

# MÔN HỌC THỐNG KÊ

17\_22 – GVLT. Ngô Minh Nhựt

## -- BÁO CÁO ĐỒ ÁN CUỐI KỲ -- SỬ DỤNG YOLO VÀ XÂY DỰNG ỨNG DỤNG NHẬN DIỆN BIỂN SỐ XE MÁY VIỆT NAM

*Báo cáo này chứa thông tin nhóm, ghi nhận lại các bước thực hiện đồ án và những phần mà nhóm đã đạt được.*



Khoa Công nghệ Thông tin  
Đại học Khoa học Tự nhiên TP HCM  
Tháng 08/2020

TP. Hồ Chí Minh, ngày 06 tháng 08 năm 2020

# MỤC LỤC

<b>1 THÔNG TIN NHÓM.....</b>	<b>3</b>
<b>2 MỨC ĐỘ HOÀN THÀNH.....</b>	<b>4</b>
<b>3 NỘI DUNG THỰC HIỆN .....</b>	<b>5</b>
3.1 Cài đặt YOLO và sử dụng được mô hình có sẵn .....	5
3.2 Áp dụng YOLO để huấn luyện mô hình nhận diện.....	8
3.3 Xây dựng ứng dụng Web .....	20
<b>4 TÀI LIỆU THAM KHẢO .....</b>	<b>30</b>

# 1 THÔNG TIN NHÓM

MSSV	Họ và tên	Email
1612406	Đặng Phương Nam	<a href="mailto:1612406@student.hcmus.edu.vn">1612406@student.hcmus.edu.vn</a>
1612423	Lê Minh Nghĩa	<a href="mailto:1612423@student.hcmus.edu.vn">1612423@student.hcmus.edu.vn</a>

## 2 MỨC ĐỘ HOÀN THÀNH

STT	Nội dung		Mức độ hoàn thành
1	Cài đặt YOLO và sử dụng được mô hình có sẵn.		100%
2	Áp dụng YOLO để xây dựng ứng dụng.	Hiểu và huấn luyện để nhận dạng thêm một loại đối tượng mới.	100%
		Áp dụng để xây dựng một ứng dụng hoàn thiện.	100%
3	Mô hình chỉ được phép load 1 lần và được dùng cho tất cả các lần phân lớp.		100%
4	Đủ độ khó ở yêu cầu 2 khi làm theo nhóm.		100%
Tổng mức độ hoàn thành			100%

### Tóm tắt các công cụ, ngôn ngữ lập trình đã sử dụng trong project:

- Ngôn ngữ lập trình chính: **Python**.
- YOLO framework: **yolo-v4**.
- Công cụ đánh nhãn **labellmg**. Thời gian đánh nhãn:
  - + Plate: **nửa ngày**.
  - + Characters: **7 ngày**.
- Sử dụng GPU trên Google Colab để train model. Thời gian train:
  - + Model Plate: **2 ngày**.
  - + Model Characters: **6 ngày**.
- Frontend: **HTML, CSS, JavaScript**.
- Backend: **Flask**.

**Link github chứa toàn bộ source của nhóm:** <https://github.com/dpnam/SL-Final-Project.git>

- **colab**: chứa tệp Python Notebook dùng để train trên Google Colab.
- **raw-data**: chứa dữ liệu gốc.
- **data**: chứa dữ liệu dùng để train và test.
- **scripts**: chứa các file scripts hỗ trợ chuẩn bị dữ liệu, ...
- **src**: chứa mã nguồn của website.
- **3 model đính kèm** nằm ở **phần Releases** của repository này (ta cần download và bỏ vào `./src/models/` để chạy web).

# 3 NỘI DUNG THỰC HIỆN

## 3.1 Cài đặt YOLO và sử dụng được mô hình có sẵn

Toàn bộ mã nguồn và data liên quan nằm tại: `./data` và `./colab` (cụ thể là file *Yolov4-Detect-License-Plate-VN.ipynb*).

### Cài đặt YOLO-v4:

**Bước 1.** Clone mã nguồn YOLO: <https://github.com/AlexeyAB/darknet>

**Bước 2.** Vào Makefile sửa lại các dòng đầu như sau:

```
GPU=0
CUDNN=0
CUDNN_HALF=0
OPENCV=0
AVX=1
OPENMP=1
LIBSO=1
ZED_CAMERA=0
ZED_CAMERA_v2_8=0
```

**Bước 3.** Sau đó, mở Terminal gõ lệnh **make** để biên dịch:

```
~/apps/darknet master* [450/450]
> make
mkdir -p ./obj/
mkdir -p backup
chmod +x *.sh
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -Wall -Wfatal-errors -Wno-unused-result -Wno-unknown-pragmas -fPIC -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a -Ofast -fopenmp -fPIC -c ./src/image_opencv.cpp -o obj/image_opencv.o
g++ -std=c++11 -std=c++11 -Iinclude/ -I3rdparty/stb/include -Wall -Wfatal-errors -Wno-unused-result -Wno-unknown-pragmas -fPIC -ffp-contract=fast -mavx -mavx2 -msse3 -msse4.1 -msse4.2 -msse4a -Ofast -fopenmp -fPIC -c ./src/http_stream.cpp -o obj/http_stream.o
```

**Kết quả tạo ra 2 file chính là:**

- **darknet:** dùng chạy để train, test, tính mAP, ...
- **libdarknet.so:** đây là thư viện để dùng trong chương trình khác, dùng chung với tệp **darknet.py** để gọi trong python.

**Sử dụng mô hình có sẵn:****Bước 1.** Download model **yolo-v4**:

- [https://github.com/AlexeyAB/darknet/releases/download/darknet\\_yolo\\_v3\\_optimal/yolov4.weights](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights)
- Được train trên tập dữ liệu **Coco 2017**: <https://cocodataset.org/#detection-2017>, là bộ dữ liệu tuyệt vời phục vụ cho việc nhận dạng đối tượng (Object detection) với 80 class, hơn 200000 bức ảnh cho cả 3 tập train, validation và test.

