HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF HIGH QUALITY TRAINING

DEPARTMENT OF COMPUTER ENGINEERING - COMMUNICATION

FINAL REPORT

AI: FOUNDATIONS AND APPLICATIONS

**VEHICLE AND TRAFFIC SIGNS DETECTION**

**MAJOR IN COMPUTER ENGINEERING TECHNOLOGY**

STUDENT: **PHAM MINH LONG**

ID: 19119067

**CAO HOANG BACH**

ID: 19119002

HO CHI MINH CITY – 11/2021

HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY OF HIGH QUALITY TRAINING

DEPARTMENT OF COMPUTER ENGINEERING - COMMUNICATION

FINAL REPORT

AI: FOUNDATIONS AND APPLICATIONS

**VEHICLE AND TRAFFIC SIGNS DETECTION**

**MAJOR IN COMPUTER ENGINEERING TECHNOLOGY**

STUDENT: **PHAM MINH LONG**

ID: 19119067

**CAO HOANG BACH**

ID: 19119067

Instructor**: Assoc. Prof .TRUONG NGOC SON**

HO CHI MINH CITY – 11/2021

LECTURER COMMENT

|  |  |  |
| --- | --- | --- |
| TT | Content | Comment |
| 1 | Introduction |  |
| 2 | Background |  |
| 3 | Design |  |
| 4 | Result |  |
| 5 | Conclusion and recommendation |  |

General comment:

Lecturer’s signature

## ABSTRACT

In the digital transformation revolution, vehicles are gradually applying many technologies to overcome problems such as traffic accidents, loss of vision while driving,... Based on these reasons, a solution The proposed method, vehicle and traffic signs detection, aims to help the driver control the traffic ahead. The application is mainly based on YOLOv5, the most optimal detection algorithm today. In addition, the trained image sets have high accuracy and fast recognition speed. However, the training process also has many disadvantages in terms of GPU and training time.

CONTENTS

[ABSTRACT vii](#_Toc88928981)

[LIST OF FIGURE ix](#_Toc88928982)

[BRIEF WORDS x](#_Toc88928983)

[Chapter 1 Introduction 1](#_Toc88928984)

[1.1. introduction 1](#_Toc88928985)

[1.2. objectives 1](#_Toc88928986)

[1.3. PROBLEM STATEMENT 1](#_Toc88928987)

[1.4. RESEARCH METHOD 2](#_Toc88928988)

[1.5. OBJECT AND SCOPE OF STUDY 2](#_Toc88928989)

[1.6. LAYOUT 2](#_Toc88928990)

[CHAPTER 2 BACKGROUND 3](#_Toc88928991)

[2.1. YOLO v5 Module Architechture 3](#_Toc88928992)

[2.2. Activation function 4](#_Toc88928993)

[2.3. OPTIMIZATION 5](#_Toc88928994)

[2.4. LOSS FUNCTION 5](#_Toc88928995)

[CHAPTER 3 DESIGN AND IMPLEMENTATION 7](#_Toc88928996)

[3.1. Environment 7](#_Toc88928997)

[3.2. Preparing the dataset for training 8](#_Toc88928998)

[3.3. Creating the datasets.yaml file 10](#_Toc88928999)

[3.4. Training phase 11](#_Toc88929000)

[3.4.1. PREPARING THE ARCHITECHTURE 11](#_Toc88929001)

[3.4.2. TRAINING MODEL 13](#_Toc88929002)

[3.4.3. INFERENCE WITH TRAINED WEIGHT 15](#_Toc88929003)

[CHAPTER 4 RESULT 16](#_Toc88929004)

[4.1. EXECUTION RESULTS 16](#_Toc88929005)

[CHAPTER 5 CONCLUSION AND RECOMMENDATION 20](#_Toc88929006)

[5.1. CONCLUSION 20](#_Toc88929007)

[5.2. RECOMMENDATION 20](#_Toc88929008)

[REFERENCE 21](#_Toc88929009)

LIST OF FIGURE

[*Figure 1. YOLO v5 Architechture* 3](#_Toc88949235)

[*Figure 2. Leaky RELU.* 4](#_Toc88949236)

[*Figure 3. Sigmoid function.* 5](#_Toc88949237)

[*Figure 4. Adam algorithm.* 6](#_Toc88949238)

[*Figure 5. Binary Cross-Entropy.* 6](#_Toc88949239)

[*Figure 6. Logits Loss function.* 7](#_Toc88949240)

[*Figure 7. Logits Loss in the case of multi-label classification.* 7](#_Toc88949241)

[*Figure 8. Colab provides Tesla K80 GPU.* 8](#_Toc88949242)

[*Figure 9. Cloning and installing the YOLOv5 repository.* 9](#_Toc88949243)

[Figure 10. *Cutting frame the video.* 9](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949244)

[*Figure 11. Labeling the bounding box.* 10](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949245)

[*Figure 12. Coordinations of each bouding box.* 10](#_Toc88949246)

[*Figure 13. The bounding box in YOLO format.* 11](#_Toc88949247)

[*Figure 14. Overwrite the empty yaml file based on the structure.* 12](#_Toc88949248)

[*Figure 15. Sample YOLOv5 architecture provided by Ultralystic.* 13](#_Toc88949249)

[*Figure 16. Overwrite the number of classes and save as custom model.* 14](#_Toc88949250)

[*Figure 17. Implement the training process.* 14](#_Toc88949251)

[*Figure 18. Detect VTS head with trained weight.* 16](#_Toc88949252)

[*Figure 19. Training progress in 30 epoches first.* 17](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949253)

[*Figure 20. Training progress in 40 epoches later.* 18](#_Toc88949254)

[*Figure 21. Path for the final result.* 18](#_Toc88949255)

[*Figure 22. . The result of the entire 70 epochs later were visualized in graphs.* 19](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949256)

[*Figure 23. The result of the entire 30 epochs first were visualized in graphs.* 19](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949257)

[*Figure 24. Predicted images were detected by using best trained weight.* 20](file:///D:\Downloads\Vehicle_and_Object_Detection_using_YOLO_v5.doc#_Toc88949258)

BRIEF WORDS

[1] YOLO: You only look once.

