ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

CÔNG NGHỆ PHẦN MỀM - EXTRA

Báo cáo

Ứng dụng AI trong thiết kế phần mềm: Bài toán ICON DETECTION

GVHD: PGS.TS. Quản Thành Thơ
SV: Thái Văn Nhật - 1813381

TP. HỒ CHÍ MINH, THÁNG 7/2021

Mục lục

1 Giới thiệu đề tài

Ngày nay, với sự bùng nổ của cuộc cách mạng công nghiệp 4.0, quá trình công nghiệp hoá, hiện đại hoá ngày càng phát triển đã dẫn đến nhu cầu về cuộc sống, sinh hoạt, giải trí của con người phải được đáp ứng một cách mạnh mẽ. Các ứng dụng phần mềm cũng được sinh ra từ đây, nhắm hướng đến việc con người có thể dễ dàng tiếp cần và sử dụng một cách nhanh chóng, dễ hiểu và tiện lợi. Do đó, hầu hết các ứng phần mềm đều do con người lập trình và hoàn thiện. Tuy nhiên, ngày nay, với sự phát triển mạnh mẽ trong lĩnh vực trí tuệ nhân tạo, ta có thể thấy con người đang hướng đến máy tính trong tương lai có thể "software writes a software", tức là máy tính với trí tuệ nhân tạo của chúng học được có thể tự thiết kế ra một giao diện phần mềm hoàn chỉnh và có thể đáp ứng tốt được nhu cầu của cuon người.

Trong đề tài lần này, mục tiêu của em chỉ tác động một phần nhỏ đến công việc đã đề cập ở trên. Mục tiêu chính là có thể phát hiện được các icon khác nhau cùng trên một giao diện phần mềm thông qua sử dụng trí tuệ nhân tạo. Do số lượng các icon là tương đối lớn nên trong đề tài lần này, em chỉ detect 5 loại icon, đó là HOME, SETTINGS, SEARCH, MENU, VERTICAL_MORE.

Do đây là đề tài đầu tiên về AI mà em nghiên cứu và tìm hiểu, chắc chắn sẽ không tránh khỏi những sai sót. Kính mong thầy có thể góp ý để đề tài này có thể được hoàn thiện hơn!

2 Phương pháp tiếp cận và phương án giải quyết vấn đề

Sau một thời gian tìm hiểu, em nhận thấy đề tài này có nét tương đồng với bài toán Object detection - một trong những bài toán kinh điển trong lĩnh vực AI. Do đó, em quyết định sử dụng giải thuật YOLOv4 - được xây dựng bởi AlexeyAB - để training tập dữ liệu của mình.

2.1 YOLO

Yolo là một mô hình mạng CNN cho việc phát hiện, nhận dạng, phân loại đối tượng. Yolo được tạo ra từ việc kết hợp giữa các convolutional layers và connected layers.Trong đóp các convolutional layers sẽ trích xuất ra các feature của ảnh, còn full-connected layers sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.

Kiến trúc YOLO bao gồm: base network là các mạng convolution làm nhiệm vụ trích xuất đặc trưng. Phần phía sau là những Extra Layers được áp dụng để phát hiện vật thể trên feature

map của base network. Base network của YOLO sử dụng chủ yếu là các convolutional layer và các fully conntected layer. Các kiến trúc YOLO cũng khá đa dạng và có thể tùy biến thành các version cho nhiều input shape khác nhau.

Sau khi xác định được giải thuật, bước tiếp, em sẽ chuẩn bị tập dataset cho quá trình training.

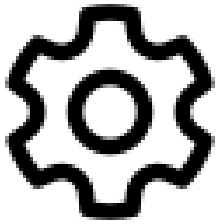
2.2 Chuẩn bị dataset

Như đã đề cập ở trên, trong đề tài này, em chỉ lấy 5 loại icon phổ biến trên các giao diện phầm mềm hiện nay.

