Research of Tortoises vs. Dogs Object Detection Algorithm Based on YOLOv5

Xueyan Shi
Beijing Lu He International Academy
Email: Aaron3963@163.com

Abstract

I created my own dataset of tortoises vs. dogs by using existing datasets and images from the Internet. Each class has 10,000 of images with labeling. I trained my model with YOLOv5 and compare it with its official data and its previous version. Besides, I merged my own dataset with YOLO's coco 80-class dataset and successfully trained it to identify 81 types of objects. I found out that YOLOv5 improved a lot on its weight size and training speed. YOLOv5 is also the first version of YOLO that got published on PyTorch. On the other hand, I discovered that YOLOv5 did not surpass YOLOv3 on accuracy. There are some potential solutions offered by Google's EfficientDet that may be useful if combined with YOLOv5 to make it more accurate.

Keywords: Object Detection; Dataset; YOLOv5

1. Introduction

The "Dogs vs. Cats" competition held by Kaggle[1] is a very famous event for anyone that is interested in computer object classification. Teams with all kinds of algorithms compete with each other to see who has the best accuracy and speed on detecting whether the picture showed contains a cat or a dog. All these algorithms are using CNN (Convolutional Neural Network). But the drawback of image classification is that the CNN can only recognize one object. This means that the image of Kaggle competition can only be a picture of a dog or a cat, it cannot be an image with more than one object. Plus, classification dataset do not show where exactly is the object located on the image, it only detects whether it is present. A more advance type of CNN, RCNN (Regional Convolutional neural Network) solved this problem. RCNN can detect multiple objects plainly because the algorithm divides the image into many regions, and each region is applied to CNN for detection. There are two types of RCNN, one is one-stage the other is two-stage. The one-stage RCNN is much faster than two-stage, and one of the most famous algorithms for one-stage RCNN is YOLO[2][3][4][5]. YOLO stands for You Only Look Once, clearly indicating that it is a one-stage RCNN. And recently scientists are developing YOLO's fifth version, and the community is excited about the new version of YOLO.

Based on my previous study about classification algorithms, I went deeper to explore about object detection. I am interested in YOLOv5 and I decide to make an algorithm research myself. I have a tortoise as my pet, but no one has created a Dogs vs. Tortoises dataset before, so I decided to be the first. Besides, by using a custom dataset for training, I can accurately verify and analysis YOLOv5's performance.

2. Tortoises and Dogs Dataset Establishment

The fundamental material for all artificial intelligence and deep learning is datasets. All kinds of algorithm need the support of massive data. The "Dogs vs. Cats" dataset on Kaggle contains 12500 images each, but it does not contain images for tortoises. I decide to create my own dataset of dogs and tortoises. There are three ways I can get all these images:

- Open datasets downloaded from ImageNet and Open Images Dataset
- Videos and pictures taken by myself
- Gathered from the Internet

The first two options are not enough, there are only 3000 images that are usable. The rest 7000 images came

from existing pictures from the internet.

2.1 Search Engine Crawler Design

I planned on using my own dataset to prove that YOLOv5 is powerful as the community said. I chose on detecting between a dog and a tortoise. The dataset for dogs were easy to find on the internet, but I could not find any usable dataset for tortoise. I had to create my own dataset for tortoise. The first step is to obtain images used for labeling. I thought of using a web crawler to download tortoise images from search engines. I chose Chrome as my browser and Google as my search Engine. The web crawler imitates human gestures of web-surfing and keep downloading every image it captures. At first, I was only searching "Tortoise" in Chinese, and the images downloaded was far less than expected. But Google supports all kinds of languages, and I tried using a dozen of different languages.

