4-2019

Field Obstacle Identification for Autonomous Tractor Applications

Caleb Lindhorst
University of Nebraska-Lincoln, clindhorst2@gmail.com

FIELD OBSTACLE IDENTIFICATION FOR AUTONOMOUS TRACTOR APPLICATIONS

by

Caleb Lindhorst

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Agricultural and Biological Systems Engineering

Under the Supervision of Professor Santosh Pitla

Lincoln, Nebraska

April 2019

FIELD OBSTACLE IDENTIFICATION USING MACHINE LEARNING TECHNIQUES FOR AUTONOMOUS TRACTOR APPLICATIONS

Caleb Lindhorst, M.S.

University of Nebraska, 2019

Advisor: Santosh Pitla

New technologies are being developed to meet the growing demand for agricultural products. Autonomous tractors are one of the many solutions to address this demand. Obstacle detection and avoidance is an important consideration for safe operation of any autonomous machine. Three field obstacles were chosen to be identified in this thesis work: tractors, round bales, and center pivots. Limited research work was found on the identification of center pivot detection.

Feasibility of using low cost LIDARs was considered for the detection of tractors, bales, and agricultural center pivots. Performance of LIDARs in different lighting conditions, different colors of obstacles, accuracy and angular resolution was evaluated. It was found that low cost LIDARs do not have a small enough angular resolution to detect pivots at a distance to avoid the obstacle. Formulas were derived to help find the distance between steps of the LIDAR.

Obstacle identification is also important so that proper corrective actions can be taken to avoid the obstacle. RGB cameras were used to aid in the detection of center pivots. SURF Feature Extraction and Matching, Viola-Jones algorithm and edge detection with a shape identification algorithm were tried but none of the algorithms could adapt to more than one orientation or class of object.

Obstacle identification using Convolutional Neural Networks (CNNs) for obstacle detection was pursued. Each obstacle was individually trained first and then all classes were combined to create one object detector. Faster Region based CNN (R-CNN) was used with GoogLeNet to give high mean Average Precision (mAP).

Acknowledgements

I want to first thank God for all the blessings in my life even the life-changing accident I was in. If it wasn't for the accident, I probably would not have even thought about pursuing a graduate degree.

Mom and Dad thank you for all your support and love. You always told me that you believed in me even when I didn't. You helped in more ways than I can even count. Thank you for all the things you did.

My entire family including those not technically a part of the family yet. You have been through it all. Thank you for encouraging and helping me get through things that were not going good at that moment.

My advisor for giving me this opportunity and committee members for always giving me enough constructive criticism to keep pushing and improving my research. You all had a significant part in helping me finish my research.

Dr. Santosh Pitla, Dr. Roger Hoy, Dr. Yufeng Ge, Dr. Joe Luck

Thank you to all the undergraduate students that helped me with my research even if it was the most boring task you have ever done.

Brian Hurst, Robert Goldsworthy, Isaac Hanson

For all my office mates. Thank you for helping me on trips and making grad school bearable.

Chandler Folkerts, Gabe Stoll, Rodney Rohrer, Abbas Atefi
Josh Murman, John Evans, Aaron Shearer, Rachel Stevens

Table of Contents

Chapter 1 Introduction and Background 1

Chapter 2 Sensors and Methodologies to Detect Agricultural Obstacles 6

2.1 Obstacle Definitions 6

2.2 Obstacle Detection Sensors 7

2.2.1 Ultrasonic sensors 7

2.2.2 LIDAR/LADAR 7

2.2.3 Cameras 9

2.2.4 Sensor Fusion 12

Chapter 3 Off-the-Shelf Inexpensive LIDAR Evaluation in Controlled Settings 13

3.1 Introduction 13

3.2 Objectives 15

3.3 Materials and Methods 15

3.4 Results and discussion 24

3.4.1 SCANSE Results and Discussion 24

3.4.2 RPLIDAR Results and Discussion 25

3.4.3 Hokuyo URG-04LX-UG01 Results and Discussion 33

3.5 LIDAR Conclusions 33

3.5.1 SCANSE Conclusion 34

3.5.2 RPLIDAR Test Conclusions 34

3.5.3 Hokuyo URG-04LX-UG01 conclusions 34

Chapter 4 Obstacle Detection and Identification Algorithms for Camera Images 35

4.1 Introduction and Objectives 35

4.2 SURF Feature Extraction and Matching 35

4.3 Edge Detection 36

4.4 Combination of SURF Feature Extraction and Matching and Edge Detection Algorithms 38

4.5 Conclusions from SURF Feature Extraction and Matching and Edge Detection 39

4.6 Object Identification 40

4.6.1 Cascade Object Detector and the Viola-Jones Algorithm 41

4.6.2 Convolutional neural networks (CNN) 42

4.7 Conclusions 54

Chapter 5 Convolutional Neural Networks for Tractor, Round Bale, and Center-Pivot Identification 55

5.1 Intoduction and Objectives 55

5.2 R-CNN 56

5.3 Faster R-CNN 57

5.4 Training Faster R-CNN 57

5.5 Results of Faster R-CNN training 61
5.5.1 Tractor 61
5.5.2 Round Bale 64
5.5.3 Pivot 66
5.5.4 Combined 69
5.5.5 Pivotv2 71
5.6 Conclusions of Faster R-CNN Training 76
Chapter 6 Conclusions and Future Work 77
6.1 Conclusions 77
6.2 Future Work 78
References 80
APPENDIX A SCANSE 94
APPENDIX A.1 Test Setup 94
APPENDIX A.2 Data Collection code (Arduino) 94
APPENDIX A.3 Plotting Data code (MATLAB) 101
APPENDIX A.4 Sample Data 102
APPENDIX A.5 Sensor Calculations 105
APPENDIX A.6 Sample Data Plot 108
APPENDIX B RPLIDAR 109
APPENDIX B.1 Test Setup 111

APPENDIX B.1.1 Occupancy Map Setup 111
APPENDIX B.1.2 Straight Line Setup 112
APPENDIX B.1.3 Final RPLIDAR Test Setup 113
APPENDIX B.2 Data Collection code (Arduino) 114
APPENDIX B.3 Plotting Data code (MATLAB) 121
APPENDIX B.4 Sample Data 122
APPENDIX B.5 Sensor Calculations 125
APPENDIX B.6 Occupancy Map Generation Code (MATLAB) 127
APPENDIX B.7 Sample Point Summaries (no false positives) 128
APPENDIX C Hokuyo URG-04LX-UG01 134
APPENDIX C.1 Test Setup 134
APPENDIX C.2 Data Collection code (C) 135
APPENDIX C.3 Plotting Data code (MATLAB) 137
APPENDIX C.4 Sample Data 138
APPENDIX D SURF Feature Extraction and Matching and Edge Detection 140
APPENDIX D.1 SURF Feature Extraction and Matching Script (MATLAB) 140
APPENDIX D.2 Edge Detection Script (MATLAB) 141
APPENDIX D.3 Shape Identification algorithm (MATLAB) 142
APPENDIX D.4 Edge detection and Shape Identification Algorithm (MATLAB) 145

APPENDIX D.5 SURF Feature Extraction and Matching and Edge Detection (MATLAB) 147

APPENDIX D.6 approxcanny 148

APPENDIX D.7 Canny 149

APPENDIX D.8 log 149

APPENDIX D.9 Prewitts 150

APPENDIX D.10 Roberts 150

APPENDIX D.11 Sobel 151

APPENDIX D.12 zerocross 151

APPENDIX E Viola-Jones Algorithm Script (MATLAB) 152

APPENDIX F Faster R-CNN Training 152

APPENDIX F.1 Code used to extract images out of video (MATLAB) 152

APPENDIX F.2 Augmentation Code (MATLAB) 153

APPENDIX F.3 Training detector Code (MATLAB) 153

APPENDIX F.4 Code for detecting objects using a trained detector for a single image (MATLAB) 156

APPENDIX F.5 Code for detecting objects using a trained detector for a multiple images (MATLAB) 156

APPENDIX F.6 Shuffling code (MATLAB) 157

APPENDIX F.7 Padding code (MATLAB) 157

APPENDIX F.8 Tractor images 157

APPENDIX F.8.1 Run 1 158

APPENDIX F.8.2 Run 2 183

APPENDIX F.9 Round Bale 204

APPENDIX F.9.1 Run 1 205

APPENDIX F.9.2 Run 2 224

APPENDIX F.10 Pivot 243

APPENDIX F.10.1 Run 1 243

APPENDIX F.10.2 Run 2 271

APPENDIX F.11 Combined Detector 286

APPENDIX F.11.1 GoogLeNet Run 1 287

APPENDIX F.11.2 GoogLeNet Run 2 305

APPENDIX F.12 Pivotv2 324

APPENDIX F.12.1 GoogLeNet 324

List of Figures

Figure 2.1 From top to bottom: tractor (dynamic, John Deere, 2019), round bales (static temporary, Pixabay 2019), barn (static permanent, 95oldcolonyrd.com, 2019)..... 6