[2] CSP: Cross Stage Partial Networks.

[3] RELU: Rectified linear unit.

[4] SGD: Stochastic gradient descent.

[5] Adam: Adaptive Moment Estimation.

[6] IDE: Integrated Development Environment.

[7] AI: Artificial Intelligent.

[8] GPU: Graphic Process Unit.

[9] TPU: Tensor Process Unit.

[10] RAM: Random Access Memory.

[11] VTSD: Vehicle and Traffic Signs Detection.

[12] VTS: Vehicle and Traffic Signs.

# Chapter 1 Introduction

## 1.1. introduction

YOLO an acronym for 'You only look once', is an object detection algorithm that divides images into a grid system. Each cell in the grid is responsible for detecting objects within itself. YOLOv5 is one of the most famous object detection algorithms due to its speed and accuracy in real-time detection. We design identification and classification of vehicles and traffic signs to support for vehicles in traffic using Yolov5 algorithm. In addition, it is applied to detect fast and return real-time results to the user. It is combined by optimization algorithms and pre-trained models to generate metrics about loss and mean average precision.

## 1.2. objectives

The application is designed to assist vehicles with visibility and recognition of traffic vehicles as well as traffic signs ahead. Therefore, drivers can easily control their vehicles. In addition, the application is deployed to reduce unfortunate traffic accidents while traveling.

## 1.3. PROBLEM STATEMENT

The training Yolo algorithm takes a lot of time. An average of 40 epochs takes up to 4 hours. In addition, due to GPU limitation from google colab, the training is divided into 2 times.

## 1.4. RESEARCH METHOD

Research methods to design and evaluate the application are mainly based on the reference to the theoretical documents and the knowledge learned in foundations and applications of artificial intelligence.

## 1.5. OBJECT AND SCOPE OF STUDY

Currently, vehicles are almost not equipped with equipment to identify the traffic ahead. Therefore, the product is created based on the application of the Yolov5 algorithm to help drivers control their vehicles to react promptly. The scope of product research mainly identifies 3 main objects: cars, motorbikes, and traffic signs.

## 1.6. LAYOUT

In order to help readers have an overview of the product, the report is divided into 5 chapters:

Chapter 2: Background

Chapter 3: Design and implementation

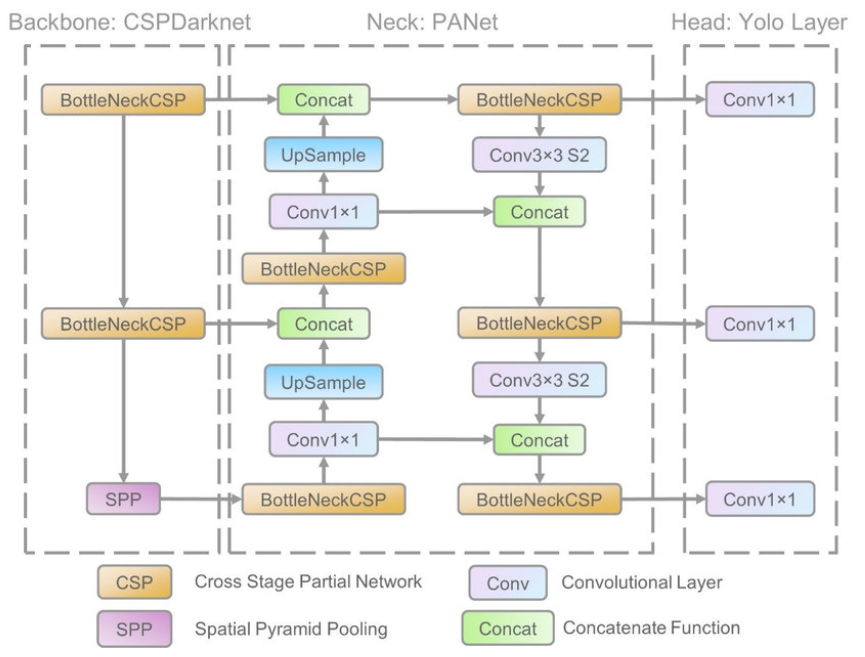
Chapter 4: Result

Chapter 5: Conclusion and recommendation

# CHAPTER 2 BACKGROUND

## 2.1. YOLO v5 Module ARCHITECTURE

As YOLO v5 is a single-stage object detector, it has three important parts like any other single-stage object detector. There are three module used in YOLO v5: Model Backbone, Model Neck, and Model Head.

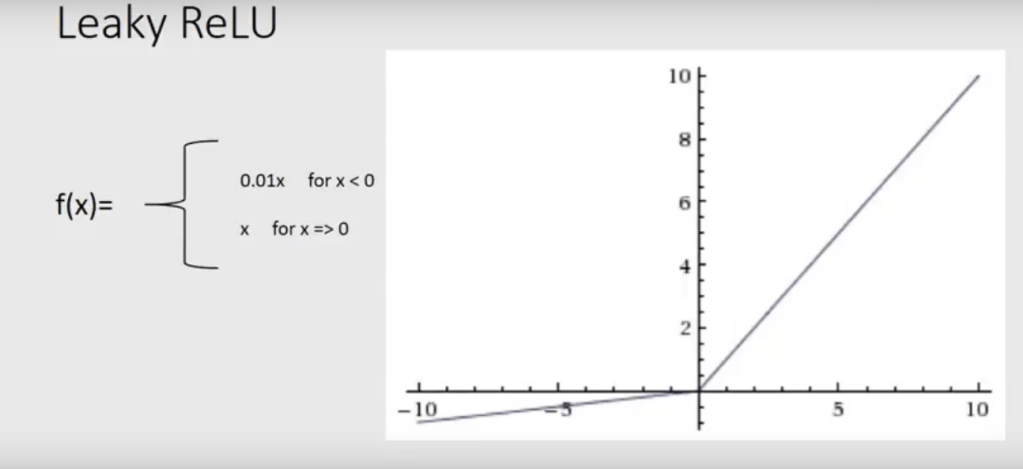


*Figure 1. YOLO v5 Architecture*

Model Backbone is mainly used to extract important features from the given input image. In YOLO v5 the CSP — Cross Stage Partial Networks are used as a backbone to extract rich in informative features from an input image. Model Neck is mainly used to generate feature pyramids. Feature pyramids help models to generalized well on object scaling. It helps to identify the same object with different sizes and scales. The model Head is mainly used to perform the final detection part. It applied anchor boxes on features and generates final output vectors with class probabilities, objectless scores, and bounding boxes.

