# Spring Midterm Progress Presentation and Report

# Unusual Object on the Road

Team (Group 32) name: Teaching AutoPilot to Dodge

Author: Xilun Guo, Tanner Fry, Basil Al Zamil

CS 462: Senior Capstone Winter 2017

Oregon State University

**Abstract**

In this document, we will discuss our Capstone project progress over the Spring term up until Week 7 of Spring term. This is includes the summary of last terms progress with the project purposes and goals. It also contains information on our current status, progress, problems, and solutions of our project. The document include sample images and their descriptions, testing results of the project test cases, data results, and data analysis.

## CONTENTS

1 Summary 3

2 Recap 3

3 Current Progress 3

4 Remaining Work 4

5 Problems Encountered and Solutions 4

6 Interesting Pieces Of Code 5

7 Write Up 5

8 Descriptions Of Final Results 6

## 1 Summary

## 2 Recap

The Unusual Objects on the Road project goal is to find objects that cannot be recognized by object recognition algorithms along with testing bad weather and lighting conditions. Based on the Tesla crash in June 2016, where there was a deficiency in Tesla's image recognition algorithm used in the car that led to the death of the driver, from this instances we believe that there exists deficiencies in image recognition algorithms that could lead to similar incidents. The software used in the Tesla vehicle incorrectly registered a Semi-truck as a cloud due to the angle of sunlight reflecting off of it, which led the algorithm to attempt to drive through the semi-truck. Our purpose is to compile an image dataset from the perspective of an autonomous car that causes a significant portion of object recognition algorithms to fail. From this we hope to give back an image set that object recognition algorithm developers can use to more thoroughly test their algorithms.

To achieve the goal, we collected video feeds from dashboard cameras. From there, we modified and cleaned up the data to pull out images from the video feeds into three major cases: sunny, rainy, and snowy. We then filtered out the suitable images which have test conditions we want. From there we downloaded individual image recognition algorithms available from the list on the Cityscapes website. For all algorithms we tested, we had to do sufficient research in the fields that we might touch, such as programming languages, numerical computing environment, Linux usages, and library functions; in order to complete the environment setup and image data testing. This gave us valuable information on the safety of these types of algorithms and where their faults lie (what objects cannot be recognized by which algorithm). The information, feedback, and data can then be handed back to the algorithm designers, so that they can improve the algorithm and hopefully be a cause of saving future lives.

## 3 Current Progress

We have got most conditions for testing the algorithms, including rainy, which contains bad sights; snowing, which was contains extremely bad weather; and sunny, which contains direct lighting effects on the camera. Also, the videos collected contains lots of different types of cars, trucks, buses, various objects and road signs, and people with diverse styles of clothing o. We followed the instructions described on Github CityScapes, where we had the labeling script system to work as suggested from the client.

For the algorithms to work as desired, we need to setup their environments and their dependency software. Thus, for every algorithm out of the four algorithms: FCN 8s, PSPNet, RefineNet, and Dilation, we have setup their environments and dependencies that reached our client's expectation. We installed the cityscape trained model for each individual algorithm, but we do not have full permission to update or install any libraries or software to the Steed server, so we had to create a specific virtual environment for updating and/or installing what we needed.

We collected our new dataset during the first three weeks of spring term by recording video footage on the road while driving through the dash camera provided by our client. We collected and processed around 2000 images of sunny and raining cases and 600 images of snowing case using the algorithms above. We got all image results from our dataset by running those algorithms and put them into labeled files. After some analysis, we found and agreed that the RefineNet algorithm is the closest to a real-world algorithm that companies would use on their auto-driving cars. FCN is not a good algorithm for recognizing objects on the road, but we still use it as a learning tool and to help get familiar with how image recognition algorithm works. Plus it was a good starting point to do research in this field. First we got expected images results by running PSPNet; however, it almost always fails on all extreme cases. For example, any objects under the sun would be recognized as empty road, and almost none of the objects can be correctly recognized under raining and snowing cases. Until now, We have manually done analysis of around 200 image results for RefineNet on all three cases, and we indeed found several major failure cases, which are more explicitly discussed later in the document. Above is the majority of the progress we went through from the end of the last term until now. We are confident that the goal of our project is achieved, and our client is satisfied with the results.