**Bước 2.** Sử dụng thử model trên **google colab**:

- Chạy lệnh bên dưới cho ảnh [sample.jpg](#):

```
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/dog.jpg
```

```
[ ] # running detection with the downloaded weights
!./darknet detector test cfg/coco.data cfg/yolov4.cfg yolov4.weights data/dog.jpg
```

```

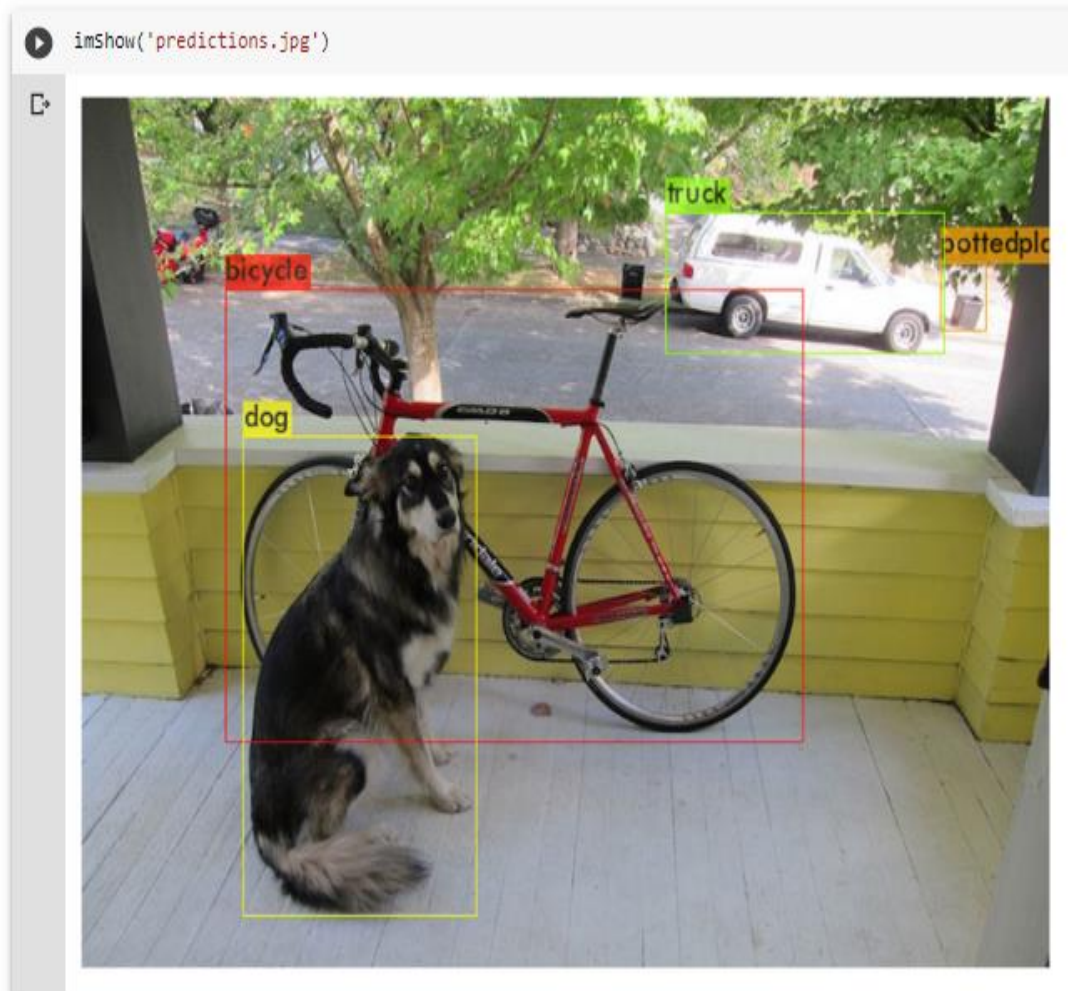
CUDA-version: 10010 (10010), cuDNN: 7.6.5, CUDNN_HALF=1, GPU count: 1
CUDNN_HALF=1
OpenCV version: 3.2.0
0 : compute_capability = 750, cudnn_half = 1, GPU: Tesla T4
net.optimized_memory = 0
mini_batch = 1, batch = 8, time_steps = 1, train = 0
layer  filters  size/strd(dil)  input  output
0 conv  32  3 x 3/ 1  608 x 608 x 3 -> 608 x 608 x 32 0.639 BF
1 conv  64  3 x 3/ 2  608 x 608 x 32 -> 304 x 304 x 64 3.407 BF
2 conv  64  1 x 1/ 1  304 x 304 x 64 -> 304 x 304 x 64 0.757 BF
3 route  1  -> 304 x 304 x 64
4 conv  64  1 x 1/ 1  304 x 304 x 64 -> 304 x 304 x 64 0.757 BF
5 conv  32  1 x 1/ 1  304 x 304 x 64 -> 304 x 304 x 32 0.379 BF
6 conv  64  3 x 3/ 1  304 x 304 x 32 -> 304 x 304 x 64 3.407 BF
7 Shortcut Layer: 4, wt = 0, wn = 0, outputs: 304 x 304 x 64 0.006 BF
8 conv  64  1 x 1/ 1  304 x 304 x 64 -> 304 x 304 x 64 0.757 BF
9 route  8 2  -> 304 x 304 x 128
10 conv  64  1 x 1/ 1  304 x 304 x 128 -> 304 x 304 x 64 1.514 BF
11 conv  128 3 x 3/ 2  304 x 304 x 64 -> 152 x 152 x 128 3.407 BF
12 conv  64  1 x 1/ 1  152 x 152 x 128 -> 152 x 152 x 64 0.379 BF
13 route  11  -> 152 x 152 x 128
14 conv  64  1 x 1/ 1  152 x 152 x 128 -> 152 x 152 x 64 0.379 BF
15 conv  64  1 x 1/ 1  152 x 152 x 64 -> 152 x 152 x 64 0.189 BF
16 conv  64  3 x 3/ 1  152 x 152 x 64 -> 152 x 152 x 64 1.703 BF
17 Shortcut Layer: 14, wt = 0, wn = 0, outputs: 152 x 152 x 64 0.001 BF
18 conv  64  1 x 1/ 1  152 x 152 x 64 -> 152 x 152 x 64 0.189 BF
19 conv  64  3 x 3/ 1  152 x 152 x 64 -> 152 x 152 x 64 1.703 BF
20 Shortcut Layer: 17, wt = 0, wn = 0, outputs: 152 x 152 x 64 0.001 BF
21 conv  64  1 x 1/ 1  152 x 152 x 64 -> 152 x 152 x 64 0.189 BF
22 route  21 12  -> 152 x 152 x 128
23 conv  128 1 x 1/ 1  152 x 152 x 128 -> 152 x 152 x 128 0.757 BF
24 conv  256 3 x 3/ 2  152 x 152 x 128 -> 76 x 76 x 256 3.407 BF
25 conv  128 1 x 1/ 1  76 x 76 x 256 -> 76 x 76 x 128 0.379 BF
26 route  24  -> 76 x 76 x 256

```

- Sau khi chạy xong cell code ở trên, mặc định kết quả ảnh output sẽ có tên là [predictions.jpg](#) nằm ở vị trí thư mục current (./). Để hiển thị ảnh, ta cần viết thêm một hàm **imShow** như bên dưới:

```
[ ] def imShow(path):  
    from google.colab.patches import cv2_imshow  
    img = cv2.imread(path, cv2.IMREAD_UNCHANGED)  
    cv2_imshow(img)
```

- Cuối cùng, gọi hàm **imShow** với ảnh `predictions.jpg` để hiện thị kết quả:





## 3.2 Áp dụng YOLO để huấn luyện mô hình nhận diện

Toàn bộ mã nguồn và data liên quan nằm tại: [./data](#), [./raw-data](#), [./scripts](#) và [./colab](#)

### Giới thiệu đề tài:

- Ứng dụng mà nhóm muốn xây dựng là **một ứng dụng có khả năng nhận diện biển số xe máy của Việt Nam**, bao gồm: **xác định được đâu là biển số xe trên tấm ảnh** và **nhận diện được từng ký tự có mặt trong biển số**.
- Bộ dữ liệu mà nhóm sử dụng được lấy từ **Bộ ảnh biển số xe máy của công ty GreenParking** trên trang <https://thigiacytinh.com/>, có 1748 tấm ảnh dạng như sau:



### Chia tập dữ liệu:

- Nhóm sẽ chia tập dữ liệu theo **80%** dữ liệu dùng cho việc training gọi là **tập training lớn (1398 ảnh)** và **20%** dữ liệu dùng cho việc **test (350 ảnh)**. Sau đó, tiếp tục chia **tập train lớn** theo theo tỉ lệ **77.5%** dùng cho việc **train (1083 ảnh)** và **22.5%** dùng cho việc **validation (315 ảnh)**, tóm lại ta thu được:
  - + Tập **train: 1083 tấm ảnh**.
  - + Tập **validation: 315 tấm ảnh**.
  - + Tập **test: 350 tấm ảnh**, sẽ **được đóng băng** không sử dụng **cho đến khi chọn được model tốt nhất** từ tập train và validation, và **chỉ được dùng duy nhất một lần** nhằm mục đích đánh giá chất lượng model trong thực tế.
- Các tập này cũng được tổ chức thành **3 folder tương ứng** là train, validation và test.



