Formula Student—Week 2

Approach research:

1. Traditional object detection with OpenCV

Firstly, we considered to use the Opencv3 to solve this problem. It is a traditional image processing approach. The technologies we researched include the Histogram of Bag of Words(BOW), image pyramid and sliding window. We tried to use the Bag of words to build the classier. By vectorizing the images with the bag of words and put each data into multi-dimensional space. Applying SVM to give the grades for the classification. Then using the sliding window to detection multi-objects in the same image and finding the location of it.

Problem: There is a very big problem when applying the sliding window. Because we do not the real scale/size of the object we want. So we have to use different size of the window to detect. That would generate lots of sub-areas and they all need the classifier to predict. Thus the calculation will be tremendous. To increase the speed, we could simplifier the classifier for that reason. The main idea is trying to decrease the number of the sub-areas. Thus, after the research, we come up with the idea that applying RCNN to solve the problem.

2. Fast-RCNN

Searching on the Internet, we find there is a mature object detection algorithm based on neural network— RCNN, and in the meanwhile a lot of algorithms related to RCNN have developed and used widely. After comparing the efficiency of each algorithm, we try to use fast-RCNN to detect cones.

Problem: The configuration of tensorflow was so complicated even though we have found the way to train our model with fast-RCNN. We were stuck into some installing errors for long which was not productive. Finally, we decided to use YOLOv3 which worked well with other students to finish our task.

3. YOLOv3

Introduction: You only look once (YOLO) is a state-of-the-art, real-time object detection system. Different with traditional object detection method or RCNN using sliding windows, YOLOv3 apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities. YOLOv3 is faster than RCNN and traditional methods and is the most promising way to go for real time object detection given limited resources.

Disadvantage: YOLOv3 is not as accurate as RCNN sometimes and easy to have some location errors.

Advantage: Darknet which YOLOv3 is based on is easy to install and configure. The tutorial in official YOLO website is friendly to use and there is a ROS package available.

This will work in Ubuntu.

Procedure (https://pjreddie.com/darknet/yolo/):

1. Git clone the depository (`https://github.com/pjreddie/darknet`). Next, open the makefile in a text editor and change “GPU=0” to “GPU=1” if you want to use a GPU and have cuda installed. In your darknet directory execute “make” to create an executable.
2. In /darknet/cfg open the yolov3-tiny.cfg configuration file and change classes = 3 in line 135 & 177 (in our case for three different cones) and the amount of filters = 24 in line 127 & 171 according to this formula (filters = (classes + 5)*3).
3. Create our own training dataset of labeled images. The final data set should contain all the images (.jpg) as well as the identically named darknet files (.txt) that containing the bounding boxes and class information. We chose 60 images from different videos that were split to 45 for the train dataset and 15 for the validation dataset.
4. Create a names.txt file containing all the labels for the classes, important is that each name is in a new line. In our case this would be (yellow_cone \newline red_cone \newline blue_cone)

5. Create a train.txt file containing all the pointers to the training pictures (the corresponding darknet files are detected automatically). Similar to before each pointer is in a new line. Example (/usr/dog_pics/img0012.jpg \newline /usr/dog_pics/img0013.jpg). Do the same for the test data set, but call it differently such as “valid.txt”.
6. Create a mydata.txt that contains the below:

classes= 3

train = #pointer to train.txt

valid = #pointer to valid.txt

names = #pointer to names.txt

backup = #directory where the trained weights will be saved, needed for step 8 e.g. /darknet/backup/

7. Train your tiny-YOLOv3

```
./darknet detector train /mydata.txt  /cfg/yolov3-tiny.cfg
```

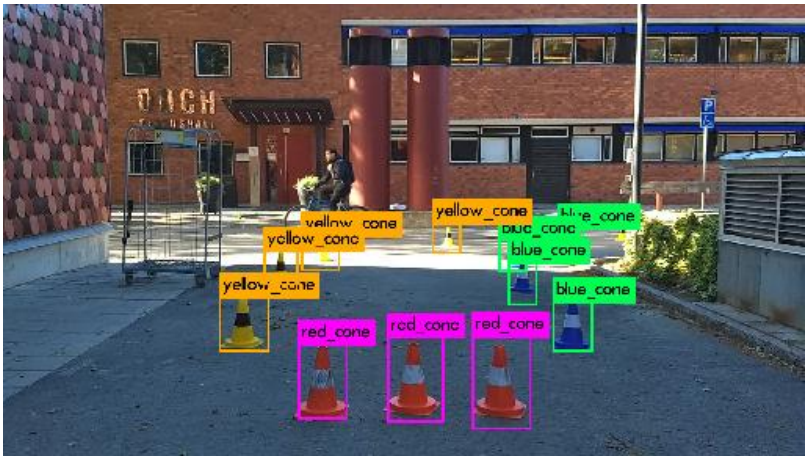
8. Test our trained model with other dataset and check the behavior and accuracy of each class.

```
./darknet detector test /mydata.txt /cfg/yolov3-tiny.cfg
/mytrained.weights
```

Future work:
We are going to research on the concept of YOLOv3 algorithm and understand more detail about it. Next step is trying to use YOLO model we trained to process live videos, and furthermore combine it with ROS system.

Here are some screen-shots of our test result:


ions
red_cone
blue_cone
(x=772, y=11) ~ R:181 G:179 B:158


DICH
yellow_cone
red_cone
blue_cone