 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 1 of 30 Date: October 27 2018
--	---	--


# WildTracker

## Object Detection Implementation Test Report

<b>Project:</b> WT18G4 <b>WP Name:</b> OD Subsystem Test <b>WP Number:</b> WT18G4-OD-TR-01	<b>Type of Test:</b> Unit Tests
<b>Test Article:</b> The Object Detection (OD) subsystem	
<b>System Requirements:</b>	<b>Test Equipment:</b> Refer to Section 4
<b>Test Operators:</b> Christian Neilsen	<b>Test Engineers:</b> Christian Neilsen
<b>Project Manager:</b> Delenn Palmer	<b>Project Supervisor:</b> Dr Felipe Gonzalez

Queensland University of Technology  
Gardens Point Campus  
Brisbane, Australia, 4001.

This document is Copyright 2018 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT


 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 2 of 30 Date: October 27 2018
--	---	--

### Revision Record

<b>Document Issue/Revision Status</b>	<b>Description of Change</b>	<b>Date</b>	<b>Approved</b>
1.0	Final Issue	24/10/2018	Delenn Palmer


## Table of Contents

Paragraph	Page No.
1 Introduction .....	5
1.1 Scope .....	5
1.2 Background .....	5
2 Reference Documents.....	6
2.1 QUT Systems Engineering Documents.....	6
2.2 Non-QUT Documents .....	6
2.3 Numbering Scheme .....	6
3 Test Objectives .....	7
4 Test Set-up and Equipment .....	8
4.1 Required Equipment and Software.....	8
4.1.1 Test Set-up of Laptop .....	<b>Error! Bookmark not defined.</b>
4.2 Graphical User Interface Software Implementations ....	<b>Error! Bookmark not defined.</b>
5 Procedure.....	12
5.1 OpenCV or Python – GUI Implementation.....	<b>Error! Bookmark not defined.</b>
6 Results .....	14
6.1 Graphical Evidence of Test Completion .....	<b>Error! Bookmark not defined.</b>
7 Analysis .....	<b>Error! Bookmark not defined.</b>
8 Conclusions and Recommendations.....	<b>Error! Bookmark not defined.</b>
9 Appendices .....	<b>Error! Bookmark not defined.</b>

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 4 of 30 Date: October 27 2018
--	---	--

## Definitions

WT	WildTracker - Software implementation of an autonomous detection and tracking tool for wildlife
QUT	Queensland University of Technology
HLO	High Level Objectives
PMP	Project Management Plan
GUI	Graphical User Interface
TF	TensorFlow
CNN	Convolutional Neural Network
OpenCV	Open Source Computer Vision
OD	Objective Detection
DL	Deep Learning
ML	Machine Learning

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 5 of 30 Date: October 27 2018
--	---	--

## 1 Introduction

A key step in the Systems Engineering process is the testing phase. The testing phase allows for the individual units that make up subsystems to be robustly tested, and through doing this, provides a good indicator of the suitability and correctness of each component that makes up the total subsystem. By verifying the individual subsystem components, the integration stage of the project is also made much easier to complete successfully, by integrating the previously tested components into functioning subsystems, and eventually complete systems.

### 1.1 Scope


This test report covers the complete testing process that was applied to the Object Detection (OD) subsystem. The sections included in this test report are:

- The testing objectives (and relevant system requirements)
- The test set-up and equipment.
- The testing procedure.
- The results of completed tests.
- Analysis of the testing results.

The key goals of this testing are: to verify the suitability OD subsystems architecture; to match successful testing results to the relevant system requirements; to provide a clear and repeatable testing environment along with a range of robust testing procedure; and finally, the examine the results of the testing procedure and to comment on the overall viability of the subsystem through this analysis.

### 1.2 Background

QUT ASL is a world leading research centre based in Brisbane, Australia. They conduct research into autonomous technologies which support the development of autonomous aircraft or drones for remote sensing with on-board sensor systems for a wide range of commercial applications. The QUT Airborne Systems Lab (ASL) has commissioned students of EGH455 in collaboration with WWF and Wildlife Australia to design and build an autonomous detection and tracking tool for wildlife. Group 4 has been tasked with designing an Unmanned Aerial System (UAS) application that must have the ability to identify and report the number of, the size of and volume of wildlife present in footage retrieved by drones. In addition, the data acquired and processed must be accessible both in the real-time use of the application and after the video has been exported.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 6 of 30 Date: October 27 2018
--	---	--

## 2 Reference Documents

### 2.1 QUT Systems Engineering Documents

RD/1	WT18G4-SUP-Customer Needs	Autonomous detection and tracking tool for wildlife
RD/2	WT18G4-SR-01	WildTracker Project: System Requirements Document 2018

### 2.2 Non-QUT Documents

RD/DF	DarkFlow Repository	<a href="https://github.com/thtrieu/darkflow">https://github.com/thtrieu/darkflow</a>
RD/AN	Anaconda Downloads	<a href="https://www.anaconda.com/download/">https://www.anaconda.com/download/</a>
RD/YO	YOLOv2 .weight and .cfg files	<a href="https://pjreddie.com/darknet/yolo/">https://pjreddie.com/darknet/yolo/</a>


### 2.3 Numbering Scheme

For ease of identification, a numbering system has been developed.

For the requirement REQ-M-01:

REQ – This is a requirement derived from the client's brief and the associated HLOs

M – Denotes a *mandatory* requirement, whereas D denotes the *desired* requirement.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 7 of 30 Date: October 27 2018
--	---	--

### 3 Test Objectives

To ensure that the OD subsystem's components are all fully functional and that the subsystem fulfils all of its necessary system requirements, robust testing must be completed for the individual sections of the subsystem. The main components of the OD subsystem that were tested are:

- The DarkFlow Neural Network (NN) model.
- The OpenCV code used to process videos alongside DarkFlow.

The following system requirements were used to evaluate whether tested components partially or wholly satisfy the client's needs:

#### **REQ-M-01**

*Description:* The WildTracker system will be capable of detecting and tracking 2 types of wildlife present.

*Rationale:* The final system will be required to track several animals (Elephants and Horses for the initial revision) at the same time. This shows the systems ability to differentiate between several animal species' and to make accurate detections.

#### **REQ-M-02**

*Description:* The WildTracker system will be capable of processing a top down view and/or oblique angle footage.