Tortoise in all kinds of languages					
Chinese	English	Japanese	French	German	Spanish
乌龟	Tortoise	カメ	Tortue	Schildkröte	Tortuga
Russian	Arabic	Portuguese	Italian	Hindi	Bengali
черепаха	سلحفاة	Tartaruga	tartaruga	टर्टलcolor	টার্টল

Table 1: All kinds of languages that stands for tortoise

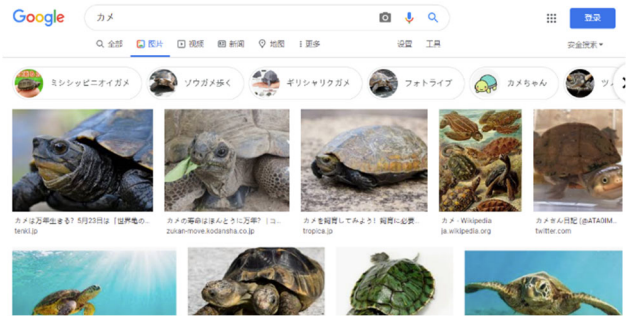

Figure 1: The Google webpage for tortoise in Japanese.

But this method created a new problem, some of the images are identical. These images have more than one language label, so they appeared more than once. I used an algorithm called MD5 to clear out all identical images.

2.2 Data Annotation

Raw images cannot be used as datasets, they need to be labeled to inform the computer where the object is located. Manually labeling every image is an impossible task, but thankfully most of my data were obtained from existing datasets, so they were labeled already. But there were still thousands of images from the internet that I had no choice but to label them myself. The annotation process was easy but repetitive. All I need to do is to draw a rectangle on the image that covers the object and label it with a tag of tortoise or dog.

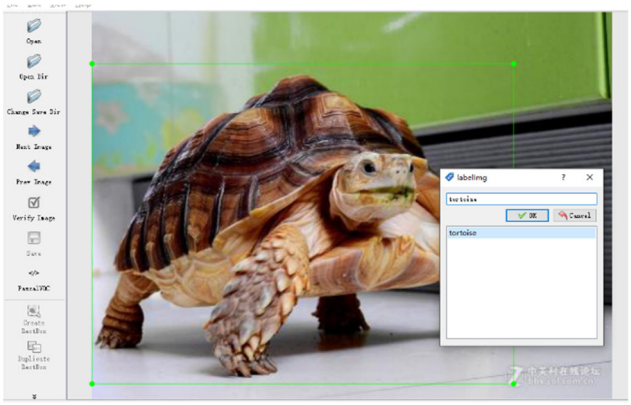

Figure 2: The interface of LabelImg.

The software above is called LabelImg, the one I had been using for data annotation. After setting up the saving directory, each label will automatically create a txt file with four corners of the rectangle and name as same as the image's name. The saved file can have different formats, one is Pascal Voc, the other is YOLO. In this case, I chose YOLO for sure. The is also an option of saving the file as a txt or a xml, both are familiar formats for labeling. YOLO only recognizes txt files.

2.3 Format Conversion of Labels

Not all of the images from free datasets are suitable for YOLO. There are two main kinds of label format, xml and txt. YOLO only recognizes txt labels, so I need to convert all the xml labels to txt format. To do this, I must first read the xml label and obtain variables and write them to an empty txt file. The xml labels' location in the image

are measured by exact pixels, but YOLO uses ratios. So, the variables need be converted.

The variables of xml labels can be represented as $\{x_{min}, y_{min}, x_{max}, y_{max}\}$. The variables used by YOLO can be written as $\{x_c, y_c, h_c, w_c\}$. x_c and y_c are the coordinates of the center of the label after normalization.

Here is example between a xml and a txt label:

```
<xmin>24</xmin>
<ymin>46</ymin>
<xmax>598</xmax>
<ymax>472</ymax>
```

Code segment from xml labeling file.

```
0 0.520067 0.525355 0.959866 0.864097
```

Code from txt labeling file, the first 0 is for class ID

The conversion formula is shown below.

$$x_c = (x_{min} + x_{\max}) * \frac{1}{2 * ImageHeight}$$

$$y_c = (y_{min} + y_{\max}) * \frac{1}{2 * ImageWidth}$$

$$h_c = (x_{max} - x_{min})/ImageHeight$$

$$w_c = (y_{max} - y_{\min})/ImageWidth$$

I developed a Python program that handles this task. The program also reads the name of the file to make sure that the name of the txt file can correspond to its image.