## 2.2. Activation function

The choice of activation functions is most crucial in any deep neural network. Recently lots of activation functions have been introduced like Leaky ReLU, mish, swish, etc. YOLO v5 authors decided to go with the Leaky ReLU and Sigmoid activation function.



*Figure 2. Leaky RELU.*

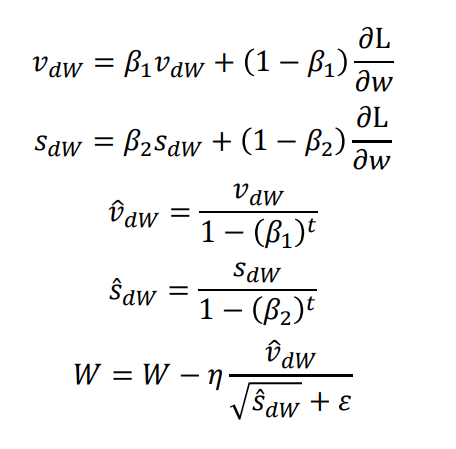


*Figure 3. Sigmoid function.*

In YOLO v5 the Leaky ReLU activation function is used in middle/hidden layers and the sigmoid activation function is used in the final detection layer.

## 2.3. OPTIMIZATION

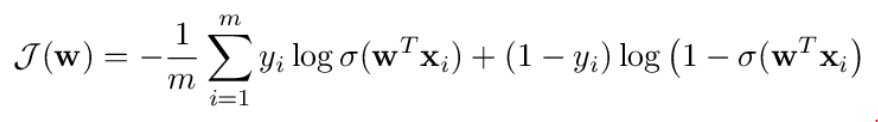
For optimization function in YOLO v5, we have two options: SGD (Stochastic Gradient Descent) and Adam (Adaptive Moment Estimation). In YOLO v5, the default optimization function for training is SGD.



*Figure 4. Adam algorithm.*

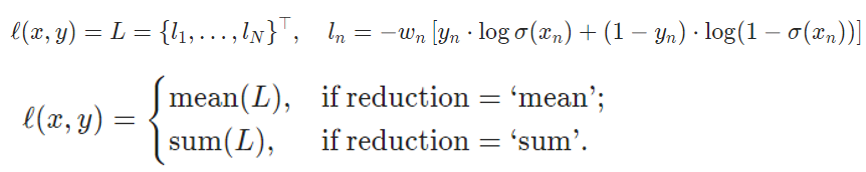
## 2.4. LOSS FUNCTION

In the YOLO family, there is a compound loss is calculated based on objectless score, class probability score, and bounding box regression score. Ultralytics have used Binary Cross-Entropy with Logits Loss function from PyTorch for loss calculation of class probability and object score.



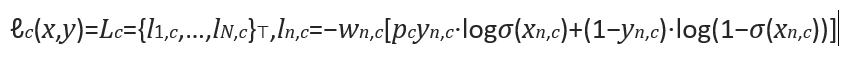
*Figure 5. Binary Cross-Entropy.*

where m is the number of samples, xᵢ is the i-th training example, yᵢ its class (i.e. either 0 or 1), σ(z) is the logistic function and w is the vector of parameters of the model. You may also know that, for logistic regression, it is a convex function. As such, any minimum is a global minimum.



*Figure 6. Logits Loss function.*

where N is the batch size. This is used for measuring the error of a reconstruction in for example an auto-encoder. Note that the targets t[i] should be numbers between 0 and 1. It’s possible to trade off recall and precision by adding weights to positive examples. In the case of multi-label classification, the loss can be described as figure 7.



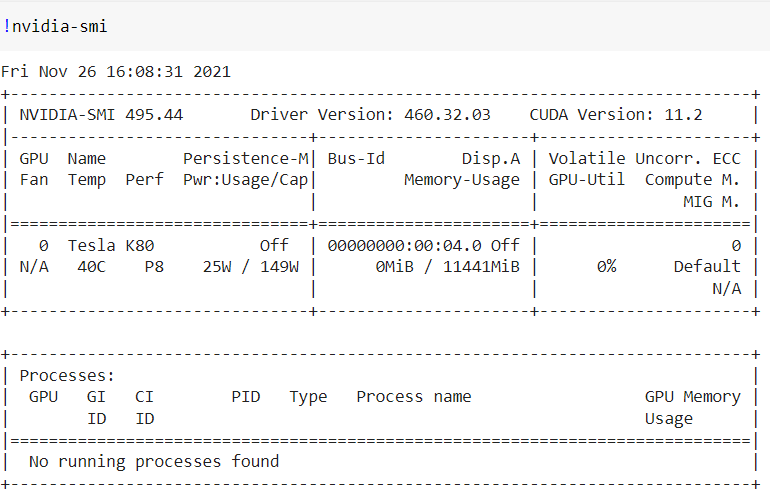
*Figure 7. Logits Loss in the case of multi-label classification.*

where c is the class number (c > 1 for multi-label binary classification, c = 1 for single-label binary classification), n is the number of the sample in the batch and *pc*​​ is the weight of the positive answer for the class c, with *pc*​​ ​ >1 increases the recall, *pc*​​ < 1 increases the precision.