Figure 2.2 Outdoor experimental results. Green is traversable red is non-traversable (Bellone et al., 2013)..... 10

Figure 2.3 Occupancy grid created by a LIDAR (Green box).Red lines are LIDAR beams being generated by the sensor. Black boxes are obstacle or edge of the sensor edge. White boxes are clear areas. Gray boxes are unknown. 11

Figure 2.4 DEM vision processing. To represent the road (blue), traffic isles (yellow) and obstacles (red). (Oniga & Nedevschi, 2010)..... 11

Figure 3.3 Arduino Mega 2560 board..... 16

Figure 3.1 The SCANSE test stand with the spinning LIDAR mounted on the top..... 16

Figure 3.2 The RPLIDAR test stand merged with an Arduino Mega 2560 on the right and a breadboard on the left..... 16

Figure 3.4 (a) Layout of a Run (b) Three runs are equal to one run..... 17

Figure 3.5 Red, Green, Blue obstacle configurations. Orange diamond is location of the sensor. 18

Figure 3.6 Obstacle used in the SCANSE Sweep test 19

Figure 3.7 Obstacle used in the RPLIDAR and Hokuyo tests..... 19

Figure 3.8 Setup for the Final Test. Obstacle in the sun RPLIDAR in the shade and vice versa 20

Figure 3.9 Location of variables 22

Figure 3.10 SCANSE Sweep plotted results for one data gathering session of the green obstacle location..... 24

Figure 3.11 Plotted RPLIDAR data..... 25

Figure 3.12 Occupancy Maps and two different path planning algorithms..... 26

Figure 3.13 Obstacle setup on the brown backside in the Straight Line Test..... 27

Figure 3.14 The proportion of false positives as distance increases and in both lighting positions. The 95% confidence interval bounds are shown..... 28

Figure 3.15 The proportion of false positives between distances and obstacle colors with a range of 95% confidence interval bounds for the number of false positives..... 29

Figure 3.16 The average amount of data points collected and 95% confidence interval bounds with the color and distance being the set variables. 30

Figure 3.17 A graph showing the LS-Means and 95% confidence interval bounds between the two color obstacles at each distance. It is noticed that at distances < six meters away the brown/black obstacles are more accurate and distances > six white obstacles are more..... 31

Figure 3.18 A graph showing the LS-Means for accuracy when the RPLIDAR in the sun and both color obstacles. A glance at the graph shows accuracy staying under 100 mm untl it reaches 9000 mm..... 32

Figure 3.19 Plotted data from Test 6 33

Figure 4.1(a) Extracted features out of the grayscale reference image. (b) Extracted features out of the grayscale test image. (c) Matching the extracted features in reference photo to features in the test photo. Only ouliers are displayed because no inliers were preent..... 36

Figure 4.2 Sobel edge detection with the "Shape Recognition" algorithm. The initial image can be seen in the top left picture. The bottom left picture is the result of using the Sobel edge detection method. The top right depicts the objects the algorithm detected based on the input binary image. This image is flipped compared to the input image. The bottom right gives scores on what the shape of the object is 38

Figure 4.3 Positively matched points (Inliers only)..... 39

Figure 4.4 Positively matched SURF Points (Including Outliers)..... 39

Figure 4.5 Viola-Jones algorithm detecting a tractor with no clutter. Yellow bounding box does not surround the entire tractor. 42

Figure 4.6 Viola-Jones algorithm detecting a pivot in the middle of a soybean field. Yellow bounding boxes do not surround the entire pivot but, false positives..... 42

Figure 4.7 Layout of the entirety of a CNN with a pivot as the input image 43

Figure 4.8 Layers in convolutional neural network..... 44

Figure 4.9 Layout of a single neuron in a CNN..... 45

Figure 4.10 Example of a convolution. A simple picture of a straight line. With a straight line filter. The input image with [1 1 1 1] is in (a) resulting matrix (b) and stride [1 1].. 46

Figure 4.11 Illustration of an image [9 9], filter size [3 3] and a stride of [1 1]..... 47

Figure 4.12 The result of convolution between the test image and the chosen filter 48

Figure 4.13 The resulting matrix after a bias of -4 is implemented..... 48

Figure 4.14 Graphical representation of the activation functions..... 49

Figure 4.15 Result of ReLu on Example 1 50

Figure 4.16 Examples of the 2 methods "VALID" and "SAME" filter width= 3 stride = 3 52

Figure 4.17 Result of "VALID" max pooling in the example 52

Figure 4.18 Result of "SAME" max pooling 52

Figure 4.19 flattening of the "VALID" max pooling (Figure 4.17)..... 53

Figure 4.20 SoftMax equation and example 53

Figure 4.21 The Final Layers of a CNN. Flattening, two fully connected layers and SoftMax to give the probability of that object in the picture 54

Figure 5.1 Steps of an R-CNN (Lee, 2017) 56

Figure 5.2 Two network layout of a Faster CNN network. The CNN can be replaced with any CNN to best fit the application applied to. (Sinhal & Sachan, 2017) 57

Figure 5.3 Tractor image 448 x 448 after padding to keep the same aspect ratio. The black part of the image is padding. 58

Figure 5.4 (a) A 1920 x 1080 pivot picture (b) pivot picture resized to 448 x 448 with no padding. 59

Figure 5.5 Components of a pivot (a) span (b) pyramid (c) tower (d) end tower..... 59

Figure 5.6 Accuracy vs time, the marker shape indicates the meta-architecture and color indficates feature extractor used. (Huang et al., 2016) 60

Figure 5.7 Tractor Faster R-CNN training graphed results 62

Figure 5.8 Tractor training results 63

Figure 5.9 Graphical representation of the training results from the round bale Faster R-CNN training. 65

Figure 5.10 Bale detector testing images. 65

Figure 5.11 Results of pivot training graphed 67

Figure 5.12 Pivot detectorv1 labeling: (a) two spans and a tower, (b) pyramid and span, (c) span and end tower 68

Figure 5.13 Accuracy vs time of combined dedectors. 70

Figure 5.14 Results from running the Run 1 combined detector on pictures from the internet. (a) A tractor and multiple bales and (b) multiple bales in a field. 70

Figure 5.15 Collage of the six classes the combined detector is suppose to be able to identify. 71

Figure 5.16 Pivot detectorv2 labeling: (a) two spans and a tower, (b) pyramid and span, (c) span and end tower 73

Figure 5.17 A graph comparing all trained detectors 74

Figure 5.18 A graphical representation of the detector testing times along with the accuracy in detection..... 75

List of Tables

Table 2.1 Examples of field obstacles and classes they are assigned to. 7

Table 3.1 Required sensor and tested LIDAR specifications (*calculated from other specifications **priced on roboshop.com priced 7/11/18) 13

Table 3.2 Data gathered during testing in the shade at a 2 meter distance red rows were eliminated and green rows were kept..... 25

Table 3.3 Sample point data summary Obstacle in the shade obstacle color is black at a distance of 1 meter 29

Table 5.1 Tractor Faster R-CNN training results..... 62

Table 5.2 Round Bale Faster R-CNN training result*(b)* 63

Table 5.3 Results from pivot training*(b)*..... 65

Table 5.4 Results from pivot training 66

Table 5.5 mAP scores for each component of the pivot*(c)*..... 68

Table 5.6 Results of training with more pivot image data..... 72

Table 5.7 Summary statistics for each feature extractor and object class in Faster R-CNN 74

Table 5.8 Detector speed and accuracy..... 75

List of Equations

Equation 1.....Angular Resolution 20
Equation 2.....Arc Length 21
Equation 3 .....Law of Cosines 22
Equation 4…….Linear Distance between two consecutive data points 22
Equation 5.....Number of data points per one foot 23
Equation 6.....Mathematical Representation of a Neuron 45
Equation 7.....Sigmoid Function 49
Equation 8……..Hyperbolic Tangent 50
Equation 9…….ReLu 50
Equation 10…….……..Leaky ReLu 50
Equation 11……..……...Parametric ReLu 51
Equation 12……..……..……Exponential Linear Units (ELU) 51

Chapter 1 Introduction and Background

Food production has significantly improved in the past 300 years. Humans have applied the knowledge developed in math and sciences during the renaissance era to improve the lives of humans around the world. The industrial revolution was the beginning of a population boom that is expected to continue growing at an exponential rate. Presently there is an estimated 7 billion people on the planet and by 2050 there will be an estimated 9.6 billion people inhabiting the earth (Kochhar, 2014). New methods and technologies have been and will be developed to satisfy the growing demand for food. The expected population increase requires increasing the overall food production by nearly 70% (FAO Director's office, 2009). Cereal grains for both human and animal consumption are projected to increase by one billion metric tons.