**Quá trình huấn luyện dữ liệu:**

Nhóm chia quá trình huấn luyện thành 2 phase tương ứng với 2 model (model nhận diện biển số xe trong ảnh và model nhận diện các ký tự trong ảnh). Cụ thể công việc như sau:

❖ **Phase 1:** Huấn luyện model nhận diện biển số xe.**Quá trình đánh nhãn dữ liệu**

- Bộ dữ liệu có cung cấp một file [location.txt](#) đã đánh nhãn biển số xe theo format:

```
<tên file> <class> <x> <y> <w> <h>
```

trong đó  $x, y$  là tọa độ top left và  $w, h$  là width, height của box bao biển số.


```
0000_00532_b.jpg 1 145 73 72 62
0000_02187_b.jpg 1 175 116 83 78
0000_05696_b.jpg 1 190 23 83 71
0000_06886_b.jpg 1 154 79 85 76
0000_08244_b.jpg 1 181 75 79 70
0001_05318_b.jpg 1 174 52 71 61
0002_02183_b.jpg 1 212 158 70 53
0002_02554_b.jpg 1 204 145 88 79
0003_02063_b.jpg 1 154 132 82 66
0003_07398_b.jpg 1 206 198 75 61
```

- Nhưng format theo YOLO để đưa dữ liệu vào huấn luyện là:

Ứng với mỗi tấm ảnh, sẽ có một file txt chứa tọa độ đánh nhãn theo format:

```
<class> <x> <y> <w> <h>
```

trong đó  $x, y$  là tọa độ tâm và  $w, h$  là width, height của box bao biển số, cả 4 giá trị này đều được scale về  $[0, 1]$ .

 0000\_00532\_b.txt - Notepad

File Edit Format View Help

```
0 0.3834745762711864 0.3432343234323432 0.15254237288135594 0.20462046204620463
```

- Do đó nhóm có code thêm một file [./scripts/convert\\_to\\_yolo\\_format.py](#) để làm nhiệm vụ chuyển đổi các giá trị đánh nhãn của file [location.txt](#) sang dạng format của YOLO. Kết quả thu được ở mỗi thư mục train, validation và test thì ứng với mỗi bức ảnh sẽ có tương ứng một file cùng tên nhưng có đuôi [.txt](#) đi kèm để đánh dấu việc gán nhãn theo format YOLO.

**Tổ chức huấn luyện dữ liệu trên colab:**

Để huấn luyện model nhanh chóng và hiệu quả, nhóm đã sử dụng **google colab**.

**Bước 1. Khâu chuẩn bị:**

- Do code để chạy trên colab nên ta cần tổ chức một **folder plate** (folder này sẽ đặt trong thư mục **darknet/data** trên colab) theo format bên dưới:

**plate**

```
|- train/
|- validation/
|- test/
|- plate.data
|- plate.names
|- yolov4-train.cfg
|- train.txt
|- validation.txt
|- test.txt
```

- + Với 3 folder train, validation và test như đã nói ở trên.
- + Tập plate.data:

```
classes = 1
train = data/plate/train.txt
valid = data/plate/validation.txt
names = data/plate/plate.names
backup = /mydrive/yolov4/plate/backup
```

- trong đó:
- *classes* là số class
  - *train* là đường dẫn tới tập *train.txt*.
  - *valid* là đường dẫn tới tập *validation.txt*.  
dòng *valid = data/plate/validation.txt* còn được dùng để tính mAP trên tập validation (nếu muốn tính mAP trên tập train thì sửa thành *valid = data/plate/train.txt*).
  - *names* là đường dẫn tới tập *plate.names*.

- + Tập *plate.names* là tập chứa tên của các class, mỗi class 1 dòng, trong trường hợp này chỉ có một dòng là "plate".
- + Tập *yolov4-train.cfg*, được lấy từ tập <https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-custom.cfg> và sửa lại như sau:

Sửa	Mục đích
width = 416 height = 416	Giảm width với height để train nhanh hơn, là bội số của 32 (nhưng có thể làm giảm performance của model).
max_batches = 6000 steps = 4800, 5400	<ul style="list-style-type: none"> <li>Sửa max_batches = 6000 = max([số class * 2000], [số ảnh dùng để train], [6000]).</li> <li>steps = 80% và 90% của max_batches.</li> </ul>
classes = 1 filters = 18	<ul style="list-style-type: none"> <li>Sửa classes = 1 ở 3 section [yolo].</li> <li>Sửa filters = 18 = (số class + 5)*3 ở 3 section [convolutional] ở ngay trên section [yolo].</li> </ul> <div> <div> <pre>[convolutional] size=1 stride=1 pad=1 filters=255 activation=linear  [yolo] mask = 0,1,2 anchors = 12, 16, 19, 36, 40, 28, 36, 75 classes=80 num=9</pre> </div> <div> <pre>[convolutional] size=1 stride=1 pad=1 filters=18 activation=linear  [yolo] mask = 0,1,2 anchors = 12, 16, 19, 36, 40, 28, 36, classes=1 num=9</pre> </div> </div>

- + Tập train.txt chứa đường dẫn tới ảnh dùng để train, mỗi ảnh một dòng, ví dụ:  
data/plate/train/0255\_03013\_b.jpg
- + Tương tự cho tập validation.txt và test.txt.

- Tải pre-trained weights-file để training:

[https://github.com/AlexeyAB/darknet/releases/download/darknet\\_yolo\\_v3\\_optimal/yolov4.conv.137](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137)

## Bước 2. Tiến hành train dữ liệu:

- Chạy lệnh trên colab:

```
./darknet detector train data/plate/plate.data data/plate/yolov4-.cfg yolov4.conv.137
```

- Thời gian chạy xong 6000 iteraters mà nhóm đã thử nghiệm là gần 2 ngày.

## Bước 3. Tính mAP để chọn best model Plate:









- Lấy weight:

- Vào yolov4/plate/backup/ trên google drive.
- Bấm chuột phải vào file yolov4-train\_last.weights → chọn Manage Versions → tải hết version về máy, đổi tên thành yolov4-train\_last\_<version>.weights rồi upload vào thư mục mAP/plate/

### Manage versions

Older versions of 'yolov4-train\_last.weights' may be deleted after 30 days or after 100 versions are stored. To avoid deletion, select **Keep forever** in the file's context menu. [Learn more](#)

UPLOAD NEW VERSION

	Current version yolov4-train_last.weights	
	Jul 18, 5:09 PM Lê Minh Nghĩa	
	Version 46 yolov4-train_last.weights	
	Jul 18, 4:51 PM Lê Minh Nghĩa	
	Version 45 yolov4-train_last.weights	
	Jul 18, 4:39 PM Lê Minh Nghĩa	
	Version 44 yolov4-train_last.weights	

CLOSE

- Tổng cộng có **60 file**.
- Tính mAP ứng với từng weight:
    - Lập qua tên 60 file có được ở trên.
    - Chạy lệnh:
 

```
!./darknet detector map data/plate/plate.data data/plate/yolov4-train.cfg  
"$file" > "$outFile" 2> /dev/null
```