# CHAPTER 3 DESIGN AND IMPLEMENTATION

## 3.1. Environment

Google Colab (Google Collaboratory) is a free Integrated Development Environment (IDE) from Google to support research and learning about Artificial Intelligent (AI). Collaboratory provides a code environment as Jupyter Notebook, and it is free to use Graphic Process Unit (GPU) and Tensor Process Unit (TPU). Google Colab has pre-installed libraries that are very popular in Deep Learning research such as PyTorch, TensorFlow, Keras, and OpenCV. Due to machine learning/deep learning algorithms require the system to have high speed and processing power (usually based on GPU), normal computers are not equipped with GPU. Therefore, Colab supplies GPU (Tesla V100) and TPU (TPUv2) on cloud, one of the highest performing GPUs at the moment, to give assistance to AI researchers. Colab provides 25 GB RAM and 150 GB main disk. Checking GPU usage status:



*Figure 8. Colab provides Tesla K80 GPU.*

As with previous versions, the YOLOv5 architecture was built theoretically based and released via a repository on GitHub. As mentioned, Ultralystic builds YOLOv5 on the PyTorch framework, one of the most popular frameworks in the AI community. However, this is only a preliminary architecture, researchers can configure the architecture to give the best results depending on their problems such as adding layers, removing blocks, integrating additional image process methods, changing the optimization methods or activation functions, etc.



*Figure 9. Cloning and installing the YOLOv5 repository.*

## 3.2. Preparing the dataset for training

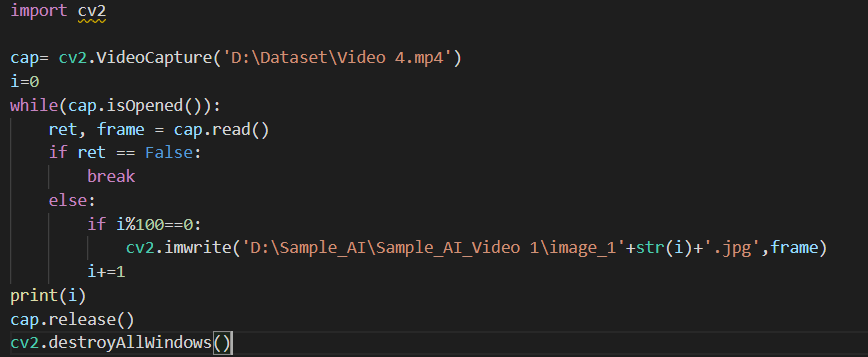
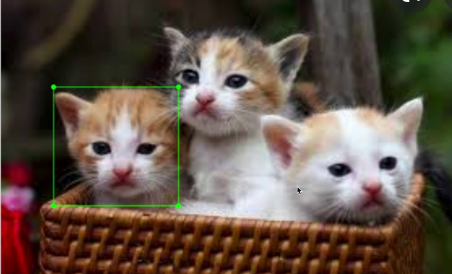
To prepare the sample, we choose one video having vehicle and traffic signs. Afterwards, we cut frame following to the code below.

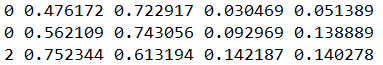
Figure 10. *Cutting frame the video.*

Let us move to choose address containing the video and set the address in order to save the sample. Afterwards, we label all of the vehicle and traffic signs in the picture that we have cut frame.



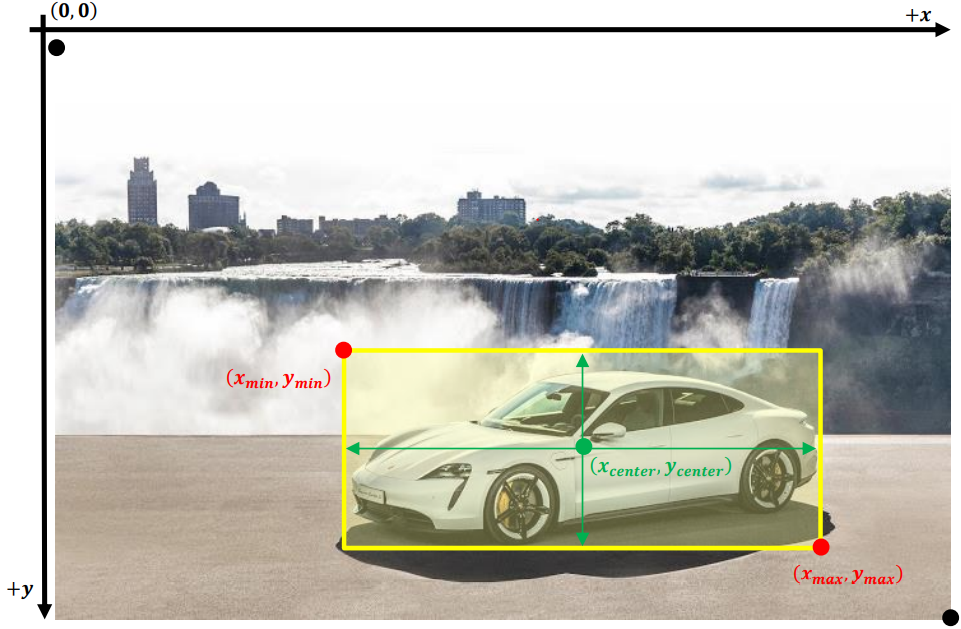
*Figure 11. Labeling the bounding box.*

After labeling, we will have a folder.txt. In each file.txt, they will contain a set of coordination as follows:



*Figure 12. Coordination of each bounding box.*

The above data file is a common format for bounding boxes in the object detection dataset. The bounding boxes data in YOLO is formatted as [𝑐𝑙𝑎𝑠𝑠, 𝑥𝑐𝑒𝑛𝑡𝑒𝑟, 𝑦𝑐𝑒𝑛𝑡𝑒𝑟, 𝑤𝑖𝑑𝑡ℎ, ℎ𝑒𝑖𝑔ℎ𝑡].

