TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Nhóm 1 - Lớp 24TNT1

Project: Automatic License Plate Recognition

ĐỒ ÁN MÔN THỰC HÀNH GIỚI THIỆU NGÀNH TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

Tp. Hồ Chí Minh, tháng 11/2024

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Nhóm 1 - 24TNT1
Project: Automatic License Plate Recognition

Nguyễn Tuấn Lâm - 24122006
Đỗ Lê Phong Phú - 24122010
Quách Lê Nhật Huy - 24122016
Phan Bảo Đức Phát - 24122020
Nguyễn Thanh Tuấn - 24122025
Trần Trọng Tú - 24122024

ĐỒ ÁN MÔN THỰC HÀNH GIỚI THIỆU NGÀNH TRÍ TUỆ NHÂN TẠO
CHƯƠNG TRÌNH CHÍNH QUY

GIÁO VIÊN HƯỚNG DẪN

ThS. Nguyễn Trần Duy Minh
ThS. Phạm Trọng Nghĩa
TS. Lê Ngọc Thành

Tp. Hồ Chí Minh, tháng 11/2024

Lời cảm ơn

Nhóm 1 xin chân thành cảm ơn sự giúp đỡ tận tình của giáo viên hướng dẫn đã hỗ trợ nhóm trong lúc làm đồ án. Sự chỉ dẫn của thầy là thành phần không thể thiếu cho đồ án của nhóm chúng em.

Mục lục

Danh sách hình

Danh sách bảng

Chương 1

Đánh Giá Thành Viên

Member	Đánh giá (Progress Bar)
Lâm	(10/10)
Phú	(10/10)
Huy	(10/10)
Phát	(10/10)
Tuấn	(10/10)
Tú	(10/10)

Bảng 1.1: Đánh thành viên theo tiến độ

Chương 2

Đánh Giá Mức Độ Hoàn Thành Cho Từng Yêu Cầu

Yêu cầu	Đánh Giá (Progress Bar)
Phân công công việc	(10/10)
Lập trình và mô hình A.I	(10/10)
Báo cáo các mục:	(10/10)
Introduction	(10/10)
Methodology	(10/10)
Conclusion	(10/10)
Lưu source code trên Github	(10/10)
Slide	(10/10)
Video	(10/10)

Bảng 2.1: Đánh thành viên theo tiến độ

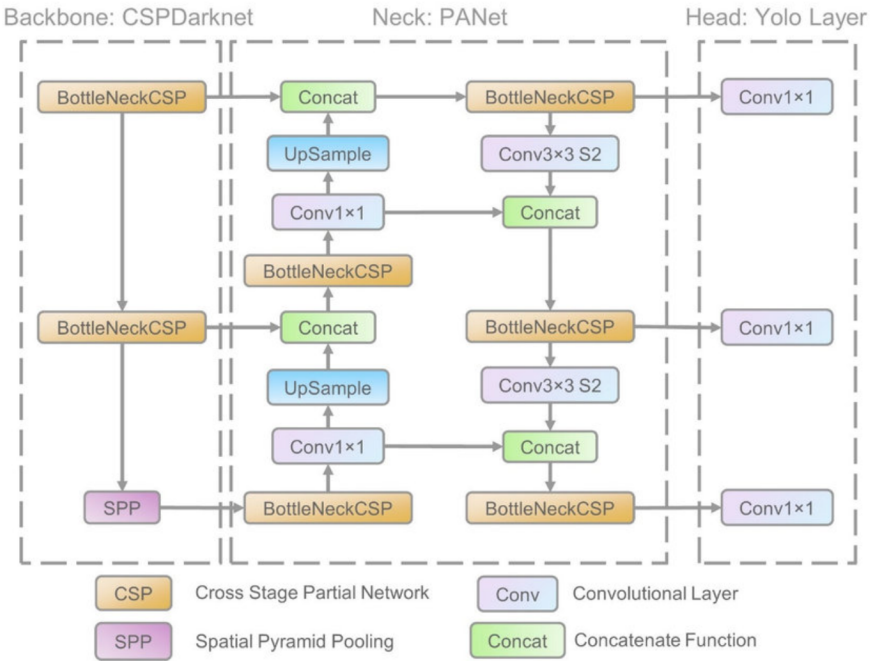
Chương 3

Nội Dung Chính

1. Introduction: Giới thiệu bài toán.

Trong dự án này, chúng ta sẽ xây dựng một ứng dụng web cho việc tự động nhận diện biển số xe bằng ngôn ngữ lập trình Python, sử dụng mô hình YOLOv5. Mục tiêu trọng tâm của dự án là xây dựng một mô hình có khả năng nhận diện và trả về những thông tin liên quan đến vị trí và nội dung biển số từ hình ảnh mà người dùng nhập. Phần lớn nội dung của dự án được xây dựng và chỉnh sửa từ tài liệu hướng dẫn: [1][2].

Link Github của dự án.



Hình 3.1: Sơ đồ nguyên lý mô hình YOLOv5

YOLO, hay "You Only Look Once", là một trong những thuật toán phát hiện đối tượng dựa trên nguyên lý học sâu được sử dụng rộng rãi nhất. YOLO hoạt động bằng cách chia hình ảnh thành hệ thống lưới, và mỗi ô lưới sẽ phát hiện các đối tượng bên trong nó. Thuật toán này có thể

được sử dụng để phát hiện đối tượng theo thời gian thực từ các luồng dữ liệu và yêu cầu rất ít tài nguyên tính toán.

Kiến trúc mạng của YOLOv5 bao gồm ba phần chính: Backbone: CSP-Darknet, Neck: PANet và Head: Yolo Layer. Dữ liệu đầu vào được đưa vào CSPDarknet để trích xuất đặc trưng, sau đó chuyển tới PANet để hợp nhất đặc trưng. Cuối cùng, Yolo Layer sẽ đưa ra các kết quả nhận diện.

2. Methodology: Giải thích phần xử lý dữ liệu, giới thiệu mô hình, giải thích mô hình

Việc xây dựng một web app cho việc tự động nhận diện biển số xe bằng mô hình YOLOv5 sẽ gồm những bước sau:

i. Xác định mục tiêu:

Như đã nói trên, mục tiêu của dự án là xây dựng một ứng dụng web có khả năng tự động nhận diện biển số xe Việt Nam từ hình ảnh do người dùng tải lên. Sau khi nhận diện xong sẽ trả về kết quả nhận diện bao gồm khung đánh dấu biển số xe trên hình ảnh kèm theo trích xuất các ký tự của biển số.