 để tính mAP cho từng "\$file" và ghi ra "\$outFile".
  - Kết quả thu được các file có dạng như sau:

```
mAP_100.txt
CUDNN_HALF=1
net.optimized_memory = 0
mini_batch = 1, batch = 16, time_steps = 1, train = 0
rms_kind: greedy (1), beta = 0.600000
rms_kind: greedy (1), beta = 0.600000
rms_kind: greedy (1), beta = 0.600000

seen 64, trained: 6 K-images (0 Kilo-batches_64)

calculation mAP (mean average precision)...
Detection layer: 139 - type = 27
Detection layer: 150 - type = 27
Detection layer: 161 - type = 27

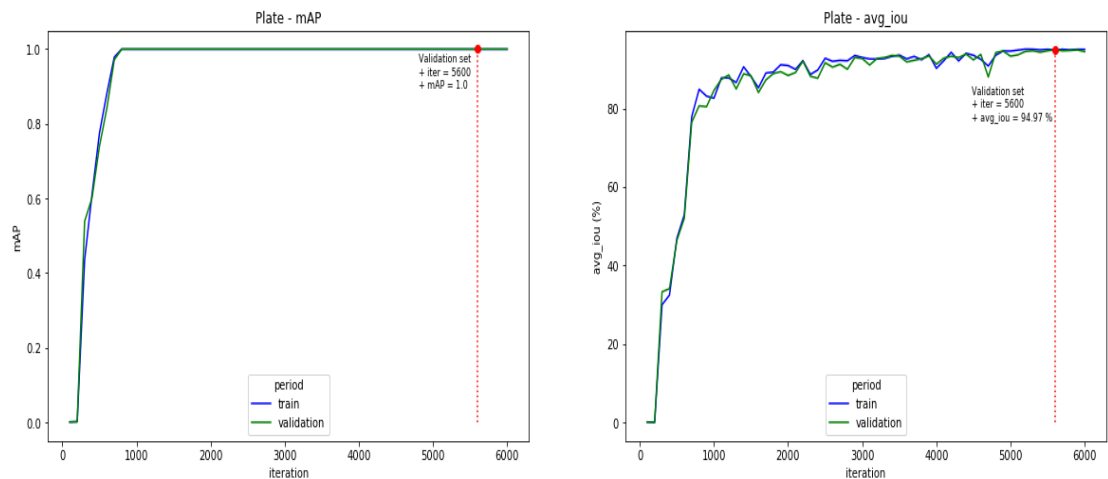
detections_count = 5478687, unique_truth_count = 1083
rank = 0 of ranks = 5478687
rank = 100 of ranks = 5478687
rank = 200 of ranks = 5478687
rank = 300 of ranks = 5478687
rank = 400 of ranks = 5478687
rank = 500 of ranks = 5478687
rank = 600 of ranks = 5478687
rank = 700 of ranks = 5478687
rank = 800 of ranks = 5478687
rank = 900 of ranks = 5478687
rank = 1000 of ranks = 5478687
rank = 1100 of ranks = 5478687
rank = 1200 of ranks = 5478687
rank = 1300 of ranks = 5478687
rank = 1400 of ranks = 5478687
rank = 1500 of ranks = 5478687
rank = 1600 of ranks = 5478687
rank = 1700 of ranks = 5478687
rank = 1800 of ranks = 5478687
rank = 1900 of ranks = 5478687
rank = 2000 of ranks = 5478687
rank = 2100 of ranks = 5478687
rank = 2200 of ranks = 5478687
rank = 2300 of ranks = 5478687
rank = 2400 of ranks = 5478687
```

- Tiếp đó, nhóm có viết một **script python** ([./scripts/map2json.py](#)) để **tổng hợp toàn bộ dữ liệu liên quan tới mAP sang json** ([./scripts/mAP](#)) cho ra kết quả như sau:

```
[
  {
    "iteration": 100,
    "detections_count": 5478687,
    "unique_truth_count": 1083,
    "classes": [
      {
        "id": 0,
        "name": "plate",
        "ap": 0.08,
        "tp": 186,
        "fp": 168565
      }
    ],
    "precision": 0.0,
    "recall": 0.17,
    "f1_score": 0.0,
    "tp": 186,
    "fp": 168565,
    "fn": 897,
    "avg_iou": 0.06,
    "map": 0.000773
  },

```

- Nhóm tiếp tục sử dụng thư viện [pandas](#) và [matplotlib](#) ([./scripts/Visualize\\_Error\\_Model\\_Yolo\\_Detect\\_License\\_VN.ipynb](#)) để **trực quan độ mAP** (mean Average Precision) và **avg\_iou** (Average Intersection over Union) **cho từng iteration**:



- Tiêu chí để chọn best model Plate là dựa trên tập validation, model phải có mAP cao nhất, nếu có nhiều model có cùng mAP cao nhất thì sẽ chọn model có avg\_iou cao nhất trong số đó. Như hình vẽ ta thấy được, **model tốt nhất ở iteration 5600** (mAP = 1.00 và avg\_iou = 94.97%).

- Nhìn vào hình vẽ, ta thấy **không có dấu hiệu bị overfitting**, thời gian train **tốn gần 2 ngày** với `max_batches = 6000`, *con số 6000 này nhóm chọn là bởi vì github darknet thì max\_batches sẽ là số class \* 2000, nhưng không được ít hơn số tấm ảnh dùng để train và không ít hơn 6000.*

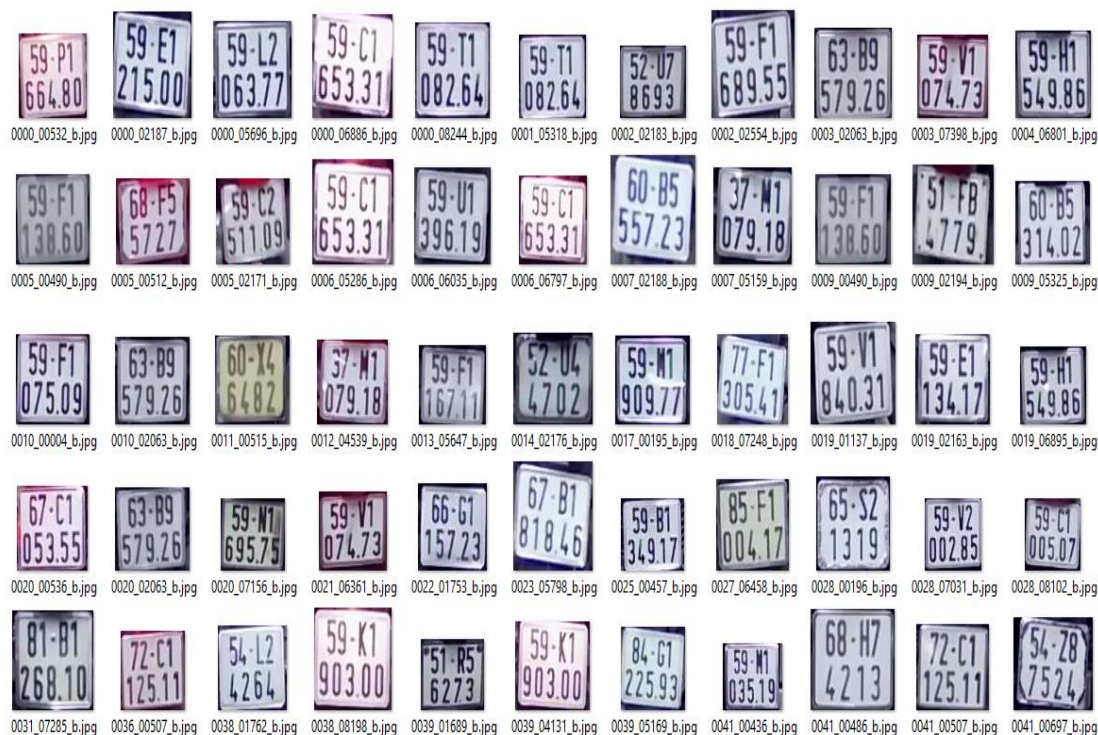
### Chạy best model Plate với tập dữ liệu test

Model Plate được chọn ở iteration 5600, chạy trên toàn bộ 350 tấm ảnh của tập test và kết quả thu được:

<b>Các giá trị độ đo</b> TP = 350 FP = 0 FN = 0	
<b>Độ đo</b>	<b>Giá trị</b>
precision	1.00
recall	1.00
F1-score	1.00
mAp	1.00
average IoU	94.72 %

❖ **Phase 2:** Huấn luyện model nhận diện từng ký tự trong biển số xe.**Quá trình rút trích biển số xe từ ảnh gốc**

- Dựa vào phần đánh nhãn dữ liệu Plate ở Phase 1, do đã có tọa độ chính xác của box bao các biển số trong hình, nên nhóm sẽ tiếp tục dựa vào đó để viết một **script python** (`./scripts/extract_plate.py`) để rút trích tất cả các biển số có trong từng tấm ảnh.
- Kết quả thu được như bên dưới:

**Quá trình đánh nhãn dữ liệu**

- Nhóm sử dụng **labellmg** để thực hiện đánh nhãn **1748 biển số** (ở mỗi biển số sẽ có từ 8 đến 9 ký tự tùy vào biển số, bao gồm ký tự chữ cái và ký tự chữ số).
- Mã nguồn labellmg: <https://github.com/tzutalin/labellmg>
- **Thời gian thực hiện đánh nhãn** cho toàn bộ dữ liệu ở cả 3 tập train, validation và test là **7 ngày**.