2.4 Dataset Showcase

To make the dataset a greater use, I made it open on GitHub for others to use. Link of the dataset: https://github.com/Aaron3963/Train_TortoiseVSDog_Dataset_with_YOLOv5. The dataset contains 10,000 images with labels for each dogs and tortoises. The format has converted into YOLO and users can download it to train without modifying the dataset. There are also few tools programs like LabelImg and ImageShow in the directory as well.

The image and their labels are separated, so it is difficult for users to see the labels on images. To solve this problem, I programmed a display program that showcases the dataset with number of class, labels, and framing so viewers can easily observe it. The labels in the image are created by OpenCV and each class has its own random color. The program can save the last displayed screenshot when it is closed (press ESC).

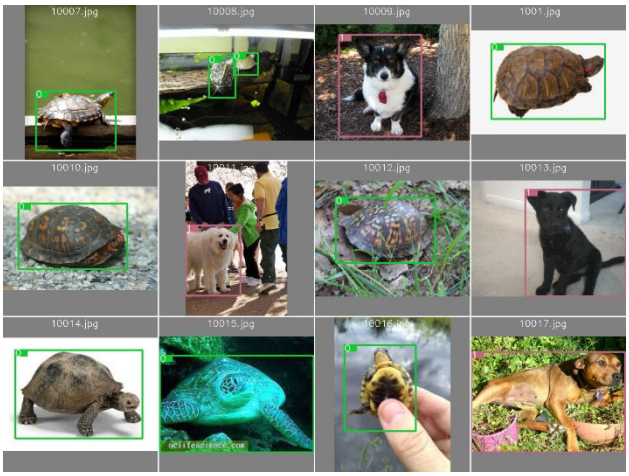

Figure 3: The screenshot of the ImageShow.py program. It can showcase multiple (In this case it is 3*4) images on the same page and auto refreshes every few seconds. The name of the file and all its labeling frames are directly displayed as well.

3 Custom YOLOv5 Dataset Training

YOLOv5 is open-sourced on GitHub, source code at: https://github.com/ultralytics/yolov5. I was using my laptop to do these training. I was using Window 10 with Visual Studio Code and PyTorch. My CPU is i7-7700HQ overclocking at 3.5GHz and my GPU is a GTX 1060 containing CUDA. I have 16G of RAM and the training is running on an SSD.

MY LAPTOP INFORMATION	
SYSTEM	Windows 10 Family Edition
WORK BENCH	Visual Studio Code
CPU CLOCK	3.5 GHz
GPU	GTX 1060 (mobile, 6GB VRAM)
RAM	16 GB
DRIVE	SSD

Table 2: My Laptop information chart.

3.1 Train Datasets with Four Models from YOLOv5

- Arrange Directory: The custom dataset should be in the same root directory as YOLOv5's folder. To train models, the yaml file must be configured first.
- Modify configuration files: The yaml file contains information like number of classes (nc) and class names. For me, I was only training tortoises and dogs, so my nc is 2 instead of the default 80. Useless classes need to be deleted as well. To be noticed, the order of classes in the yaml file should be the same with labels in the custom dataset. I was using tortoise as class NO.0, so the names section in the yaml should be like [tortoise, dog].The yaml file in the model should also be rewritten, changing the nc from 80 to 2.

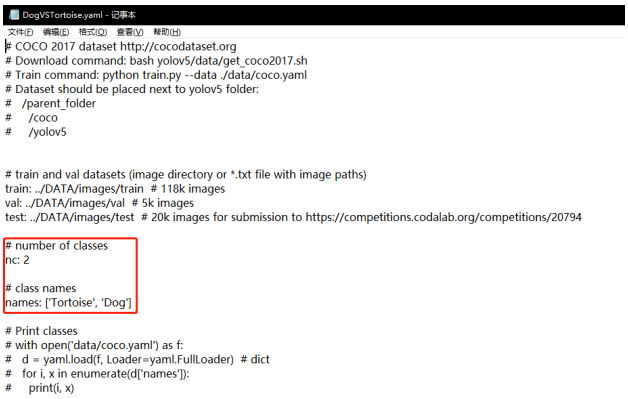

Figure 4: The parts that are modified in a yaml file.