Trước khi đi vào quá trình xây dựng mô hình, ta sẽ cài đặt những thư viện cần thiết, tổ chức các thư mục và cài đặt mô hình YOLOv5.

```
import os
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
import pytesseract as pt
import plotly.express as px
import matplotlib.pyplot as plt
import xml.etree.ElementTree as xet

from glob import glob
from skimage import io
from shutil import copy
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import TensorBoard
from sklearn.model_selection import train_test_split
from tensorflow.keras.applications import InceptionResNetV2
```

```
from tensorflow.keras.layers import Dense, Dropout, Flatten, Input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
git clone https://github.com/ultralytics/yolov5
pip install -qr ./yolov5/requirements.txt comet_ml

from google.colab import drive
drive.mount('/content/drive', force_remount = True)
```

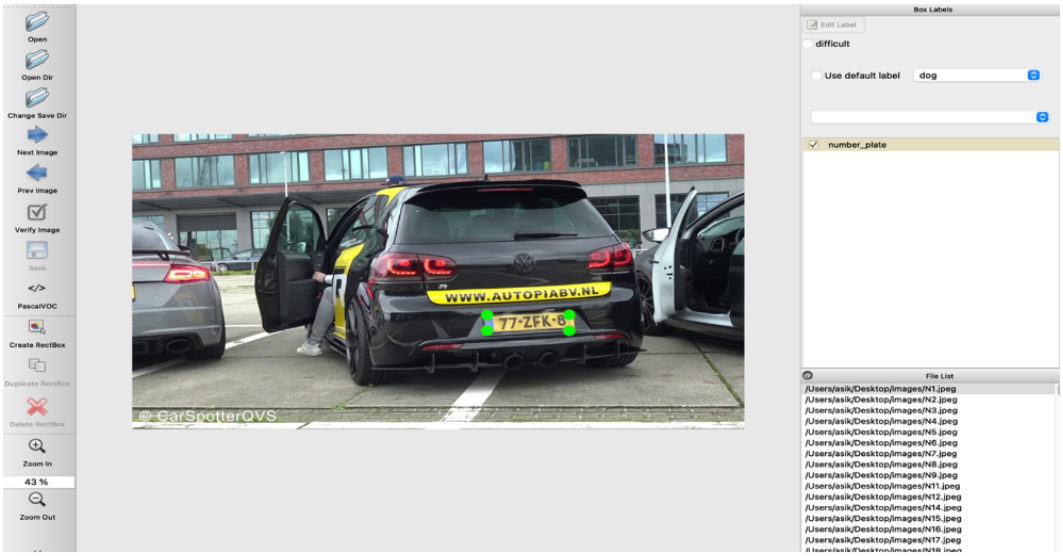
ii. Thu thập dữ liệu:

Để xây dựng và huấn luyện một hệ thống nhận diện biển số xe, cần một lượng lớn hình ảnh các loại biển số xe Việt Nam trong nhiều điều kiện khác nhau (về góc nhìn, ánh sáng, màu sắc, kích thước...). Ban đầu ta sẽ huấn luyện mô hình với 250 hình ảnh, được lưu trong folder *images/*. Sau khi chạy thử mô hình, ta sẽ tiếp tục huấn luyện với 1000 hình ảnh được lấy từ nguồn trên mạng [3] trước khi đưa lên web.

iii. Xử lý dữ liệu:

Sau khi có được dữ liệu, ta thực hiện các bước xử lý trước khi đưa dữ liệu đi huấn luyện mô hình bao gồm:

- **Chú thích dữ liệu:** Đánh dấu vị trí, hay còn được gọi là gắn nhãn cho biển số xe trên mỗi bức ảnh, được thực hiện như sau:
  - Sử dụng Labellmg (image annotation tool) để xác định vị trí của biển số xe bằng **hộp giới hạn** (hay **bounding box**), dữ liệu được xuất ra gồm tọa độ bốn góc của bounding box dưới dạng file .xml và được lưu trong cùng thư mục *images/*. Công việc này được làm hoàn toàn thủ công.



Hình 3.2: Gắn nhãn bằng Labellmg

– Sau khi đã có các file .xml, ta thực hiện trích xuất dữ liệu từ file.

```
path = glob('DataSet/images/*.xml')
labels_dict = dict(filepath=[],xmin=[],xmax=[],ymin=[],ymax=[])
for i in path:
    info = xet.parse(i)
    root = info.getroot()
    member_object = root.find('object')
    labels_info = member_object.find('bndbox')
    xmin = int(labels_info.find('xmin').text)
    xmax = int(labels_info.find('xmax').text)
    ymin = int(labels_info.find('ymin').text)
    ymax = int(labels_info.find('ymax').text)

    labels_dict['filepath'].append(i)
    labels_dict['xmin'].append(xmin)
    labels_dict['xmax'].append(xmax)
    labels_dict['ymin'].append(ymin)
    labels_dict['ymax'].append(ymax)

df = pd.DataFrame(labels_dict)
```

Đoạn mã trên đọc các file .xml chứa thông tin nhãn và trích xuất các giá trị tọa độ của bounding box (xmin, xmas, ymin, ymax) từ từng file, sau đó lưu trữ vào một DataFrame df.

– Từ DataFrame đã có, ta chuyển dữ liệu đã có sang dữ liệu mà mô hình YOLO có thể xử lý được, bao gồm kích thước ảnh, tọa

độ điểm trung tâm và kích thước bounding box như được minh họa trong hình 3.3.

Hình 3.3: Cách chú thích dữ liệu cho mô hình YOLO

Cụ thể như sau:

```
def parsing(path):
    parser = xet.parse(path).getroot()
    name = parser.find('filename').text
    filename = f'DataSet/images/{name}'

    parser_size = parser.find('size')
    width = int(parser_size.find('width').text)
    height = int(parser_size.find('height').text)

    return filename, width, height
df[['filename','width','height']] = df['filepath'].apply(parsing
    ).apply(pd.Series)

df['center_x'] = (df['xmax'] + df['xmin']) / (2 * df['width'])
df['center_y'] = (df['ymax'] + df['ymin']) / (2 * df['height'])

df['bb_width'] = (df['xmax'] - df['xmin']) / df['width']
df['bb_height'] = (df['ymax'] - df['ymin']) / df['height']
```

Đoạn mã này tiếp tục trích xuất thông tin về chiều rộng width và chiều cao height của ảnh, từ đó tính được các giá trị đã được chuẩn hóa theo kích thước ảnh (tức giá trị nằm trong khoảng từ 0 đến 1, dựa theo kích thước ảnh) gồm: tọa độ điểm trung tâm bounding box center_x, center_y và kích thước bounding box bb_width, bb_height.