## 4 Remaining Work

Nothing that needs to be done for our project requirements. However, we do plan on doing more image processing just for fun in an attempt to find result outliers from the algorithms.

## 5 Problems Encountered and Solutions

We encountered an a sequence of obstacles with setting up the environment for the dilation algorithm. For the dilation algorithm to work, we needed the following software to be installed on the environment: caffe, numpy, numby, and opencv. The first obstacle we encountered was with caffe. The virtual machine environment on the steed server indicates that it had caffe installed. However, the system on the steed server gives a message indicating otherwise. Thus, we contacted the IT, and asked them to install caffe for us. However, the error persisted. After a few emails with the IT, we found that the libraries and caffe source were not set on the environment, so we had to set it from our end every time a user logs in the steed server. We set up the libraries through the following commands:

```
$ export PYTHONPATH=/scratch/caffe-master/python
$ export LD_LIBRARY_PATH='/usr/local/apps/cuda/cuda-8.0/lib64:/usr/local/lib:/usr/local
/lib64:/usr/lib64'
```

The second obstacle we faced was with installing numpy, which is another dependency of dilation algorithm. Since we do not have permission to installed software on the server, we need to use set up a virtual environment to install software. We could successfully installed numpy on the virtual machine on steed, but the system would not find caffe on the virtual machine, although caffe was installed on the actual system. We approached this problem by seeking help from Jialin, who is a Ph.D student of our client. Jialin was helpful and she met with us to help us through this obstacle. She successfully had caffe and numpy installed interactively on the virtual machine. We attempted to install the remaining software, numby and opencv, but we faced errors and Jialin had to leave.

The third obstacle was installing numby and opencv. Through a lot of trial and error and reading a great amount of documentations, we were able to install numby and opencv through running a python script on our virtual environment on steed the server.

The final obstacle that we encounter was with using a model with the algorithm, which was relatively easy to overcome. After we had all the software installed on our environment, we still could not have the algorithm to run, where the algorithm would stuck without feedback. After reading in the dilation documentation, we found that we needed to have a model to run with the algorithm in order for the algorithm to processes images and to function properly. We finally had the dilation algorithm to processes our data sets and product image labeled results.

## 6 Interesting Pieces Of Code

```
case 'cityscapes'
        isVal = true;
        step = 125; %125=500/4
        data_root = '/data2/hszhao/dataset/cityscapes';
        eval_list = 'list/cityscapes_val.txt';
        save_root = 'mc_result/cityscapes/val/pspnet101_713/';
        model_weights = 'model/pspnet101_cityscapes.caffemodel';
        model_deploy = 'prototxt/pspnet101_cityscapes_713.prototxt';
        fea_cha = 19;
        base_size = 2048;
        crop_size = 713;
        data_class = 'objectName19.mat';
        data_colormap = 'colormapcs.mat';
```

The step and base size are two magic number, and we can't figure out the logic behind. The step is based on the number of image data and the number of GPU we can access to. The base size number is the based size of scaling.

```
cd ./libs/matconvert
run(' ./matlab/vl_setupnn.m')

vl_compilenn('enableGpu', true, ...
             'cudaRoot', '/usr/local/cuda-7.0', ...
             'cudaMethod', 'nvcc', ...
             'enableCudnn', true, ...
             'cudnnRoot', '../../libs/cudnn5')
```

Downloading and installing Matconvnet and call the path to matconvet. Compiling MatConvNet using the MATLAB function vl compilenn. First check GPU support and cuda are available with the specific version. MatConvNet is compiled with cuDNN support and point Matlab to the paths of both the CUDA and cuDNN libraries.

## 7 Write Up

The structure is including in the Recap Section.