Food production advancements started in the late 1700's with the threshing machine (Spielmaker, 2018). Farmers went from feeding an estimated 10.7 people per farmer in the 1940's to farmers in Iowa being able to produce enough food to feed 155 people per farmer in 2010 (Sullivan, 2014). A timeline of farm mechanization advancements is shown in the list below (Spielmaker, 2018):

- 1788 – Invention of the threshing machine in by Andrew Meikle (a Scottish engineer)
- 1830's – invention of the steel plow
- 1868 – Steam tractors are tested
- 1892 – John Froelich produced the first gasoline tractor
- 1918 – First combine with auxiliary engine introduced

- 1930s – Rubber tires first introduced
- 1945-1970s – Farmers change from horses to tractors
- 1980s – Introduction of GPS
- 2010s – Introduction of the autonomous tractor concept

Current applications of electronics in agriculture could be the current Global Positioning Systems (GPS), monitors and displays, or computers and sensors on the future autonomous equipment. GPS was first introduced in the 1980's which significantly contributed to the progress of precision farming. The idea of an autonomous tractor has been around since the introduction of precision farming in the 1980's (Big Ag, 2018). Today's technology has advanced far enough that equipment manufacturers such as Case IH, New Holland (Case IH, 2016) and AGCO are promising autonomous tractors for commercial in the coming years (Bedord, 2018)

An autonomous tractor may have three substantial benefits for farmers and farm workers (Big Ag, 2018). The first benefit is that autonomous tractors could have significantly better accuracy in field applications by increasing fertilizer and chemical application rate uniformity and reducing planting population inaccuracies. The second way that an autonomous tractor could benefit farmers is by collecting information on soil conditions and plant health during the growing season with minimal human supervision. The third benefit is that it could reduce stress and workload of farmers and farm workers during the growing season.

The five levels of autonomy as defined by Case IH ((Case IH, 2018, Vogt, 2018) are listed as follows:

1. **Guidance** – In Guidance, a tractor can be controlled by a GPS while driving and an operator is inside the tractor, like many tractors today. An operator is always present.
2. **Coordinate and optimization** – This level of autonomy is like a tractor and grain cart linked to the combine. The tractor and grain cart try to match the speed of the combine.
3. **Operator Assisted Automation** – This style of autonomy is much like a leader/follower architecture. One machine is fully autonomous while another machine has an operator present. The autonomous machine follows the operator through the field.
4. **Supervised autonomy** – This level of autonomy allows all machines to run on their own with a supervisor managing the machines nearby.
5. **Full autonomy** – The highest level of autonomy is full autonomy and no local supervision. However, there can be remote supervision.

An area of interest for any autonomous robot is obstacle detection. The objects that need to be detected depend on the environment of the robot. A household robot may have to detect floor obstacles like shoes or stairs; however, agricultural robots need to detect obstacles that are unique to agriculture – such as other tractors, bales, and agricultural center pivots in irrigated agricultural fields.

A center pivot is a unique obstacle to agricultural robots. A center pivot is an agricultural machine that is used to water crops during dry weather patterns and drought. These machines are found on farms throughout the world.

In this research, different sensors, cameras, and methodologies were explored to detect and identify agricultural obstacles such as tractors, bales, and center pivots.

Chapter 2: Sensors and Methodologies to Detect Agricultural Obstacles

1. Determine which sensors are used for obstacle detection
2. Methods used in Obstacle detection

Chapter 3: Off-the-Shelf Inexpensive LIDAR Evaluation in Controlled Settings

1. Test selected off-the-self sensors
2. Determine if off-the-self sensors meet performance needs

Chapter 4: Obstacle Detection and Identification Algorithms for Camera Images

1. Explore different machine vision algorithms for obstacle identification
2. Preview of a Convolutional Neural Networks (CNN)

Chapter 5: Convolutional Neural Networks for Tractor, Round Bale, and Center-Pivot Identification

1. Identify an obstacle identification algorithm that has over 85% mean Average Precision (mAP) at 5 Hz or greater for real time obstacle detection.
2. Train an object detector for tractors, round bales and center-pivots

Chapter 6:

Conclusions and Future Work

Chapter 2 Sensors and Methodologies to Detect Agricultural Obstacles

2.1 Obstacle Definitions

An obstacle is an obstruction in the tractor's path. Obstacles can be classified into three categories based on the obstacle's movement. The first category is a dynamic obstacle, which is moving or self-powered in nature such as a tractor, pivot, or an animal. The second category obstacle is static temporary and requires outside power to be moved. Examples of static temporary obstacles are a round bale, implement, or a temporary snow fence. The final category is static permanent. Obstacles that cannot be moved like barns, trees, or fence lines are considered static permanent. Pictures of these obstacles are shown in Figure 2.1 and examples are given in Table 2.1.

Figure 2.1 From top to bottom: tractor (dynamic, John Deere, 2019), round bales (static temporary, Pixabay 2019), barn (static permanent, 95oldcolonyrd.com, 2019)

2.2 Obstacle Detection Sensors

2.2.1 *Ultrasonic sensors*

Ultrasonic sensors use soundwaves to detect obstacles. These types of sensors measure distance by measuring the time it takes the sound waves to rebound off the obstacle. Ultrasonic sensors use higher frequency sound waves (20-100 kHz) to overcome most noise pollution that occurs at lower wave frequencies (20-200 Hz).

Table 2.1 Examples of field obstacles and classes they are assigned to.

Field Obstacle	Obstacle class
Pivot	Dynamic
Tractor	Dynamic
Animal	Dynamic
Agricultural Implements	Static temporary
Round Bale	Static temporary
Small Piles	Static temporary
Wells	Static permanent
Field Boundaries	Static permanent
Trees and Tree Line	Static permanent

Benefits from using an ultrasonic sensor are that they can be used in dark environment and are unaffected by color of the object or dust (Gillespie, 2018). Some of the disadvantages to an ultrasonic sensor include no location of objects in 3-D space, no orientation information of the object to the sensor and surface temperatures may impact wave speeds (Jo & Jung, 2014; Senix Corporation, 2018).

2.2.2 *LIDAR/LADAR*

The terms LIDAR and LADAR come from LIght Detection And Ranging (LIDAR) and LAser Detection And Ranging (LADAR) sensors. LIDAR and LADAR are similar, which for the scope of this review and thesis, will be technically equivalent and referenced as LIDAR.

Two styles of LIDAR are currently found off-the shelf, spinning LIDARs and solid state LIDARs (Mokey, 2018). Both styles of LIDAR used for navigation and obstacle detection can be 2-D or 3-D LIDARs. Spinning LIDAR sensors that are 2-D have a single LIDAR that spins in a circle collecting distance data on a single plane. 3-D LIDARs have a mechanism that spins an array of LIDARs stacked on top of each other, gathering a 3-D point cloud or multiple 2-D planes, one for each LIDAR beam. The Velodyne HDL-64E has an electromechanical system that spins a small housing at a very high speed collecting up to 300,000 data points per second. These sensors can cost upwards of $75,000 and have a life span of 1,000 to 2,000 hours (Mapanauta, 2018).

Unlike spinning LIDARs, Solid State LIDARs have no moving mechanical parts (Dubois, 2018). Quanergy's S3 solid state LIDAR uses silicon to "steer" the light. Currently a solid-state LIDAR can be purchased for around $900, but through making several chips one solid chip, Louay Eldada the CEO of Quanergy, foresees "At that point, our sales price will become under $100."

LIDARs are also shown to work well in agriculture. In a study by Freitas et al. (2012), an autonomous vehicle traveled between apple tree rows of an apple orchard that used wheel and steering encoders, one SICK LMS 291 laser scanner (LIDAR), and one Inertial Measurement Unit (IMU). Using a push – broom method, the researchers in this study implemented a 3-D point cloud strategy to avoid obstacles that were within 4 meters of the vehicle. Experiments were conducted in a lab setting in Pittsburg and then at an apple orchard in Washington state. The conclusions of this research showed that this method

was able to detect people moving across the row at walking speed and objects at least 15 cm tall and not covered by grass.

