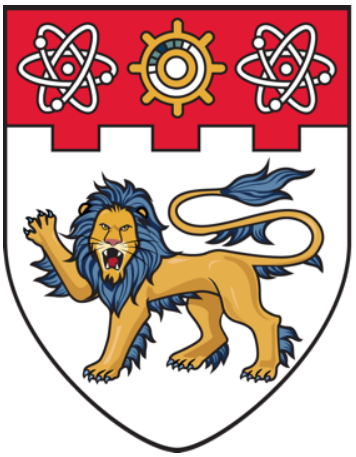

NANYANG TECHNOLOGICAL UNIVERSITY

SINGAPORE

# FYP MID-TERM REPORT

Hu Xiaoxiang
U1521319A
EEE

11 Nov, 2017

# Contents

# 1 Introduction

Object detection is a technology related to computer vision and image processing. In terms of algorithm, classical techniques, including point processing, edge detection and morphological operation, can be used to detect objects edge of a certain class(such as humans, buildings or cars) in digital images and videos. With the development of deep learning in recent years, new techniques(such as Convolutional Neural Network(CNN)) appears to show its high accuracy on object detection task.

The objective of my project is to apply deep learning techniques on object detection in the auto-driving environment. The deep learning techniques includes Extreme Learning Machine(ELM) and CNN. By the end of this project, the neural network is supposed to provide capability to differentiate object such as vehicles, bicycle riders and pedestrians.

# 2 Research Progress

Object detection can use both classical method and deep learning method. Each of them has its own advantages and disadvantages. Typically, the classical method is used as a pre-processing method for noise removal, contrast enhancement or image segmentation. Especially, the pre-processing method can be used for data augmentation to improve overfitting problems, which is caused by having too few samples for a deep learning model to learn from. On the other side, deep learning method has its advantages over feature engineering since it can learn features of object by itself. This is very essential for object detection under auto driving environment, because the complexity of image features has becomes too high that no classical algorithm can handle it both efficiently and accurately. To combine and utilize different methods on object detection in future, my research mainly focuses on two field:

1. Fundamentals of deep learning techniques and utilization of mature deep learning framework, such as TensorFlow and Keras.
2. Computer vision basic and utilization of computer vision tools, such as OpenCV.