**Tổ chức huấn luyện dữ liệu trên colab:**

Giống như ở Phase 1, ta cũng có 3 bước chính là:

**Bước 1. Khâu chuẩn bị:**

- Ta cũng tổ chức một **folder characters** (folder này sẽ đặt trong thư mục **darknet/data** trên colab) theo format bên dưới:

**characters**

```
|- train/
|- validation/
|- test/
|- characters.data
|- characters.names
|- yolov4-train.cfg
|- train.txt
|- validation.txt
|- test.txt
```

- + Các thư mục train, validation, test hoàn toàn tương tự như bên Plate, nhưng ở đây là chứa ảnh các biển số được cắt ra và file đánh nhãn kèm theo.
- + Tập plate.data:

```
classes = 36
train = data/characters/train.txt
valid = data/characters/validation.txt
names = data/characters/characters.names
backup = /mydrive/yolov4/characters/backup
```

- + Tập plate.names:

```
0
1
2
...
X
Y
Z
```

- + Tập yolov4-train.cfg, được sửa lại như sau:

Sửa	Mục đích
width = 416 height = 416	Giảm width với height để train nhanh hơn, là bội số của 32 (nhưng có thể làm giảm performance của model).
max_batches = 72000	<ul style="list-style-type: none"> <li>Sửa max_batches = 72000 = max([số class * 2000], [số ảnh dùng để train], [6000]).</li> </ul>

steps = 57600, 64800	<ul style="list-style-type: none"> <li>steps = 80% và 90% của max_batches.</li> </ul>
classes = 36 filters = 123	<ul style="list-style-type: none"> <li>Sửa classes = 26 ở 3 section [yolo].</li> <li>Sửa filters = 123 = (số class + 5)*3 ở 3 section [convolutional] ở ngay trên section [yolo].</li> </ul> <div> <div> <pre> [convolutional] size=1 stride=1 pad=1 filters=255 activation=linear  [yolo] mask = 0,1,2 anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, classes=80 num=9 </pre> </div> <div> <pre> [convolutional] size=1 stride=1 pad=1 filters=123 activation=linear  [yolo] mask = 0,1,2 anchors = 12, 16, classes=36 num=9 </pre> </div> </div>

- + Tập train.txt chứa đường dẫn tới ảnh dùng để train, mỗi ảnh một dòng, ví dụ: data/characters/train/0255\_03013\_b.jpg
- + Tương tự cho tập validation.txt và test.txt.

- Vẫn dùng pre-trained weights-file để training:

[https://github.com/AlexeyAB/darknet/releases/download/darknet\\_yolo\\_v3\\_optimal/yolov4.conv.137](https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.conv.137)

## Bước 2. Tiến hành train dữ liệu:

- Chạy lệnh trên colab:

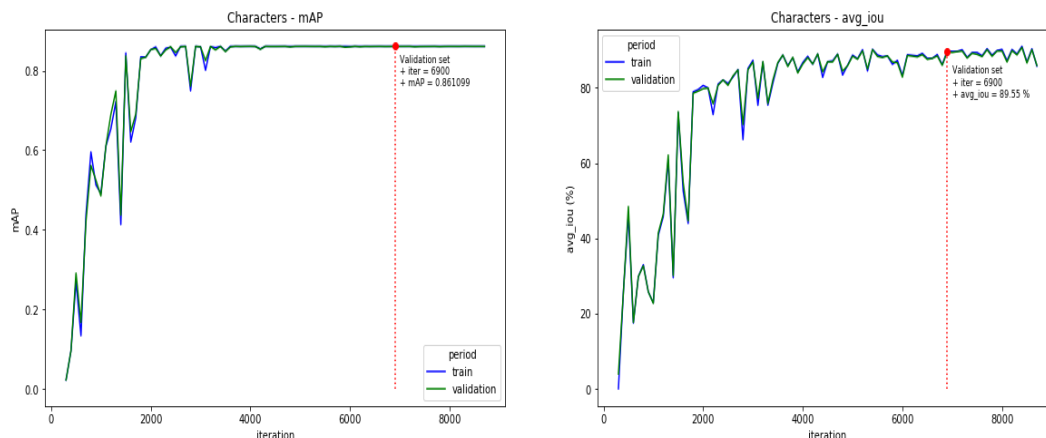
```
./darknet detector train data/characters/characters.data data/characters/yolov4-
.cfg yolov4.conv.137
```

- Thời gian chạy xong 8700 iteraters** mà nhóm đã thử nghiệm là **gần 6 ngày**. Trong file yolov4-train.cfg nhóm có ghi là max\_batches = 72000, nhưng ở đây nhóm đã cho dừng việc train model ở iteraters 8700 là vì một số lý do sẽ giải thích bên dưới.

## Bước 3. Tính mAP để chọn best model Plate:

Các thao tác hoàn toàn tương tự như bên Phase 1, và kết quả thu được:

- Ở đây có **87 version** cho **weight**.
- Và kết quả trực quan:



- Tiêu chí để chọn best model Characters là dựa trên tập Validation, model phải có mAP cao nhất, nếu có nhiều model có cùng mAP cao nhất thì sẽ chọn model có avg\_iou cao nhất trong số đó. Như hình vẽ ta thấy được, **model tốt nhất ở iteration 6900** (mAP = 0.861 và avg\_iou = 89.55%).
- Nhìn vào hình vẽ, ta thấy **không có dấu hiệu bị overfitting**.

### Lý giải lý do dừng train model sớm:

Sở dĩ ban đầu nhóm muốn huấn luyện với max\_batches = 72000, nhưng ở đây nhóm đã quyết định dừng huấn luyện model tại iterations là vì hai lý do chính:

- Một là, khi vẽ ra biểu đồ trực quan mAP và avg\_iou trên tập train và validation, nhóm thấy rằng model đã fix khá tốt với tập train và cho ra kết quả dự đoán trên tập validation cũng khá cao, với mAP = 0.861099 (lưu ý ở đây có tới 36 class nên mAP là rất ổn) và avg\_iou = 89.55% (cho biết việc tìm ra box chứa các ký tự trong biển số cũng khá chính xác). Đồng thời qua khoảng 7000 iterations thì mAP dường như không cải thiện được gì hơn và avg\_iou bắt đầu nhảy lên xuống dao động quanh một giá trị khoảng từ 85 đến 90. Tới đây model cũng đã rất tốt, có thể đem ra dự đoán ngoài thực tế.
- Hai là việc train model trên colab, phải yêu cầu dữ liệu truy cập vào file colab của mình liên tục, nhưng do vấn đề mạng internet bị chập chờn và việc di chuyển máy tính thường xuyên của nhóm (do các thành viên trong nhóm đã đi làm) nên tạo sự bất tiện, khó khăn cho việc tiếp tục train model. Đồng thời mới chỉ train tới 8700 iterations mà đã tốn đến 7 ngày, trong khi đó thời gian nộp đồ án sắp diễn ra và còn phải viết một web app để hiện thị kết quả. Nên nhóm đã quyết định dừng tại iterations 8700.