2.2.3 *Cameras*

All cameras can be either Red, Green and, Blue (RGB) or black and white, to gather image information about an object. A camera produces an image that is 2-D with no depth data. These captured images can be used in image processing. Cameras become 3-D by implementing stereo vision. Stereo vision extracts 3-D information from two different camera angles and calculates distance using triangulation. Human vision is a good example of stereo vision.

2.2.3.1 RGB

RGB cameras give a matrix of three-color values for Red, Green and Blue. The color intensity of the pixel is given in the form of a three-layer matrix the size of the image. The combination of these three primary colors can provide any color that is needed. A digital camera is an example of an RGB camera.

Pictures from RGB cameras be can used in various algorithms to identify obstacles. MATLAB includes various obstacle detection algorithms using RGB cameras. The Algorithms include SURF Feature Extraction and Matching, edge detection methods, Viola-Jones Algorithm and CNNs.

2.2.3.2 RGB-D

An RGB-D camera is a RGB camera that can output the distance of an object it is capturing; hence the D in RGB-D. A low cost RGB-D camera is the Microsoft Kinect, which is a popular consumer RGB-D camera. This device uses a per-pixel depth sensing

technology developed by PrimeSense (Litomisky, 2012). The Microsoft Kinect's distance is found using an infrared camera with an infrared projector/emitter. Stereo vision from the infrared systems is used to obtain a 3-D image.

A study by Sabale and Vaidya (2016) found the accuracy of the measured distance of the Microsoft Kinect sensor. Kinect has a range from about 800 millimeters to about 4,000 millimeters. In this experiment, measurements were taken every 100 mm from 800-3500 mm using the manufacture's calibrations for the camera. It was found that the percentage error with respect to the actual distance to be approximately 3.6%. The Microsoft Kinect had an increasing difference the further the object was from the camera, but the mean percentage error slightly decreased. Bellone et al. (2013), used an RGB-D camera to create an Unevenness Point Descriptor (UPD), a combination of the roughness and the inclination indices (Reina, 2013). Vectors normal to the terrain were calculated and could be classified into two categories: traversable surfaces (green vectors) and non-traversable objects (red vectors) (Figure 2.2). Experiments were performed in indoor and outdoor settings. The Microsoft Kinect was used for indoor experiments, while the Point Grey Bumblebee XB3 stereo system was used for the outdoor experiments. A different camera was used outdoors due to inability of the Microsoft Kinect to collect data in bright outdoor conditions. The results were successful in both the outdoor and indoor conditions, even when detecting dynamic obstacles.

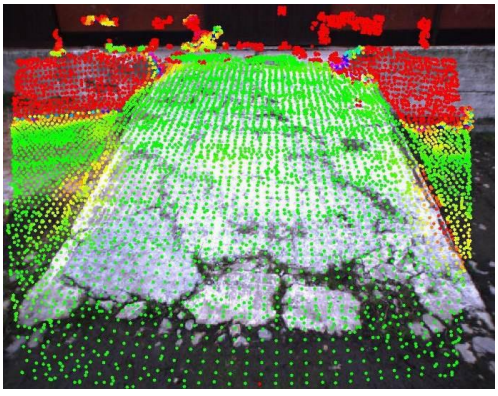


Figure 2.2 Outdoor experimental results. Green is traversable red is non-traversable (Bellone et al., 2013).

2.2.3.3 Obstacle detection algorithms for RGB-D, RGB cameras and stereovision

Obstacle detection systems using stereo vision implement tessellation or clustering strategies, that can be categorized into 4 possible models: (Bernini et al., 2014)

- Probabilistic occupancy maps
- Digital elevation maps
- Scene flow segmentation
- Geometry-based cluster

The probabilistic occupancy map was proposed by Elfes in 1989 (Elfes, 1989). This method breaks the field of vision into a grid system. Grid blocks are then given one of three possible states: occupied, free, or unknown (Figure 2.3). The computer tells the machine which necessary actions need to be taken to avoid the obstacle. Processing time is not given for this visual processing method nor the accuracy of the system.

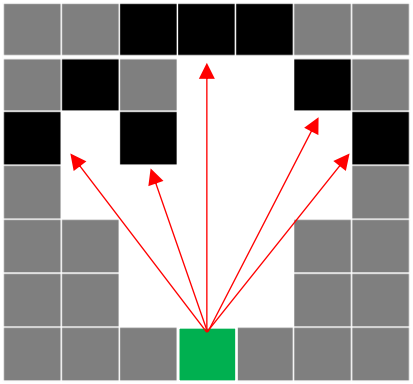


Figure 2.3 Occupancy grid created by a LIDAR (Green box).Red lines are LIDAR beams being generated by the sensor. Black boxes are obstacle or edge of the sensor edge. White boxes are clear areas. Gray boxes are unknown.

Digital elevation maps plot the heights of objects onto a Cartesian plane like the occupancy grid map (Oniga & Nedevschi, 2010). The grid cells are often identified into three different classes, road, traffic isles, and obstacles (Figure 2.4).

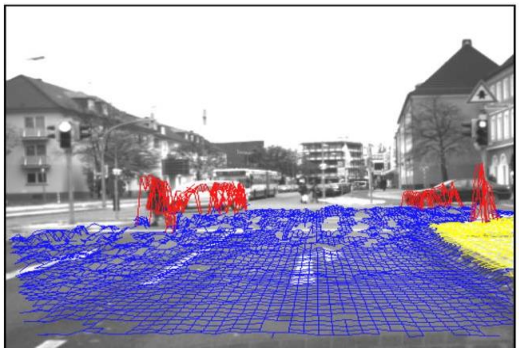


Figure 2.4 DEM vision processing. To represent the road (blue), traffic isles (yellow) and obstacles (red). (Oniga & Nedevschi, 2010)

Scene flow segmentation uses temporal correlation between two frames captured by a camera at two different times using stereo vision. (Lenz et al., 2011). To accommodate the slow processing speed, a rural traffic setting was considered for validation of the proposed obstacle detection algorithm. The study concluded that the algorithm worked well in rural settings and it detected objects up to 60 meters away. However, in urban settings it was only able to detect obstacles up to 50 meters away. Geometry based clustering will not be discussed.

2.2.4 *Sensor Fusion*

Many sensors are needed to avoid obstacles. Sensor fusion is the process of combining the outputs of multiple sensors to accomplish a complex task such as obstacle detection. The RGB camera and LIDARs may be used in sensor fusion. These sensors can provide information to the machine to help identify and classify obstacles. There are methods to detect obstacles using images. LIDARs can help determine distance to an obstacle detected by an RGB camera.

The Extended Kalman Filter (EKF) is an algorithm that can be applied to sensor fusion and fuse sensor information together (Barbosa et al., 2016). In this study GPS and IMU data is fused together to track the yaw rate, lateral velocity and longitudinal velocity of a car. The first obstacle detection sensors used in this study were inexpensive off-the-self sensors (Chapter 3). Obstacle detection and identification using camera images is discussed in Chapter 4 and Chapter 5.

Chapter 3 Off-the-Shelf Inexpensive LIDAR Evaluation in Controlled Settings

3.1 Introduction

This chapter characterizes the performance of three off-the-shelf LIDAR sensors and proposes equations when selecting a LIDAR. The LIDARs were selected on 5 specifications:

- Price
- Range
- angular resolution
- sampling rate
- sunlight resistance

The driving factors were initially price and range. As more testing was completed the angular resolution, sampling rate and sunlight resistance became important characteristics. Table 3.1 list specifications of the three sensors tested. The first LIDAR explored was the SCANSE Sweep, second was the RPLIDAR and finally the HOKUYO URG-04LX-UG-01. All three sensors output 2-D polar coordinates.

Table 3.1 Required sensor and tested LIDAR specifications (*calculated from other specifications **priced on roboshop.com priced 7/11/18)

Spec	SCANSE Sweep	RPLIDAR A2M8	Hokuyo URG-04LX-UG01
Scan Angle	360°	360°	240°
Angular Resolution	≈0.36° -7.20°*	≈0.45°-1.35°*	≈0.36°
Range	40m	8m	4m
Spinning Frequency	1-10Hz	5-15Hz	Solid State
Max Sampling Rate	1000Hz	4000Hz	6650Hz*
Sunlight resistant	Yes	Yes	No
Price (USD)	349**	319**	1080**

Information regarding these sensors can be found in APPENDIX A SCANSE, APPENDIX B RPLIDAR, and APPENDIX C Hokuyo URG-04LX-UG01.

The SCANSE was the first LIDAR purchased for testing. Only one test was done with the SCANSE Sweep before it malfunctioned. This sensor is no longer available in the market because the company that produces them went out of business on May 10th, 2018 (Magneon, 2018).

