_{color}(r_i,r_j)+a_2 s_{texture}(r_i,r_j)+a_3 s_{size}(r_i,r_j)+a_4 s_{fill}(r_i,r_j)$$

圖7. 相似度計算公式

其中a_i的取值為0或1，表示某個相似度是否被採納。

候選框搜索階段:

實現方式:當我們輸入一張圖片時，我們要搜索出所有可能是物體的區域，這個採用的方法是傳統文獻的算法:《search for object recognition》，通過這個算法我們搜索出2000個候選框。然後從上面的總流程圖中可以看到，搜出的候選框是矩形的，而且是大小各不相同。然而CNN對輸入圖片的大小是有固定的，如果把搜索到的矩形選框不做處理就扔進CNN中肯定不行。因此對於每個輸入的候選框都需要縮放到固定的大小。下面我們講解要怎麼進行縮放處理，為了簡單起見我們假設下一階段CNN所需要的輸入圖片大小是個正方形圖片227x227。因為我們經過selective search 得到的是矩形框，paper試驗了兩種不同的處理方法。

1.各項異性縮放:方法很簡單，就是不管圖片的長寬比例，全部缩放到CNN輸入的大小227x227，如下图(D)所示。

2.各項同性縮放:因為圖片扭曲後，估計會對後續CNN的訓練精度有影響，於是作者也測試了＂各項同性縮放＂，有兩種方法:

A.在原始圖片中，把bounding box的邊界進行擴展延伸成正方形，然後再 進行裁剪，如果已經延伸到原始圖片的外邊界，那麼就用bounding box中的顏色均值填充，如下圖(B)所示。

B.先把bounding box圖片裁剪出來，然後用固定的背景顏色填充成正方形圖片(背景顏色也是用bounding box的像素顏色均值)，如下圖(C)所示。

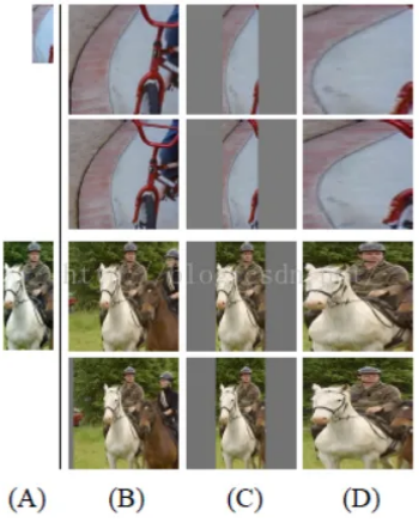


圖8. 測試圖片

3.動機及目的

成功準確辨識交通號誌之狀態

4.執行方法及步驟

使用kaggle Traffic Light Detection Dataset資料集，總共有2600訓練集，把其中20%提取出當驗證集，共有400張測試集。

使用colab編寫程式與線上訓練。

(1)從 json 檔案中提取有關邊界框和燈號的標籤
(2)準備訓練資料集
(3)建立模型
(4)訓練模型

```
with open("/content/train_dataset/train.json") as f:
    data_dict = json.load(f)

#建立一個包含表格資料行的列表
data = []

#循環遍歷註釋清單中的元素
for annotation in data_dict['annotations']:
    #取得一般邊界框信息
    filename = annotation['filename']
    xmin = annotation['bndbox']['xmin']
    ymin = annotation['bndbox']['ymin']
    xmax = annotation['bndbox']['xmax']
    ymax = annotation['bndbox']['ymax']

    if annotation['inbox']:
        for inbox in annotation['inbox']:
            color = inbox['color']
            data.append({
                'filename': filename,
                'xmin': xmin,
                'ymin': ymin,
                'xmax': xmax,
                'ymax': ymax,
                'color': color,
            })

#從資料清單建立 DataFrame 並將其儲存到 CSV 文件
df = pd.DataFrame(data)
df.to_csv('traffic_lights.csv', index=False)
data_tf = pd.read_csv('/content/traffic_lights.csv')
data_tf['filename'] = data_tf['filename'].str.replace('train_images\\\\', '', regex=True)
data_tf.to_csv('traffic_lights_labels.csv', index=False)
df = pd.read_csv('/content/traffic_lights_labels.csv')
df.head()
```

	filename	xmin	ymin	xmax	ymax	color
0	00001.jpg	1026.5	741.8	1077.5	910.9	red
1	00002.jpg	1418.7	1136.8	1428.3	1149.4	red
2	00003.jpg	2574.3	868.7	2632.3	887.1	red
3	00003.jpg	2252.1	899.6	2300.8	918.7	green
4	00004.jpg	833.3	1163.2	843.6	1184.8	green