- **Phân chia dữ liệu:** Khi đã có dữ liệu đã được gắn nhãn, ta thực

hiện tổ chức phân chia dữ liệu thành dữ liệu huấn luyện (dùng để huấn luyện mô hình), lưu trong *yolov5/data_images/test/*, và dữ liệu kiểm tra (dùng để đánh giá sự hiệu quả của mô hình sau khi đã hoàn tất quá trình huấn luyện), lưu trong *yolov5/data_images/train/*. Cụ thể là:

```
import shutil
import os

# Split the data into train and test
df_train = df.iloc[:200]
df_test = df.iloc[200:]

train_folder = 'yolov5/data_images/train'

values = df_train[['filename', 'center_x', 'center_y', 'bb_width', '
    bb_height']].values
for fname, x, y, w, h in values:
    image_name = os.path.split(fname)[-1]
    txt_name = os.path.splitext(image_name)[0]

    dst_image_path = os.path.join(train_folder, image_name)
    dst_label_file = os.path.join(train_folder, txt_name + '.txt')

    # Copy each image into train folder
    shutil.copy(fname, dst_image_path)

    # Generate .txt with label info
    label_txt = f'0 {x} {y} {w} {h}'
    with open(dst_label_file, mode='w') as f:
        f.write(label_txt)

        f.close()

test_folder = 'yolov5/data_images/test'

values = df_test[['filename', 'center_x', 'center_y', 'bb_width', '
    bb_height']].values
for fname, x, y, w, h in values:
    image_name = os.path.split(fname)[-1]
    txt_name = os.path.splitext(image_name)[0]

    dst_image_path = os.path.join(test_folder, image_name)
    dst_label_file = os.path.join(test_folder, txt_name + '.txt')

    # Copy each image into test folder
```

```
    copy(fname, dst_image_path)

    # Generate .txt with label info
    label_txt = f'0 {x} {y} {w} {h}'
    with open(dst_label_file, mode='w') as f:
        f.write(label_txt)

        f.close()
```

Từ dữ liệu DataFrame df đã có, đoạn mã trên phân chia thành dữ liệu huấn luyện df_train và dữ liệu kiểm tra df_test. Từ đó, ta sao chép hình ảnh có sẵn vào thư mục tương ứng và tạo ra file .txt chứa thông tin nhãn cho mỗi ảnh.

Cuối cùng, ta tạo thêm một file *data.yaml* để chú thích đường dẫn đến thư mục huấn luyện và kiểm tra cho mô hình YOLO.

```
train: data_images/train
val: data_images/test
nc: 1
names: [
    'license_plate'
]
```

Dữ liệu giờ đã sẵn sàng cho việc huấn luyện mô hình.

iv. Xây dựng và huấn luyện mô hình:

- **Cài đặt YOLOv5:** Cài đặt các thư viện cần thiết, cấu hình môi trường cho YOLO và cài đặt mô hình YOLOv5. Việc này ta đã làm ở mục i.
- **Huấn luyện mô hình:** Sử dụng những dữ liệu đã chuẩn bị sẵn, tiến hành huấn luyện mô hình nhận diện biển số xe.

```
!python ./yolov5/train.py --data ./data.yaml --cfg ./yolov5/models/
    yolov5s.yaml --batch-size 8 --name Model --epochs 5
```

Các thông số cần quan tâm:

  - **batch-size:** Số lượng ảnh được học qua trước khi mô hình điều chỉnh tham số, giá trị càng nhỏ thì yêu cầu càng nhiều bộ nhớ và độ chính xác càng cao.

- **epochs:** Số lần huấn luyện qua toàn bộ dữ liệu, giá trị càng nhiều thì mô hình học được càng nhiều từ dữ liệu, nhưng quá nhiều có thể dẫn đến hiện tượng **overfitting** (quá khớp với dữ liệu huấn luyện).

- **Lưu mô hình:** Sau khi huấn luyện xong, lưu lại phiên bản mô hình đã được huấn luyện với độ chính xác cao nhất để sử dụng sau này.

```
!python ./yolov5/export.py --weight ./yolov5/runs/train/Model/weights/
    best.pt --include torchscript onnx
```

Lệnh này giúp ta lưu mô hình YOLOv5 ở định dạng TorchScript và ONNX, cho phép ta áp dụng được mô hình trong nhiều môi trường làm việc khác nhau.

v. Chạy mô hình:

Sau khi quá trình huấn luyện hoàn tất, ta đã có được một mô hình AI có khả năng tự động nhận diện biển số xe. Ở phần này ta sẽ tạo một pipeline và chạy thử mô hình đã huấn luyện trên ảnh biển số xe.

- **Tiền xử lý:** Trước khi chạy mô hình trên ảnh, ta cần thực hiện vài cấu hình để đảm bảo việc chạy mô hình được suôn sẻ:

```
# Settings
INPUT_WIDTH =  640
INPUT_HEIGHT = 640
```

Đoạn mã đặt kích thước chuẩn đầu vào cho ảnh là 640x640 pixel, việc này dùng để chuẩn hóa kích thước ảnh trước khi đưa vào mô hình, đảm bảo nhất quán với dữ liệu mà mô hình đã được huấn luyện.

```
# LOAD THE IMAGE
img = io.imread('TEST/TEST2.png')

fig = px.imshow(img)
fig.update_layout(width=700, height=400, margin=dict(l=10, r=10, b=10,
    t=10))
fig.update_xaxes(showticklabels=False).update_yaxes(showticklabels=
    False)
fig.show()
```

Tải mẫu ảnh mà ta sẽ dùng để kiểm tra, ở đây ta sẽ dùng ảnh TEST2.png, đồng thời ẩn nhãn, cập nhật các kích thước hiển thị (không liên quan đến kích thước ảnh đưa vào mô hình) và hiển thị ảnh ra màn hình.

Hình 3.4: Ảnh TEST2.png

Cuối cùng, ta nạp mô hình đã lưu ở bước iv., đồng thời thiết lập backend của mô hình là **OpenCV DNN** và thiết bị sử dụng là **CPU**.

```
# LOAD YOLO MODEL
net = cv2.dnn.readNetFromONNX('./yolov5/runs/train/Model/weights/best.
    onnx')
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
```

- **Dự đoán, lọc và xử lý kết quả:** Ở phần này ta sẽ hoàn thiện pipeline cho mô hình. Ta có các hàm quan trọng sau đây:

  - **get_detections(img, net):**

```
def get_detections(img, net):
    # 1. CONVERT IMAGE TO YOLO FORMAT
    image = img.copy()
    row, col, d = image.shape

    max_rc = max(row,col)
    input_image = np.zeros((max_rc,max_rc,3),dtype=np.uint8)
    input_image[0:row,0:col] = image

    # 2. GET PREDICTION FROM YOLO MODEL
    blob = cv2.dnn.blobFromImage(input_image,1/255,(INPUT_WIDTH,
    INPUT_HEIGHT),swapRB=True,crop=False)
```

```
    net.setInput(blob)
    preds = net.forward()
    detections = preds[0]

  return input_image, detections
```

Hàm này chuẩn bị ảnh đầu vào và đảm bảo cho ảnh luôn có kích thước vuông 640x640 khi truyền vào YOLO, trả về ảnh đã được điều chỉnh input_image. Sau đó chạy YOLO để lấy dự đoán và lưu kết quả dự đoán vào detections gồm tọa độ điểm trung tâm và kích thước bounding box cùng hai chỉ số confidence (độ tin cậy của mô hình về việc bounding box chứa một đối tượng biển số) và class_score (còn gọi là probability, là xác suất mà bounding box thuộc lớp biển số xe).