The RPLIDAR is a low cost sensor ($319), that had acceptable angular resolution based on evaluations using equations developed from SCANSE testing. As shown in Table 3.1 the angular resolution was found to be between 0.45° and 1.35°, depending on the spinning frequency. The equations developed to calculate these specifications can be found in 3.4.1. The RPLIDAR did not have the range of the SCANSE Sweep.

The Hokuyo was the most expensive LIDAR tested and the only solid-state LIDAR tested ($1,080). This sensor offered a high sampling rate and a high angular resolution but, only had a 4 meter range and was tested just once.

Both SCANSE and RPLIDAR outputs were distance, azimuth and signal strength. The SCANSE measured distance to the obstacle in centimeters, the azimuth was measured in degrees and the signal strength was 0 – 255. 255 was the best signal strength and 0 was the worst. The data produced by the RPLIDAR were the distance in millimeters, azimuth in degrees, and the signal strength as an integer from 0 – 15. 15 was the highest signal strength and 0 was the weakest signal strength. The Hokuyo outputted X and Y coordinates in millimeters, azimuth measured in radians, radial distance in millimeters,

and a timestamp but does not give signal strength. All three sensors used serial communication.

3.2 Objectives

The objectives for LIDAR testing were:

- Test selected low cost off-the-self sensors (SCANSE, RPLIDAR, Hokuyo URG-04LX-UG01) to observe if low cost sensors have the capabilities of detecting bales, tractors and center pivots.
- Determine the statistical effect that distance, color of obstacle and lighting condition have on the ability of being detected by LIDAR.

If the first objective could not be met, develop equations that would aid in the sizing of LIDARs to detect obstacles. Data points would be collected to create a binary occupancy grid and see the possibility of creating probabilistic road maps. Statistical comparisons would be made on accuracy, number of data points and number of false positives between different distances, lighting conditions and color of obstacle.

3.3 Materials and Methods

The two parameters that could be set on the SCANSE were sampling rate and the spinning frequency. The sample rate of this sensor could be set to 500 Hz, 750 Hz or 1,000 Hz. The spinning frequency of the head of the sensor could be set from 0 Hz to 10 Hz. The defaults of 500 Hz sampling rate and 5 Hz spinning speed were used for the SCANSE Sweep. The RPLIDAR had a set sampling rate of 4,000 Hz and a range of spinning frequency from 5 Hz to 15 Hz. The default settings of 4,000 Hz sampling rate and a spinning rate of 10 Hz were used for RPLIDAR. Solid State LIDARs do not have a

spinning head. Therefore, no spinning rate settings were adjusted on the Hokuyo. The sampling rate could not be adjusted as well.

The SCANSE and RPLIDAR were mounted on a test stand as shown in Figure 3.2 and Figure 3.3. Both sensors were connected to an Arduino Mega 2560 (Figure 3.1). Arduino hardware and software were used to collect data from the SCANSE Sweep and the RPLIDAR. There was no trouble with data collection with the SCANSE, using the default settings so, a similar approach was attempted with the RPLIDAR using the default parameters. However, the dynamic memory of the Arduino could not write and save data points fast enough during RPLIDAR data collection.

Figure 3.2 The SCANSE test stand with the spinning LIDAR mounted on the top

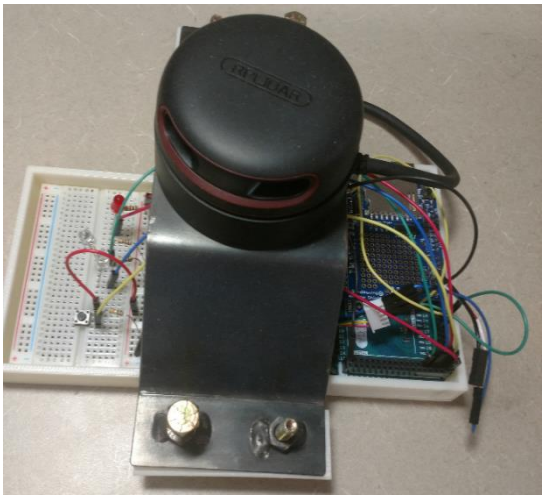


Figure 3.3 The RPLIDAR test stand merged with an Arduino Mega 2560 on the right and a breadboard on the left

Figure 3.1 Arduino Mega 2560 board

To work around this problem, the Arduino accumulated one scan's worth of data points three times then stored the data to an SD card for a collection point. Then three collection points constituted one run and three runs were gathered for each test (Figure 3.4).

A Raspberry Pi using C code was used instead of the Arduino for interfacing and data collection with the Hokuyo LIDAR because the Arduino could not handle the amount of data coming from the Hokuyo.

Example code and wiring diagram for the SCANSE Sweep wired to an Arduino MEGA 2560 was found on GitHub (*sweep-arduino*, 2017/2017). Example code for the RPLIDAR was also found for Arduino on GitHub (Repos, 2014/2019). Example code for the Hokuyo was found on Sourceforge (Satofumi Kamimura, 2013). All code used in LIDAR data collection can be found in SCANSE APPENDIX A.2 Data Collection code (Arduino), RPLIDAR APPENDIX B.2 Data Collection code (Arduino) and Hokuyo APPENDIX C.2 Data Collection code (C).

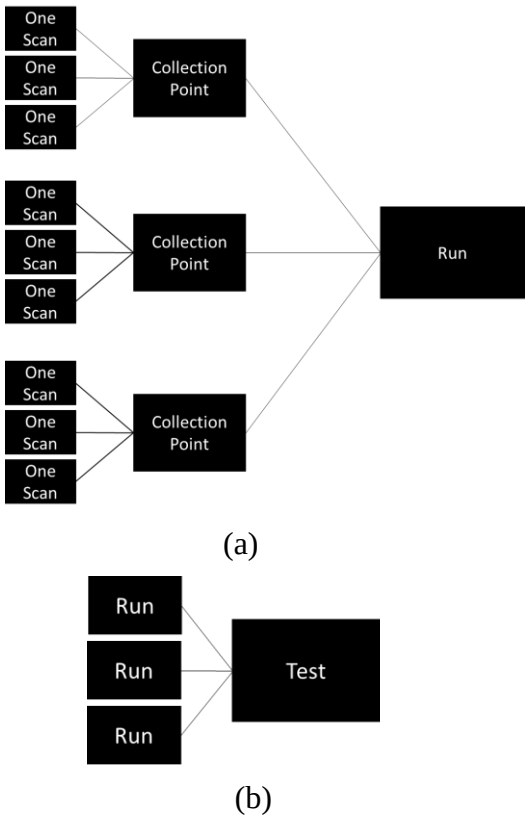


Figure 3.4 (a) Layout of a Run (b) Three runs are equal to one run

In the SCANSE and RPLIDAR tests, three sets of four obstacles were placed for the sensor to detect. An obstacle was located on each positive and negative X and Y axis and two obstacles in each quadrant of the X and Y graph. Figure 3.5 shows a typical obstacle

layout. The three generated configurations were called Red, Green and Blue configurations. The distances to the obstacles were randomly generated by a computer. The obstacle layouts for tests can be found APPENDIX A.1 Test Setup, APPENDIX

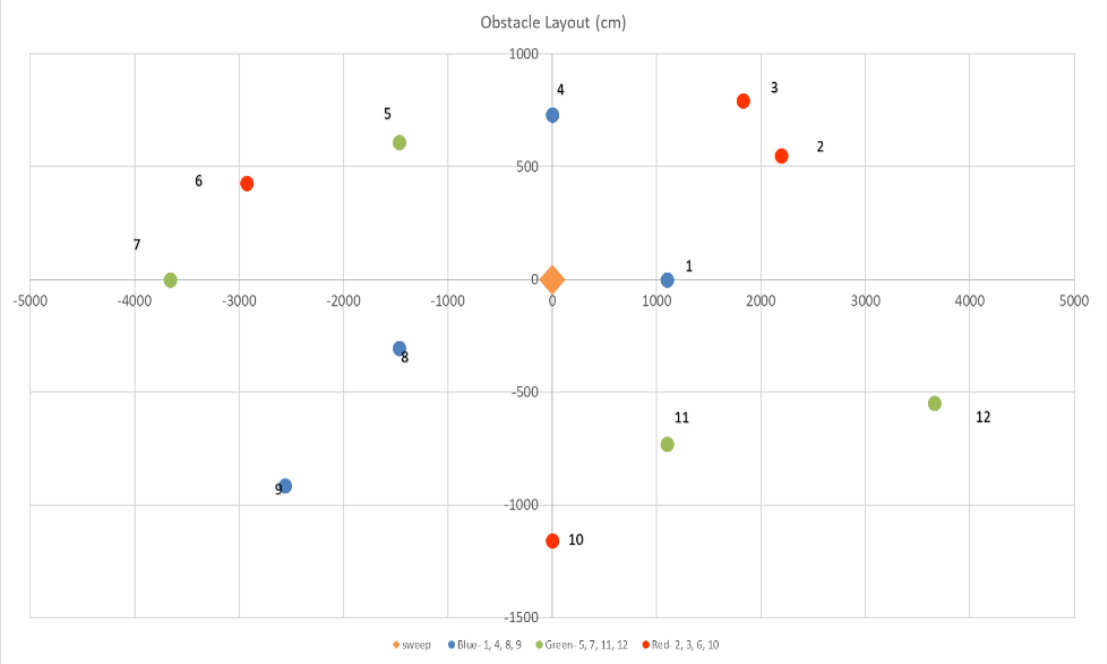


Figure 3.5 Red, Green, Blue obstacle configurations. Orange diamond is location of the sensor.