**Chạy best model Plate với tập dữ liệu test**

Model Plate được chọn ở iteration 5600, chạy trên toàn bộ 350 tấm ảnh chỉ là biển số (mỗi tấm ảnh trong đó có thể chứa 8 hay 9 ký tự) của tập test và kết quả thu được:

Các giá trị độ đo	
TP = 3031	
FP = 17	
FN = 3	
Độ đo	Giá trị
precision	0.99
recall	1.00
F1-score	1.00
mAp	0.8608
average IoU	89.06 %

### 3.3 Xây dựng ứng dụng Web

Toàn bộ mã nguồn và data liên quan nằm tại: `./src` và `./data`

#### Các tính năng của trang web:

- + Server khởi động, load model lên một lần và dùng cho tất cả các lần phân lớp.
- + User upload file ảnh thông qua giao diện web.
- + Server nhận data của ảnh, gửi tới bộ classifier với model load sẵn để detect ảnh.
- + Server trả kết quả về cho user.
- + Trang web được mô tả như sau:
  - Có một trang chủ cho phép user chọn nhận diện bằng model train sẵn của **YOLO** hay sử dụng model được train bởi nhóm.
  - Tùy thuộc vào lựa chọn của user mà nhảy đến trang tương ứng để user thực hiện quá trình upload ảnh, và server sẽ tiến hành detect ảnh rồi hiển thị kết quả lên web, user có thể download ảnh về local.

#### Công cụ sử dụng:

- + Backend: Flask.
- + Frontend: HTML, CSS, JavaScript.
- + Model có sẵn của YOLO (yolo-v4) và 2 model được train bởi nhóm (1 cái để detect biển số và 1 cái để detect các ký tự có mặt trong biển số).

#### Chạy website:

**Bước 1.** Cài đặt Flask:

```
pip install flask
```

**Bước 2.** Clone mã nguồn:

```
git clone https://github.com/dpnam/SL-Final-Project  
cd SL-Final-Project/src
```

**Bước 3.** Download 3 model cần thiết ([yolov4-characters.weights](#), [yolov4-plate.weights](#) và [yolov4.weights](#)) tại phần Releases của repository (<https://github.com/dpnam/SL-Final-Project/releases/tag/v1.0.0>) và bỏ vào `./src/models/`

**Bước 4.** Chạy app:

- Trên linux:

```
export FLASK_ENV=development
export FLASK_APP=main.py
python -m flask run
```

- Trên Windows:

- Command Prompt:

```
set FLASK_ENV=development
set FLASK_APP=main.py
python -m flask run
```

- PowerShell:

```
$env:FLASK_ENV = "development"
$env:FLASK_APP = "main.py"
python -m flask run
```

**Cấu trúc chương trình:**

```
models
├── characters.data
├── characters.names
├── coco.data
├── coco.names
├── plate.data
├── plate.names
├── yolov4-characters.cfg
├── yolov4-characters.weights
├── yolov4-plate.cfg
├── yolov4-plate.weights
├── yolov4.cfg
└── yolov4.weights
static
├── css
│   ├── # detector.css
│   └── # index.css
├── js
│   └── js detector.js
└── templates
    ├── <> detector.html
    └── <> index.html
✓ Session.vim
⚙ darknet.py
⚙ darknet_detector.py
≡ libdarknet.so
⚙ main.py
≡ pthreadGC2.dll
≡ pthreadVC2.dll
≡ yolo_cpp_dll_no_gpu.dll
```

**Thư mục models:**

```
| yolov4.weights
| yolov4.cfg
| coco.data
| coco.names
```

→ Đây là các tệp cần thiết để load model có sẵn của yolo.

```
| yolov4-plate.weights
| yolov4-plate.cfg
| plate.data
| plate.names
```

→ Đây là các tệp cần thiết để load model nhận diện biển số xe.

```
| yolov4-characters.weights
| yolov4-characters.cfg
| characters.data
| characters.names
```

→ Đây là các tệp cần thiết để load model nhận diện các ký tự trên biển số xe.

**File darknet.py:** là tệp hỗ trợ gọi các hàm từ thư viện C của darknet (tệp libdarknet.so cho linux và yolo\_cpp\_dll\_no\_gpu.dll cho windows)

**File darknet\_detector.py:**

- **Class Detection** với 3 field là:

```
class Detection:
    def __init__(self, name, confidence, bbox):
        self.name = name
        self.confidence = confidence
        self.bbox = bbox
```

- name: tên class của object detect được (là một trong các tên được định nghĩa trong file \*.names).
- bbox: bounding box của object detect được, với format là (left, top, right, bottom).
- confidence: độ tự tin/độ chính xác của detection này.

- **Class DarknetDetector** gồm 2 hàm chính:

+ Hàm `__init__`:

```
class DarknetDetector:
    def __init__(self, config_path, weight_path, meta_path):
        self.net_main = darknet.load_net_custom(
            config_path.encode("ascii"), weight_path.encode("ascii"), 0, 1
        )
        self.meta_main = darknet.load_meta(meta_path.encode("ascii"))
        self.network_width = darknet.network_width(self.net_main)
        self.network_height = darknet.network_height(self.net_main)
```

- Dùng để load model khi một object của class này được khởi tạo.
- Gồm 3 tham số:
  - config\_path: đường dẫn tới file \*.cfg.
  - weight\_path: đường dẫn tới file \*.weights.
  - meta\_path: đường dẫn tới file \*.data.
- Gồm các field:
  - net\_main: model đã được load.
  - net\_meta: metadata của model (gồm số class, tên các classes, ...).
  - network\_width, network\_height: kích thước ảnh dùng trong lúc train model (416x416).



+ Hàm **detect** :

```
def detect(self, im):
    im_rgb = cv2.cvtColor(im, cv2.COLOR_BGR2RGB)
    im_resized = cv2.resize(
        im_rgb,
        (self.network_width, self.network_height),
        interpolation=cv2.INTER_LINEAR,
    )

    darknet_image = darknet.make_image(
        self.network_width, self.network_height, 3
    )
    darknet.copy_image_from_bytes(darknet_image, im_resized.tobytes())

    yolo_detections = darknet.detect_image(
        self.net_main, self.meta_main, darknet_image, thresh=0.75
    )

    detections = DarknetDetector.yolo_detections_to_detections(yolo_detections)

    h, w = im.shape[:2]
    width_ratio = w / self.network_width
    height_ratio = h / self.network_height

    scale = (width_ratio, height_ratio, width_ratio, height_ratio)

    for detection in detections:
        detection.bbox = tuple(int(l * r) for l, r in zip(detection.bbox, scale))

    return detections
```

- Dùng để nhận diện object có trong 1 tấm ảnh.
- Gồm 1 tham số:
  - im: là ảnh đã được load bằng **OpenCV** (có kiểu `numpy.ndarray`).
- Quá trình detect:
  1. Do **OpenCV** lưu ảnh trong bộ nhớ ở dạng BGR nên ta sẽ chuyển về RGB trước.
  2. Resize ảnh về kích thước dùng trong lúc train (network\_width x network\_height).
  3. Gọi hàm `darknet.make_image` để cấp phát vùng nhớ cho tấm ảnh sẽ được dùng để detect trong darknet.
  4. Copy ảnh từ **OpenCV** vào `darknet_image` vừa cấp phát.
  5. Gọi hàm `darknet.detect_image` để detect các object có trong ảnh.
  6. Gọi hàm `DarknetDetector.yolo_detections_to_detections` để chuyển kết quả trả về từ hàm `detect_image` sang **class Detection** được định nghĩa ở trên.
  7. Do tọa độ bbox trả về sẽ tương ứng với ảnh sau khi được resize (network\_width x network\_height), nên ta sẽ chỉnh lại tọa độ này về ứng với kích cỡ của ảnh gốc.
  8. Trả về **list Detection** chứa các object có trong ảnh.