- **non_maximum_supression(input_image, detections):**

```
def non_maximum_supression(input_image,detections):

    # 3. FILTER DETECTIONS BASED ON CONFIDENCE AND PROBABILIY SCORE

    # center x, center y, w , h, confidence, probability
    boxes = []
    confidences = []

    image_w, image_h = input_image.shape[:2]
    x_factor = image_w/INPUT_WIDTH
    y_factor = image_h/INPUT_HEIGHT

    for i in range(len(detections)):
        row = detections[i]
        confidence = row[4] # confidence of detecting license plate
        if confidence > 0.4:
            class_score = row[5] # probability score of license plate
            if class_score > 0.25:
                cx, cy , w, h = row[0:4]

                left = int((cx - 0.5*w)*x_factor)
                top = int((cy-0.5*h)*y_factor)
                width = int(w*x_factor)
                height = int(h*y_factor)
                box = np.array([left,top,width,height])

```

```
                confidences.append(confidence)
                boxes.append(box)

    # 4.1 CLEAN
    boxes_np = np.array(boxes).tolist()
    confidences_np = np.array(confidences).tolist()

    # 4.2 NMS
    index = cv2.dnn.NMSBoxes(boxes_np,confidences_np,0.25,0.45)

    return boxes_np, confidences_np, index
```

Hàm này xử lý kết quả từ mô hình YOLO, cụ thể là lọc qua tất cả các dự đoán trong detections dựa trên các chỉ số confidence và class_score. Nếu confidence > 0.4 và class_score > 0.25 thì dự đoán được tin tưởng là chính xác và hàm thực hiện đưa kích thước của bounding box từ hệ chuẩn hóa về kích thước gốc.

Cuối cùng, hàm áp dụng cv2.dnn.NMSBoxes để loại các bounding box bị trùng lặp và trả về số thứ tự của các bounding box được giữ lại sau quá trình lọc trong biến index, cùng với đó là danh sách tọa độ các bounding box boxes_np và danh sách các chỉ số độ tin cậy tương ứng confidences_np.

– **extract_text(image, bbox):**

```
# Extracting text
def extract_text(image,bbox):
    x, y, w, h = bbox
    roi = image[y : y + h, x : x+w]

    if 0 in roi.shape:
        return 'no number'

    else:
        text = pt.image_to_string(roi)
        print(text)
        text = text.strip()

        return text
```

Hàm này dùng thư viện **Tesseract OCR** (gọi qua **pytesseract**) để thực hiện việc trích xuất và trả về dữ liệu văn bản từ bounding box qua biến text.

– **drawings(image, boxes_np, confidences_np, index):**

```
def drawings(image,boxes_np,confidences_np,index):
    # 5. DRAWINGS
    for ind in index:
        x, y, w, h =  boxes_np[ind]
        bb_conf = confidences_np[ind]
        conf_text = 'plate: {:.0f}%'.format(bb_conf*100)

        license_text = extract_text(image,boxes_np[ind])

        cv2.rectangle(image,(x, y), (x + w, y + h), (255, 0,
    255), 2)
        cv2.rectangle(image,(x, y - 30), (x + w, y), (255, 0,
    255), -1)
        cv2.rectangle(image, (x, y + h), (x + w, y + h + 25),
    (0, 0, 0), -1)

        cv2.putText(image, conf_text,(x, y - 10), cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 1)
        cv2.putText(image, license_text,(x, y + h + 27), cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 1)

    return image
```

Hàm này có nhiệm vụ vẽ bounding box từ dữ liệu đã xử lý ở trên. Cụ thể, hàm sẽ đi qua các bounding box đã được lọc, lấy về các chỉ số tọa độ, kích thước và vẽ ra một bounding box thật trên hình ảnh (những bounding box được nhắc đến ở trên thực chất chỉ là giá trị tọa độ được lưu trữ). Kèm theo đó, hàm sẽ ghi thêm độ tin cậy confidence và áp dụng OCR extract_text để lấy dữ liệu văn bản và ghi thêm vào trên bounding box.

– **yolo_predictions(img, net):**

```
# Predictions flow with return result
def yolo_predictions(img,net):
    # Step 1: detections
    input_image, detections = get_detections(img, net)

    # Step 2: NMS
    boxes_np, confidences_np, index = non_maximum_supression(
    input_image, detections)

    # Step 3: drawings
    result_img = drawings(img,boxes_np, confidences_np, index)
    return result_img
```

Đây là hàm tổng hợp của pipeline, tích hợp các bước tiền xử lý, xử lý kết quả dự đoán và biểu hiện kết quả của chương trình dưới dạng bounding box và dữ liệu văn bản, trả về hình ảnh sau khi đã được vẽ bounding box result_img.

- **Chạy code:**

```
# test
img = io.imread('TEST/TEST2.png')
results = yolo_predictions(img,net)

fig = px.imshow(img)
fig.update_layout(width=700, height=400, margin=dict(l=10, r=10, b=10,
    t=10))
fig.update_xaxes(showticklabels=False).update_yaxes(showticklabels=
    False)
fig.show()
```

Sau khi chạy code bằng mô hình qua hàm tổng hợp yolo_predictions, ta nhận lại được hình sau:

Hình 3.5: Ảnh TEST2.png sau khi đã dự đoán

- **Chạy code với video:**

```
from google.colab.patches import cv2_imshow
import cv2

cap = cv2.VideoCapture('TEST/TEST.mov')

nFrame = 50 # Show nFrame only
```

```
cFrame = 0
while cap.isOpened():
    ret, frame = cap.read()

    if ret == False:
        print('Unable to read video')
        break

    results = yolo_predictions(frame,net)

    # cv2.namedWindow('YOLO',cv2.WINDOW_KEEPRATIO)
    cv2_imshow(results)
    # if cv2.waitKey(1) == 27 :
    #     break
    cFrame = cFrame+1
    if (cFrame > nFrame):
        break
# cv2.destroyAllWindows()
cap.release()
```

Đoạn code trên dùng OpenCV để đọc video và đưa từng khung hình qua mô hình YOLOv5 để thực hiện dự đoán. Kết quả trả về là tối đa 50 khung hình đã được dự đoán và vẽ bounding box.