- Training model: Training model uses the train.py file in YOLOv5, and YOLO provides a variety of options for training. There are four types of weights that I could choose. S, M, L, and X, as the size of the weight goes higher, it consumes more computing power, but it's performs better.

The training process should be running on the command line. And my command looks like this:

```
python train.py --img 320 --batch 24 --epochs
100 --data ./data/DogVSTortoise.yaml --
cfg ./models/yolo5m.yaml – weights ./weights/
yolov5m.pt
```

After training, YOLO will generate two weights as results. One is "best.pt", the other one is "last.pt."

"--img" represent the sample size. The training program will resize all images that are bigger than this value. I set it to 320, which means all images bigger than 320 will be resized to 320. Bigger sample size will use more resource, and I decide to use a relatively small one to save resources.

"--batch" means batch size, the number of runs at the same time. I tried bigger values like 32 but after a few runs the program will abort due to lack of memory. At last, I settled down for 24 and used this value for all of the other models (except x, because it was too big)

"--epochs" indicates the number of runs. To save time, I set it to 100. Later I found out that YOLOv5 need much less runs to get its curves stable.

Below is when training in Visual Studio Code. The program can display its loss values directly when training.

```
Epoch   gpu_mem     GIoU     obj      cls       total   targets  img_size

15/100   2.88G   0.02159  0.06682  0.0009749   0.08938   71     320:

 24%  181/750 [02:11<06:30,  1.46it/s]
```

The command lines during training can demonstrate parameters like losses, progress and losses, estimated time etc.

Figure 5: Ground truth labels of test batch 0.

Figure 6: Prediction labels of batch 0.

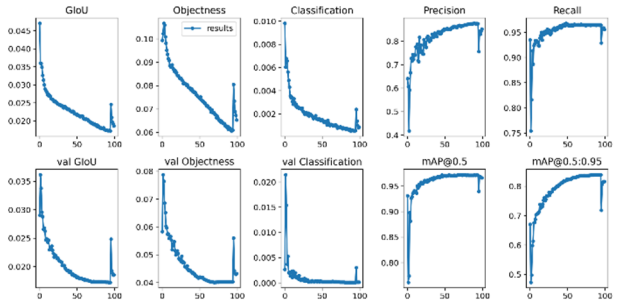

Figure 7: The result.jpg created after training. For each training, the loss values are logged in result.txt, and are graphed at the end of training. The txt file can also be used for comparison later.

3.2 Detection Results from Models

After training, YOLO will generate a best.pt weight, and we can use the detect.py to check our actual detecting results. The command of detection:

```
python detect.py --weights ./weights/best.pt --source ./sample --conf-thres 0.4
```

There are three kinds of ways to detect. Changing directory of the -- source into:

- A single file for detecting a mp4 video or image
- A directory for detecting all images in it
- Zero which will do real time detection by using the webcam

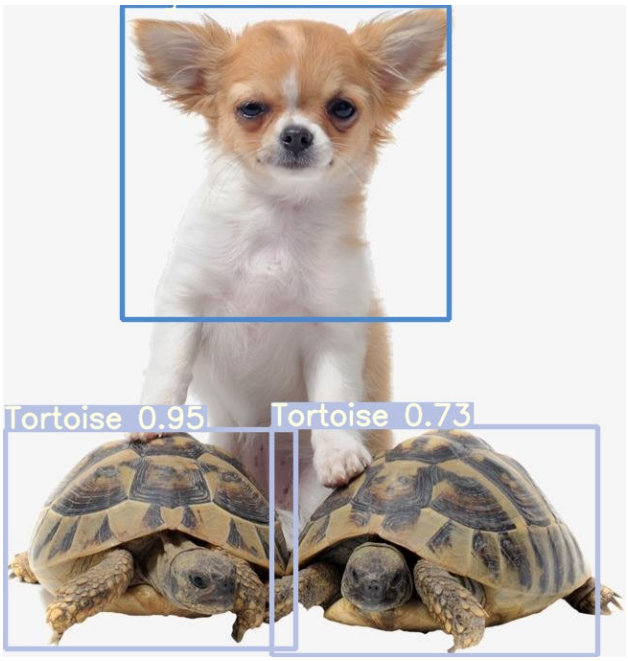

Figure 8: Detection result of a dog and two tortoises.