*Rationale:* The video footage that will be input into the system will be taken from drone recordings, and as such it is likely that this footage will be taken from elevation and either a top-down or oblique angle.

#### **REQ-M-03**


*Description:* The WildTracker system shall be capable of achieving 95% accuracy.

*Rationale:* The system needs to be able to accurately make detections, minimizing false negatives and false positives that may skew data collection. This is also important for accurately estimating animals sizes as it relates to the bounding box coordinates generated by the DarkFlow model.

#### **REQ-M-04**

*Description:* The WildTracker system shall have less than 8% error in animal counts.

*Rationale:* Like with REQ-M-03, accurate detections should be made and the system should be suitable for the use in automating animal counts (for wildlife preservation, research etc.)

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 8 of 30 Date: October 27 2018
--	---	--

## 4 Test Set-up and Equipment

The following section outlines the hardware and software requirements of the testing environment.

### 4.1 Required Equipment and Software

#### 4.1.1 Hardware Requirements

In order to efficiently test the system, a workstation computer using a dedicated graphics card was employed. The following computer specifications were used:

- OS: Windows 10 Home (primary) and Ubuntu 16.0.4 LTS (using VirtualBox software)
- Storage: 500 GB hard drive
- GPU: GTX 1070 (8GB VRAM)
- CPU: Intel Core i7-4790K @ 4.00 GHZ (4 core)

Due to the data and GPU intensive nature of ML, I definitely recommend have upwards of 100GB storage free for storing the DarkFlow model weights, and also using a GPU equivalent or better than an NVIDIA GTX 1050 (or the Linux equivalent).



Figure 1 - NVIDIA GTX 1070 GPU

### 4.1.2 Software Requirements


#### 4.1.2.1 DarkFlow

DarkFlow was the primary software used in order to train the model used in the OD subsystem. DarkFlow provides an easy-to-install Python wrapper for the Darknet NN framework (which is written in the low level C language) and features a number of command line tools that sped up the training and testing process. The YOLOv2 architecture was used with DarkFlow to train the model.

#### 4.1.3 DarkFlow Prerequisites (CUDA, CUDNN)

In order to make use of DarkFlow's ML training, the NVIDIA CUDA Toolkit (v 9.0) and cuDNN (CUDA Deep Neural Network library) (v 7.0) had to be installed so that the testing machines GPU could be fully utilized by the model.



 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 9 of 30 Date: October 27 2018
--	---	--

#### 4.1.3.1 Anaconda

Anaconda is an open-source data science platform, which is widely used in industry for the development, testing and training of ML models. Anaconda was used as the primary method for managing the testing environment and its required software packages and dependencies.

#### 4.1.3.2 OpenCV

OpenCV is a critical component in the OD subsystem, and it allowed us to perform many necessary video processing functions. These include: loading and reading/writing to video files; displaying rectangular bounding boxes over objects detected by the DarkFlow model; displaying the relevant object data (label, confidence values) on top of the bounding box; performing any required resolution/colour normalization with video files.

#### 4.1.4 Test Environment Setup

##### 4.1.4.1 DarkFlow Setup

In order to setup the DarkFlow model so that training and testing could begin, the DarkFlow GitHub repository (refer to RD/DF) was cloned to the testing machine, and using Anaconda Prompt with a Python 3.6 and navigating to the repository folder, the following command was used to install DarkFlow locally into this folder:

```
python3 setup.py build_ext -inplace
```

After the initial installation was complete, two new folders were created in the main DarkFlow directory, with the names 'bin' and 'cfg'. The YOLOv2 weight and configuration files were downloaded (refer to RD/YO) and stored in these directories respectively.

##### 4.1.4.2 Anaconda Environment Setup


A new Anaconda environment was created for the purpose of testing the system with only the necessary dependencies. This was also done so that this test environment could be exported to a different machine by simply packaging the model into a .yaml configuration file and using this to install the environment to a new machine. The test environment was created with the least amount of dependencies needed, with the aim of making the final system as portable as possible. The following command was used to create the environment:

```
conda create -name odtest python=3.6 pip cython numpy tensorflow-gpu opencv
```

After creating the environment, the environment was activated, and any other necessary dependencies were installed (while using the activated environment) using the following commands:

```
conda activate ObjectDetection
```

```
conda install NEW_PACKAGE_NAME
```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 10 of 30 Date: October 27 2018
--	---	---

#### 4.1.4.3 OpenCV Setup

Several Python files were created in order to speed up the process of testing the OD subsystem. Two separate Python scripts (*process\_vid.py* and *export\_vid.py*) were created, with each file making use of OpenCV to read raw video files in and to show the bounding box, label and confidence values of each object detecting using the DarkFlow model. The key difference between the two scripts were:

- Process\_vid simply streams the final video file containing all detections while they are being made.
- Export\_vid performs object detection on the raw video and exports the final video to a new file specified in the code, with the tqdm module being used to display the export progress.

If a slower graphics card was being used then it is recommended that the export\_vid script be used to test out the model on video files, as the process\_vid file is heavily reliant on the power of the chosen graphics card.

#### 4.1.5 YOLOv2 Model Configuration

In order to use the pre-trained YOLOv2 model, slight changes had to be made to the 'labels.txt' file (contained within the root DarkFlow directory) and also to the selected YOLOv2 .cfg file 'yolov2.cfg' which was contained within the cfg folder (see Section 4.1.4.1).

For the labels file, the inner content was simply deleted and replaced with the classes that were to be detected. For our test cases, the label files inner content was simple changed to 'elephant horse' with a line break in between each class, as shown in the figure below:

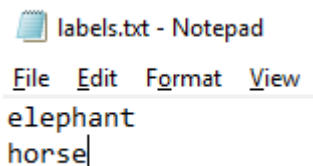


Figure 2 - labels.txt File Change

In terms of setting up the .cfg file to work for our particular model, a copy of the 'yolov2.cfg' file was made, and this file was renamed 'yolov2-2c.cfg'. Inside of this new file, two alterations were made at the end of the file. The first change made was to set the value of the 'classes' variable in the region section to 2 (representing elephants and horses), and the second change was to alter to value of 'filters' in the last convolutional layer to be equal to:  $5 * (5 + \text{num\_classes}) = 5 * 7 = 35$ .