圖9. 從 json 檔案中提取有關邊界框和燈號的標籤

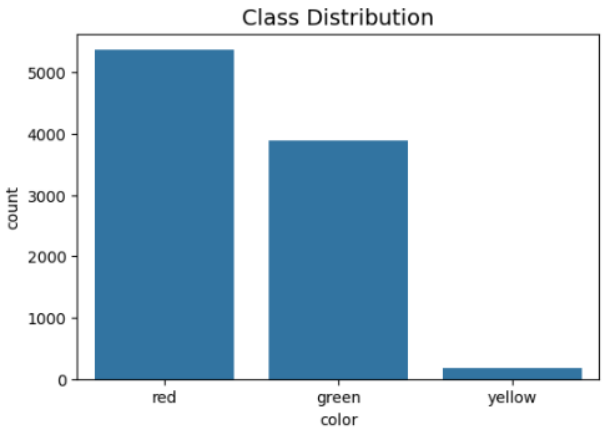


圖10. 總共的紅綠燈數量

```
class TrafficLightDetectionDataset(Dataset):

    def __init__(self, dataframe, image_dir, mode='train', transforms=None, resize_factor=0.25):
        super().__init__()

        self.image_names = dataframe['filename'].unique()
        self.df = dataframe
        self.image_dir = image_dir
        self.transforms = transforms
        self.mode = mode
        self.resize_factor = resize_factor

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, index: int):
        #從 df 中檢索影像名稱及其記錄 (x1、y1、x2、y2、classname)
        image_name = self.image_names[index]
        records = self.df[self.df['filename'] == image_name]

        #讀取圖片
        image_path = f"{self.image_dir}/{image_name}"
        image = cv2.imread(image_path, cv2.IMREAD_COLOR)

        if image is None:
            print(f"Error: Image not found or cannot be read: {image_path}")
            return None, None, image_name

        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        image /= 255.0

        if self.mode == 'train':
            #取得每個框的邊界框座標
            boxes = records[['xmin', 'ymin', 'xmax', 'ymax']].values

            #取得每個邊界框的標籤
            temp_labels = records['color'].values
            labels = [class_to_int[label[0]] for label in temp_labels]

            #將邊界框和標籤轉換為torch張量
            boxes = torch.as_tensor(boxes, dtype=torch.float32)
            labels = torch.as_tensor(labels, dtype=torch.int64)
```

圖11. 建立Dataset讀取class(上)

```
        #調整影像和邊界框的大小
        height, width, _ = image.shape
        new_height, new_width = int(height * self.resize_factor), int(width * self.resize_factor)
        image = cv2.resize(image, (new_width, new_height))
        boxes = boxes * self.resize_factor
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])

        #建立目標
        target = {
            'boxes': boxes,
            'labels': labels,
            'area': torch.as_tensor(area, dtype=torch.float32)
        }

        #轉換
        if self.transforms:
            image = self.transforms(image)

        return image, target, image_name

    elif self.mode == 'test':
        #調整圖片大小
        height, width, _ = image.shape
        new_height, new_width = int(height * self.resize_factor), int(width * self.resize_factor)
        image = cv2.resize(image, (new_width, new_height))

        if self.transforms:
            image = self.transforms(image)

        return image, image_name

    return None, None, image_name
```

圖12. 建立Dataset讀取class(下)

```
#準備訓練和驗證數據

def get_transform():
    return T.Compose([T.ToTensor()])

def collate_fn(batch):
    batch = list(filter(lambda x: x[0] is not None, batch))    #過濾掉無效影像
    return tuple(zip(*batch)) if batch else ([], [], [])


#資料集對象
dataset = TrafficLightDetectionDataset(df, DIR_IMAGES, transforms = get_transform())


#將資料集拆分為訓練集和測試集 - 使用 80% 用於訓練，20% 用於驗證
indices = torch.randperm(len(dataset)).tolist()
train_dataset = torch.utils.data.Subset(dataset, indices[:-490])
valid_dataset = torch.utils.data.Subset(dataset, indices[-490:])


#準備資料載入器
train_data_loader = DataLoader(
    train_dataset,
    batch_size = 2,
    shuffle = True,
    num_workers = 2,
    collate_fn = collate_fn
)


valid_data_loader = DataLoader(
    valid_dataset,
    batch_size = 2,
    shuffle = True,
    num_workers = 2,
    collate_fn = collate_fn
)
```

圖13. 準備訓練和驗證數據

```python
#建立/載入模型

#Faster - RCNN 模型
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=None, weights_backbone=None)
num_classes = len(class_to_int)

#取得分類器的輸入特徵數量
in_features = model.roi_heads.box_predictor.cls_score.in_features

#將預先訓練好的頭部更換為新頭部資料
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

#準備訓練模型
#從模型中檢索所有可訓練參數（用於最佳化器）
params = [p for p in model.parameters() if p.requires_grad]
#定義優化器
optimizer = torch.optim.Adam(params, lr = 0.0001)
#LR
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=2, gamma=0.1)
model.to(device)
#epochs
epochs = 5
```