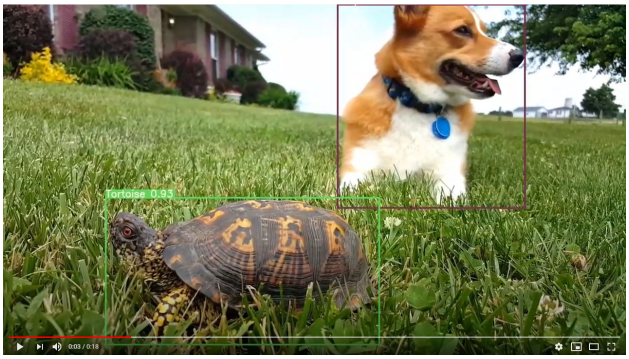

Figure 9: A screenshot from the detection video I uploaded. Video detection is based on frame by frame image detection and then merge them back to a new video. Link of video: https://youtu.be/bpQvIc9RKzQ

3.3 Train Dataset of 81 Classes Object

The original dataset provide in YOLOv5 is a coco dataset with 80 classes. Unfortunately, the class tortoise was not included. I want to make a well-rounded dataset but creating a dataset with two classes costed me a whole week for labeling, so I started trying to see if I can add my class of tortoise into the existing dataset of 80 classes.

- First, the config files must be changed. For example, the nc in yaml files should be 81 now and there should

be a new class of tortoise.

- Secondly, the labels of dog in my custom dataset was 1, in the 81-class model, it was 16. So, all the labels need to be replaced to new ones. And the tortoise's label is added at the end of the list so it should be number 80 instead of zero. I designed a helper program for changing labels. It simply reads which label it is and change it to the corresponding number.

```
# number of classes
nc: 81

# class names
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
        'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
        'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
        'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
        'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
        'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
        'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
        'hair drier', 'toothbrush', 'tortoise']
```

Figure 10: The classes and their corresponded label. The ones marked with red is the dog in place 16th and the added 80st class of tortoise.

The result turned out that my tortoise and dog images merged perfectly in to the 81-class model and this new dataset with 81 classes can be successfully trained.

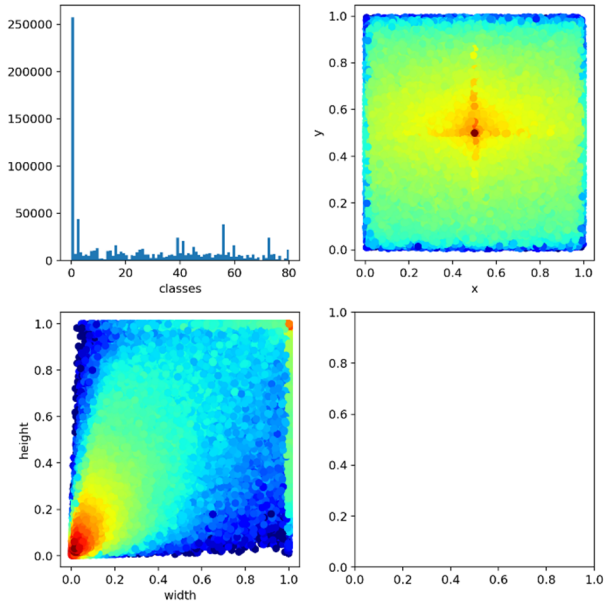

Figure 11: The distribution graphs of labels. The upper left corner displays the distribution of different kinds of labels. Label 0 is people, and that is why it got the highest percentage. The upper right corner is showing the center of the rectangular frame, and they are spread evenly. The lower left corner is how big the object is.

The command that runs dataset is slightly different from the previous one:

```
python train.py --img 320 --batch 24 --epochs 20 --
data ./data/coco81.yaml --cfg ./models/yolov5m.yaml
--weights ./weights/ yolov5m.pt
```

Besides on the changes of the config files, I changed the epochs from 100 to 20. This is because my dataset increased from 18,000 to 135,000. Training 20 times of 135,000 images exceeds training 18,000 images for 100 times already.