**File main.py** làm 2 nhiệm vụ chính:

- Một là khởi tạo 3 object **DarknetDetector** là `yolo_detector`, `plate_detector` và `characters_detector`. Đây cũng là lúc 3 models này sẽ được load lên:

```
app = Flask(__name__)

yolo_detector = DarknetDetector(
    config_path="./models/yolov4.cfg",
    weight_path="./models/yolov4.weights",
    meta_path="./models/coco.data",
)

plate_detector = DarknetDetector(
    config_path="./models/yolov4-plate.cfg",
    weight_path="./models/yolov4-plate.weights",
    meta_path="./models/plate.data",
)

characters_detector = DarknetDetector(
    config_path="./models/yolov4-characters.cfg",
    weight_path="./models/yolov4-characters.weights",
    meta_path="./models/characters.data",
)
```

- Hai là định nghĩa các flask routes:

- route '/' sẽ trả về trang `templates/index.html`, trang này gồm 2 buttons dẫn tới trang dùng model có sẵn của yolo và model nhận diện biển số xe tương ứng.

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

- route '/yolo' sẽ trả về trang `templates/detector.html` với **title** là **YOLOv4**.

```
@app.route('/yolo')
def yolo():
    return render_template('detector.html', title="YOLOv4")
```

- route '/plate' sẽ trả về trang `templates/detector.html` với **title** là **Plate**.

```
@app.route('/plate')
def plate():
    return render_template('detector.html', title="Plate")
```

- route `/yolo/upload` sẽ nhận ảnh user upload lên từ trang `/yolo`, gửi ảnh này vào model `yolo_detector` đã load lúc đầu để `detect`, sau đó vẽ `boundingbox` và tên của các object detect được lên ảnh và trả về cho user ảnh này.

```
@app.route('/yolo/upload', methods = ["POST"])
def process_yolo():
    global yolo_detector
    f = request.files['image'].read()
    im = cv2.imdecode(np.frombuffer(f, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
    detections = yolo_detector.detect(im)
    for detection in detections:
        x1, y1, x2, y2 = detection.bbox
        cv2.rectangle(im, (x1, y1), (x2, y2), (255, 0, 0), 1)
        cv2.putText(
            im, detection.name, (x1, y2), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 145), 1
        )

    buf = cv2.imencode(".jpg", im)
    response = make_response(buf.tobytes())
    response.headers["Content-Type"] = "image/jpeg"
    return response
```

- route `/plate/upload` sẽ nhận ảnh user upload lên từ trang `/plate`, gửi ảnh này vào model `plate_detector` đã load lúc đầu để `detect` các biển số xe có trong ảnh, sau đó trích các biển số xe detect được và gửi vào model `characters_detector` đã load lúc đầu để `detect` các ký tự có trong biển số. Cuối cùng thì vẽ `bounding box` của biển số và các ký tự detect được, cùng với tên của các ký tự đó lên ảnh và trả về cho user ảnh này.

```
@app.route('/plate/upload', methods = ["POST"])
def process_plate():
    global plate_detector
    global characters_detector

    f = request.files['image'].read()
    im = cv2.imdecode(np.frombuffer(f, dtype=np.uint8), cv2.IMREAD_UNCHANGED)
    detections = plate_detector.detect(im)
    for detection in detections:
        x1, y1, x2, y2 = detection.bbox
        cv2.rectangle(im, (x1, y1), (x2, y2), (0, 255, 0), 1)
        plate_im = im[y1:y2, x1:x2]
        chars = characters_detector.detect(plate_im)
        draw_chars(im, chars, detection.bbox)

    buf = cv2.imencode(".jpg", im)
    response = make_response(buf.tobytes())
    response.headers["Content-Type"] = "image/jpeg"
    return response
```

```
def draw_chars(im, chars, plate_bbox):~
    y_min = min(char.bbox[1] for char in chars)~
    y_max = max(char.bbox[1] for char in chars)~
    middle = (y_min + y_max) / 2~
    for char in chars:~
        x1, y1, x2, y2 = char.bbox~
        top = y1 <= middle~

        x1 += plate_bbox[0]~
        y1 += plate_bbox[1]~
        x2 += plate_bbox[0]~
        y2 += plate_bbox[1]~
        h = y2 - y1~

        cv2.rectangle(im, (x1, y1), (x2, y2), (255, 0, 0), 1)~

    if top:~
        drawToRect(im, char.name, (x1, plate_bbox[1] - h, x2, plate_bbox[1]))~
    else:~
        drawToRect(im, char.name, (x1, plate_bbox[3], x2, plate_bbox[3] + h))~
```

**File detector.js:**

- File này được load bởi file [detector.html](#).
- Có hàm chính là hàm **upload**, hàm này sẽ được gọi khi người dùng chọn/kéo thả ảnh từ giao diện web:

```
function upload(input) {~
    if (input.files && input.files[0]) {~
        clearAll();~

        const img = createImage(input.files[0], "Uploaded Image");~
        originalImageContainer.append(img);~

        const loader = document.createElement('div');~
        loader.className = 'loader';~
        resultImageContainer.append(loader);~

        const formData = new FormData();~
        formData.append('image', input.files[0]);~

        fetch(`${window.location.pathname}/upload`, {~
            method: 'POST', body: formData, signal: controller.signal })~
            .then(response => response.blob())~
            .then(blob => {~
                const img = createImage(blob, "Result Image");~

                removeChildren(resultImageContainer);~
                resultImageContainer.append(img);~
            })~
            .catch(error => {~
                console.error('Fetch error:', error);~
            });~
    }~
}
```

Quá trình làm việc của hàm **upload**:

1. Chạy hàm **clearAll** xóa ảnh gốc và ảnh kết quả nếu có.
2. Hiển thị ảnh người dùng vừa **upload** lên giao diện web.
3. Hiện thị **icon loading** cho người dùng biết là ảnh kết quả đang được xử lý.
4. Dùng hàm **fetch** để gửi ảnh lên server. Sau khi nhận được ảnh kết quả trả về từ server thì sẽ cho hiển thị lên giao diện web.