The results of these alterations can be seen in the figure below:

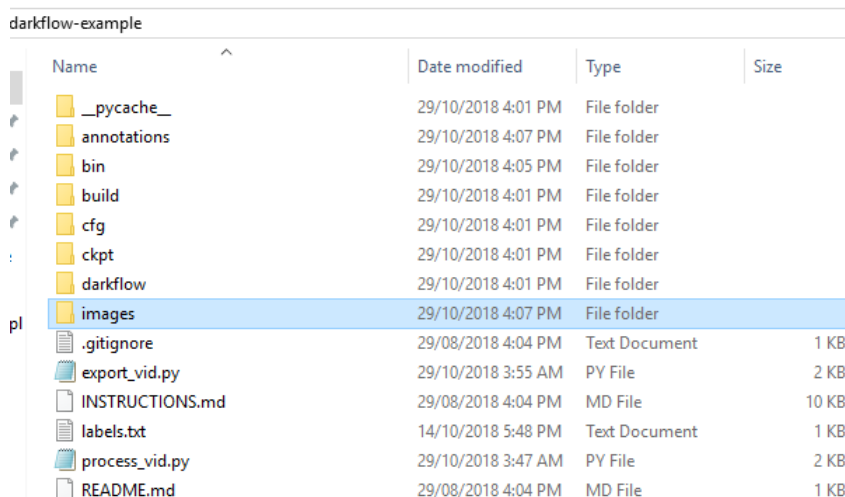
```
[convolutional]
size=1
stride=1
pad=1
filters=35
activation=linear

[region]
anchors = 0.57273, 0.677385, 1.87446, 2.06253, 3.33843, 5.47434, 7.88282, 3.52778, 9.77052, 9.16828
bias_match=1
classes=2
coords=4
num=5
softmax=1
jitter=.3
rescore=1
```

Figure 3 - Alterations to yolov2.cfg

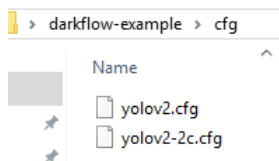
#### 4.1.6 Final Directory Structure

The following figures show an example of how the testing directory looked after DarkFlow was correctly installed and all relevant cfg and weight files were copied/configured:



Name	Date modified	Type	Size
__pycache__	29/10/2018 4:01 PM	File folder	
annotations	29/10/2018 4:07 PM	File folder	
bin	29/10/2018 4:05 PM	File folder	
build	29/10/2018 4:01 PM	File folder	
cfg	29/10/2018 4:01 PM	File folder	
ckpt	29/10/2018 4:01 PM	File folder	
darkflow	29/10/2018 4:01 PM	File folder	
images	29/10/2018 4:07 PM	File folder	
.gitignore	29/08/2018 4:04 PM	Text Document	1 KB
export_vid.py	29/10/2018 3:55 AM	PY File	2 KB
INSTRUCTIONS.md	29/08/2018 4:04 PM	MD File	10 KB
labels.txt	14/10/2018 5:48 PM	Text Document	1 KB
process_vid.py	29/10/2018 3:47 AM	PY File	2 KB
README.md	29/08/2018 4:04 PM	MD File	1 KB

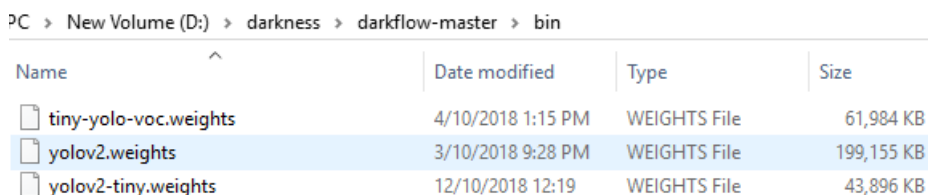
Figure 4 - Example test directory for DarkFlow



Name
yolov2.cfg
yolov2-2c.cfg

Figure 5 - Example cfg Folder Containing Model Configuration Files

One note regarding the cfg folder shown above is that it is important to keep the original .cfg file for a custom model, as this file is referenced by DarkFlow when performing training.




Name	Date modified	Type	Size
tiny-yolo-voc.weights	4/10/2018 1:15 PM	WEIGHTS File	61,984 KB
yolov2.weights	3/10/2018 9:28 PM	WEIGHTS File	199,155 KB
yolov2-tiny.weights	12/10/2018 12:19	WEIGHTS File	43,896 KB

Figure 6 - Example bin Folder Containing Model Weights

#### 4.1.7 Dataset Information

A dataset consisting of 1217 fully annotated images was used to train the DarkFlow model used for testing. The majority of annotated images contained multiple (often > 5) instances of an animal for a test class, with the majority of images being sampled from drone footage (both top-down and oblique angles) and a smaller portion of training images consisting of ground level shots of the animal with and without other animals. In order to prevent false detections, we ensured that images for both test classes contained a variety of backgrounds and other objects so that a more accurate model could be trained.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 12 of 30 Date: October 27 2018
--	---	---

## 5 Test Procedure

The following section will detail the test setup and the procedures that will be used to test the various components of the OD subsystem.

### 5.1 Python / Anaconda Install Test

The Anaconda program was installed, and an environment named ‘od\_test’ was created using the code shown in Section 4.1.4.2. After the successful creation of the environment, the ‘conda activate’ command will be used to start using the environment, and then the ‘conda list’ command will be used to verify that all requested modules are installed correctly.