圖14. 建立/載入模型

```python
itr = 1
total_train_loss = []
for epoch in range(epochs):
        start_time = time.time()
        train_loss = []
        for images, targets, image_names in tqdm(train_data_loader):
                if not images:
                        print("No valid images for this batch.")
                        continue

                #在colab上載入圖像和目標
                images = [image.to(device) for image in images]
                targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

                #前向傳播
                out = model(images, targets)
                losses = sum(loss for loss in out.values())

                #重置漸變
                optimizer.zero_grad()

                #反向傳播
                losses.backward()
                torch.nn.utils.clip_grad_norm_(model.parameters(), 1)
                optimizer.step()

                #平均損失
                loss_value = losses.item()
                train_loss.append(loss_value)

                if itr % 300 == 0:
                        print(f"\n Iteration #{itr} loss: {loss_value:.4f} \n")
                itr += 1
        lr_scheduler.step()

        epoch_train_loss = np.mean(train_loss)
        total_train_loss.append(epoch_train_loss)
        print(f'Epoch: {epoch+1}')
        print(f'Epoch train loss is {epoch_train_loss:.4f}')

        time_elapsed = time.time() - start_time
        print("Time elapsed: ", time_elapsed)

        torch.save({
                        'epoch': epoch,
                        'model_state_dict': model.state_dict(),
                        'optimizer_state_dict': optimizer.state_dict(),
                        'loss': epoch_train_loss
                        }, "checkpoint.pth")
```

圖15. 設定訓練方式

```
 30%|███       | 300/990 [03:25<07:51,  1.46it/s]
 Iteration #300 loss: 0.7204

 61%|██████    | 600/990 [06:56<04:36,  1.41it/s]
 Iteration #600 loss: 0.8463

 91%|█████████ | 900/990 [10:27<01:03,  1.41it/s]
 Iteration #900 loss: 0.4704

100%|██████████| 990/990 [11:30<00:00,  1.43it/s]
Epoch: 1
Epoch train loss is 0.6766
Time elapsed:  690.3060350418091
 21%|██        | 210/990 [02:26<09:17,  1.40it/s]
 Iteration #1200 loss: 0.6791

 52%|█████▏    | 510/990 [05:56<05:44,  1.40it/s]
 Iteration #1500 loss: 0.2399

 82%|████████▏ | 810/990 [09:26<02:04,  1.44it/s]
 Iteration #1800 loss: 0.4944

100%|██████████| 990/990 [11:32<00:00,  1.43it/s]
Epoch: 2
Epoch train loss is 0.5087
Time elapsed:  692.2469909191132
 12%|█▏        | 120/990 [01:23<10:25,  1.39it/s]
 Iteration #2100 loss: 0.3669

 42%|████▏     | 420/990 [04:53<06:38,  1.43it/s]
 Iteration #2400 loss: 0.5122

 73%|███████▎  | 720/990 [08:23<03:05,  1.45it/s]
 Iteration #2700 loss: 0.4188

100%|██████████| 990/990 [11:33<00:00,  1.43it/s]
Epoch: 3
Epoch train loss is 0.4163
Time elapsed:  693.3164749145508
  3%|▎         | 30/990 [00:21<10:52,  1.47it/s]
 Iteration #3000 loss: 0.3013

 33%|███▎      | 330/990 [03:50<07:34,  1.45it/s]
 Iteration #3300 loss: 0.1883

 64%|██████▎   | 630/990 [07:20<04:16,  1.40it/s]
 Iteration #3600 loss: 0.3078

 94%|█████████▍| 930/990 [10:51<00:41,  1.43it/s]
 Iteration #3900 loss: 0.4007

100%|██████████| 990/990 [11:33<00:00,  1.43it/s]
```

圖15. 訓練

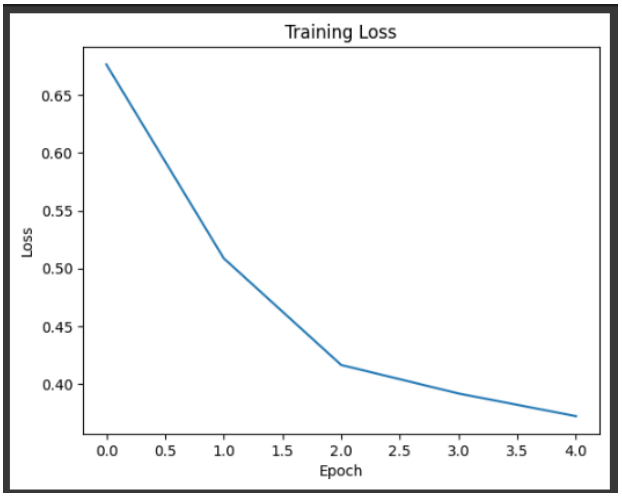


圖16. 訓練loss率

```
itr = 1
v_loss = []

start_time = time.time()

for images, targets, image_names in tqdm(valid_data_loader):

        #在colab上載入圖像和目標
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        #前向傳播
        out = model(images, targets)
        losses = sum(loss for loss in out.values())

        #平均損失
        loss_value = losses.item()
        v_loss.append(loss_value)

val_loss = np.mean(v_loss)
print(f'Val loss is {val_loss:.4f}')

time_elapsed = time.time() - start_time
print("Time elapsed: ",time_elapsed)
```

```
100%|██████████| 245/245 [01:06<00:00, 3.67it/s]Val loss is 0.4093
Time elapsed:  66.84853315353394
```