Một cách khác:

```
!python yolov5/detect.py --weights yolov5/runs/train/Model2/weights/
    best.pt --source "TEST/TEST.mov"
```

Chạy câu lệnh này sẽ thực hiện dự đoán tương tự như trên, điểm khác biệt là kết quả được trả về sẽ là một file video với mỗi khung hình đã được dự đoán và vẽ bounding box.

Như đã nói ở phần ii., ta tiếp tục huấn luyện mô hình với 1000 ảnh bổ sung, thực hiện lại những bước xử lý hình ảnh từ iii. và tiến hành train mô hình với **batch-size 16** và **epochs 50** trước khi tích hợp vào web.

vi. Xây dựng web:

Trong mục này, ta tiến hành xây dựng ứng dụng web để tích hợp với mô hình, ta bắt đầu bằng việc tạo thư mục *WebbApp/* lưu trữ thông tin của web. Tiếp theo:

- **Tạo file app.py:**

```python
from flask import Flask, render_template

# Webserver gateway interface
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('layout.html')

if __name__ == "__main__":
    app.run(debug=True)
```

- Khởi tạo server Flask: tạo một đối tượng ứng dụng Flask để làm server xử lý các yêu cầu từ người dùng.
- Định nghĩa cách xử lý các yêu cầu HTTP (qua "route"): xác định các URL và gắn chúng với các hàm xử lý cụ thể. Trong trường hợp này, chỉ định rằng hàm index() sẽ xử lý các yêu cầu đến URL "/" (trang chủ) của trang web.
- Hàm index(): trả về nội dung của file *templates/layout.html* (file sẽ được tạo sau bước này) và gửi nó về trình duyệt của người dùng.

- **Tạo thư mục *WebbApp/templates:*** chứa định dạng của web, bao gồm 2 file layout.html và index.html.

  - **Tạo file layout.html:**

```html
<!DOCTYPE html>

<html>
  <head>
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <title>AUTOMATIC NUMBER-PLATE RECOGNITION</title>
    <link
      <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6"
      crossorigin="anonymous"
```

```
    />
    <script
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"
      integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"
      crossorigin="anonymous"
    ></script>
    <meta name="description" content="" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link rel="stylesheet" href="" />
  </head>
  <body>
    <!-- Navigation bar -->
    <nav class="navbar navbar-light" style="background-color: #c1dce0">
      <div class="container">
        <a class="navbar-brand" href="/">
          <h1 class="display-6">NUMBER PLATE OCR</h1>
        </a>
      </div>
    </nav>
    {% block body %} {% endblock %}
    <!-- Footer -->
    <footer>
      <hr />
      <a href="http://aslanahmedov.com"> CONTACT ME</a>
    </footer>
    <script src="" async defer></script>
  </body>
</html>
```

File layout.html sử dụng thư viện front-end Bootstrap để tạo ra một giao diện web có cấu trúc chung, gồm các thành phần cơ bản như thanh điều hướng (navbar), phần thân (body) có thể được thay đổi thông qua kế thừa, và một phần chân trang (footer).

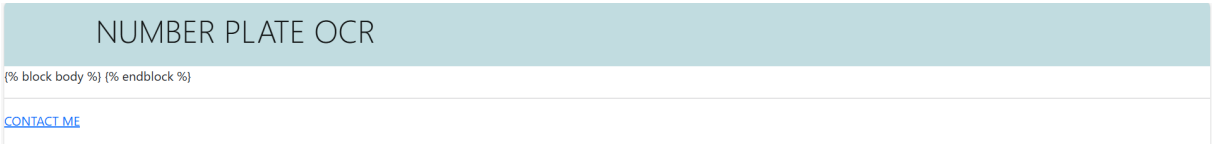
* **Thẻ <head>:**
  - <meta charset="utf-8" />: Đảm bảo mã hóa ký tự UTF-8, hỗ trợ hiển thị các ngôn ngữ khác nhau.
  - <title>AUTOMATIC NUMBER-PLATE RECOGNITION

</title>: Đặt tiêu đề trang.

- Dùng để tải thư viện JavaScript của Bootstrap từ CDN.
- Thuộc tính src chỉ định URL của file.
- Thuộc tính integrity đảm bảo tính toàn vẹn của file.
- Thuộc tính crossorigin="anonymous" cho phép tải file mà không kèm thông tin nhận dạng người dùng.
- Thẻ <meta name="description">: Cung cấp mô tả ngắn về nội dung trang web (SEO).
- Thẻ <meta name="viewport">: Điều chỉnh giao diện trên thiết bị di động.
- width=device-width: Đặt chiều rộng phù hợp với màn hình thiết bị.
- initial-scale=1: Đặt tỷ lệ thu phóng ban đầu.
- Thẻ <link rel="stylesheet">: Dùng để liên kết file CSS bên ngoài.

* **Thẻ <body>:**
  - <nav>: Tạo thanh điều hướng và lớp navbar.
  - : Chứa nội dung chính của thanh điều hướng, bao gồm logo hoặc tiêu đề trang web.
  - {% block body %} {% endblock %}: Đây là khối code phân định phần nội dung mà một layout con sẽ kế thừa và chèn nội dung riêng vào, trong phần sau ta sẽ tạo file **index.html** để làm việc này.
  - <footer>: Phần chân trang với đường kẻ ngang <hr /> và liên kết "CONTACT ME".
* **Thẻ <script>:**
  - Thẻ <script src="" async defer></script>: Cho phép chèn JavaScript với các thuộc tính async (tải đồng thời) và defer (chạy sau khi các tài nguyên đã tải).

– **Tạo file kế thừa index.html:**



Hình 3.6: Ảnh layout khi chạy.

```
{% extends 'layout.html' %} {% block body %}
<div class="container">
  <br />
  <form action="#" method="POST" enctype="multipart/form-data">
    <div class="input-group">
      <input type="file" class="form-control" name="image_name"
    required />
      <input type="submit" value="UPLOAD" class="btn btn-outline
    -secondary" />
    </div>
  </form>
</div>

{% if upload %}
<div class="container">
  <br />
  <table>
    <tr>
      <td>
        <img
          class="rounded float-left img-fluid"
          src="/static/upload/{{upload_image}}"
          alt=""
        />
      </td>
      <td>
        <img
          class="rounded float-right img-fluid"
          src="/static/predict/{{upload_image}}"
          alt=""
        />
      </td>
    </tr>
  </table>
  <br />
  <table style="border: solid black; width: 100%">
    <tr style="border: solid black">
      <th>CROPPEED LICENCE PLATE IMAGE</th>
      <th>TEXT</th>
```