Figure 12: Ground truth labels of test batch 0.

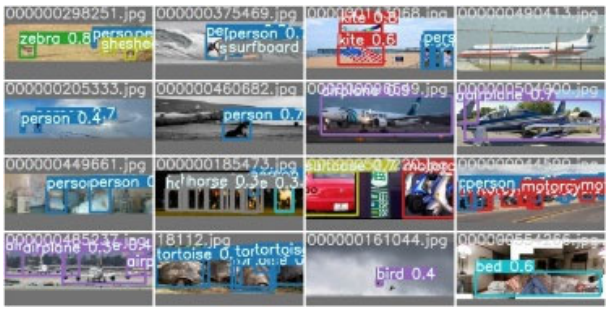

Figure 13: Prediction labels of batch 0.

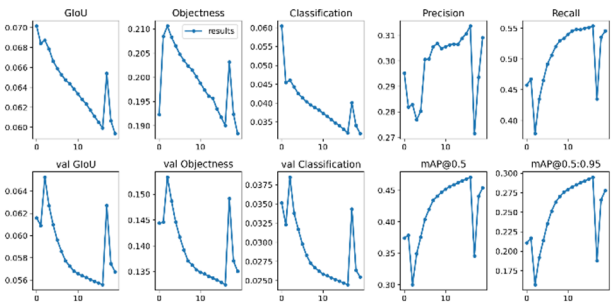

Figure 14: Training results of the 81-class model.

3.4 Detection Results from 81-Class Model

After I finished training, I used the same command in detect.py to check for detection results. The command was the same as when I was detecting my custom dataset.

```
python detect.py --weights ./weights/best.pt --
source ./sample --conf-thres 0.4
```

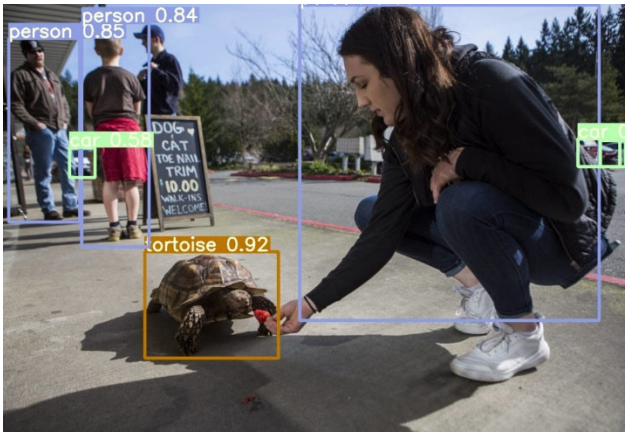

Figure 15: Detection result of one tortoise, three person and two cars from an image.

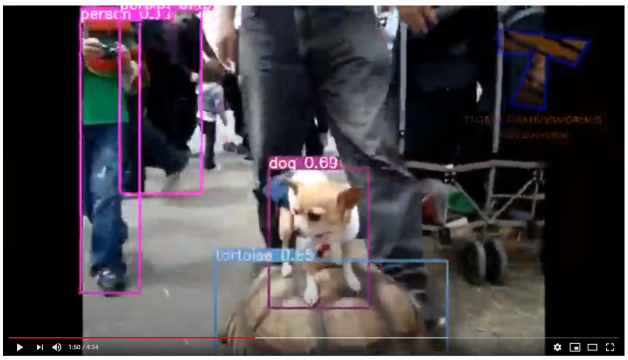

Figure 16: A screenshot of my detection video. Link of video: https://youtu.be/qC-RBYiOe9o

The conclusion is that the method of merging two datasets and train it as a whole is a success.

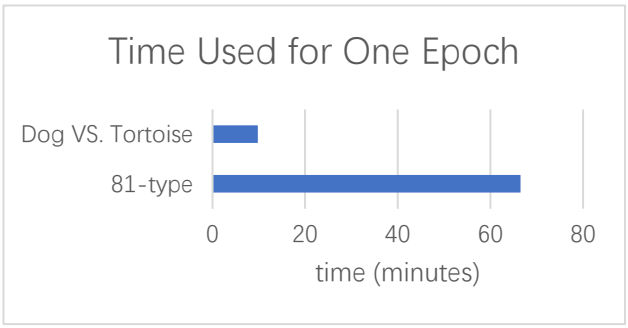

Figure 17: The training time increased 7 times as the amount of data increased from 18,000 to 135,000 images.