圖17. 訓練驗證集

```
submission = pd.DataFrame(columns = ["filename", "xmin", "ymin", "xmax", "ymax", "color"])

#準備訓練數據
images = os.listdir(DIR_IMAGES_TEST)

df_test = df_test.drop_duplicates(subset='filename', keep="first")

#測試資料集
test_dataset = TrafficLightDetectionDataset(df_test, DIR_IMAGES_TEST, mode = 'test',
                                            transforms = get_transform())


#測試資料載入器
test_data_loader = DataLoader(
        test_dataset,
        batch_size=2,
        shuffle=False,
        num_workers=2,
        drop_last=False,
        collate_fn=collate_fn
)
```

圖18. 準備測試集

```
import gc

threshold = 0.7
model.eval()

for images, image_names in test_data_loader:
        for i, image in enumerate(images):
                #前向傳播 -
                image = image.to(device).unsqueeze(0)   # 逐張圖像處理
                output = model(image)

                #將張量轉換為數組
                boxes = output[0]['boxes'].data.cpu().numpy()
                scores = output[0]['scores'].data.cpu().numpy()
                labels = output[0]['labels'].data.cpu().numpy()

                #閾值化
                boxes_th = boxes[scores >= threshold].astype(np.int32)
                scores_th = scores[scores >= threshold]

                # int_to_class - labels
                labels_th = [int_to_class[labels[x]] for x in range(len(labels)) if scores[x] >= threshold]

                #建立暫存 DataFrame 用於批次追加
                temp_df = pd.DataFrame()

                for y in range(len(boxes_th)):
                        # 邊界框、類別名稱和圖片名稱
                        x1 = boxes_th[y][0] * 4
                        y1 = boxes_th[y][1] * 4
                        x2 = boxes_th[y][2] * 4
                        y2 = boxes_th[y][3] * 4
                        class_name = labels_th[y]

                        # 為 df 建立行
                        row = {
                                "filename": image_names[i],
                                "xmin": x1,
                                "xmax": x2,
                                "ymin": y1,
                                "ymax": y2,
                                "color": class_name
                        }

                        # 將新行追加到暫存 DataFrame
                        temp_df = pd.concat([temp_df, pd.DataFrame([row])], ignore_index=True)

                # 將暫存 DataFrame 追加到 submission DataFrame
                submission = pd.concat([submission, temp_df], ignore_index=True)

                # 清理 GPU 內存
                del image
                torch.cuda.empty_cache()
                gc.collect()
```

圖19. 輸出測試集之邊界框、類別名稱

```
[65] def plot_img_test(image_name):

        fig, ax = plt.subplots(2, 1, figsize = (14, 14))
        ax = ax.flatten()

        bbox = df[df['filename'] == image_name]
        img_path = os.path.join(DIR_IMAGES_TEST, image_name)

        image = cv2.imread(img_path, cv2.IMREAD_COLOR)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        image /= 255.0
        image2 = image

        ax[0].set_title('Original Image')
        ax[0].imshow(image)

        for idx, row in bbox.iterrows():
                x1 = row['xmin']
                y1 = row['ymin']
                x2 = row['xmax']
                y2 = row['ymax']
                label = row['color']
                if label == 'red':
                        color_brg = (255,0,0)
                elif label == 'green':
                        color_brg = (0,255,0)
                elif label == 'yellow':
                        color_brg = (0,255,255)

                cv2.rectangle(image2, (int(x1),int(y1)), (int(x2),int(y2)), color_brg, 3)
                font = cv2.FONT_HERSHEY_SIMPLEX
                cv2.putText(image2, label, (int(x1),int(y1-10)), font, 1, color_brg, 2)

        ax[1].set_title('Image with Bondary Box')
        ax[1].imshow(image2)

        plt.show()
```

圖20. 輸出結果圖片

預期成果
正確辨識圖片中紅綠燈位置

圖21. 輸出原圖片

圖22. 輸出預測圖片

參考文獻

[1] Rich feature hierarchies for accurate object detection and semantic segmentation (2014)。
[2] Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition (2014)。
[3] Selective Search for Object Recognition (2012)。