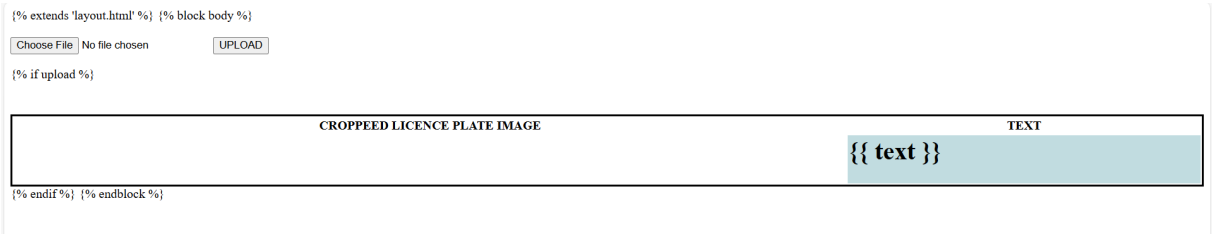
```
38      </tr>
39      <tr style="border: solid black">
40        <td>
41          <img class="img-fluid" src="/static/roi/{{ upload_image
        }}" alt="" />
42        </td>
43        <td style="background-color: #c1dce0">
44          <h1 class="display-8">{{ text }}</h1>
45        </td>
46      </tr>
47    </table>
48  </div>
49
50  {% endif %} {% endblock %}
```

Mục đích chính của file index.html là xây dựng giao diện tải lên và hiển thị kết quả dự đoán nhận diện biển số xe cho người dùng.

Giải thích file:

* {% extends 'layout.html' %} {% block body %}: kế thừa file **layout.html** và chèn nội dung file **index.html** vào file **layout.html**.
* Form upload ảnh: cho phép người dùng tải lên hình ảnh
  - action="#": Hành động của form (nơi gửi dữ liệu) được chỉ định là #, có thể là URL xử lý ảnh tải lên (ở đây chỉ là placeholder).
  - method="POST": Dữ liệu sẽ được gửi đi bằng phương thức POST.
  - enctype="multipart/form-data": Cấu hình này cho phép gửi tệp hình ảnh trong form.
  - <input type="file">: Đây là trường để người dùng chọn tệp hình ảnh từ máy tính của họ.
  - <input type="submit">: Nút gửi form "UPLOAD".
* Hiển thị hình ảnh sau khi upload và dự đoán:
  Hình ảnh gốc (image from upload) được hiển thị ở bên trái (float-left), và hình ảnh đã qua xử lý (ví dụ ảnh nhận diện biển số xe) được hiển thị ở bên phải (float-right).

* Bảng hiển thị biển số xe có 2 cột:
  - Một cột cho ảnh biển số xe đã cắt ra (cropped) từ ảnh gốc (cột bên trái).
  - Một cột cho văn bản đã nhận diện từ biển số xe (biển số được chuyển thành văn bản) (cột bên phải).
* Tạo thêm folder *static/upload* trong WebbApp để chứa ảnh.



Hình 3.7: Ảnh index khi chạy

- **Tạo file deeplearning.py:** File này có vai trò xây dựng pipeline cho mô hình khi chạy trên web, khá tương tự với pipeline mà chúng ta xây dựng ở phần v., bao gồm các phần sau:

  – **Cài đặt thư viện và tải mô hình:**

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.preprocessing.image import load_img,
    img_to_array
import pytesseract as pt
import torch
from PIL import Image

model = torch.hub.load(
    'yolov5',
    'custom',
    path='yolov5/runs/train/Model2/weights/best.pt',
    source="local")
```

  Phần này ta cài đặt các thư viện cần thiết cho việc tạo pipeline cũng như tải mô hình đã lưu.

  – **object_detection(path, filename):**

```python
def object_detection(path, filename):
    # Read image
    image = Image.open(path)  # PIL object
    image = np.array(image, dtype=np.uint8)  # 8 bit array
    (0,255)

    result = model(image, size=640)
    result.save()
    coords = result.xyxy[0].tolist()

    new_coords = []
    for obj_class in coords:
        obj_class = obj_class[:4]
        new_coords.append([int(x) for x in obj_class])

    image_bgr = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

    cv2.imwrite(
        'WebbApp/static/predict/{}'.format(filename), image_bgr)
    return new_coords
```

Mục đích: Hàm này dùng để thực hiện phát hiện đối tượng (object detection) trong ảnh sử dụng mô hình YOLOv5.

Các bước thực hiện:

* Đọc ảnh từ đường dẫn path bằng thư viện PIL và chuyển ảnh thành mảng NumPy.
* Sử dụng mô hình YOLOv5 để phát hiện các đối tượng trong ảnh.
* Kết quả trả về là tọa độ của các bounding box cho các đối tượng phát hiện được. Những tọa độ này được chuẩn hóa, sau đó nhân với kích thước ảnh ban đầu để có tọa độ thực tế.
* Sử dụng vòng lặp for để xác định những biển số ở trong hình ảnh được người dùng UPLOAD lên.
* Mô hình trả về các các tọa độ của bounding box dưới dạng tọa độ [xmin, ymin, xmax, ymax].
* Cuối cùng, ảnh sẽ được lưu vào *static/predict/* với tên là filename và tọa độ của các đối tượng được trả về.

– **save_text(filename, text):**

```python
def save_text(filename, text):
    name, ext = os.path.splitext(filename)
    with open('WebbApp/static/predict/{}.txt'.format(name), mode='w') as f:
        f.write(text)
    f.close()
```

Mục đích: Hàm này có nhiệm vụ lưu văn bản vào một tệp tin .txt.

Các bước thực hiện:

* Nhận tham số filename và text, tách phần mở rộng của tệp tin (tên tệp mà không có phần mở rộng).
* Lưu văn bản text vào tệp .txt có cùng tên với ảnh đã được xử lý. Tệp này sẽ được lưu vào thư mục *static/predict/*.

– **OCR(path, filename):**

```python
def OCR(path, filename):
    img = np.array(load_img(path))
    cods = object_detection(path, filename)
    final_text = ""

    for i in range(len(cods)):
        xmin, ymin, xmax, ymax = cods[i].copy()
        roi = img[ymin:ymax, xmin:xmax]
        roi_bgr = cv2.cvtColor(roi, cv2.COLOR_RGB2BGR)
        gray = cv2.cvtColor(roi_bgr, cv2.COLOR_BGR2GRAY)

        blur = cv2.GaussianBlur(gray, (3,3), 0)
        thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]

        cv2.imwrite(
            'WebbApp/static/roi/{}'.format(filename), roi_bgr)