**File `css/index.css`** dùng để làm đẹp trang [index.html](#).

**File `css/detector.css`** dùng để làm đẹp trang [detector.html](#).

### Một số hình ảnh demo:

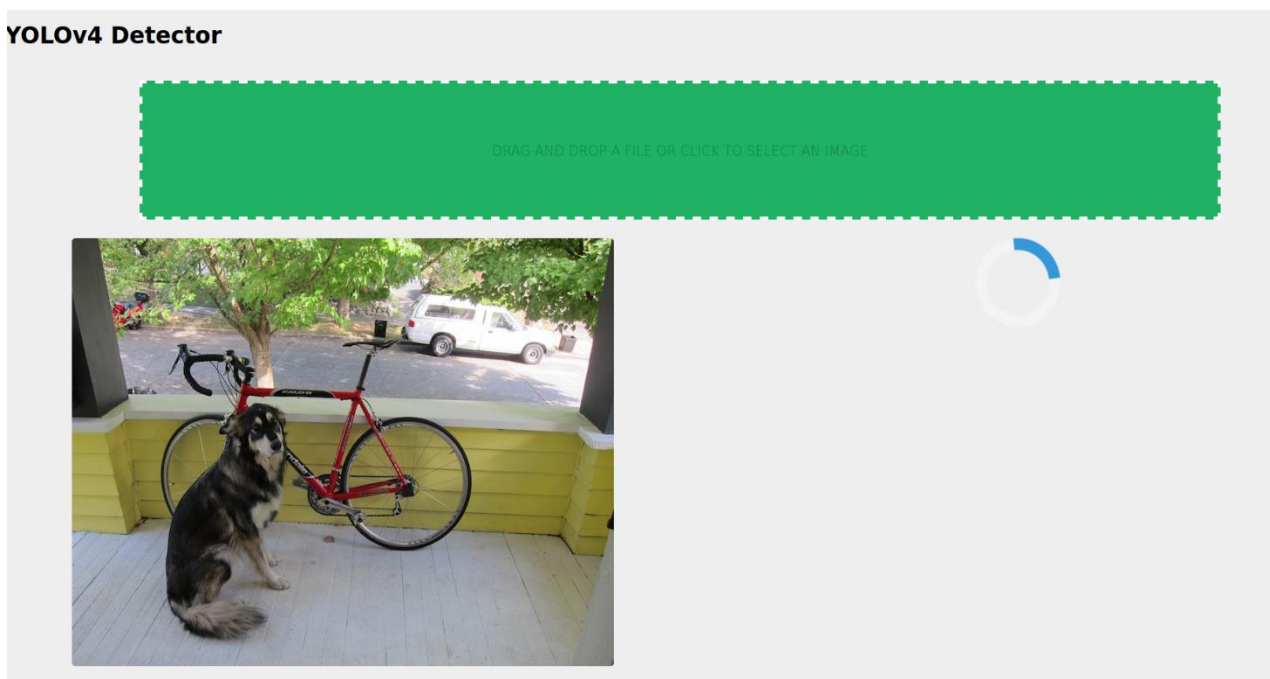
**Trang chính:**

Final Project



**Giao diện sử dụng model đã train sẵn của YOLO để nhận diện ảnh:**

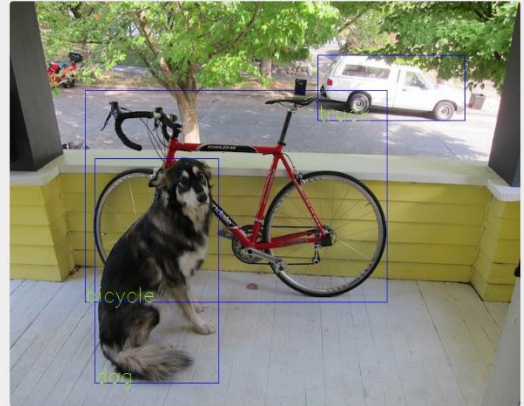
YOLOv4 Detector



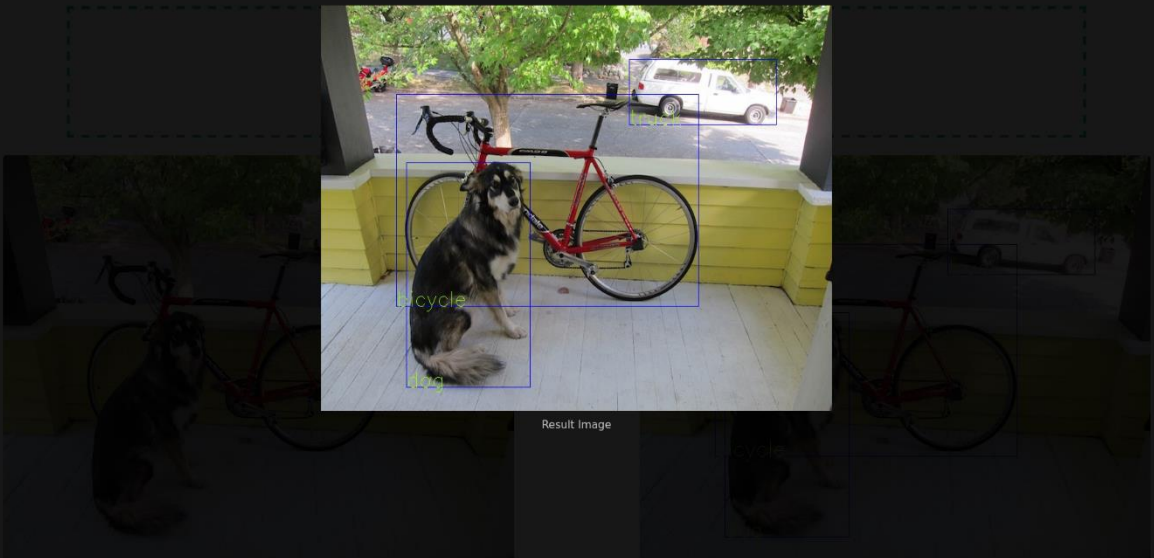


YOLOv4 Detector

DRAG AND DROP A FILE OR CLICK TO SELECT AN IMAGE



YOLOv4 Detector

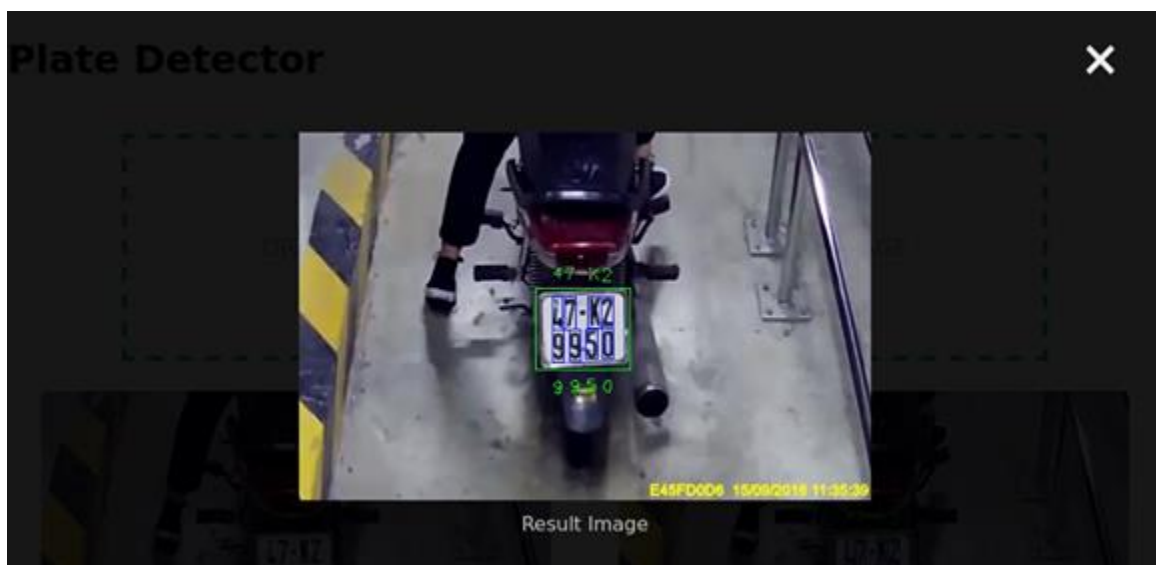


**Giao diện sử dụng model nhận diện biển số xe và từng ký tự trong biển số xe của nhóm:**

### Plate Detector



### Plate Detector





## 4 TÀI LIỆU THAM KHẢO

- [1] <https://github.com/AlexeyAB/darknet#how-to-train-to-detect-your-custom-objects>
- [2] <https://speckyboy.com/custom-file-upload-fields/>
- [3] <https://flask.palletsprojects.com/en/1.1.x/quickstart/>
- [4] <https://auth0.com/blog/developing-restful-apis-with-python-and-flask/>
- [5] <https://pjreddie.com/darknet/yolo/>