Hình 1: Icon home

Hình 2: Icon search

Hình 3: Icon menu



Hình 4: Icon settings

Hình 5: Icon vertical more

Các icon này đều có background ở dạng transparent và được lấy từ nguồn sau https://icons8.com/

Sau khi có được những icon tiêu chuẩn, em sử dụng thư viện Image Augmentation (imgaug) để sinh thêm nhiều dữ liệu cho quá trình traning. Code của tác vụ này được thể hiện dưới đây

```
import imageio
import imgaug as ia
from imgaug import augmenters as iaa

STANDARD_SIZE = 74

seq_0 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE)
])

seq_1 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    iaa.GaussianBlur(sigma=0.2),
    iaa.LogContrast(gain=0.75)
])
```

```
seq_2 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    # iaa.MedianBlur(k=5),
    iaa.Add(value=-45),
    # iaa.Fog(),
    # iaa.Affine(cval=3)
])

seq_3 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    # iaa.MotionBlur(angle=30),
    iaa.Add(value=-45),
    iaa.GammaContrast(gamma=1.44),
    iaa.pillike.EnhanceSharpness(factor=0.15),
    # iaa.Clouds(),
    iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0))
])

seq_4 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    iaa.PiecewiseAffine(scale=0.03),
    # iaa.Affine(cval=20),
    iaa.Pepper(p=(0.05, 0.1))
])

seq_5 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    # iaa.AveragePooling(kernel_size=4),
    iaa.AdditiveGaussianNoise(loc=0, scale=(0.0, 0.05*255), per_channel=0.5),
    iaa.Emboss(alpha=(0, 1.0), strength=(0, 2.0))
])

seq_6 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    iaa.GaussianBlur(sigma=0.2),
    iaa.Dropout(p=0.1)
])
```

```
seq_7 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    # iaa.imgcorruptlike.Snow(severity=1),
    # iaa.Fog(),
    # iaa.GammaContrast(gamma=1.12)
    # iaa.GammaContrast((0.5, 2.0), per_channel=True)
    iaa.LogContrast(gain=(0.6, 1.4), per_channel=True),
    iaa.MedianBlur(k=3),
    iaa.Emboss(alpha=(0.0, 1.0), strength=(0.5, 1.5))
])

seq_8 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    # iaa.Cartoon(blur_ksize=2, segmentation_size=1.0, saturation=2.0)
    iaa.Add(value=(15, 35), per_channel=0.5),
    iaa.Sharpen(alpha=(0.0, 1.0), lightness=(0.75, 2.0))
])

seq_9 = iaa.Sequential([
    iaa.Resize(size=STANDARD_SIZE),
    iaa.ImpulseNoise(0.1),
    iaa.Flipud(0.5)
])


def generator(path):
    image = imageio.imread(path)
    image_aug = [
        seq_0(image=image),
        seq_1(image=image),
        seq_2(image=image),
        seq_3(image=image),
        seq_4(image=image),
        seq_5(image=image),
        seq_6(image=image),
        seq_7(image=image),
        seq_8(image=image),
    ]
```

```
    return image_aug
```

Theo đoạn chương trình trên, 1 icon tiêu chuẩn sẽ cho sinh ra thành 9 icon đã qua các bước xử lí của image augmentation. Tiếp đó, em sẽ lấy các icon tiêu chuẩn đã chuẩn bị để làm dữ liệu đầu vào cho quá trình phát sinh dữ liệu. Lưu ý rằng, đoạn code dưới đây xử lí với từng file icon riêng biệt chứa các loại icon tương ứng.

```
from generator_methods import generator
import imageio
import imgaug as ia
import time

NUM_OF_HOME_DATA = 10
NUM_OF_SETTINGS_DATA = 11
NUM_OF_SEARCH_DATA = 7
NUM_OF_VERT_MORE_DATA = 6
NUM_OF_MENU_DATA = 6

path = 'data/'

path_result = 'data/'

index = 1
index_output = 1


def process_label(label, maxData):
    global index
    global index_output

    index = 1
    while index <= maxData:
        images = generator(path + label + '/' + label + str(index) + '.png')
        for i in range(len(images)):
            imageio.imwrite(path_result + label + str(index_output) + '.png', images[i])
            index_output += 1
        index += 1
```

```
# Generating icon
process_label('home', NUM_OF_HOME_DATA)
print("Complete home data !!!")
process_label('settings', NUM_OF_SETTINGS_DATA)
print("Complete settings data !!!")
process_label('search', NUM_OF_SEARCH_DATA)
print("Complete search data !!!")
process_label('vert_more', NUM_OF_VERT_MORE_DATA)
print("Complete vertical more data !!!")
process_label('menu', NUM_OF_MENU_DATA)
print("Complete menu data !!!")
```

Sau khi đã có đầy đủ tập dataset, em tiến hành công đoạn đánh nhãn dữ liệu tương ứng với từng loại icon.