        text = pt.image_to_string(thresh, lang='eng', config='--psm 6')
        final_text += text.strip() + ";" +"\n"
    print(final_text)
    save_text(filename, final_text)
    return final_text
```

Mục đích: Hàm này có nhiệm vụ tích hợp các hàm rời rạc khác trong file thành một pipeline hoàn chỉnh, đồng thời sử dụng kỹ thuật nhận dạng ký tự quang học (OCR) để trích xuất văn bản

từ các đối tượng phát hiện trong ảnh.

Các bước thực hiện:

* Đầu tiên, hàm gọi object_detection để phát hiện các đối tượng trong ảnh.
* Sau đó, cho mỗi đối tượng (có tọa độ bounding box), hàm cắt vùng chứa đối tượng đó từ ảnh gốc (region of interest - ROI).
* Tiếp theo, ảnh ROI được chuyển sang dạng ảnh xám (grayscale), sau đó làm mờ và áp dụng ngưỡng nhị phân (binary thresholding) để làm rõ các văn bản.
* Ảnh sau khi xử lý được lưu lại trong thư mục *static/roi/* để kiểm tra kết quả.
* Cuối cùng, sử dụng thư viện **pytesseract** để nhận diện văn bản từ ảnh đã xử lý và lưu kết quả vào một tệp văn bản thông qua hàm save_text.

- **Chỉnh sửa lại file app.py:**

  – **Thiết lập đường dẫn cơ bản:**

```
BASE_PATH = os.getcwd()
UPLOAD_PATH = os.path.join(BASE_PATH, 'WebbApp/static/upload')

```

  * BASE_PATH: Lấy đường dẫn hiện tại của tệp *app.py*.
  * UPLOAD_PATH: Tạo đường dẫn đến thư mục lưu trữ tệp tải lên *static/upload*.

  – **Định nghĩa route chính:**

```
@app.route('/', methods=['POST', 'GET'])
def index():
    ...

```

  * URL gốc '/' được định nghĩa bằng decorator @app.route.
  * Chấp nhận cả yêu cầu **POST** và **GET**:
    Khi yêu cầu POST được gọi:

```
if request.method == 'POST':
    upload_file = request.files['image_name']
    filename = upload_file.filename
    path_save = os.path.join(UPLOAD_PATH, filename)
    upload_file.save(path_save)
    text = OCR(path_save, filename)

return render_template('index.html', upload=True,
    upload_image=filename, text=text)
```

- Nhận tệp tải lên từ input image_name.
- Xây dựng đường dẫn từ UPLOAD_PATH đã được lưu ở trên và lưu file vào đường dẫn.
- Sử dụng mô hình YOLOV5 để thực hiện dự đoán hình ảnh qua hàm OCR.
- Trả về kết quả qua giao diện HTML bằng **index.html** (đã được chỉnh sửa so với **layout.html** ở vi.).

Khi yêu cầu GET được gọi (khi mới vào trang web):

```
return render_template('index.html', upload=False)
```

Chạy template **index.html** với biến upload=False cho biết chưa có file nào được đăng lên web.

– **Chạy ứng dụng Flask:**