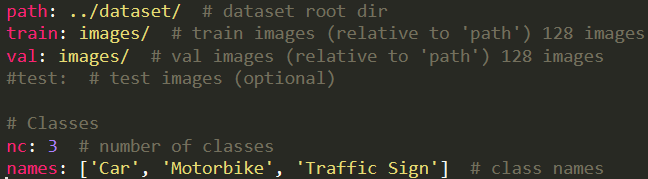


*Figure 13. The bounding box in YOLO format.*

The VTSD dataset contain 2068 images, but only images containing objects (i.e., bounding boxes) are written in csv files. Meanwhile, images that do not contain or contain near VTSD-like objects (such as weeds) can be helpful in training the model, preventing the model from being fooled by similar objects. Images containing objects are called positive\_images, and images that do not contain objects are called negative\_images. In order for the YOLOv5 model to access negative images and use them during training, there also should be label text files corresponding to those negative images. Since these images do not contain any objects, they do not contain any bounding boxes. Therefore, their label text files will be empty.

## 3.3. Creating the datasets.yaml file

The YOLOv5 model on PyTorch accesses the images and uses them as input through a yaml file containing summary information about the data set. The data.yaml file used in the YOLO model has the following structure as figure 14.

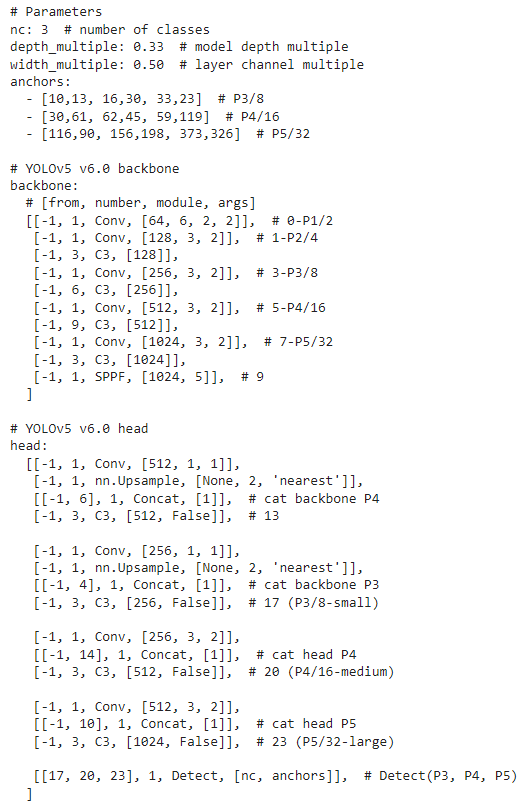


*Figure 14. Overwrite the empty yaml file based on the structure.*

## 3.4. Training phase

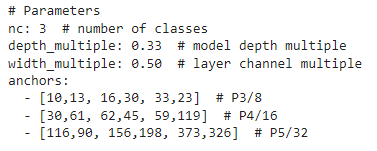
### 3.4.1. PREPARING THE ARCHITECTURE

Glenn Jocher also provides some sample YOLOv5 models built on previous theory. The YOLOv5 model on PyTorch will read these architectures from the yaml file and build it in the train.py file. This also makes it easier to configure the architecture depending on the different object detection problems.



*Figure 15. Sample YOLOv5 architecture provided by Ultralystic.*

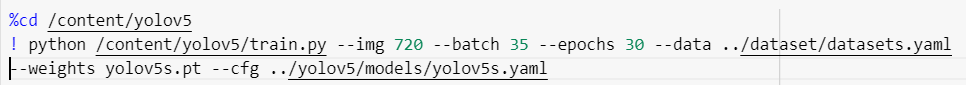
The purpose of this thesis is to evaluate the performance of the YOLOv5 algorithm, so the original architecture will be used and will temporarily not configure or add other algorithms and optimization methods to the model. Because this sample YOLOv5 architecture is used to train the COCO dataset, the number of classes defined is 80. For the VTSD dataset, the number of classes needs to be adjusted, 3 classes especially. Since the anchor box auto-learning has been integrated, the anchor box parameters can be ignored as default.



*Figure 16. Overwrite the number of classes and save as custom model.*

### 3.4.2. TRAINING MODEL

With the command line shown in Figure 17, the model will be trained by compile file train.py along with its configurable arguments.



*Figure 17. Implement the training process.*

These following arguments represent for:

**img**: define input image size. The original image size is 1280 × 720, compress to smaller size make the training process faster. After many experiments, many computer vision researchers agreed that the size 384 × 640 is the ideal size to use as input without losing much detail.

**batch**: determine the batch size. The forwarding of thousand images into the neural network at the same time makes the number of weights that the model learns in one time (one epoch) to increase a lot. Thus, the dataset is usually divided into multiple batches of 𝑛 images and training batch by batch. The results of each batch are then saved to RAM and aggregated after the training for all batches is completed. Because the weights learned from the batches are stored in RAM, so the larger the number of batches, the more memory consumption will be consumed.

The training set contains 2068 images, with 𝑏𝑎𝑡𝑐ℎ 𝑠𝑖𝑧𝑒 = 35. The number of batches will be 2068 ÷ 35 = 60 𝑏𝑎𝑡𝑐ℎ𝑒𝑠.

**epochs**: define the number of training epochs. An epoch is responsible for learning all input images, in other words, training all input. Since the dataset is split into multiple batches, one epoch will be responsible for training all the batches. The number of epochs represents the number of times the model trains all the inputs and updates the weights to get closer to the ground truth labels. Often chosen based on experience and intuition. The number of epochs more than 3000 is normal. In this case, it will be trained 70 with 2 times due to not having enough GPU.

**data**: the path to datasets.yaml file containing the summary of the dataset. The model evaluation process is executed immediately after each epoch, so the model will also access the validation directory via the path in data.yaml file and use its contents for evaluation at that moment.