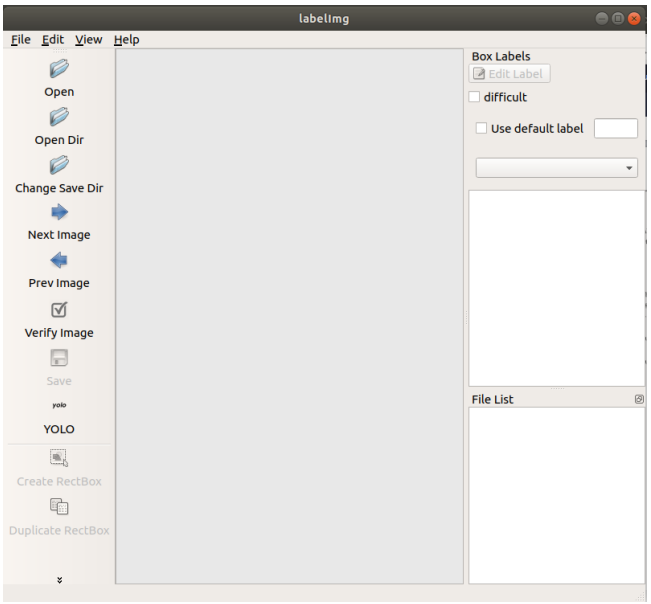
2.3 Đánh nhãn dữ liệu

Trong đề tài lần này, em sử dụng thư viện Image Label (labelImg) để tiến hành đánh nhãn cho từng data một. Việc này khá mất thời gian vì đây là công việc hoàn toàn thủ công, số lượng data rất nhiều nên đòi hỏi trong quá trình đánh nhãn cần phải có sự chính xác, tập trung và tính kiên nhẫn. Thư viện labelImg được tham khảo từ repository https://github.com/tzutalin/labelImg
Sau khi cài đặt thành công thư viện labelImg, ta đánh dấu nhãn cho dataset như sau:

1. Đầu tiên, chúng ta chạy tool labelImg, bằng cách gõ câu lệnh

```
python3 labelImg.py
```

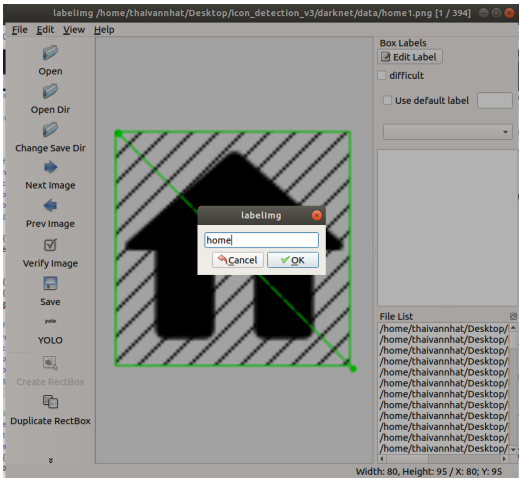
Sau đó, sẽ có giao diện xuất hiện như sau:



Hình 6: Giao diện của tool Label Img

2. Sau đó, chúng ta thiết lập các đường dẫn cho Open Dir và Change Save Dir. Trong đề tài này, chúng ta sẽ lưu chung file kết quả đánh label chung với dataset, tức là file này sẽ chung một đường dẫn.

3. Sau khi mở được đường dẫn, icon sẽ xuất hiện. Sau đó, chúng ta bấm phím **w** để bật chế độ bound box, đánh nhãn hết những vùng có chứa icon và lưu label đó tương ứng với loại icon mà chúng ta cần detect.



Hình 7: Đánh nhãn Home cho icon hiện tại

Cứ làm như vậy cho đến khi hết tập dataset, ta sẽ có được bộ label hoàn chỉnh.

2.4 Google Colab

Google Colab (Colaboratory) là một dịch vụ đám mây miễn phí, hiện nay có hỗ trợ GPU (Tesla K80) và TPU (TPUv2). Do được phát triển dựa trên Jupiter Notebook nên việc sử dụng Google Colab cũng tương tự như việc sử dụng Jupyter Notebook. Google Colab là một công cụ lý tưởng để chúng ta rèn luyện kĩ năng lập trình với ngôn ngữ Python thông qua các thư viện của deep learning. Google Colab cài đặt sẵn cho chúng ta những thư viện rất phổ biến trong nghiên cứu Deep Learning như PyTorch, TensorFlow, Keras và OpenCV.