### 5.2 TensorFlow Install Test

After successfully installing the TensorFlow GPU module using Anaconda, the following code was used within the Anaconda terminal to run a basic ‘hello world’ application to verify that TensorFlow was downloaded and worked within the system:

```
python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

### 5.3 DarkFlow

#### 5.3.1 DarkFlow Install Test


After installing DarkFlow using the instructions shown in Section 4.1.4.1, the YOLOv2 weight and configuration files were downloaded and stored in the bin and cfg folders respectively (refer to RD/YO for download links). Several test commands provided in the DarkFlow repository (refer to RD/DF) were used to test whether the DarkFlow model was functioning correctly. The following command was run to test the basic ‘flow’ function and to display the model layers:

```
python flow --model cfg/yolov2-2c.cfg --load bin/yolov2.weights --gpu 0.8
```

#### 5.3.2 DarkFlow Start Training New Model Test

After verifying that the DarkFlow program was correctly installed, the ‘train’ flag was used with the flow function to begin training a custom model using previously annotated images. For this test two folders were created in the base DarkFlow directory (named ‘images’ and ‘annotations’) and the previously annotated images were stored in a folder and the annotation files in the other (.xml files annotated using the labeling program). The following command was run to test training the DarkFlow model:

```
python flow --model cfg/yolov2-2c.cfg --load bin/yolov2.weights --gpu 0.8
--train --dataset images --annotation annotations --trainer adam --batch 8
```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 13 of 30 Date: October 27 2018
--	---	---

### 5.3.3 DarkFlow Resume Training Model Test

Due to the long periods of time needed to train a model, it was important to ensure that training could resume on a partially trained model by loading a previously stored training checkpoint (using -1 for the last value or using a set value to reference a file). The following example code shows how training was resumed using a checkpoint at 1000 iterations for the yolov2-2c model:

```
python flow --model cfg/yolov2-2c.cfg --load 1000 --gpu 0.8
--train --dataset images --annotation annotations --trainer adam --batch 8
```

## 5.4 OpenCV

After the DarkFlow model was trained to an adequate level, its prediction capabilities were tested using OpenCV to visually display the object detection results. The three distinct testing methods used were:

- Performing predictions on single image files.
- Streaming predicted output using the ‘process\_vid.py’ file.
- Exporting the final predicted video file using the ‘export\_vid.py’ file.

The code used for these three methods can be seen in Appendix A.

All testing results and an analysis of the overall model accuracy can be seen in Section 6 and 7 of this report.

## 6 Results

This section documents the results from the tests described in Section 5.

### 6.1 Python / Anaconda Installation - SUCCESS

Following the steps provided in Section 5.1, the Anaconda program was installed and a test environment named 'odtest' was created using Python v 3.6. In addition to this base install, the following python modules were also installed: pip, cython, numpy, tensorflow-gpu and opencv.

```

Anaconda Prompt - conda create --name odtest python=3.6 pip cython numpy tensorflow-gpu
(base) C:\Users\MLORD>conda create --name odtest python=3.6 pip cython numpy tensorflow-gpu
WARNING: A space was detected in your requested environment path
'D:\Program Files (x86)\Miniconda3\envs\odtest'
Spaces in paths can sometimes be problematic.
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 4.5.4
latest version: 4.5.11

Please update conda by running

$ conda update -n base conda

## Package Plan ##

environment location: D:\Program Files (x86)\Miniconda3\envs\odtest

added / updated specs:
- cython
- numpy
- pip
- python=3.6
- tensorflow-gpu

The following packages will be downloaded:

package | build | size
-----|-----|-----
tensorflow-base-1.11.0 | py36he025d50_0 | 3.1 MB
numpy-base-1.15.3 | py36h8128ebf_0 | 3.8 MB
tensorflow-select-2.1.0 | gpu | 3 KB
certifi-2018.10.15 | py36_0 | 138 KB
keras-preprocessing-1.0.5 | py36_0 | 52 KB
markdown-3.0.1 | py36_0 | 125 KB
wheel-0.32.2 | py36_0 | 52 KB
numpy-1.15.3 | py36ha559c80_0 | 36 KB
python-3.6.7 | h33f27b4_0 | 21.1 MB
keras-applications-1.0.6 | py36_0 | 49 KB
tensorflow-1.11.0 | gpu_py36h5dc63e2_0 | 5 KB
cython-0.29 | py36ha925a31_0 | 2.0 MB
tensorflow-gpu-1.11.0 | h0d30ee6_0 | 3 KB
tensorflow-base-1.11.0 | gpu_py36h6e53903_0 | 175.1 MB
-----|-----|-----
Total: | | 205.7 MB

```

Figure 7 - Anaconda / Python Installation 1

The above figure shows the environment creation command being entered, which resulted in the following two figures showing all packaged which were to be installed and finally showing that the environment creation was successful:

```

The following NEW packages will be INSTALLED:

_tflow_select:      2.1.0-gpu
absl-py:            0.5.0-py36_0
astor:              0.7.1-py36_0
blas:               1.0-mkl
certifi:            2018.10.15-py36_0
cudatoolkit:        9.0-1
cudnn:              7.1.4-cuda9.0_0
cython:             0.29-py36ha925a31_0
gast:               0.2.0-py36_0
grpcio:             1.12.1-py36h1a1b453_0
h5py:               2.8.0-py36h3bdd7fb_2
hdf5:               1.10.2-hac2f561_1
icc_rt:             2017.0.4-h97af966_0
intel-openmp:       2019.0-118
keras-applications: 1.0.6-py36_0
keras-preprocessing: 1.0.5-py36_0
libprotobuf:        3.6.0-h1a1b453_0
markdown:           3.0.1-py36_0
mkl:                2019.0-118
mkl_fft:            1.0.6-py36hdbbee80_0
mkl_random:         1.0.1-py36h77b88f5_1
numpy:              1.15.3-py36ha559c80_0
numpy-base:         1.15.3-py36h8128ebf_0
pip:                10.0.1-py36_0
protobuf:           3.6.0-py36he025d50_0
python:             3.6.7-h33f27b4_0
scipy:              1.1.0-py36h4f6bf74_1
setuptools:         40.4.3-py36_0
six:                1.11.0-py36_1
tensorboard:        1.11.0-py36he025d50_0
tensorflow:          1.11.0-gpu_py36h5dc63e2_0
tensorflow-base:     1.11.0-gpu_py36h6e53903_0
tensorflow-gpu:      1.11.0-h0d30ee6_0
termcolor:          1.1.0-py36_1
vc:                 14.1-h0510ff6_4
vs2015_runtime:     14.15.26706-h3a45250_0
werkzeug:           0.14.1-py36_0
wheel:              0.32.2-py36_0
winertstore:        0.2-py36h7fe50ca_0
zlib:               1.2.11-h8395fce_2