```
if __name__ == "__main__":
    app.run(debug=True)
```

* Chạy ứng dụng Flask trên server cục bộ.
* Sử dụng chế độ debug để hỗ trợ phát triển (hiển thị lỗi chi tiết, tự động tải lại ứng dụng khi thay đổi).

vii. Kiểm tra ứng dụng và mô hình:

Như vậy ta đã hoàn thành việc xây dựng ứng dụng web cho việc tự động nhận diện biển số xe. Việc chạy mô hình và nhận xét kết quả sẽ được ghi rõ hơn ở phần 3. và 4..

Để mô hình được hoàn thiện hơn nữa, ta tiếp tục chạy mô hình trên nhiều dữ liệu để đánh giá độ chính xác dựa trên các chỉ số liên quan. Đồng thời kiểm tra trong điều kiện thực tế với các biển số khác nhau trong điều kiện môi trường khác nhau để đảm bảo mô hình hoạt động tốt và ổn định.

3. Experiment:

Phần này sẽ được thuyết minh theo video đi kèm.

i. Dataset: Giới thiệu dataset, mô tả dataset

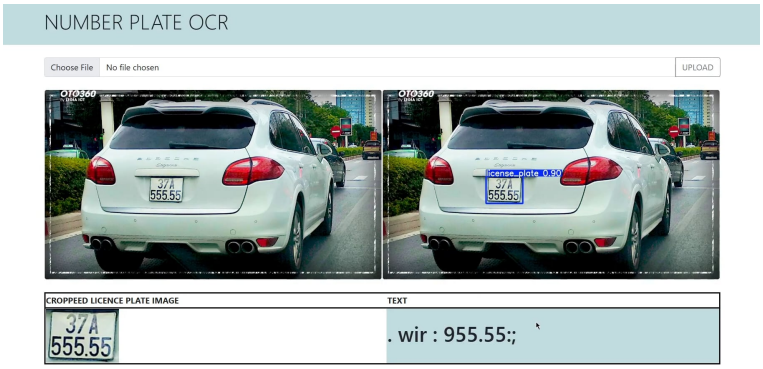
Dữ liệu huấn luyện mô hình gồm 250 hình ảnh biển số xe (toàn bộ biển nước ngoài) từ file hướng dẫn [1] và 1000 hình ảnh biển số xe (một phần nước ngoài, một phần Việt Nam) từ nguồn tìm thêm [3]. Các hình ảnh được chụp ở nhiều điều kiện ánh sáng, góc độ,...khác nhau. Dữ liệu được chia thành 80% huấn luyện và 20% kiểm tra.

ii. Cấu hình chạy thí nghiệm

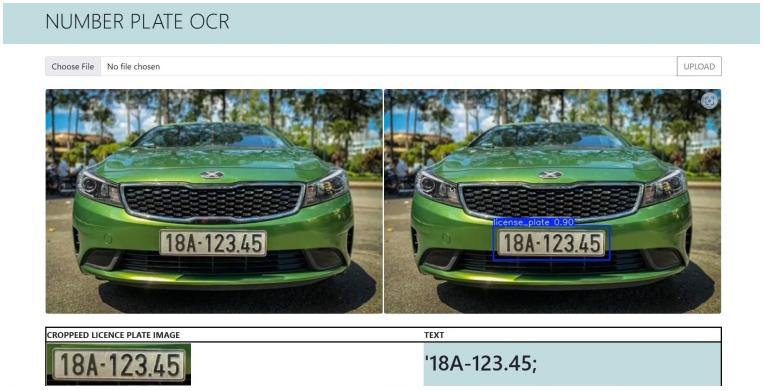
- **Môi trường thực thi:**
  - Các Framework sử dụng: OS module, OpenCV, NumPy, Pandas, TensorFlow, Pytesseract, Plotly Express, Matplotlib, ElementTree, Glob, Scikit-Image, Scikit-learn, Shutil.
  - Phần cứng: Project được chạy trên nền tảng Google Colaboratory, sử dụng runtime T4-GPU.
- **Mô hình được sử dụng:** YOLOv5.

iii. Kết quả chạy thí nghiệm:

Ta chạy web application với 3 hình ảnh chọn ngẫu nhiên (không nằm trong dataset):



Hình 3.8: Hình 1



Hình 3.9: Hình 2

Hình 3.10: Hình 3

Từ kết quả, ta thấy:

- **Hình 1:** Dùng để kiểm tra khả năng đọc biển số có nhiều dòng ký tự. Kết quả trả về có bounding box vẽ chính xác, chỉ số confidence cao nhưng ký tự văn bản có nhiều lỗi sai.

- **Hình 2:** Dùng để kiểm tra khả năng đọc biển số thông thường. Kết quả trả về có bounding box vẽ chính xác, chỉ số confidence cao và ký tự văn bản được nhận diện chính xác.

- **Hình 3:** Dùng để kiểm tra khả năng nhận diện nhiều biển số. Kết quả trả về có các bounding box được vẽ chính xác, chỉ số confidence cao và ký tự văn bản được nhận diện chính xác.

4. Kết luận

Như vậy trong dự án này, ta đã hoàn thành việc xây dựng một mô hình YOLOv5 thực hiện chức năng tự động nhận diện biển số xe bằng ngôn ngữ Python, và dùng model để xây dựng một web application thực hiện dự đoán từ dữ liệu hình ảnh của người dùng. Mô hình của chúng ta có khả năng đọc biển số xe Việt Nam và nước ngoài, mang độ chính xác tương đối và còn gặp nhiều khó khăn trong việc đọc những biển số có góc lệch so với khung hình (biển số bị "nghiêng") và biển số có nhiều hơn 1 hàng ký tự.

Với sự gia tăng số lượng phương tiện, việc theo dõi phương tiện đã trở thành một lĩnh vực nghiên cứu quan trọng nhằm kiểm soát giao thông một cách hiệu quả, nhận diện và tìm kiếm xe bị đánh cắp. Để đạt được mục tiêu này, việc phát hiện và nhận diện biển số xe theo thời gian thực là rất quan trọng. Do sự đa dạng về nền, màu sắc và kiểu chữ, kích thước biển số, cùng với các ký tự chưa được chuẩn hóa, việc nhận diện biển số xe là một thách thức lớn đối với các quốc gia đang phát triển. Để khắc phục các vấn đề này, nghiên cứu này đã áp dụng học sâu để cải thiện hiệu quả nhận diện biển số xe. Các hình ảnh thu thập được đã được chụp dưới các điều kiện ánh sáng/độ tương phản khác nhau, khoảng cách từ camera, góc quay thay đổi và được xác minh để đạt tỷ lệ nhận diện cao. Phương pháp này có thể được sử dụng hiệu quả bởi các cơ quan thực thi pháp luật và các tổ chức tư nhân để cải thiện an ninh quốc gia. Tương lai của mô hình

này có thể bao gồm việc huấn luyện và xác minh thuật toán hiện tại bằng phương pháp phân loại lai và cải thiện độ bền vững của hệ thống nhận diện biển số xe trong các điều kiện thời tiết khác nhau.

Chương 4

Báo cáo làm việc hàng tuần

Ngày họp	Thành viên tham dự	Phân công
09/11/2024	Lâm, Phú, Huy, Phát, Tuấn, Tú	**Lâm**: Viết báo cáo **Phú**: Lập trình đồ án **Huy**: Viết báo cáo **Phát**: Viết báo cáo **Tuấn**: Lập trình đồ án **Tú**: Viết báo cáo
Thời gian: 9-23/11/2024 Tiến độ: 25% Công việc đã làm: - Xác định mục tiêu, phân công công việc, thời hạn ☑ - Lập trình và mô hình A.I ☑ - Báo cáo các mục: + Introduction ✖ + Methodology ✖ + Conclusion ✖ - Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ✖ - Powerpoint thuyết minh ✖ - Video giới thiệu và demo đồ án ✖		
14/11/2024	Lâm, Phú, Huy, Phát, Tuấn, Tú	**Lâm**: Viết báo cáo **Phú**: Lập trình đồ án **Huy**: Viết báo cáo **Phát**: Viết báo cáo **Tuấn**: Lập trình đồ án **Tú**: Viết báo cáo

<table>
<tr><td colspan="3">Thời gian: 14-23/11/2024 Tiến độ: 50%
Công việc đã làm:
- Xác định mục tiêu, phân công công việc, thời hạn ☑
- Lập trình và mô hình A.I ☑
- Báo cáo các mục:
+ Introduction ☑
+ Methodology ☑
+ Conclusion ☑
- Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ✖
- Powerpoint thuyết minh ✖
- Video giới thiệu và demo đồ án ✖</td></tr>
<tr><td>24/11/2024</td><td>Lâm, Phú, Huy, Phát, Tuấn, Tú</td><td>Lâm: Thiết kế PowerPoint, làm video
Phú: Lập trình đồ án
Huy: Viết báo cáo
Phát: Viết báo cáo
Tuấn: Lập trình đồ án
Tú: Viết báo cáo</td></tr>
<tr><td colspan="3">Thời gian: 14-23/11/2024 Tiến độ: 50%
Công việc đã làm:
- Xác định mục tiêu, phân công công việc, thời hạn ☑
- Lập trình và mô hình A.I ☑
- Báo cáo các mục:
+ Introduction ☑
+ Methodology ☑
+ Conclusion ☑
- Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ☑
- Powerpoint thuyết minh ✖
- Video giới thiệu và demo đồ án ✖</td></tr>
</table>

28/11/2024	Lâm, Phú, Huy, Phát, Tuấn, Tú	**Lâm**: Thiết kế PowerPoint, làm video **Phú**: Lập trình đồ án, làm video **Huy**: Viết báo cáo, làm video **Phát**: Viết báo cáo **Tuấn**: Lập trình đồ án, làm video **Tú**: Viết báo cáo
Thời gian: 24-28/11/2024 Tiến độ: 100% Công việc đã làm: - Xác định mục tiêu, phân công công việc, thời hạn ☑ - Lập trình và mô hình A.I ☑ - Báo cáo các mục: + Introduction ☑ + Methodology ☑ + Conclusion ☑ - Hoàn thiện phần mềm, tải lên github sản phẩm hoàn chỉnh ☑ - Powerpoint thuyết minh ☑ - Video giới thiệu và demo đồ án ☑		

Tài liệu tham khảo

Tiếng Anh

[1] *File Github của mô hình mẫu.* URL: https://github.com/Asikpalysik/Automatic-License-Plate-Detection.

[2] *File google colab hướng dẫn xây dựng mô hình.* URL: https://universe.roboflow.com/utech-susnq/license-plate-detection-dataset.

[3] *Tài liệu hình ảnh.* URL: https://universe.roboflow.com/utech-susnq/license-plate-detection-dataset.