Do trong đề tài này, lượng data khá nhiều và quá trình training để máy học được tập dữ liệu cũng khá lâu, một phần do máy em không có GPU nên việc training trên môi trường Colab cũng sẽ là một lực chọn hợp lí, vì nó miễn phí và có GPU. Tuy nhiên, khuyết điểm lớn nhất của Google Colab là session sẽ bị kill sau 10 tiếng, nên đòi hỏi trong quá trình train, người sử dụng phải liên tục chuẩn bị các phương án để lưu trữ cũng như backup kết quả

2.5 Sử dụng file config và Make file từ Open source YOLO của AlexeyAB

Trong đề tài này, em sử dụng YOLO v4 để tiến hành training. Do đó, cần phải có Makefile và Config file để có thể chạy thành công. Với config file, chúng ta có thể download tại link https://github.com/AlexeyAB/darknet/blob/master/cfg/yolov4-custom.cfg và Makefile tại link https://github.com/AlexeyAB/darknet/blob/master/Makefile.

Sau đó, chúng ta chỉnh sửa lại một vài thông tin như sau:
1. Trong Makefile, chúng ta sửa thông tin của **GPU, CUDNN, OPENCV** đều bằng 1. Do chúng ta train trên Google Colab, nên các thông số này sẽ cho phép sử dụng GPU.
2. Trong file config, chúng ta chỉnh sửa các thông số sau:

- Trong đề tài này, em train 5 class ứng với 5 loại icon cần được detect, cho nên cần sửa lại ở **dòng 20** *max_batches = 5*2000 = 10000*
- Ở **dòng 22**, sửa steps = 80%, 90% của max_batches
- Replace toàn bộ các dòng có "classes=80" thành "classes=<số class>". Tức là classes = 5
- Replace toàn bộ các dòng có "filters=255" thành "filters=<(số class+5)*3>". Tức là filters=30
- Nếu trong quá trình train bị out memory, chỉnh **dòng 7** thành 32 (hoặc 64)

Sau khi chuẩn bị xong dataset, config file và Makefile, em sẽ upload chúng lên Google Drive để chuẩn bị training

2.6 Sử dụng Colab bắt đầu training

Sau khi tạo thành công một Notebook mới trên Colab, em thiết lập chế độ Runtime type của nó là GPU. Sau đó, chúng ta tiến hành thực hiện các bước sau

1. Kết nối với Google Drive
Thực hiện các câu lệnh sau:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Sau khi xác nhận xong, chúng ta sẽ thấy dữ liệu từ Google Drive đã xuất hiện trong thư mục /content/gdrive/My Drive/

2. Tải mã nguồn YOLO v4 về Drive

```
!rm -rf darknet
%cd /content/gdrive/My\ Drive
!git clone https://github.com/AlexeyAB/darknet
%cd /content/gdrive/My\ Drive/darknet
!rm -rf data
!mkdir data
```

3. Upload file data, config file và Makefile đã chuẩn bị
Đầu tiên chúng ta zip toàn bộ thư mục data. Sau đó vào Google Drive và upload tât1 cả những file đã chuẩn bị

4. Giải nén file data.zip

```
%cd /content/gdrive/My\ Drive/darknet/data
!unzip data.zip
```

5. Tạo file yolo.names chứa các tên class

```
%cd /content/gdrive/My\ Drive/darknet
!echo "home" > yolo.names
!echo "search" >> yolo.names
!echo "settings" >> yolo.names
!echo "menu" >> yolo.names
!echo "vert_more" >> yolo.names
```