B.1.1 Occupancy Map Setup, APPENDIX B.1.2 Straight Line Setup, APPENDIX B.1.3 Final RPLIDAR Test Setup and APPENDIX C.1 Test Setup.

Obstacles were created from a piece of cardboard that was white on the front side and black/brown on the back.

The SCANSE obstacle measured one foot tall by three feet wide and clamped to a frame as depicted in Figure 3.6. The cardboard backing was rotated 90° for the RPLIDAR and Hokuyo as seen in Figure 3.7. Both sides were used in test to see if color of the obstacle would affect detection by the LIDARs.

Figure 3.6 Obstacle used in the SCANSE Sweep test

Figure 3.7 Obstacle used in the RPLIDAR and Hokuyo tests

In the Straight Line Test for the RPLIDAR, all of the obstacles were randomized and placed within the range of 802 – 777 cm in the Y direction, the obstacles would form a line along a wall in the X direction to document the smallest change that the LIDAR would be able to detect. Obstacles were three feet from a wall to allow room for the obstacle stands. The obstacles were placed three feet apart from each other in the X direction. Obstacles 4 – 12 were placed out of range of the sensor to test the true range of the sensor.

For The Final RPLIDAR Test, the test started at one meter and moved back in one meter steps, after collecting data two times on the white side and two times on the black/brown side, until 10 meters were reached. Obstacles were placed in the sun and shade

(Figure 3.8). For this test obstacles were only placed in the Y axis. Preprocessing the collected data was done manually to filter out unwanted data.

A BeagleBone microcontroller was programmed in C to collect data from the Hokuyo LIDAR. The code can be seen in APPENDIX C.2 Data Collection code (C) . To simulate fence posts, the poles of the obstacle stands were used which are similar in size and cross section to a steel fence post. The obstacles were place from 0 - 180° relative to the Hokuyo. Like the previous LIDAR tests, three sets of four obstacles were used.

Figure 3.8 Setup for the Final Test. Obstacle in the sun RPLIDAR in the shade and vice versa

An Excel Worksheet was created to evaluate the angular resolution of the SCANSE, based on the given parameters of the sensor. It became apparent that this sensor did not have a high enough angular resolution. The angular resolution was calculated using the following equation.

Equation 1.....Angular Resolution

$$\varphi = \frac{360\omega}{f_s}$$

Where: φ = angular resolution (degrees)

f_s = sampling frequency (Hz)

ω = spinning frequency (Hz)

Angular resolution varied with sensor settings. Values were spread between 0.36 and 7.20 degrees respectively. The results from these calculations on the SCANSE Sweep can be seen in APPENDIX A.5 Sensor Calculations. The data collected had an angular resolution of 3.60 degrees. The lowest angular resolution occurred at 500 Hz sampling rate and spinning frequency of 10 Hz, whereas the highest resolutions occurred at 1000 Hz sampling rate and 1 Hz spinning frequency. The arc length and the linear length were calculated using the arc length equation for arc length (Equation 2) and law of cosines for linear length (Equation 3). Figure 3.9 shows the locations of the variables.

Equation 2.....Arc Length

$$a_{\,l} = \frac{2\pi r\omega}{f_s}$$

Where: $a_{\,l}$ = arc length (cm)

r = distance to obstacle (cm)

Equation 3 .....Law of Cosines

$$a^2 = b^2 + c^2 - 2bc * cos(A)$$

Where: a = desired side's unknown length (cm)

A = angle opposite side of length a (degrees)

b = side length (cm)

c = side length (cm)

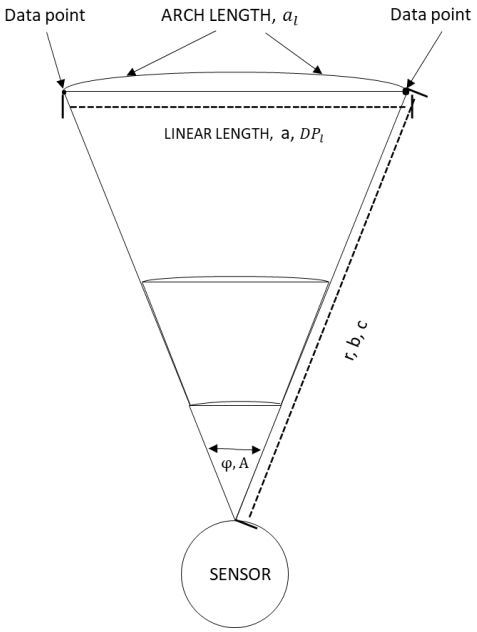


Figure 3.9 Location of variables

The Law of Cosines (Equation 3) was simplified to the following equation to find the linear distance between two data points:

Equation 4…….Linear Distance between two consecutive data points

$$DP_l = \sqrt{2r^2(1 - cos(\varphi))}$$

Where: DP_l = linear distance (cm)

The best average number of points per obstacle was 9.7 data points per foot. The settings for this calculation was for an obstacle that was 500 cm away at a 1 Hz rotating speed and 1000 Hz sampling rate. This was calculated using the following formula.

Equation 5.....Number of data points per one foot

$$DP_n = \frac{f_c}{DP_l}$$

Where: DP_n = average number of data points per one foot

f_c = conversion factor from foot to cm

The results from all calculations on the SCANSE Sweep can be seen in APPENDIX A.5 Sensor Calculations. The highest angular resolution occurred at 1 Hz spinning frequency and 1000 Hz sampling rate. The SCANSE Sweep's lowest angular resolution of 7.20° occurred at 10 Hz spinning frequency and at 500 Hz sampling rate. The average distance between data points at this settings, 20 meters away would be 251.33 cm or about 8 ¼ feet between data points.

Using Equation 1 to find the angular resolution and Equation 2 and Equation 4…….Linear Distance between two consecutive data points to find arc length and linear distance between two consecutive points. In a calculation using the highest angular resolution of 0.36° for the SCANSE Sweep, and 20 meters away for the obstacle. The arc length between two consecutive data points would be 12.57 cm or about five inches between each data point and the linear distance would be 12.57 cm. Therefore, the arc length and linear distance between two points will technically be equivalent.

3.4 Results and discussion

3.4.1 *SCANSE Results and Discussion*

The SCANSE Sweep completed only one run during the data gathering session. It malfunctioned during the second test of the SCANSE. More data was needed to be collected in order to draw complete conclusions. Figure 3.10 shows the raw data plotted using MATLAB code. The blue circles are data points, green circles indicate the placed obstacles, and the orange circle shows the ground interference and red arrows point to some of the false positives. The center of the graph is the sensor. The plotting code can be seen in APPENDIX A.3 Plotting Data code (MATLAB). Two of the four obstacles that were placed were detected, along with false positives by the sensor. Obstacles 5 and 11 were detected, 7 and 12 were not detected in the green configuration.

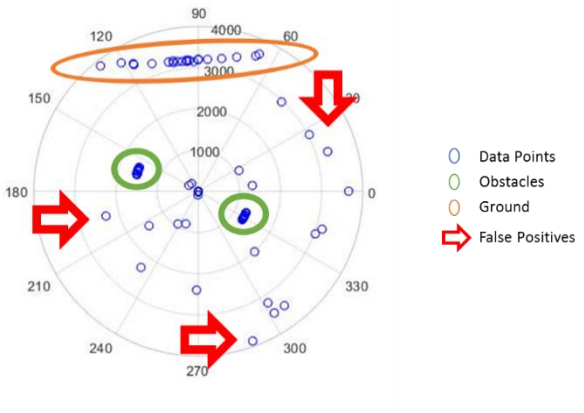


Figure 3.10 SCANSE Sweep plotted results for one data gathering session of the green obstacle location.