**cfg**: specify our model configuration path. Based on the architecture defined in the model yolov5s.yaml file previously, this command line allows the train.py file to compile and build this architecture for training input images.

**weights**: specify a path to weights. A pretrained weight can be used for saving training time. If it is left blank, the model will automatically initialize random weights for training.

### 3.4.3. INFERENCE WITH TRAINED WEIGHT

Trained weights can be used to identify the VTS head on any image. If the presence of a VTS head is detected, a bounding box is drawn to encase the object and display the probability that the object is the head of VTS.

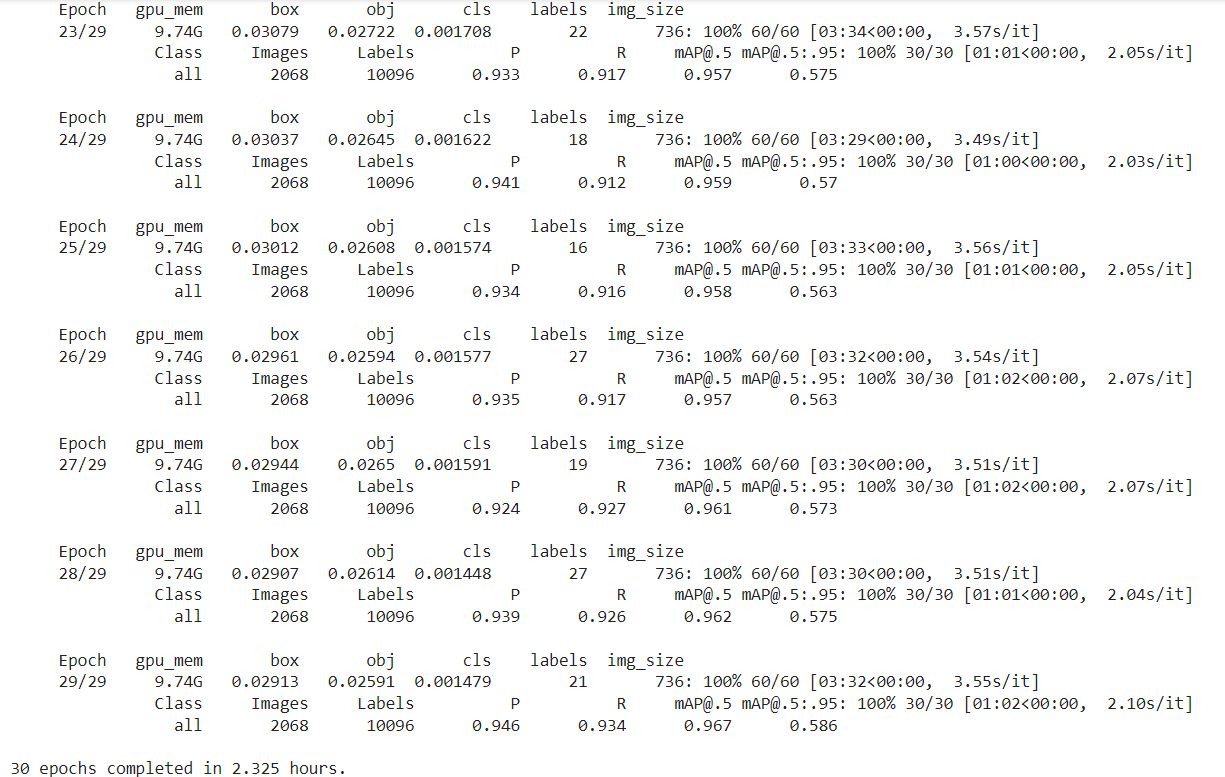
The method of performing object detection with trained weights is similar to training the model. Using the command shown in Figure 23, detect.py file will be compiled, and it rebuilds the architecture used in the training. Trained weights will be used to predict objects and limit boxes for them with 97.9% accuracy.