In figure 17, I used the same yolov5m weight and trained these two datasets under the same condition. The time used to train one epoch is directly proportional with the number of images contained in the dataset, it almost has nothing to do with the number of classes.

4 Discussion

4.1 Analysis of Five YOLO Models

In summary, I did 5 different kinds of models. Four of them are based on YOLOv5 and an additional one based on YOLOv3 is used as a control group.

The result.txt files created after each model training session can be used to plot graphs. YOLOv5 made a utils.py file and in this utility program there is a function that support multiple result.txt files plotted in one graph. This was perfect for a comparison between different models. I saved the results.txt file in every session and by combining all four kinds of results crated by four weights, I got a comparison graph. The command lines and the combined graph (Figure 18) are shown below.

```
from utils.utils import *
plot_results()
```

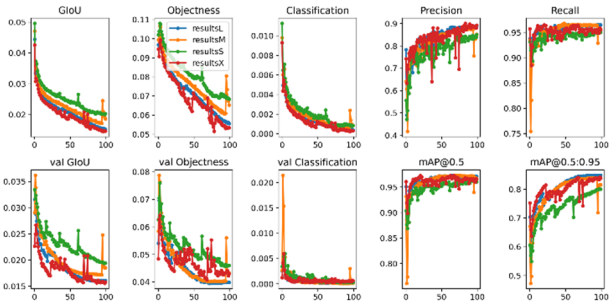

Figure 18: Training results of four model tortoise vs. dog based YOLOv5.

Results shows that the precision and losses values are similar between all kinds of weights. The more complex the model is, the more precisive the results are.

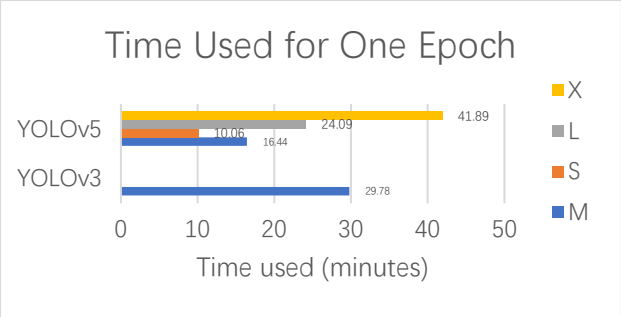

Figure 19: Comparison graph of amount of time used to train one epoch under controlled condition.

All models were trained with the same dataset and variables but there are great differences between their time spend on training. By comparing in YOLOv5, Figure 19 shows clearly that as the size of the model goes bigger, the longer the time it will take to train. Between different versions of YOLO, v5 did a better job than v3. The training time of YOLOv5m model is half comparing to YOLOv3 ones.

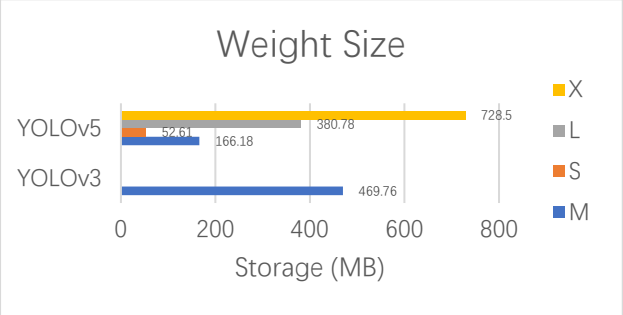

Figure 20: Comparison graph of weight size between YOLOv5 and YOLOv3.

On weight sizes, I see that YOLOv5 improved a lot on its weight size. I think this is a step forward to make YOLO more adaptive to mobile platforms since they have less calculating power and a lighter model can boost their performance.

