Convolutional Neural Network on Visual Recognition

Nguyen Dai Dong - M10707823
Andi Rahmadiansah - D10707829
Willybrordus Harsa Prasetya Muda - M10707825

I. INTRODUCTION

With the dawn of a new era of A.I., machine learning, and robotics, its time for the machines to perform tasks characteristic of human intelligence. Machines use their own senses to do things like planning, pattern recognizing, understanding natural language, learning and solving problems. And Image Recognition is one of its senses!!!

From Automated self-driven cars to Boosting augmented reality applications and gaming, from Image and Face Recognition on Social Networks to Its application in various Medical fields, Image Recognition has emerged as a powerful tool and has become a vital for many upcoming inventions.

II. System Architecture

The system use Yolo and Darknet as the algorithm and network. The dataset is combined from coco dataset and NTUST campus dataset. The images, video and real-time object will be import into system to get the result. We can evaluate the output of system and from the result we can enhance the system step by step. The figure 1 show the system architecture

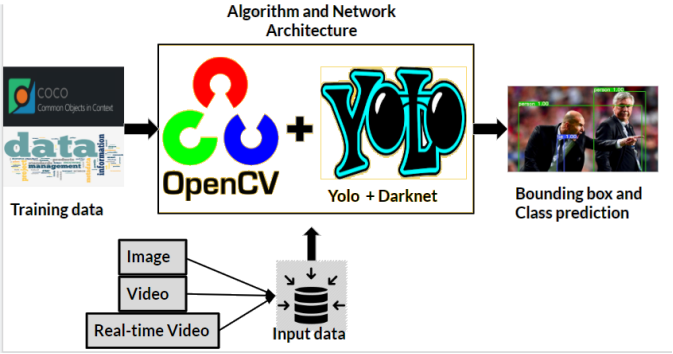


Figure 1: System architecture

III. TOOL AND TECHNOLOGY

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications, that aims to simplify package management and deployment.

VS Code is a free, open source streamlined cross-platform code editor with excellent support for Python code editing, IntelliSense, debugging, linting, version control, and more. Additionally, the Python Extension for Visual Studio Code tailors VS Code into a Python IDE. For Anaconda users, VS Code is a great IDE choice on Windows, mac OS, or Linux. Free for private or commercial use, VS Code is lightweight and fast, yet still offers debugging, code completion, and Git integration. It is also openly extensible—users can choose from a long list of additional extensions to tailor it to their specific needs. VS Code provides developers with the tools they need for a quick code-build-debug cycle.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code

OpenCV dnn module - DNN (Deep Neural Network) module was initially part of opencv_contrib repo. It has been moved to the master branch of opencv repo last year, giving users the ability to run inference on pre-trained deep learning models within OpenCV itself.

IV. DATASET AND TRAINING CUSTOM DATASET

A. Coco Dataset

The Microsoft Common Objects in COntext (MS COCO) dataset contains 91 common object categories with 82 of them having more than 5,000 labeled instances, Fig. 6. In total the dataset has 2,500,000 labeled instances in 328,000 images. In contrast to the popular ImageNet dataset [1], COCO has fewer categories but more instances per category. This can aid in learning detailed object models capable of precise 2D localization. The dataset is also significantly larger in number of instances per category than the PASCAL VOC [2] and SUN [3] datasets. Additionally, a critical distinction between our dataset and others is the number of labeled instances per image which may aid in learning contextual information, Fig. 5. MS COCO contains considerably more object instances per image (7.7) as compared to ImageNet (3.0) and PASCAL (2.3). In contrast, the SUN dataset, which contains significant contextual information, has over 17 objects and "stuff" per image but considerably fewer object instances overall.