From this one test and plotting the data points, two possible sources of error could have been:

1) The obstacles were not in the Field of Vision (FOV) of the sensor

2) The sensor missed obstacles because of the angular resolution of the sensor

3.4.2 *RPLIDAR Results and Discussion*

The unsorted data was first plotted using the code in APPENDIX B.3 Plotting Data code (MATLAB). An example of the plotted RPLIDAR data can be seen in Figure 3.11. Data was manually sorted to contain only detected obstacle data to use for data analysis. When preprocessing the data it was noticed that data contained false positives in between two positive data points (Table 3.2). When the data was used for statistical analysis the false positives were omitted to give a better reflection of how the sensor performed. The data in the green rows are positive data points and were used in the statistical analysis of the data. Rows in red were omitted. It was noticed that the signal strength was zero when no distance data was available.

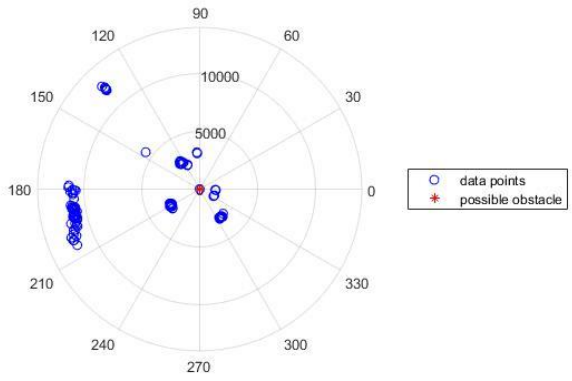


Figure 3.11 Plotted RPLIDAR data

Table 3.2 Data gathered during testing in the shade at a 2 meter distance red rows were eliminated and green rows were kept

Angle	358.719	Distance	2055.5	Signal Strength	15
Angle	358.406	Distance	0	Signal Strength	0
Angle	358.406	Distance	2045	Signal Strength	15
Angle	358.375	Distance	0	Signal Strength	0
Angle	358.359	Distance	2043.75	Signal Strength	14
Angle	358.313	Distance	2040.75	Signal Strength	13
Angle	358.266	Distance	0	Signal Strength	0
Angle	358.234	Distance	2026.75	Signal Strength	13

It was assumed that the sensor reported a 0 distance when a false positive occurred. The Final RPLIDAR Test had statistics done with and without false positives to determine the statistical difference between data that included false positives and the data that did not include false positives.

The beginning runs with the RPLIDAR were successful and occupancy maps could be made (Mathworks, 2019). The path planning algorithm would choose a different path every time it ran (Figure 3.12). On the left side, data points captured by the LIDAR are inserted into a binary occupancy grid to show where obstacles were located. In the middle and right pictures, the red dots are gathered data points or objects detected and the objects are increased in size and a path was planned around the obstacles creating a probabilistic roadmap. The blue lines are possible paths a robot could take to avoid the obstacles. The orange line was the route chosen to get around the obstacles. The path

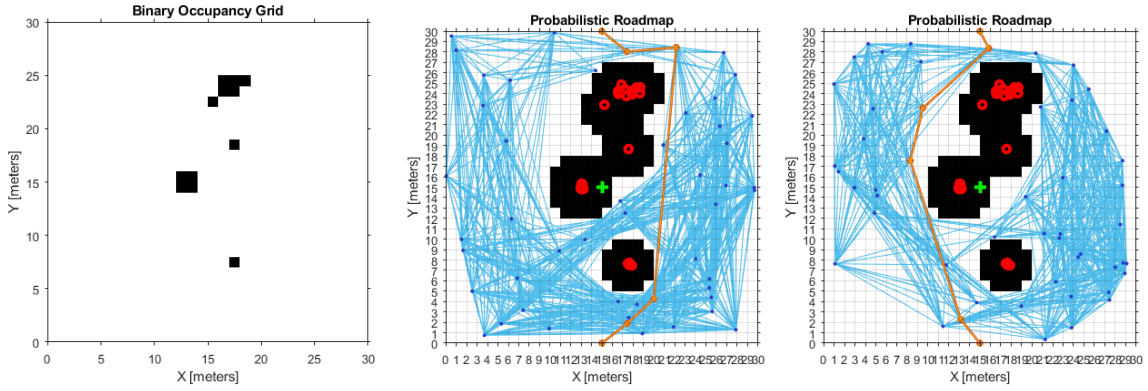


Figure 3.12 Occupancy Maps and two different path planning algorithms.

planning algorithm did not choose the same path every time so, it was thought that a minimization algorithm, like the genetic algorithm, could be tried to find the shortest path but it was out of the scope of this research so that was not pursued. Obstacle locations and distances can be seen in APPENDIX B.1.1 Occupancy Map Setup.

In the Straight Line RPLIDAR Test, the sensor was able to detect obstacles 1 – 9 and 11 on the white side and obstacles 1 – 7 on the brown/black side indicating that the sensor had more than an eight meter range in the shade Figure 3.13. This supported the hypothesis that white obstacles were more likely to return sensor light packets than the brown/black obstacles. This test also showed that the RPLIDAR was inaccurate to reliably detect small obstacles. The average error was -12.56 cm on the brown/black side and -16.29 cm on the white side. Locations and obstacles can be seen in APPENDIX B.1.2 Straight Line Setup.

Figure 3.13 Obstacle setup on the brown backside in the Straight Line Test.

In the Final RPLIDAR Test, the statistical analyses were completed by the UNL Statistics Help Desk.

When obstacles were in the sun 42.10% or 570 data points were false positive readings. When the obstacles were in the shade 41.65% or 1,251 data points were false positive readings.

When comparing the number of data points collected, analysis combines both the front and back sides and lighting positions (Figure 3.14). Statistical analysis found that the proportion of false positives increases from 1 – 4 meters and then plateaus around a proportion of .5 at distances over 5 meters

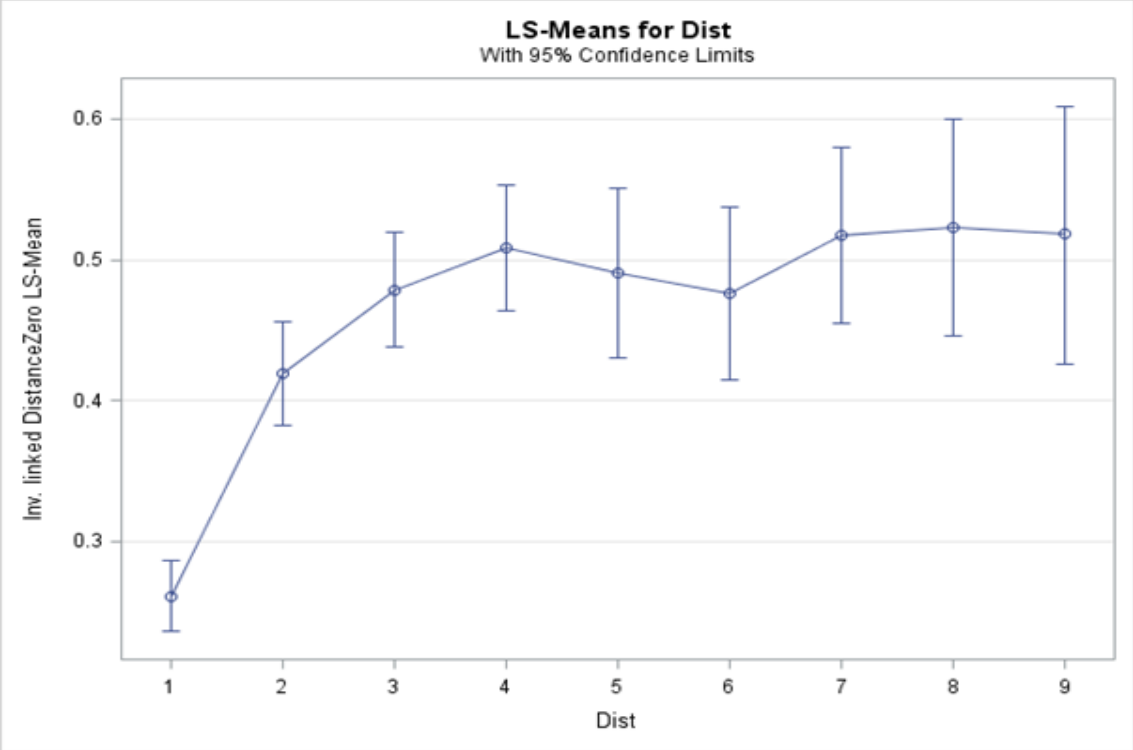


Figure 3.14 The proportion of false positives as distance increases and in both lighting positions. The 95% confidence interval bounds are shown.

