

**HO CHI MINH UNIVERSITY OF SCIENCE**

**FACULTY OF INFORMATION TECHNOLOGY**

Final Semester Project

yolo project

**Student in charge:**

1. Huỳnh Cao Nhật Hiếu – 19127399
2. Ngô Đăng Khoa – 19127444

Contents

[1. Requirement 1: Install YOLO 2](#_Toc92273006)

[2. Requirement 2: Apply YOLO to build application 7](#_Toc92273007)

[2.1 explain step by step: 7](#_Toc92273008)

[2.2 Evaluate the model 9](#_Toc92273009)

[3. References: 10](#_Toc92273010)

# Requirement 1: Install YOLO

* Step 1: We install Yolo from darknet on Google colab

Graphical user interface, application

Description automatically generated

Graphical user interface, text

Description automatically generated

* Step 2: Import library and implement predict function

Text

Description automatically generated

Then we implemented **darknet\_helper** to run detection on image, this function we refered from <https://github.com/MINED30/Face_Mask_Detection_YOLO?fbclid=IwAR1uVLNqP3l9_6Wi3El_F9wA0QeHDM-ENJ8Iw_99_nUSxv-zq23pKu-i4Kk>

Text

Description automatically generated

Graphical user interface

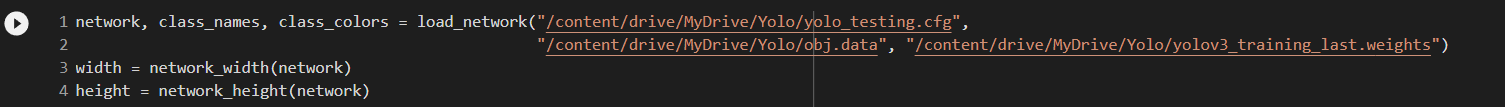
Description automatically generated with low confidenceNext, we implemented **yolo\_predict** draw a bounding box and classify object. This function’s input is a path of image, then we use OpenCV to read that image to a 3D array then draw a bounding box on it and save back to its file path.

* Step 3: Implement web application to demo Yolo

To make a web application on Colab we used **flask-ngrok** to make a local server.

Text

Description automatically generated

To run this server, we have to load the model first.

Backend Flask server

Text

Description automatically generated

Text

Description automatically generatedFrontend html file:

Idea: After upload an image to server, the image will be saved to a folder on my Google Drive then call function **yolo\_predict**  to draw bounding box and save it again. After that the backend server will return predicted image back to front end to show the result.

Graphical user interface, website

Description automatically generatedA sign with a building in the background

Description automatically generated with low confidence

# Requirement 2: Apply YOLO to build application

## explain step by step:

* Step 1: We prepare the dataset. My group choose the dataset “Road Sign Detection” on the Kaggle.

*The link of dataset:* [*https://www.kaggle.com/andrewmvd/road-sign-detection*](https://www.kaggle.com/andrewmvd/road-sign-detection)

In this dataset, there are 4 classes: Trafic Light, Stop, Speedlimit and Crosswalk. Therefore, we need to add a class named Paking.

* Step 2: We annotate the dataset. We use LabelImg to annotate the dataset.
* Step 3: We prepare a folder yolo in the Google drive to contain the data of the model that we will train.

*Note: From step 3, we refered from* [*YOLOv3 Custom Object Detection with Transfer Learning | by Nitin Tiwari | Medium*](https://tiwarinitin1999.medium.com/yolov3-custom-object-detection-with-transfer-learning-47186c8f166d)

Graphical user interface, application

Description automatically generated

* Step 3: We clone, configure, and compile the Darknet

A picture containing graphical user interface

Description automatically generated

* Step 4: We fine-tune the model by configure yolov3.cfg file:

Graphical user interface, application

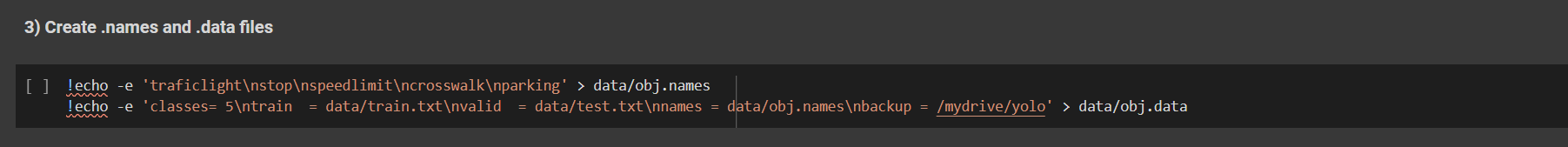
Description automatically generated

* + Firstly, we create a copy of yolov3.cfg file and names it as yolov3\_training.cfg so that all the changes and configurations that we will make for our custom model would be reflected in the copy and not the original file. It’s a good practice to keep a backup of the original .cfg file if in case anything goes wrong.
  + Secondly, we edit the yolov3\_training.cfg:

As darknet recommend, if the custom object detection model includes ‘n’ no. of classes, then max\_batches = 2000 \* n and filters = (n + 5) \* 3

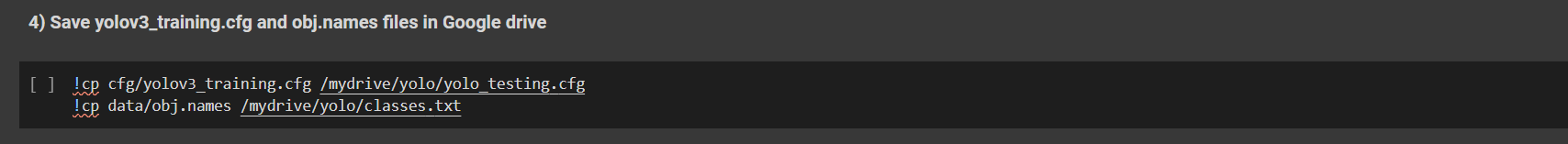
My model has 5 classes so max\_batches =10000, classes=5, filters=30.

* Step 5: we Create.names and .data files.

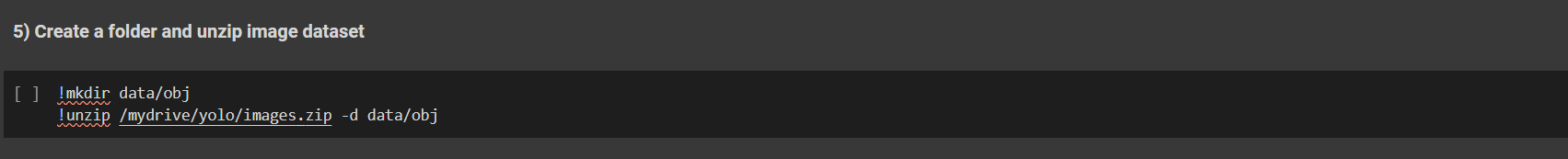


This cell creates obj.names and obj.data files inside the darknet/data/obj directory. These files include metadata such as class names and no. of classes required for training.

* Step 6: We save the yolov3\_training.cfg and obj.name files in the folder of the Drive.



* Step 7: We unzip the dataset and move it into data/obj of the Darknet



* Step 8: We create train.txt and test.txt that 2 files to contain the images to train and valid. In this project, we split dataset into 8 training and 2 validation.

A screenshot of a computer

Description automatically generated with medium confidence

* Step 9: We are applying Transfer Learning by adding our own layers to an already trained model. So, we download the pre-trained weights called darknet53.conv.74. Thus, our custom model will be trained using these pre-trained weights instead of randomly initialized weights which in turn will save a lot of time and computations while training our own model.

**Text

Description automatically generated**

* Step 10: We start to train the model.

**A screenshot of a computer

Description automatically generated**

## Evaluate the model

* Average Precision

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Class\_id*** | ***Name*** | ***TP*** | ***FP*** | ***ap*** |
| 0 | traficlight | 16 | 10 | 55.02% |
| 1 | stop | 10 | 7 | 81.07% |
| 2 | speedlimit | 105 | 4 | 98.43% |
| 3 | crosswalk | 34 | 5 | 93.67% |
| 4 | parking | 8 | 0 | 70.91% |

* F1-score & Average IoU

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| conf\_thresh | precision | recall | F1-score | TP | FP | FN | average IoU |
| 0.25 | 0.89 | 0.85 | 0.87 | 220 | 26 | 40 | 72.34% |

* Mean Average Precision

mean average precision (mAP@0.50) = 79.68%

Text

Description automatically generated

# References:

[1] <https://github.com/MINED30/Face_Mask_Detection_YOLO>

[2] [YOLOv3 Custom Object Detection with Transfer Learning | by Nitin Tiwari | Medium](https://tiwarinitin1999.medium.com/yolov3-custom-object-detection-with-transfer-learning-47186c8f166d)