6. Tạo hai file train.txt và val.txt chứa danh sách các file ảnh
Trong giải thuật YOLO, chúng ta cần có dữ liệu để train và dữ liệu dùng để validate, đánh giá xem kết quả train như thế nào. Thông thường, người ta sẽ chia theo tỉ lệ 8-2, tức là 80% data giành cho training và 20% data dùng để validate. Do data chúng ta đã có sẵn, giờ chỉ viết code đoạn sau đây để tiến hành chia data

```
%cd /content/gdrive/My\ Drive/darknet

import glob2
import math
import os
import numpy as np

files = []
for ext in ["*.png", "*.jpeg", "*.jpg"]:
  image_files = glob2.glob(os.path.join("data/data/", ext))
  files += image_files

# Nếu không chọn chia tỉ lệ 8-2, chúng ta sẽ thay đổi tỉ lệ ở số 0.2 dưới đây
nb_val = math.floor(len(files)*0.2)
rand_idx = np.random.randint(0, len(files), nb_val)

# Tạo file train.txt
with open("train.txt", "w") as f:
  for idx in np.arange(len(files)):
    if (os.path.exists(files[idx][:-3] + "txt")):
      f.write(files[idx]+'\n')

# Tạo file vali.txt
with open("val.txt", "w") as f:
  for idx in np.arange(len(files)):
```

```
    if (idx in rand_idx) and (os.path.exists(files[idx][:-3] + "txt")):
      f.write(files[idx]+'\n')
```

7. Tạo file yolo.data chứa tham số train

```
%cd /content/gdrive/My\ Drive/darknet
!mkdir backup
!echo classes=1 > yolo.data
!echo train=train.txt >> yolo.data
!echo valid=val.txt >> yolo.data
!echo names=yolo.names >> yolo.data
!echo backup=backup >> yolo.data
```

8. Tiến hành biên dịch mã nguồn Darknet

```
%cd /content/gdrive/My\ Drive/darknet
!rm darknet
!make
```

9. Tải pretrain weights
Trong đề tài này, em tận dụng lại phần Convolution đã được train bởi Alexey. Download file weight bằng câu lệnh sau:

```
%cd /content/gdrive/My\ Drive/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/
yolov4.conv.137
```

10. Tiến hành training

```
%cd /content/gdrive/My\ Drive/darknet
!./darknet detector train yolo.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show
```

3 Kiểm thử kết quả thu được

Sau khi quá trình train dần hội tụ, chúng ta có thể dừng train và lấy file weight trong thư mục backup

Download file weight về, chúng ta tiến hành kiểm thử kết quả như sau:

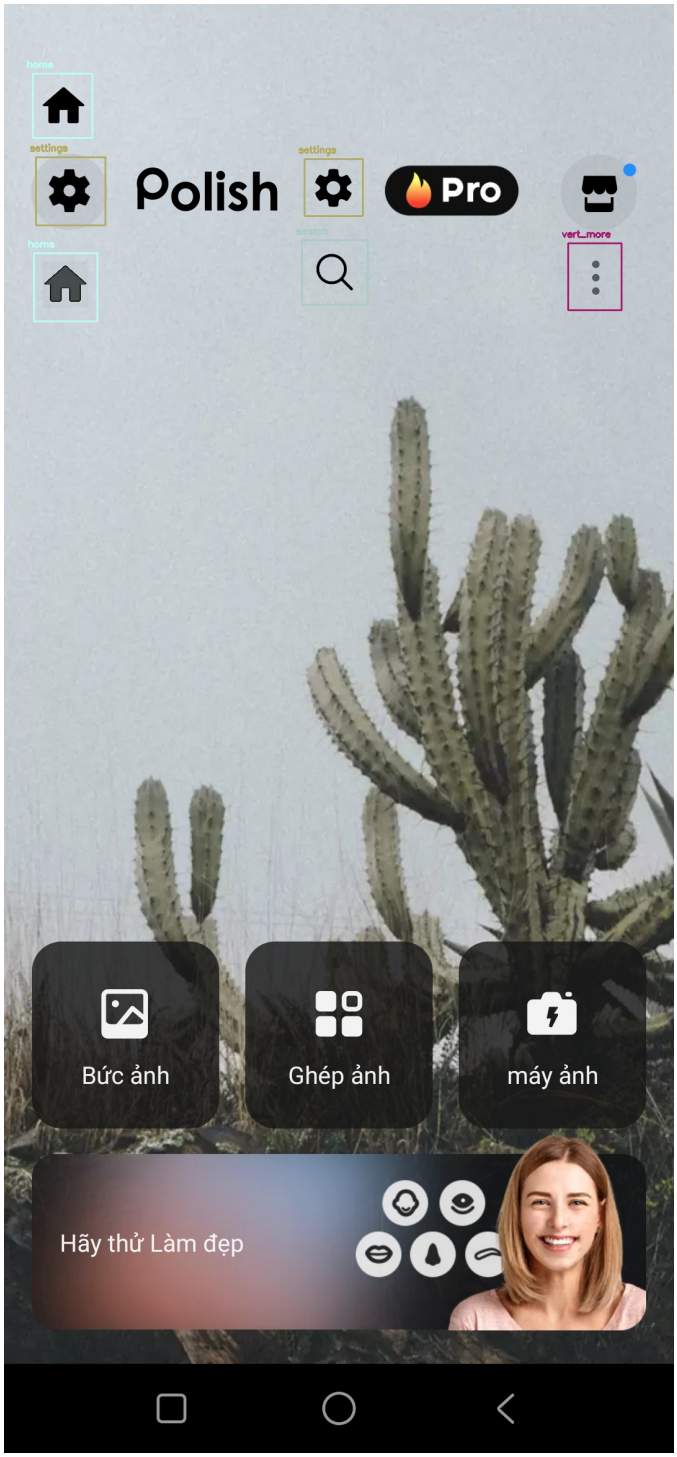
```
%cd /content/gdrive/My\ Drive/darknet
python YOLO.py -i test/test3.png -cl yolo.names -w backup/yolov4-custom_last.weights
-c cfg/yolov4-custom.cfg
```