	S	M	L	X
Mean time (min/epoch)	4.6	9.8	14.45	23
Total time (h)	8.16	16.4	19.5	42.8

Table 3: Results show that there is a positive relationship between weight size and time spend.

The results are astonishing. Every weight reaches the precision rate of 90% within 100 epochs. And as expected, the larger the scale of the weight goes, the longer it takes for completing one epoch. I had to pause the training because I am using my computer to do other things, so there are some spikes within the curves, but after a few more epochs the curve went back to normal.

4.2 Optimization and Future Works

Although YOLOv5 has better training speed and lighter weight size, it has its own problem. The precision of v5 is lower than YOLOv3. YOLOv5 sacrifices its precision in order to increase its adaptivity in different platforms.

I know that Google has find a new neural network structure called EfficientDet[4], and the BiFPN (Bi-directional feature pyramid network) structure inside may help improving YOLO's precision.

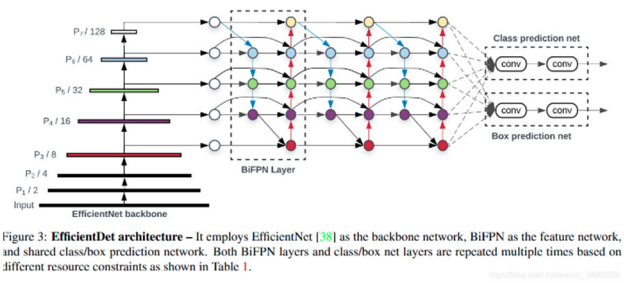

Figure 3: **EfficientDet architecture –** It employs EfficientNet [38] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

Figure 21: The architecture of EfficientDet from its

official papers.

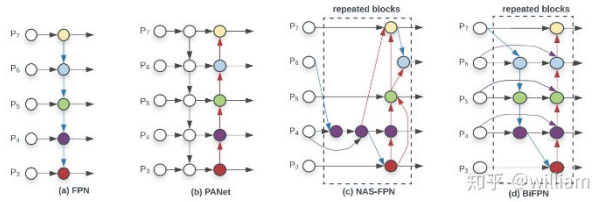

Figure 22: Structures of different algorithms. YOLO is applying PANet, but it appears that BiFPN is more efficient.

The great adaptivity of YOLOv5 made it suitable for cellphones and other machines with little calculating power. Many applications can make good use of it. And in the future, I will research more about neural networking and try to combine YOLO with BiFPN. But currently, YOLOv5's official papers are still unreleased. This means that Ultralytics is still working on improving YOLO, I hope they can gain progress and make this algorithm even better.

5 Conclusion

After I did some research on YOLOv5, I conducted a research performance of it. To make sure the validity of this test, I used my own dataset. The 20,000 Dogs vs. Tortoises dataset contains images from both the Internet and myself. Besides on testing YOLOv5, the dataset is also open for others to use. I applied all four different kinds of weights (S, M, L, X), as well as another one from YOLOv3.

The method of merging two datasets and trained them as a whole is a success. No one has tried it so I figured out the way be myself. By changing the configuration of classes and merge the dataset together, I successfully combined my tortoise dataset with the coco 80 class dataset provided by YOLO.

The detection result shows that YOLOv5 improved by decreasing its weight size and shortening both the training time and detection spend. But on the other hand, YOLOv5 sacrifices its accuracy on detection compare to YOLOv3, which may be fixed if combined with BiFPN.

References

[1] Official web site of Kaggle Dogs vs. Cats: https://www.kaggle.com/c/dogs-vs-cats

[2] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. *You Only Look Once: Unified, Real-Time Object Detection*

[3] Joseph Redmon, Ali Farhadi. *YOLO9000: Better, Faster, Stronger*

[4] Joseph Redmon, Ali Farhadi. *YOLOv3: An Incremental Improvement*

[5] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*

[6] Mingxing Tan, Ruoming Pang, Quoc V. Le. *EfficientDet: Scalable and Efficient Object Detection*

Acknowledgement

I would express my gratitude to my computer science teacher, Guangliang Qiu, who provided me the information of this competition and gave me insightful guidance and instructions. I would also thank my parents who provide me a laptop with a dedicated GPU so I could perform all these tests with ease.