## 2.1 Deep Learning Field

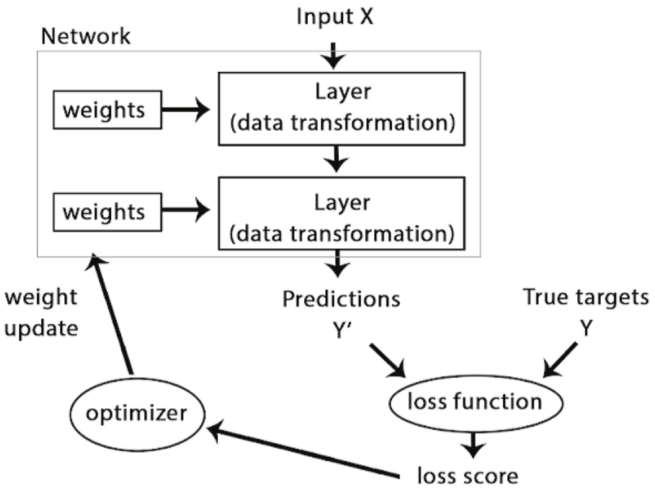


Figure 1: Neural Network Model

The major deep learning technique on which I researched for the first semester is CNN. The core building block of CNN is the "layer". The layer is a data-processing module which can be conceived as a "filter" for data. It extracts representations or features out of the data fed into them. This representations determines the weight of nodes within each layers and the weight will change dynamically after each iteration of learning progress according to input samples and difference between output prediction and target label, which is usually known as the loss score. To update the weight on each node, the CNN use "backpropagation" algorithm.

```
from keras import layers
from keras import models
from keras import optimizers

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu',
                        input_shape=(150, 150, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(128, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

List 1: Use Keras creating CNN model

The CNN framework in TensorFlow and Keras consists of two main layers, which are convolution layers and pooling layers. The fundamental difference between a normal densely-connected layer and a convolution layer is this: dense layers learn global patterns in their input feature space, while convolution layers learn local patterns through a convolution kernel. Due to this characteristic, the patterns CNN learns are translation-invariant. Unlike convolution layers splitting entire image into many small features, the role of pooling layers is to aggressively downsample feature maps. By doing this, it allows successive convolution layers to learn spatial hierarchies of patterns through increasingly large window and hence reduce the influence of overfitting [1].

Knowing the fundamental component of CNN, what I have exercised is to use a pre-trained model, which is trained by Microsoft Common Objects in Context dataset, and detect the bicycle rider on images. The purpose to use a pre-trained model is that it can help to reduce the training time significantly.

To train a CNN model using TensorFlow backend, it needs to prepare some labeled training data. What I tried is to download pictures from google search directly using a image crawler.

```
# Command for fetch urls from google image search on chrome:
# a = document.querySelectorAll('img')
# document.body.innerText = Array.prototype.map.call(a,x=>x.currentSrc)

import os
import re
import urllib.request

root_path = os.path.abspath('~/Documents/images/')
```

```
object_class = 'car'
object_path = 'img_'+object_class

url_path = os.path.join(root_path, object_path, object_class+'URL.txt')

result = []
with open(url_path, 'r') as file:
    while True:
        line = file.read(1024)
        if not line:
            break
        a = line.split(',')
        for i in a:
            item = re.match('^https', i)
            if item is not None:
                result.append(i.strip())
n=0
for url in result:
    try:
        figure_name = ''.join([object_class, '_fig_', str(n), '.jpg'])
        figure_path = os.path.join(root_path, object_path,figure_name)
        urllib.request.urlretrieve(url,figure_path)
        n += 1
    except:
        pass
```

List 2: Image Crawler

The next step is to label the images. The tools used for labeling is called "labelImg" [2] (Figure 2). After labeling, this tool helps to generate a ".xml" file to store the label information (List 3).


Figure 2: Labeled Image

```
<annotation>
  <folder>img_bicycle</folder>
  <filename>bicycle_fig_1.jpg</filename>
  <path>/home/seanhxx/Documents/images/img_bicycle/bicycle_fig_1.jpg</path>
  <source>
```

```
    <database>Unknown</database>
  </source>
  <size>
    <width>275</width>
    <height>183</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>bicycle rider</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>85</xmin>
      <ymin>15</ymin>
      <xmax>257</xmax>
      <ymax>173</ymax>
    </bndbox>
  </object>
</annotation>
```

List 3: Label Infomation

The final step of training data preparation is to generate a TensorFlow-recognizable data format [3] (List 4). The number of training dataset is 150 and validation dataset is 16.

```
def create_tf_example(group, path):
    with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:
        encoded_jpg = fid.read()
    encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(features=tf.train.Features(feature={
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
```