Nội dung của file YOLO.py như sau:

```
import time
import cv2
import argparse
import numpy as np

ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True,
                help='path to input image')
ap.add_argument('-c', '--config', required=True,
                help='path to yolo config file')
ap.add_argument('-w', '--weights', required=True,
                help='path to yolo pre-trained weights')
ap.add_argument('-cl', '--classes', required=True,
                help='path to text file containing class names')
args = ap.parse_args()


def get_output_layers(net):
    layer_names = net.getLayerNames()

    output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

    return output_layers


def draw_prediction(img, class_id, confidence, x, y, x_plus_w, y_plus_h):
    label = str(classes[class_id])

    color = COLORS[class_id]
```

```
    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)

    cv2.putText(img, label, (x - 10, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)


image = cv2.imread(args.image)

Width = image.shape[1]
Height = image.shape[0]
scale = 0.00392

classes = None

with open(args.classes, 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416, 416), (0, 0, 0), True, crop=False)

net.setInput(blob)

outs = net.forward(get_output_layers(net))

class_ids = []
confidences = []
boxes = []
conf_threshold = 0.5
nms_threshold = 0.4

# Thực hiện xác định bằng HOG và SVM
start = time.time()

for out in outs:
    for detection in out:
```

```
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * Width)
            center_y = int(detection[1] * Height)
            w = int(detection[2] * Width)
            h = int(detection[3] * Height)
            x = center_x - w / 2
            y = center_y - h / 2
            class_ids.append(class_id)
            confidences.append(float(confidence))
            boxes.append([x, y, w, h])

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf_threshold, nms_threshold)

for i in indices:
    i = i[0]
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]
    print(round(x), round(y), round(x + w), round(y + h), str(classes[class_ids[i]]))
    draw_prediction(image, class_ids[i], confidences[i], round(x), round(y),
    round(x + w), round(y + h))

cv2.imshow("object detection", image)


end = time.time()
print("YOLO Execution time: " + str(end-start))


cv2.waitKey()

cv2.imwrite("object-detection.jpg", image)
cv2.destroyAllWindows()
```



Hình 8: Kết quả thu được

Thông tin về từng bound box được thể hiện dưới đây:

```
480 381 586 485 search
484 250 578 342 settings
51 247 163 357 settings
```

```
46 112 142 216 home
909 386 995 494 vert_more
48 402 150 512 home
YOLO Execution time: 13.523146152496338
```

Tài liệu

[1] *[YOLO Series] Train YOLO v4 train trên COLAB chi tiết và đầy đủ (A-Z)*, Nguyễn Cao Thắng, truy cập ngày 08/07/2021 tại https://www.miai.vn/2020/05/25/yolo-series-train-yolo-v4-train-tren-colab-chi-tiet-va-day-du-a-z/

[2] *Darknet*, AlexeyAB, truy cập ngày 08/07/2021 tại https://github.com/AlexeyAB/darknet