B. Combination between coco dataset and custom dataset

YOLO v3 is an improvement of YOLO v2 with faster FPS and higher mAP. Tiny YOLO is the simplified version of YOLO that has smaller weight size Tiny YoloV3 use Darknet-53 as the backbone (YOLO v2 use Darknet-19). Hardware specification during training a data-set:

- Intel i7 7th generation
- 16gb of RAM
- NVIDIA 1050 4GB

We use 229 image as training data-set and 4 image as validation, learning rate 0.001, and batch equal to 64. Before training the data-set, training image is labeled using LabelImg. LabelImg is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. Besides, it also supports YOLO format. Next, we define the number of class and filter in tiny-yolo-obj.cfg file. We changed filter value from filter=255 to filters= (classes + 5) x3 in the 3 before each yolo layer. So if classes=1 then should be filters=18. If classes=1 then write filter=21.

After that data-set is ready for trained by using this command line:

```
darknet.exe detector train data/obj.data tiny-yolo-obj.cfg tiny-yolo-voc.conv.13.
```

We used AlexeyAB darknet as our reference: https://github.com/AlexeyAB/darknet. Flowchart for training data in this project shown in fig.2

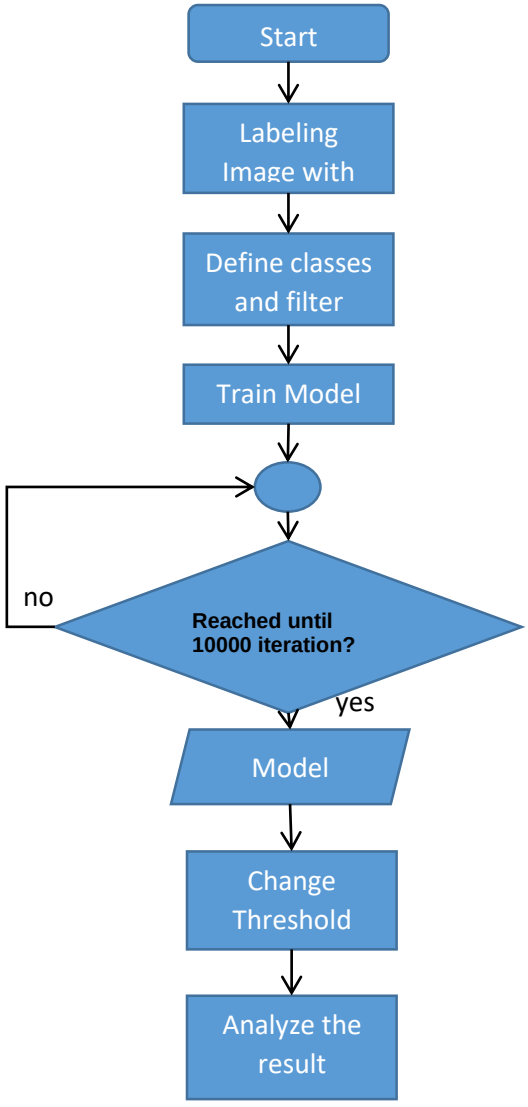


Figure 2. Flowchart trained process

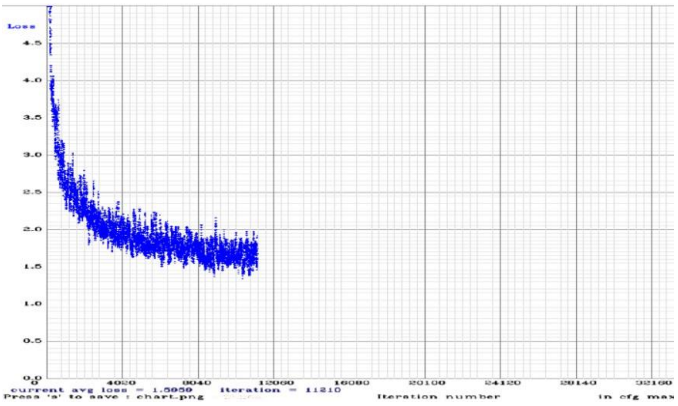


Figure 3. Lost function value

V. ALGORITHM AND NETWORKS

A. Yolo algorithm

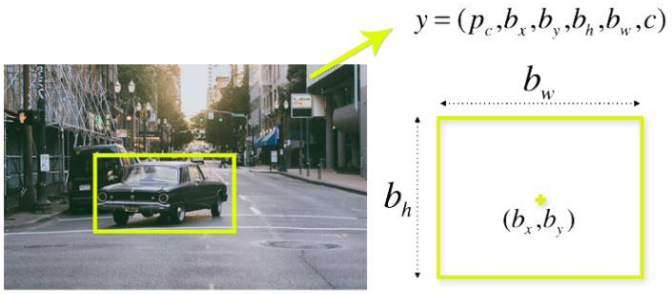
There are a few different algorithms for object detection and they can be split into two groups:

1. Algorithms based on classification – they work in two stages. In the first step, we're selecting from the image interesting regions. Then we're classifying those regions using convolutional neural networks. This solution could be very slow because we have to run prediction for every selected region. Most known example of this type of algorithms is the Region-based convolutional neural network (RCNN) and their cousins Fast-RCNN and Faster-RCNN.
2. Algorithms based on regression – instead of selecting interesting parts of an image, we're predicting classes and bounding boxes for the whole image **in one run of the algorithm**. Most known example of this type of algorithms is **YOLO (You only look once)** commonly used for real-time object detection.

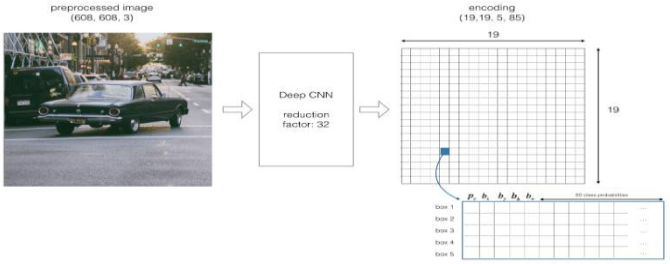
Before we go into YOLOs details we have to know what we are going to predict. Our task is to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

1. center of a bounding box ($\mathbf{b_x b_y}$)
2. width ($\mathbf{b_w}$)
3. height ($\mathbf{b_h}$)
4. value **c** is corresponding to a class of an object (f.e. car, traffic lights, …).

We've got also one more predicted value p_c which is a probability that there is an object in the bounding box, I will explain in a moment why do we need this.



With YOLO algorithm we're not searching for interested regions on our image that could contain some object. Instead of that we are splitting our image into cells, typically its 19×19 grid. Each cell will be responsible for predicting 5 bounding boxes (in case there's more than one object in this cell). This will give us 1805 bounding boxes for an image and that's a really big number.



Majority of those cells and boxes won't have an object inside and this is the reason why we need to predict p_c. In the next step, we're removing boxes with low object probability and bounding boxes with the highest shared area in the process called **non-max suppression**.

B. Darknet

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation.

Installation is very simple, just run these 3 lines (in order to use GPU modify settings in Makefile script after cloning the repository). For more details go here:

git clone https://github.com/pjreddie/darknet
cd darknet
make
After installation, we can use a pre-trained model or build a new one from scratch

VI. OBJECT-DETECTION USING EXTERNAL CAMERA

The external camera is used for object-detection and get the data outside. The system is connected with camera via IP. The figure 4 show the way camera connect to the system

Figure 4: Camera connect to the system

VII. RESULT

The system can identify objects in a relatively accurate way. The figure 5 show the example of test image with the object recognition system.

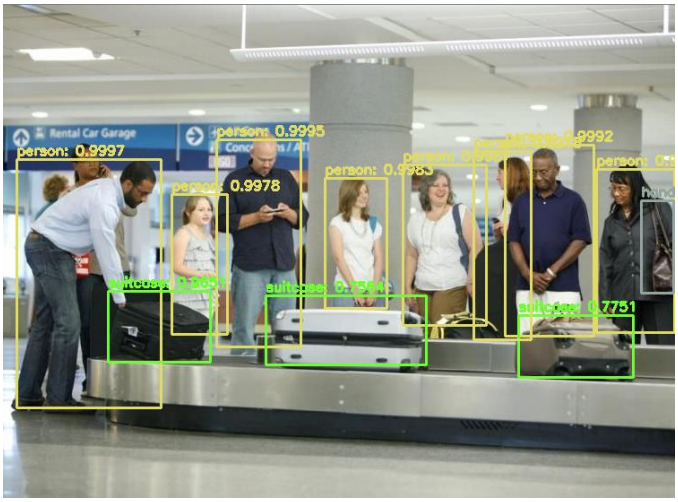


Figure 4: The image test output

Here, we see that YOLO has not only detected each person in the input image, but also the suitcases as well.