## 8 Descriptions Of Final Results

Figure 2 is the processed image date result we get from Figure 1 by running the RefineNet algorithm. The ground truth image in figure 1 is under sunny test cases, where the direct sun light is affecting the camera in campturing accurate and clear image of the objects. In figure 2, the purple color represents the road, pink represents the side of the road, blue represents cars, gray represents buildings, red represents people, and green represents trees. As it shown, the car in front of our vehicle was not recognized. This is a huge potential of an accident by driving into the front car, since the algorithm thinks that the front car does not exist. The reason of this bug could be the lighting reflection on the windshield shelter from the car, or the sun is exposure in the ground truth image, which make the algorithm unable to correctly go interpret that part of the image.

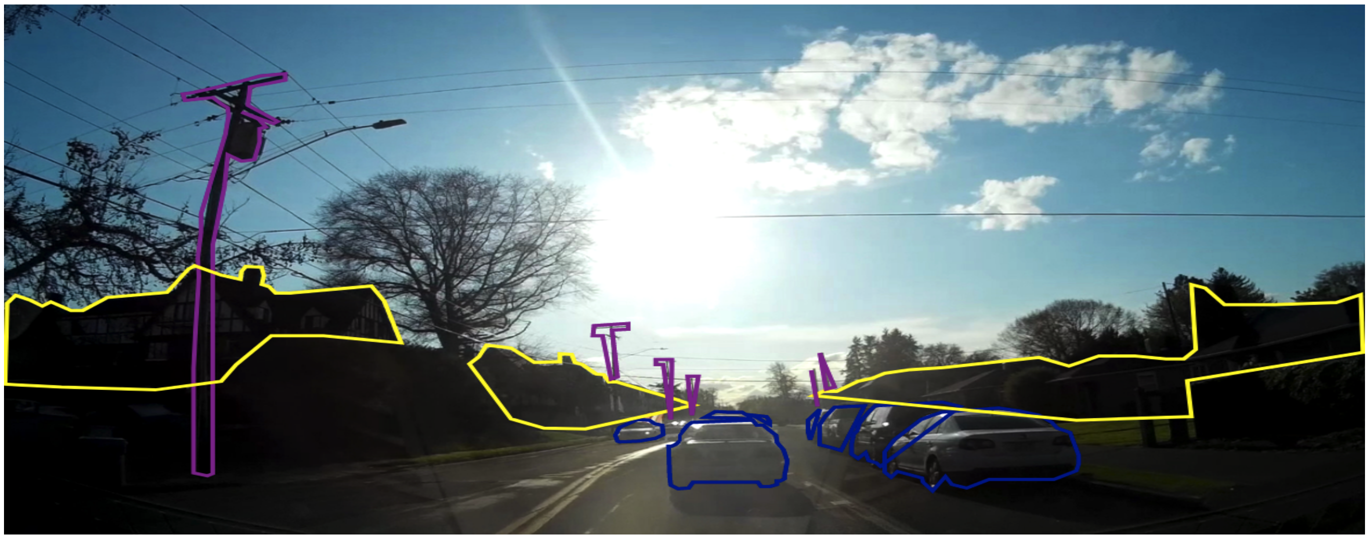
Figure 1: Ground Truth Image

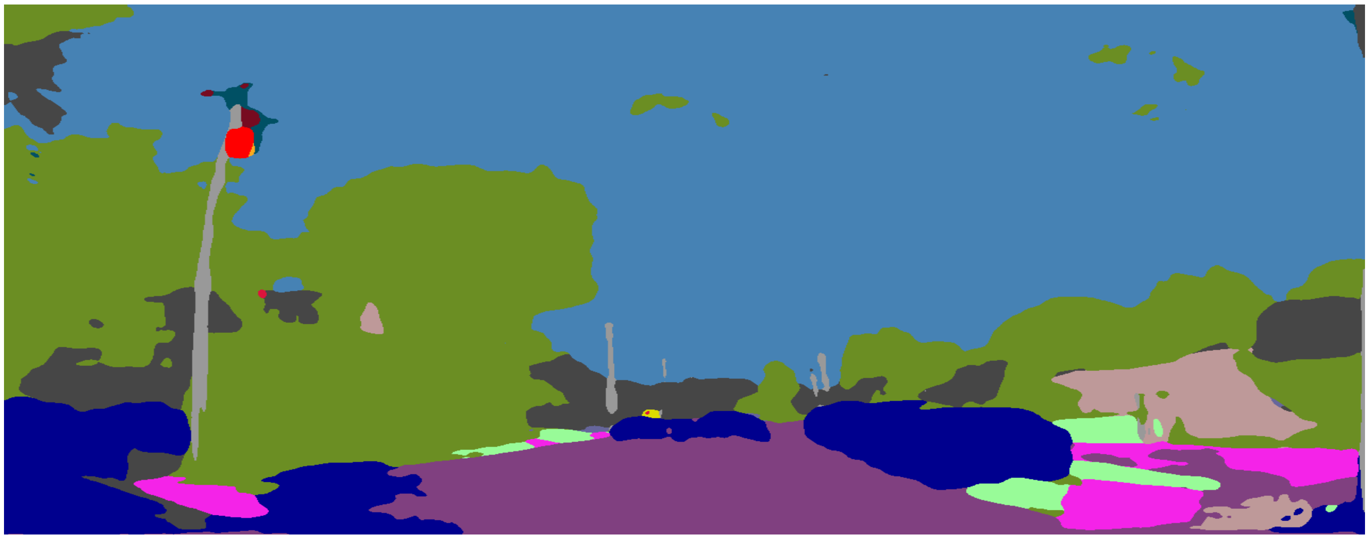
Figure 2: Processed Image

In figure 3 and 4, the color settings are the same as in figure 1 and 2. The yellow color in figure 4 represents the road signs and the pink is the curb. The ground truth image is under raining test cases so the water on the windshield could affect the camera. From the ground truth image, you can clearly see there is a person behind fence. The algorithm should be designed to recognize objects or at least people behind fence or other similar building, which are legally put on the side road. Since the person is not recognized by the algorithm, there is a potential accident of hitting the person.