```
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text': dataset_util.bytes_list_feature(classes_text),
        'image/object/calass/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example
```

List 4: Generate TensorFlow Record

Since the training data is prepared, the execution of following procedure can feed data into the neural network. As what I mentioned before, to reduce training time and increase detection accuracy with small number of training data set, a pre-trained model is used at current stage of research. Besides, TensorFlow Object Detection API provides the function to train pre-trained model with new data set [5]. The training loss value is shown in Figure 3.

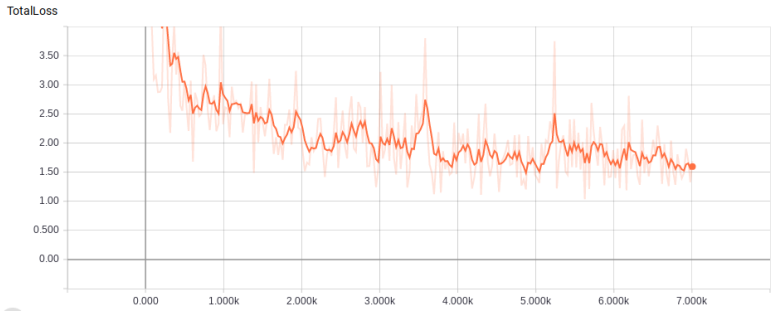


Figure 3: Training Total Loss

As shown in Figure 4 and 5, the test image shows the percentage of reliability of the detection. Due to the limit number of training data, the loss value is relative high and occasionally some test images actually cannot be recognized even. This is the overfitting problem. To solve it, some image processing techniques are needed for data augmentation.


Figure 4: Test Result 1

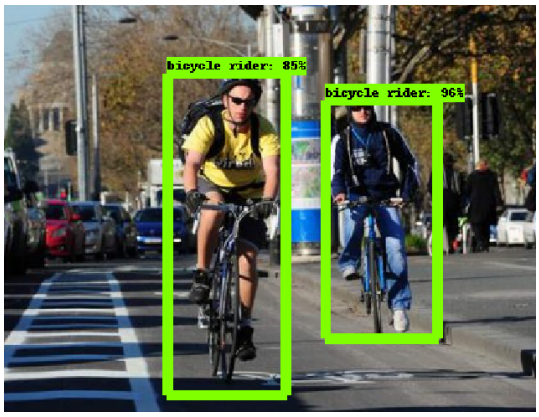


Figure 5: Test Result 2

## 2.2 Computer Vision Field

The research target on computer vision field for the first semester is to learn some basic image processing techniques and use openCV via python API.

To reduce the calculation cost, a color image is typically converted to a gray scale image before fed into deep neural network. To reduce the noise, Gaussian filter or alpha-trimmed mean filter can be applied. Point processing tools like contrast stretching is used to improve the illumination of images which are under exposure. As what I mentioned before, image transformation or degradation function can also be used for data augmentation to mitigate overfitting problems.

OpenCV is a open source computer vision library. It provides plenty of embedded computer vision functions to use immediately, as well as some APIs to control camera devices on different hardwares. For example, the web camera on laptop can be accessed directly through openCV and is used as a testing of the pre-trained model at real time (Figure 6).

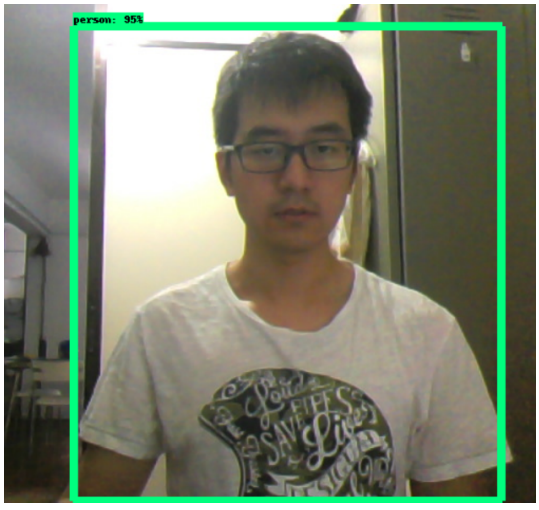


Figure 6: Real Time Object Detection

# 3 Plan For Next Stage

Based on my current research progress, the plan for the next step is to study and try to implement one of the state-of-the-art deep learning techniques, such as CNN-ELM and Mask R-CNN. Another potential research path is about Recursive Cortical Network (RCN) on object detection area [4]. Currently, CNN can achieve a good accuracy on object detection task with enough number of lossless training samples. However, in a real-time auto driving environment, it is impossible to collect all kinds of perfect images of street under different situations. Moreover, long pre-processing and training time and high computation cost required by CNN also limit its application on auto driving car. To solve this problem, a new neural network RCN, which mimics human brain and has achieved high accuracy on CAPTCHA recognition, may be used for object detection on auto driving environment.

# References


[1] Chollet, F. (2017). Deep Learning With Python. 1st ed. Manning Pubns Co, pp. 111-112

[2] GitHub (2017) LabelImg [online] Available at: https://github.com/tzutalin/labelImg

[3] Learning Python (2017) Object Detection [online] Available at: https://pythonprogramming.net/

[4] Vicarious (2017) Common Sense, Cortex, and CAPTCHA
[online] Available at: https://www.vicarious.com/2017/10/26/common-sense-cortex-and-captcha/

[5] TensorFlow (2017) TensorFlow Object Detection API
[online] Available at: https://github.com/tensorflow/models/tree/master/research/object$_d$*etection*