Proceed ([y]/n)? y

```

Figure 8 - Anaconda / Python Installation 2

```

Downloading and Extracting Packages
tensorflow-1.11.0 | 3.1 MB | ##### | 100%
numpy-base-1.15.3 | 3.8 MB | ##### | 100%
_tflow_select-2.1.0 | 3 KB | ##### | 100%
certifi-2018.10.15 | 138 KB | ##### | 100%
keras-preprocessing- | 52 KB | ##### | 100%
markdown-3.0.1 | 125 KB | ##### | 100%
wheel-0.32.2 | 52 KB | ##### | 100%
numpy-1.15.3 | 36 KB | ##### | 100%
python-3.6.7 | 21.1 MB | ##### | 100%
keras-applications-1 | 49 KB | ##### | 100%
tensorflow-1.11.0 | 5 KB | ##### | 100%
cython-0.29 | 2.0 MB | ##### | 100%
tensorflow-gpu-1.11. | 3 KB | ##### | 100%
tensorflow-base-1.11 | 175.1 MB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate odtest
#
# To deactivate an active environment, use
#
#     $ conda deactivate
#

(base) C:\Users\MLORD>activate odtest

(odtest) C:\Users\MLORD>

```

Figure 9 - Anaconda / Python Installation 3



Following this installation, a simple test was done to ensure that the correct version of Python was installed and working, by entering 'python' into the Anaconda prompt window (after activating the environment):

```
(odtest) C:\Users\MLORD>python
Python 3.6.7 [Anaconda, Inc.] (default, Oct 24 2018, 09:45:24) [MSC v.1912 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 10 - Python Test

The following output resulted from entering the 'conda list' command, showing all installed packages:

```
(odtest) C:\Users\MLORD>conda list
# packages in environment at D:\Program Files (x86)\Miniconda3\envs\odtest:
#
# Name                      Version                      Build      Channel
tensorflow-select           2.1.0                        gpu        gpu
absl-py                     0.5.0                        py36_0     py36_0
astor                       0.7.1                        py36_0     py36_0
blas                        1.0                          mkl         mkl
certifi                     2018.10.15                   py36_0     py36_0
cudatoolkit                 9.0                          1           1
cudnn                       7.1.4                        cuda9.0_0  cuda9.0_0
cython                      0.29                         py36ha925a31_0
gast                        0.2.0                        py36_0     py36_0
grpcio                      1.12.1                       py36h1a1b453_0
h5py                       2.8.0                        py36hf7173ca_2
hdf5                        1.8.20                       hac2f561_1  hac2f561_1
icc_rt                      2017.0.4                     h97af966_0  h97af966_0
intel-openmp                2019.0                       118         118
jpeg                        9b                           hb83a4c4_2  hb83a4c4_2
keras-applications          1.0.6                        py36_0     py36_0
keras-preprocessing         1.0.5                        py36_0     py36_0
libopencv                   3.4.2                        h20b85fd_0  h20b85fd_0
libpng                      1.6.35                       h2a8f88b_0  h2a8f88b_0
libprotobuf                 3.6.0                        h1a1b453_0  h1a1b453_0
libtiff                     4.0.9                        h36446d0_2  h36446d0_2
markdown                    3.0.1                        py36_0     py36_0
mkl                         2019.0                       118         118
mkl_fft                     1.0.6                        py36hdbbbee80_0
mkl_random                  1.0.1                        py36h77b88f5_1
numpy                       1.15.3                       py36ha559c80_0
numpy-base                  1.15.3                       py36h8128ebf_0
opencv                      3.4.2                        py36h40b0b35_0
pip                         10.0.1                       py36_0     py36_0
protobuf                    3.6.0                        py36he025d50_0
py-opencv                   3.4.2                        py36hc319ecb_0
python                      3.6.7                        h33f27b4_0  h33f27b4_0
scipy                       1.1.0                        py36h4f6bf74_1
setuptools                  40.4.3                       py36_0     py36_0
six                          1.11.0                       py36_1     py36_1
tensorboard                 1.11.0                       py36he025d50_0
tensorflow                   1.11.0                       gpu_py36h5dc63e2_0
tensorflow-base             1.11.0                       gpu_py36h6e53903_0
tensorflow-gpu              1.11.0                       h0d30ee6_0  h0d30ee6_0
termcolor                   1.1.0                        py36_1     py36_1
vc                           14.1                         h0510ff6_4  h0510ff6_4
vs2015_runtime              14.15.26706                  h3a45250_0  h3a45250_0
werkzeug                    0.14.1                       py36_0     py36_0
wheel                       0.32.2                       py36_0     py36_0
winertstore                 0.2                          py36h7fe50ca_0
zlib                        1.2.11                       h8395fce_2  h8395fce_2
```

Figure 11 - Anaconda Environment List



## 6.2 TensorFlow Installation - SUCCESS

The following results confirmed the successful installation of TensorFlow by running a simple ‘hello world’ program using TF:

```
(odtest) C:\Users\MLORD>python
Python 3.6.7 [Anaconda, Inc.] (default, 0
Type "help", "copyright", "credits" or "
>>> import tensorflow as tf
>>> hello = tf.constant('Hello, TF!')
>>> sess = tf.Session()
2018-10-29 02:29:18.306878: I tensorflow
to use: AVX AVX2
2018-10-29 02:29:18.883000: I tensorflow
name: GeForce GTX 1070 major: 6 minor: 1
pciBusID: 0000:01:00.0
totalMemory: 8.00GiB freeMemory: 6.63GiB
2018-10-29 02:29:18.889782: I tensorflow
2018-10-29 02:29:19.350672: I tensorflow
2018-10-29 02:29:19.353560: I tensorflow
2018-10-29 02:29:19.355391: I tensorflow
2018-10-29 02:29:19.356998: I tensorflow
with 6384 MB memory) -> physical GPU (de
>>> print(sess.run(hello))
b'Hello, TF!'
>>>
```