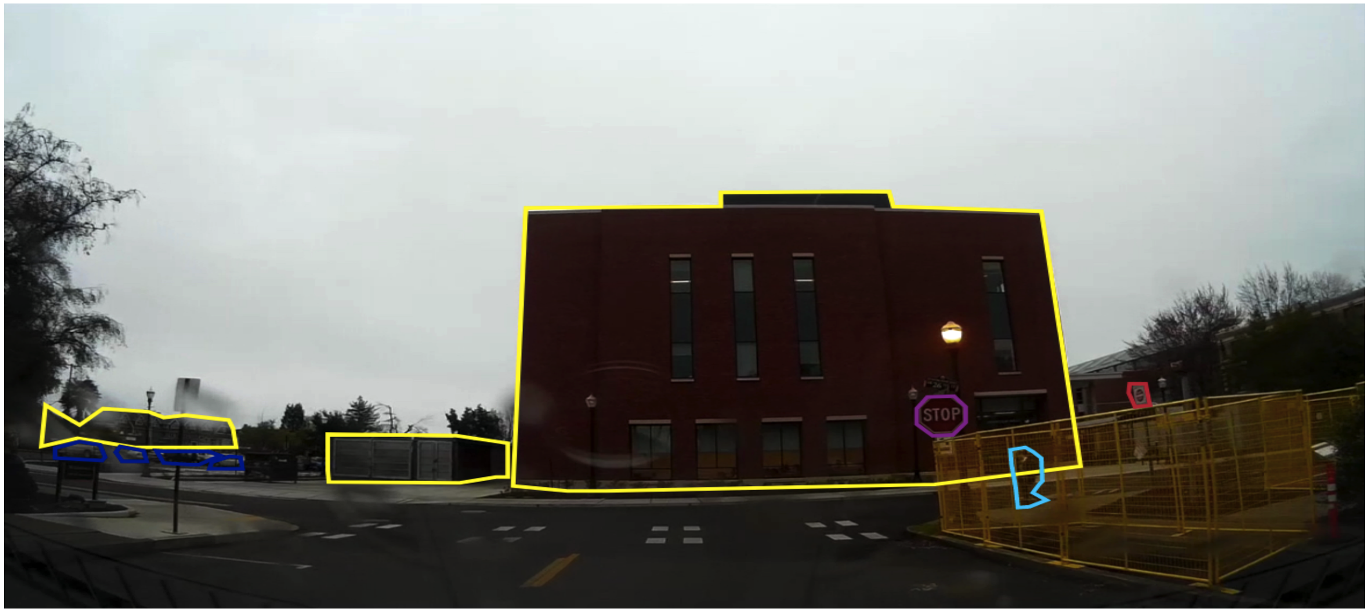

Figure 3: Ground Truth Image

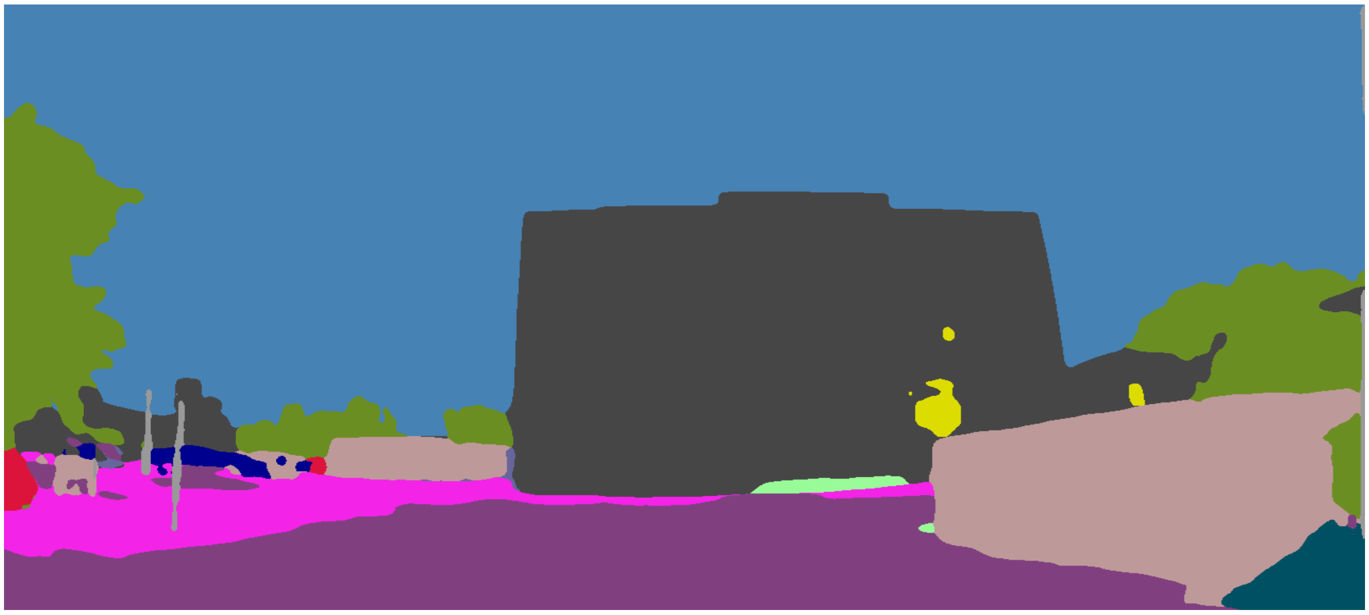

Figure 4: Processed Image

## References

[1] https://github.com/mcordts/cityscapesScripts. *Part of the Cityscapes team* Marius Cordts, Mohamed Omran.

[2] https://www.cityscapes-dataset.com/ *Cityscapes subset of Daimler* 2016 Cityscapes Dataset by Marius Cordts

[3] http://pedrokroger.net/choosing-best-python-ide/ *Choosing the Best Python IDE* Dr. Pedro Kroger