*Figure 18. Detect VTS head with trained weight.*

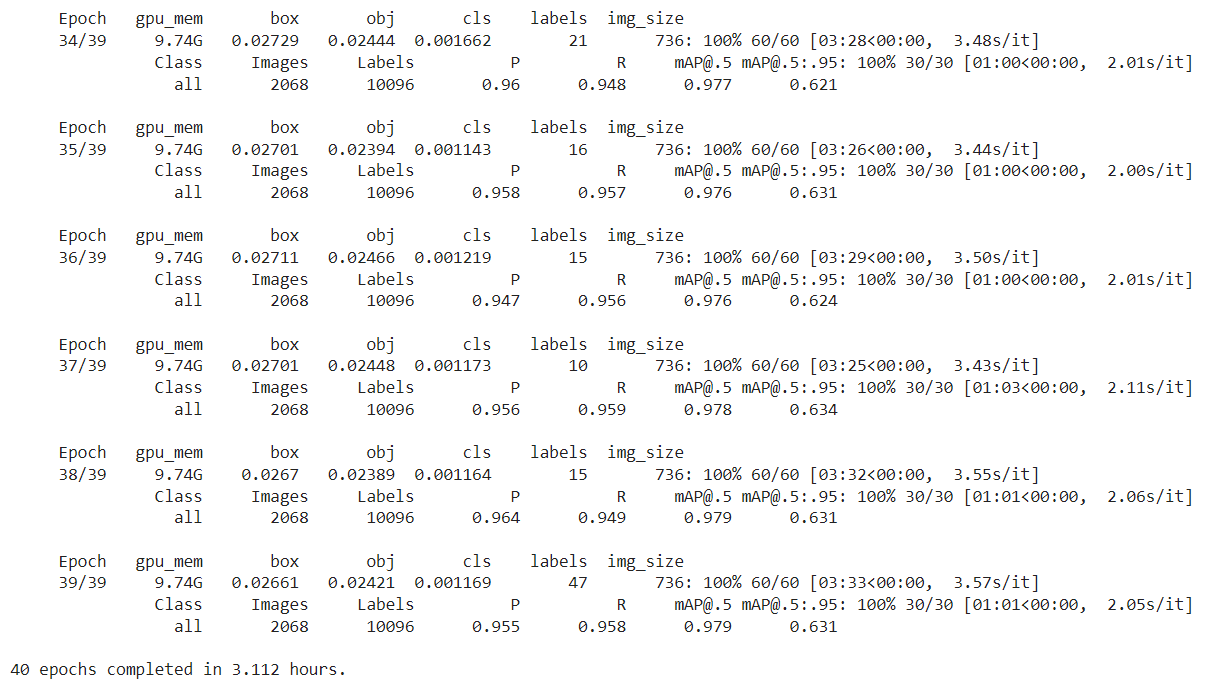
# CHAPTER 4 RESULT

## 4.1. EXECUTION RESULTS

**After training, we receive the result below:

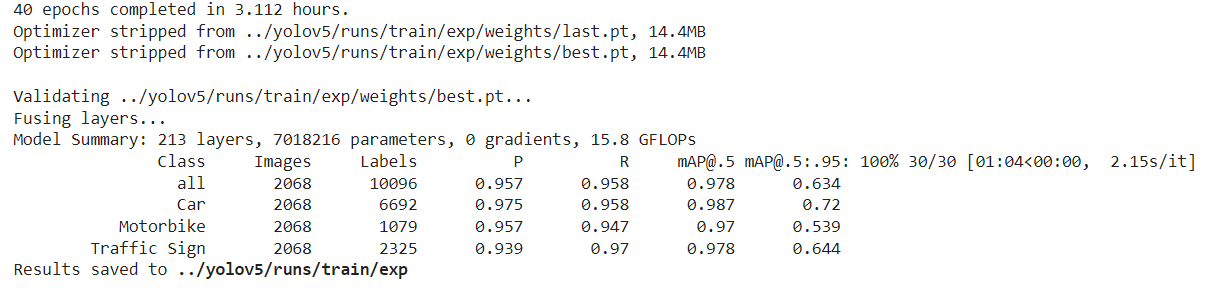
*Figure 19. Training progress in 30 epochs first.*

For 1 epoch, the average time to perform the training process on 60 batches were 3 minutes 34 seconds and for evaluation on 30 batches were 1 minute 1 second. The total execution time was 2.325 hours for 30 epochs first on the dataset containing 2068 images and the mAP of last epoch is 96.7%.

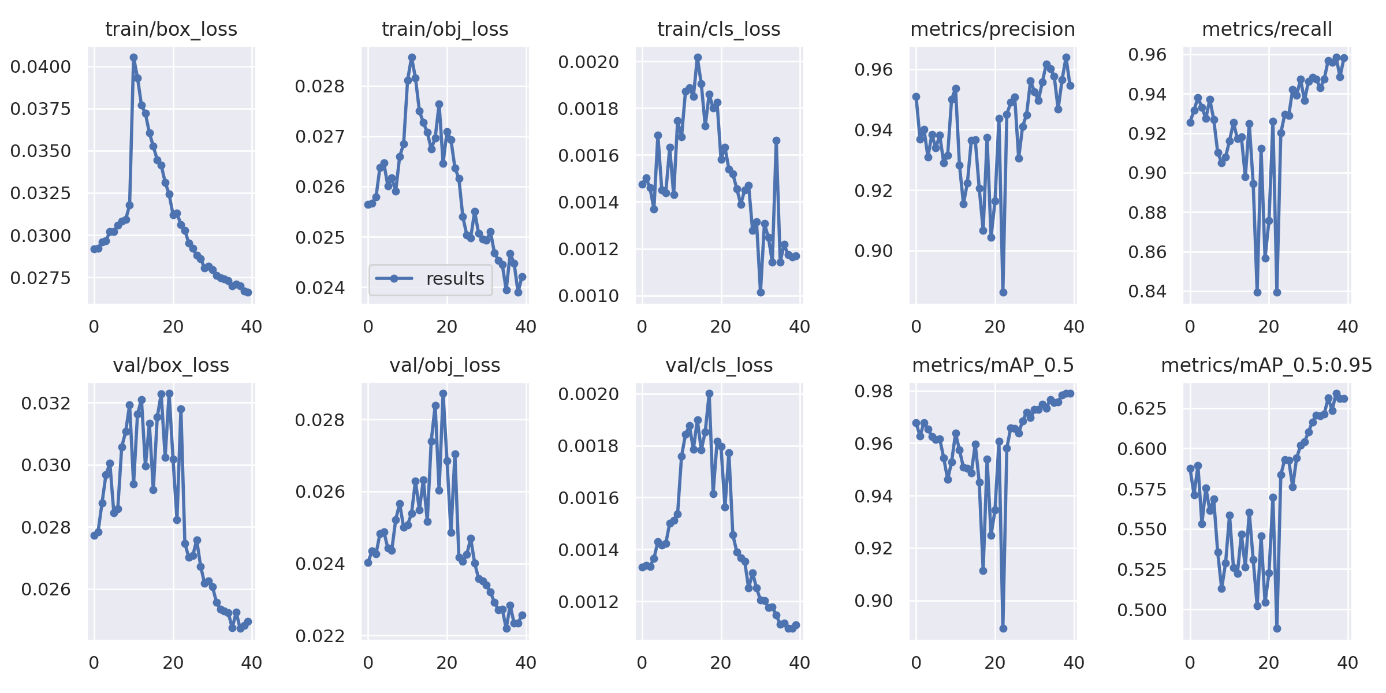
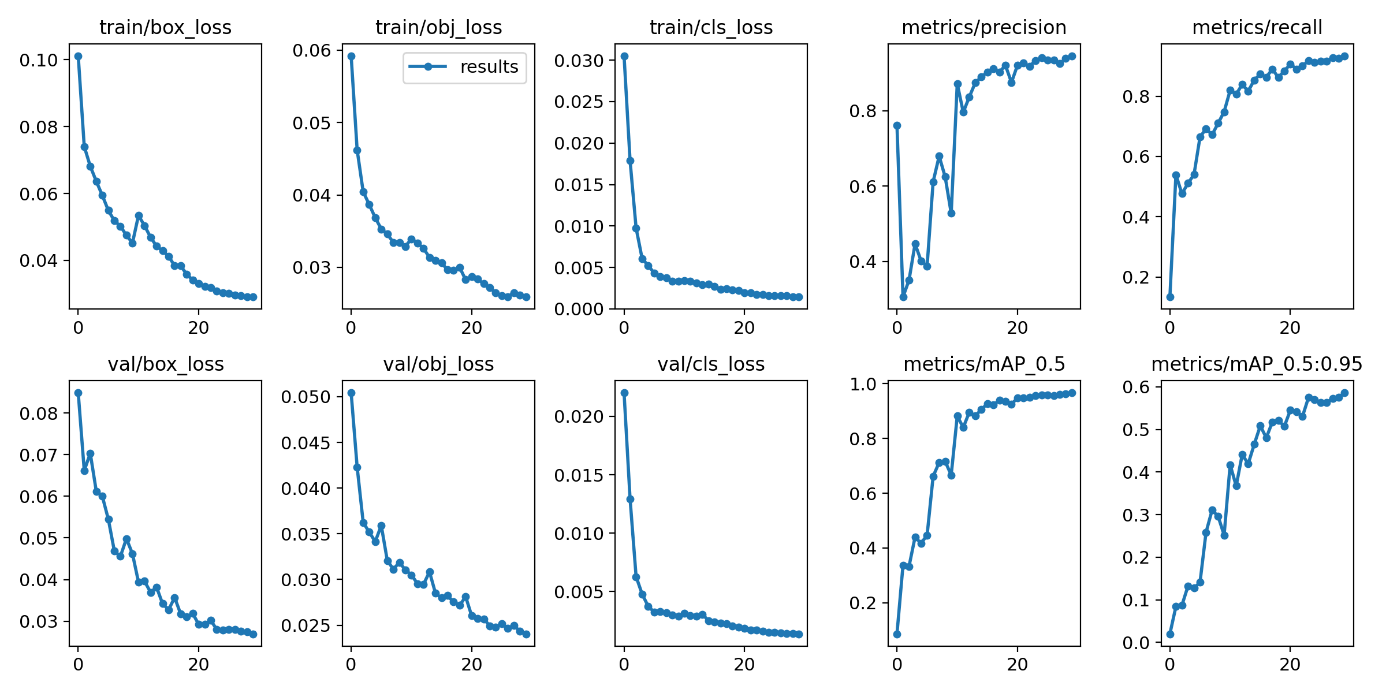