Figure 12 - TensorFlow Test

## 6.3 DarkFlow Test Results

### 6.3.1 DarkFlow Install Test - SUCCESS


The following figure shows the testing output after running a simple ‘flow’ command using the customized YOLOv2 configuration file:

```
(odtest) D:\darkness\darkflow-master>python flow --model cfg/yolov2-2c.cfg --load bin/yolov2.weights --gpu 0.8

Parsing ./cfg/yolov2.cfg
Parsing cfg/yolov2-2c.cfg
Loading bin/yolov2.weights ...
Successfully identified 203934260 bytes
Finished in 0.05605006217956543s

Building net ...
Source | Train? | Layer description | Output size
-----|-----|-----|-----
Load | Yes! | input | (?, 608, 608, 3)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 608, 608, 32)
Load | Yes! | maxp 2x2p0_2 | (?, 304, 304, 32)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 304, 304, 64)
Load | Yes! | maxp 2x2p0_2 | (?, 152, 152, 64)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 152, 152, 128)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 152, 152, 64)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 152, 152, 128)
Load | Yes! | maxp 2x2p0_2 | (?, 76, 76, 128)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 76, 76, 256)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 76, 76, 128)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 76, 76, 256)
Load | Yes! | maxp 2x2p0_2 | (?, 38, 38, 256)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 38, 38, 512)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 38, 38, 256)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 38, 38, 512)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 38, 38, 256)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 38, 38, 512)
Load | Yes! | maxp 2x2p0_2 | (?, 19, 19, 512)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 19, 19, 1024)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 19, 19, 512)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 19, 19, 1024)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 19, 19, 512)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 19, 19, 1024)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 19, 19, 1024)
Load | Yes! | concat [16] | (?, 38, 38, 512)
Load | Yes! | conv 1x1p0_1 +bnorm leaky | (?, 38, 38, 64)
Load | Yes! | local flatten 2x2 | (?, 19, 19, 256)
Load | Yes! | concat [27, 24] | (?, 19, 19, 1280)
Load | Yes! | conv 3x3p1_1 +bnorm leaky | (?, 19, 19, 1024)
Init | Yes! | conv 1x1p0_1 linear | (?, 19, 19, 35)
```

Figure 13 - DarkFlow Install Test

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 18 of 30 Date: October 27 2018
--	---	---

### 6.3.2 DarkFlow Training New Model Test and Resume Training Tests - SUCCESS

The following figures shows the test output when beginning training on a new DarkFlow model. The custom YOLOv2 configuration file was used with the standard YOLOv2 weights. Other training parameters used were: a batch size of 8 (constrained by GPU memory), the adam training model. The second figure shows the successful continuation of training using a checkpoint value of 94500.

```

Enter training ...

cfg/yolov2-2c.cfg parsing final_annots
Parsing for ['elephant', 'horse']
[=====>]100% 957.xml
Statistics:
elephant: 3437
horse: 2643
Dataset size: 1220
Dataset of 1220 instance(s)
090.jpg
1246.jpg
1410.jpg
1320.jpg
618.jpg
1585.jpg
512.jpg
365.jpg
Training statistics:
  Learning rate : 1e-05
  Batch size    : 8
  Epoch number  : 1000
  Backup every  : 2000
2018-10-29 02:51:25.221038: W tensorflow/core/common_runtime/bfc_
lller indicates that this is not a failure, but may mean that ther
2018-10-29 02:51:25.256691: W tensorflow/core/common_runtime/bfc_
lller indicates that this is not a failure, but may mean that ther
2018-10-29 02:51:25.745118: W tensorflow/core/common_runtime/bfc_
lller indicates that this is not a failure, but may mean that ther
2018-10-29 02:51:25.755965: W tensorflow/core/common_runtime/bfc_
lller indicates that this is not a failure, but may mean that ther
2018-10-29 02:51:25.801579: W tensorflow/core/common_runtime/bfc_
lller indicates that this is not a failure, but may mean that ther
step 1 - loss 243.236572265625 - moving ave loss 243.236572265625

```

Figure 14 - DarkFlow Training New Model Test

```


Loading from ./ckpt/yolov2-2c-94500
Finished in 30.50046992301941s

Enter training ...

cfg/yolov2-2c.cfg parsing final_annots
Parsing for ['elephant', 'horse']
[=====>]100% 957.xml
Statistics:
elephant: 3437
horse: 2643
Dataset size: 1220
Dataset of 1220 instance(s)
013.jpg
915.jpg
1609.jpg
502.jpg
699.jpg
183.jpg
462.jpg
348.jpg
Training statistics:
  Learning rate : 1e-05
  Batch size    : 8
  Epoch number  : 1000
  Backup every  : 2000
2018-10-29 02:53:39.681296: W tensorflow/core/common_runtime/bfc_alloca
lller indicates that this is not a failure, but may mean that there coul
2018-10-29 02:53:39.721198: W tensorflow/core/common_runtime/bfc_alloca
lller indicates that this is not a failure, but may mean that there coul
2018-10-29 02:53:40.211597: W tensorflow/core/common_runtime/bfc_alloca
lller indicates that this is not a failure, but may mean that there coul
2018-10-29 02:53:40.230137: W tensorflow/core/common_runtime/bfc_alloca
lller indicates that this is not a failure, but may mean that there coul
2018-10-29 02:53:40.284725: W tensorflow/core/common_runtime/bfc_alloca
lller indicates that this is not a failure, but may mean that there coul
step 94501 - loss 2.481337785720825 - moving ave loss 2.481337785720825

```

Figure 15 - DarkFlow Continue Training Model Test

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 19 of 30 Date: October 27 2018
--	---	---

## 6.4 OpenCV Test Results

### 6.4.1 OpenCV Single Image Detection - SUCCESS

The code shown in Appendix A1 was run through a jupyter notebook file using a test image of an elephant. The figure shown below shows the output of this test:

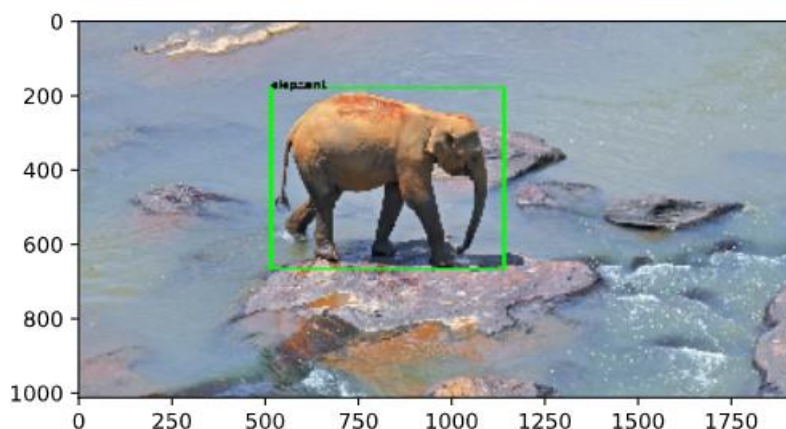


Figure 16 - Single Image Detection Test Result

### 6.4.2 OpenCV Video Streaming and Exporting Tests

Using the process\_vid.py and export\_vid.py functions shown in Appendix A2 and A3, the functions were called from Anaconda prompt and both methods successfully performed image detection, with an example being shown in the figure below:




Figure 17 - OpenCV Object Detection Example

The following figure shows the output of a successfully exported video in the Anaconda prompt window:

```
OpenCV: FFMPEG: tag 0x44495658/'XVID' is not supported with codec id 13 and format 'mp4 / MP4 (MPEG-4 Part 14)'
OpenCV: FFMPEG: fallback to use tag 0x7634706d/'mp4v'
100% | 771/771 [00:44<00:00, 18.20it/s]
```

Figure 18 - Example of TQDM progress bar

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 20 of 30 Date: October 27 2018
--	---	---

## 6.5 Overall Object Detection Accuracy Tests

After verifying that all of the necessary system components worked correctly, several test videos containing Elephants and Horses were used with the export\_vid.py file and the final trained model's weights. The final weight file that was used for system testing and validation, was taken after 132500 steps, with the model having an average moving loss value of less than 2.

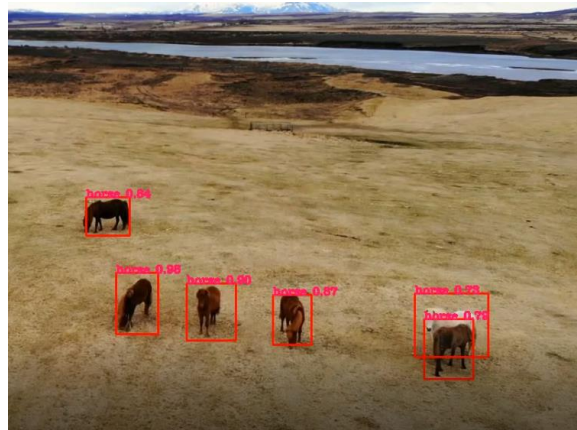
### 6.5.1 Definition of Accuracy in Systems Context

In the context of testing the systems overall accuracy, a detection was deemed accurate when the Intersection over Union (IoU) value was below above 50%. Due to the scale of and time restraints of this project, the majority of testing IoU values were taken visually instead of through calculations.

### 6.5.2 Object Detection Model Accuracy


When performing object detection on image frames that closely fit the format of the training dataset, the DarkFlow model was able to make accurate predictions with accuracy values upwards of the 95% requirement set by the Client (REQ-M-03) and with less than 8% error in animal count (REQ-M-04).

A testing threshold value of 0.5 was used for most tests, as this value enabled the model to accurately predict objects with the least amount of false positives (extra false bounding boxes). By having a reasonably large and equally distributed dataset of both classes (Elephants and Horses) the model was in most cases able to make correctly predict elephant images as being elephants, and vice versa for horses. The following figures show screenshots of exported videos with detections made on both test classes:



As can be seen in the above figures, the model was able to make accurate detections on Horses and Elephants in both top-down and oblique video footage, thereby meeting the Clients functional requirements (REQ-M-01, REQ-M-02). More examples of test screenshots are shown in Appendix B.



 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 21 of 30 Date: October 27 2018
--	---	---


### 6.5.3 Detection Errors and Model Issues

Throughout the process of testing the finished subsystem, a number of small errors and test cases which did not meet the client requirements became apparent. The following list highlights these cases:

- Higher than desired error in counts for overlapping animals.
- Lower than desired accuracy when testing on videos that differ substantially from training datasets.
- Errors when trying to detect grey coloured horses from top-down angles (detected as elephants).
- Issues with false detections on certain objects (some rocks, bodies of water, other untrained animals detected as elephants).

For most of the issues and edge cases highlighted above, it was simply not viable to further train the model to the point of eliminating these issues. The primary reason for this was that this would require substantial amounts of data (thousands of images for each class) and exponentially more time for training and testing of the subsystem.

Despite this, the subsystem performed well given its context, and all of the clients functional and performance requirements were reasonably met in most cases. Examples of some detection errors throughout testing can be seen in Appendix C.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 22 of 30 Date: October 27 2018
--	---	---


## 7 Conclusions and Recommendations

After performing all test procedures (ref. Section 4) of this report and analysing the results (ref. Section 5, 6) the OD subsystem was deemed to meet all of the functional and performance requirements for the overall system (REQ-M 1,2,3,4).

Performance requirements of the system (REQ-M 3,4) were met wholly in most cases, with a few test cases resulting in the requirements being partially met. In all cases where requirements were only partially met this was the result of the test case being outside the scope of the project (ref. Section 6).

At the time of testing, the OD subsystem was deemed to be working at an acceptable level and was therefore ready for integration with the GUI subsystem. Recommendations that arose from this testing are:

- Increased dataset size (preferably over 5000 images for each class, introduction of more classes). This could also be achieved through the use of more advanced image augmentation techniques beyond the default set provided by the DarkFlow model.
- Longer training time in order to fine-tune the model and to play with batch size, learning rate and other optional parameters.
- Training specifically on cases that caused errors in detection.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 23 of 30 Date: October 27 2018
--	---	---

## Appendix A1 – Single Image Detection Code

```
import cv2

from darkflow.net.build import TFNet

import matplotlib.pyplot as plt

%config InlineBackend.figure_format = 'svg'

options = {

    'model': 'cfg/yolov2-2c.cfg',

    'load': 132500,

    'threshold': 0.3,

    'gpu': 0.8

}

tfnet = TFNet(options)

img = cv2.imread('testelephant.jpg', cv2.IMREAD_COLOR)

img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# use YOLO to predict the image

result = tfnet.return_predict(img)

img.shape

tl = (result[0]['topleft']['x'], result[0]['topleft']['y'])

br = (result[0]['bottomright']['x'], result[0]['bottomright']['y'])

label = result[0]['label']


# add the box and label and display it

img = cv2.rectangle(img, tl, br, (0, 255, 0), 7)

img = cv2.putText(img, label, tl, cv2.FONT_HERSHEY_COMPLEX, 1, (0, 0, 0), 2)

plt.imshow(img)

plt.show()
```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 24 of 30 Date: October 27 2018
--	---	---