Furthermore, if you take a look at the right corner of the image you'll see that YOLO has also detected the handbag on the lady's shoulder.

VIII. CONCLUSION

We introduce YOLO, a unified model for object detection. Our model is simple to construct and can be trained directly on full images. Unlike classifier-based approaches, YOLO is trained on a loss function that directly corresponds to detection performance and the entire model is trained jointly.

Fast YOLO is the fastest general-purpose object detector in the literature and YOLO pushes the state-of-the-art in real-time object detection. YOLO also generalizes well to new domains making it ideal for applications that rely on fast, robust object detection.

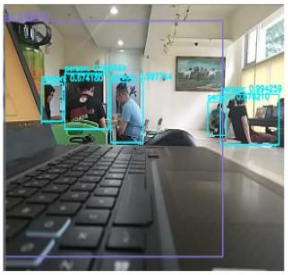
REFERENCES

[1]https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/
[2] https://github.com/AlexeyAB/darknet
[3] Joseph Redmon, Santosh Divvala, Roos Girshick, Ali Farhadi. You only look Once: Unified, Real-Time Object Detection
[4] https://pjreddie.com/darknet/
[5] Tsung-Yi Lin, Michael Maire, Serge Belongie Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar. Microsoft COCO: Common Objects in Context
[6] Joseph Redmon, Ali Farhadi. YOLOv3: An incremental Improvement

APPENDIX: Output evaluation

This appendix show the out from the different output with the value of iteration, threshold and confident.

Form the figure we can evaluate the output



Learning rate : 0.001, Confident: 0.5, Threshold: 0.3

Learning rate : 0.0001, Confident: 0.5, Threshold: 0.3

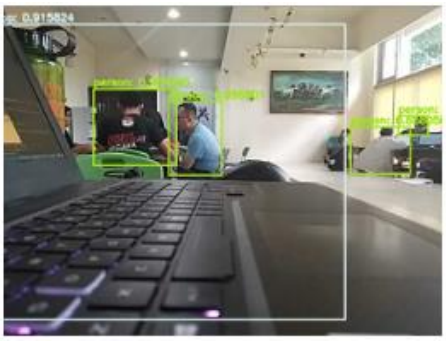
Learning rate : 0.1, Confident: 0.5, Threshold: 0.3

Learning rate : 0.01, Confident: 0.5, Threshold: 0.3

We can see that the smaller value of the learning rate, the system can recognize more object

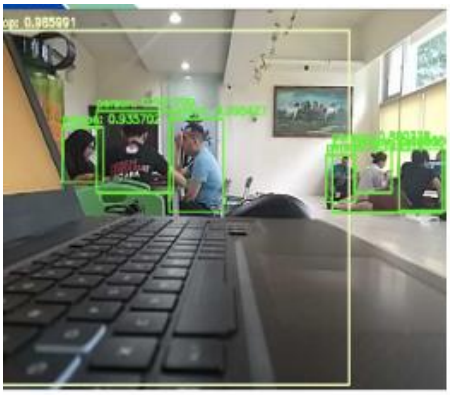
Learning rate : 0.01, Confident: 0.1, Threshold: 0.3

Learning rate : 0.01, Confident: 0.3, Threshold: 0.3



Learning rate : 0.01, Confident: 0.7, Threshold: 0.3

We can see that the smaller value of the confident, the system can recognize more object



Learning rate : 0.01, Confident: 0.5, Threshold: 0.5

Learning rate : 0.01, Confident: 0.5, Threshold: 0.8

Learning rate : 0.01, Confident: 0.1, Threshold: 0.8

We can see that the smaller value of the threshold, the system can recognize more object. And in the last figure if the threshold and confident is not suitable it will occur overlapping