*Figure 20. Training progress in 40 epoches later.*

After training 30 epoches completely, we continue to train more 40 epoches. Therefore, the total epoch of training is 70 that we use best.pt file from 30 epoches first. For 1 epoch, the average time to perform the training process on 60 batches were 3 minutes 33 seconds and for evaluation on 30 batches were 1 minute 1 second2. The total execution time was 3.112 hours for 40 epochs later on the dataset containing 2068 images and the mAP of last epoch is 97.9%. This mAP also belongs to 70 epoches training. Totally, it executes 5.437 hours for 2 times training. The weighting result obtained in the last epoch is not always the weight for the highest accuracy.



*Figure 21. Path for the final result.*

**After training completely, the results will be showed the path that can view and appriciate the final result. We access to the result and use cv2 module to read the graph.

*Figure 22. . The result of the entire 70 epochs later were visualized in graphs.*

*Figure 23. The result of the entire 30 epochs first were visualized in graphs.*

As shown in Figure 21 and Figure 22, with a dataset containing 2068 images, the model takes about 4 minutes 30 seconds to complete one epoch, and only with 70 epochs, the accuracy of model is about 98%. This proves that, with the mere original architecture of YOLOv5, the model is not only fast, but the accuracy is also high without any optimization methods integrated into it.

*Figure 24. Predicted images were detected by using best trained weight.*

As shown in Figure 23, the model gives out extremely impressive predictive results on even the images it has not seen before. Although the density of VTS in each image is very high, the model almost detects that all VTS appear in the image. However, because the model's accuracy was only 97.9%, some VTS were still missing in detection. As can be seen in Figure 23, not only the VTS are correctly predicted, but the probability for them is also high. As mentioned, the model predicts an object based on the probability for that object, if the probability is less than a given threshold (here is 0.3) then the model predicts it is not a VTS. With a low prediction probability near the threshold, some VTS may have a probability lower than the threshold and be predicted not VTS.

# CHAPTER 5 CONCLUSION AND RECOMMENDATION

## 5.1. CONCLUSION

The model gives out extremely impressive predictive results on even the images it has not seen before. Although the mAP of VTS in the video is very high, the model almost detects that all VTS appear in the video. However, because the model's accuracy was only 97.9%, some VTS were still missing in detection. Not only the VTS are correctly predicted, but the probability for them is also high. The model expenditures up to 5.437 hours for 70 epochs that be trained slowly and takes more to wait. Because of not enough GPU to train, the model can not lay out the best mAP for the whole training process.

## 5.2. RECOMMENDATION

Instead of using logistic function, we can use another function to avoid vanishing gradient. In addition, it can be tried other optimization in the future.

REFERENCE

[1] Linlin Zhu, Xun Geng, Zheng Li and Chun Liu, Improving YOLOv5 with Attention Mechanism for Detecting Boulders from Planetary Images, 20 September 2021.

[2] Ioffe, S., & Szegedy, C. (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv, 20.11.2020.

[3] Mihir Rajput, Elephant Detector Training Using Custom Dataset & YOLOV5, 15 June 2020.

[4] Xingkui Zhu, Shuchang Lyu, Xu Wang, Qi Zhao, TPH-YOLOv5: Improved YOLOv5 Based on Transformer Prediction Head for Object Detection on Drone-captured Scenarios, 26 August 2021.