## Appendix A2 – Streaming Video Code (process\_vid.py)

```
import cv2

from darkflow.net.build import TFNet

import numpy as np

import time

option = {
    'model': 'cfg/yolov2-2c.cfg',
    'load': 132500,
    'threshold': 0.5,
    'gpu': 0.8
}

tfnet = TFNet(option)

capture = cv2.VideoCapture('Elephant3.mp4')

counter = 0

while (capture.isOpened()):
    stime = time.time()
    ret, frame = capture.read()
    if ret:
        counter+=1

        results = tfnet.return_predict(frame)
        for color, result in zip(colors, results):
            tl = (result['topleft']['x'], result['topleft']['y'])
            br = (result['bottomright']['x'], result['bottomright']['y'])
            label = result['label']
            conf = result['confidence']

            numStr = ('There are {} elephants'.format(len(results)))

            frame = cv2.rectangle(frame, tl, br, (255,0,0), 2)

            frame = cv2.putText(frame, "{0} {1:.2f}".format(label, conf), tl,
cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 255), 1)
```





```
        frame = cv2.putText(frame, numStr, (0, 25),
cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 1)

        cv2.imshow('frame', frame)

        if (counter % 10 == 0):

            print('FPS {:.1f}'.format(1 / (time.time() - stime)))

            #print('There are {} elephants'.format(len(results)))

        if cv2.waitKey(1) & 0xFF == ord('q'):


            break

    else:

        capture.release()

        cv2.destroyAllWindows()

        break
```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-OD-TR -01 Issue: 1 Page: 26 of 30 Date: October 27 2018
--	---	---

### Appendix A3 – Export Video Code (export\_vid.py)

```
import cv2

from darkflow.net.build import TFNet

import numpy as np

import time

from tqdm import tqdm

option = {
    'model': 'cfg/yolov2-2c.cfg',
    'load': -1,
    'threshold': 0.5,
    'gpu': 0.8
}

tfnet = TFNet(option)

video_in = 'Elephant2.mp4'
video_out = 'Elephant2_exported.mp4'
video_reader = cv2.VideoCapture(video_in)

nb_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
frame_h = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
frame_w = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))

video_writer = cv2.VideoWriter(video_out, cv2.VideoWriter_fourcc(*'XVID'),
50.0, (frame_w, frame_h))

for i in tqdm(range(nb_frames)):
    ret, frame = video_reader.read()

    input_image = cv2.resize(frame, (416, 416))
    input_image = input_image / 255.
    input_image = input_image[:, :, :-1]
    input_image = np.expand_dims(input_image, 0)
```

```
results = tfnet.return_predict(frame)

for result in results:

    tl = (result['topleft']['x'], result['topleft']['y'])
    br = (result['bottomright']['x'], result['bottomright']['y'])
    label = result['label']
    conf = result['confidence']

    numStr = ('There are {} elephants'.format(len(results)))

    frame = cv2.rectangle(frame, tl, br, (255,0,0), 2)

    frame = cv2.putText(frame, "{0} {1:.2f}".format(label, conf), tl,
cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 255), 1)

    frame = cv2.putText(frame, numStr, (0, 25), cv2.FONT_HERSHEY_COMPLEX,
1, (255, 255, 255), 1)

video_writer.write(np.uint8(frame))

if cv2.waitKey(1) & 0xFF == ord('q'):
    break

video_reader.release()
video_writer.release()
```

## Appendix B – Successful Subsystem Test Screenshots

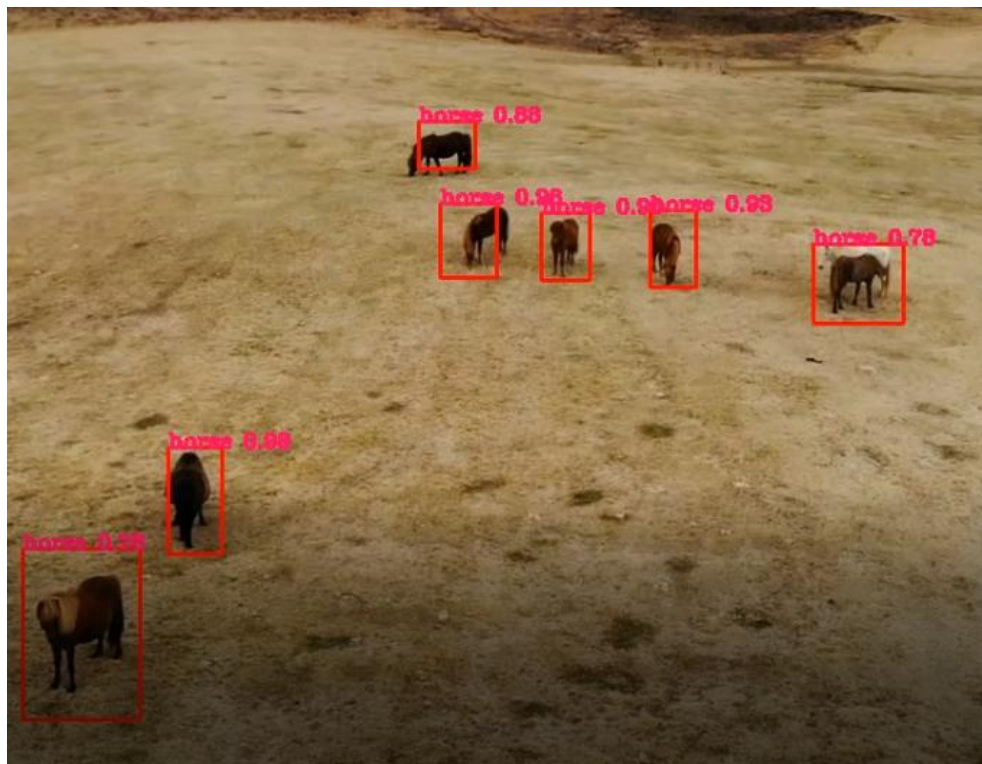


Figure 19 - Successful Detection of Horses (Oblique)



Figure 20 - Successful Detection of Elephants 1 (Top-down)





Figure 21 - Successful Detection of Elephants 2 (Top-down)



Figure 22 - Successful Detection of Elephants (Oblique)

Appendix C – Subsystem Detection Error Screenshots



Figure 23 - Error Case (Horse Detected as Elephant)



Figure 24 - Untrained Animal Detected as Elephant