It was found that of the data points gathered on the back side and 45.59% of the data points recorded on the white side would be false positives. With a p-value of 0.4579, there is no significant interaction between the number of false positives and the color of the obstacle (Figure 3.15).

This analysis showed there was only a statistically significant difference in the number of false positives collected when one meter results were compared to every other distance (all p-values were <.0001). It was concluded that false positives would be removed as to not skew statistics.

There were 40 sample points gathered and a summary table was created for each point. The summary tables provide the mean, standard deviation, minimum and maximum values for accuracy and measured distance. This table also gives the number of data

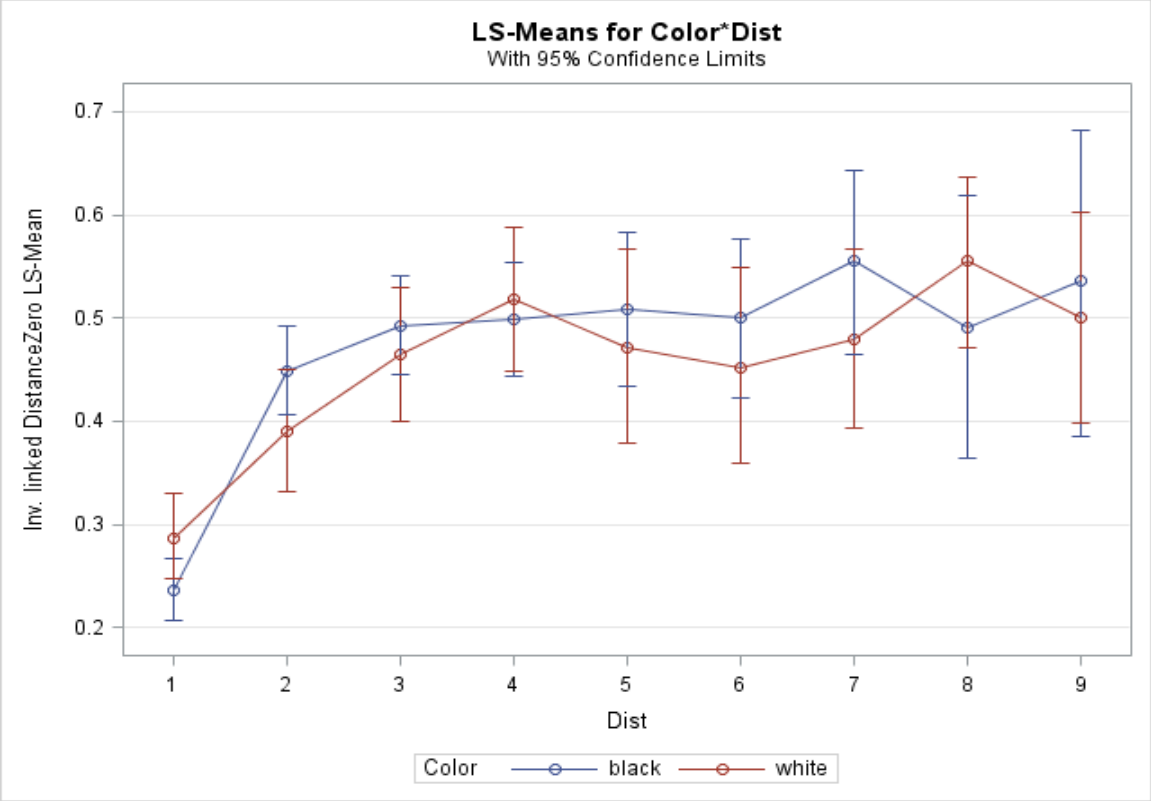


Figure 3.15 The proportion of false positives between distances and obstacle colors with a range of 95% confidence interval bounds for the number of false positives

points collected. The highlighted words above the table gives information on the lighting condition of the obstacle in "weather". The color of the obstacle and the distance in meters the obstacle is away. An example of sample point data summary can be seen in Table 3.3. These summary tables can be seen in APPENDIX B.7 Sample Point Summaries.

Table 3.3 Sample point data summary Obstacle in the shade obstacle color is black at a distance of 1 meter

Weather=Shade Color=black Dist=1

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	**Accuracy**	283	-39.64	12.14	-75.00	-15.50
MeasuredDistance	**MeasuredDistance**	283	1039.64	12.14	1015.50	1075.00

Analysis was done on the number of data points collected for each color and distance. There was a significant difference also found between the number of data points collected and the distances the obstacles were from the sensor (Figure 3.16). Obstacles closer to the sensor had more data points than those further away. There was a statistically significant difference in the number of data points collected between the front and back sides when the distances were one (p-value of 0.0205), three (p-value of 0.0322) and four (p-value of 0.0445) meters away.

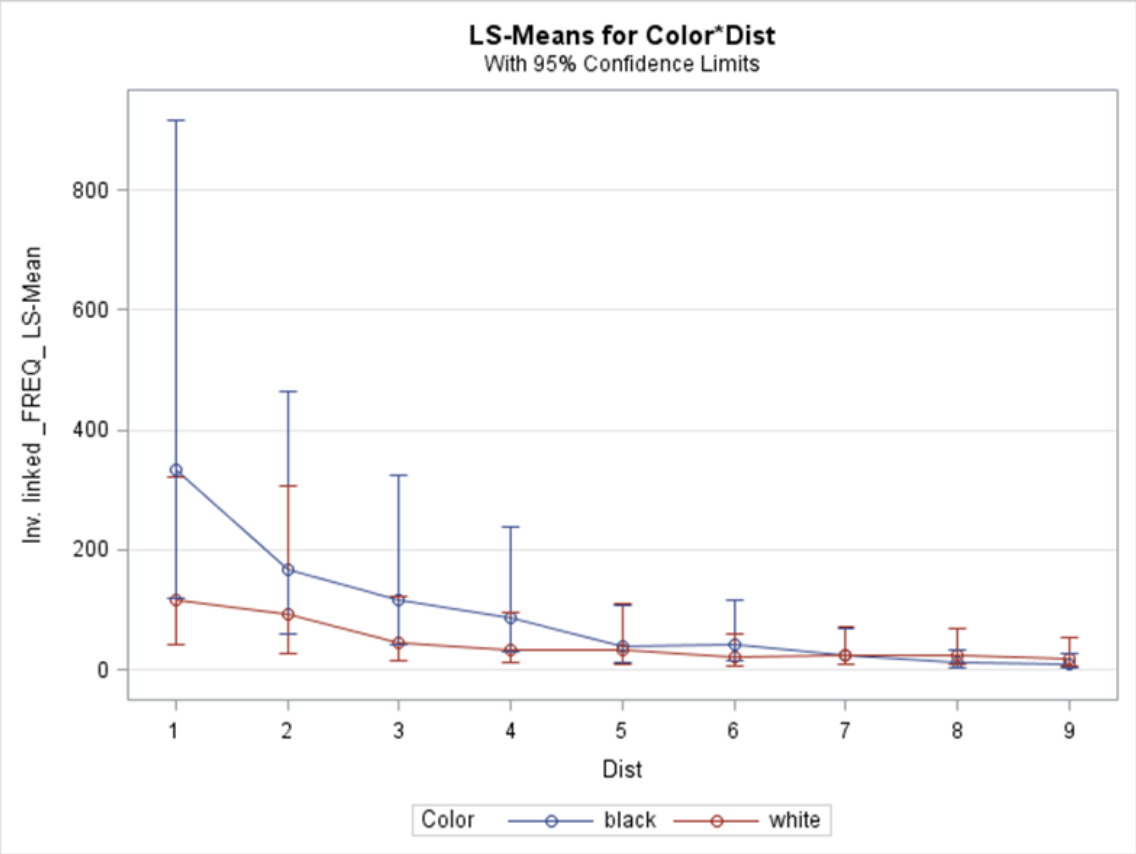


Figure 3.16 The average amount of data points collected and 95% confidence interval bounds with the color and distance being the set variables.

The accuracy of the RPLIDAR was found by taking the actual distance in millimeters then subtracting the distance measured by the RPLIDAR. The closer the number is to

zero the better the accuracy. While the RPLIDAR was in the shade and obstacles in the sun, black/brown obstacles at six meters or less were within -100 mm. But, when obstacles were six meters or further, the black/brown side had a lower accuracy than -200 mm (Figure 3.17). The white obstacles were between -100 mm and -200 mm throughout the test.

When the same comparison is made for the obstacles that were in the shade and RPLIDAR was in sun, it was noticed that the accuracy was improved. For both the white and brown/black obstacle colors, the accuracy is less than 100 mm at obstacles less than

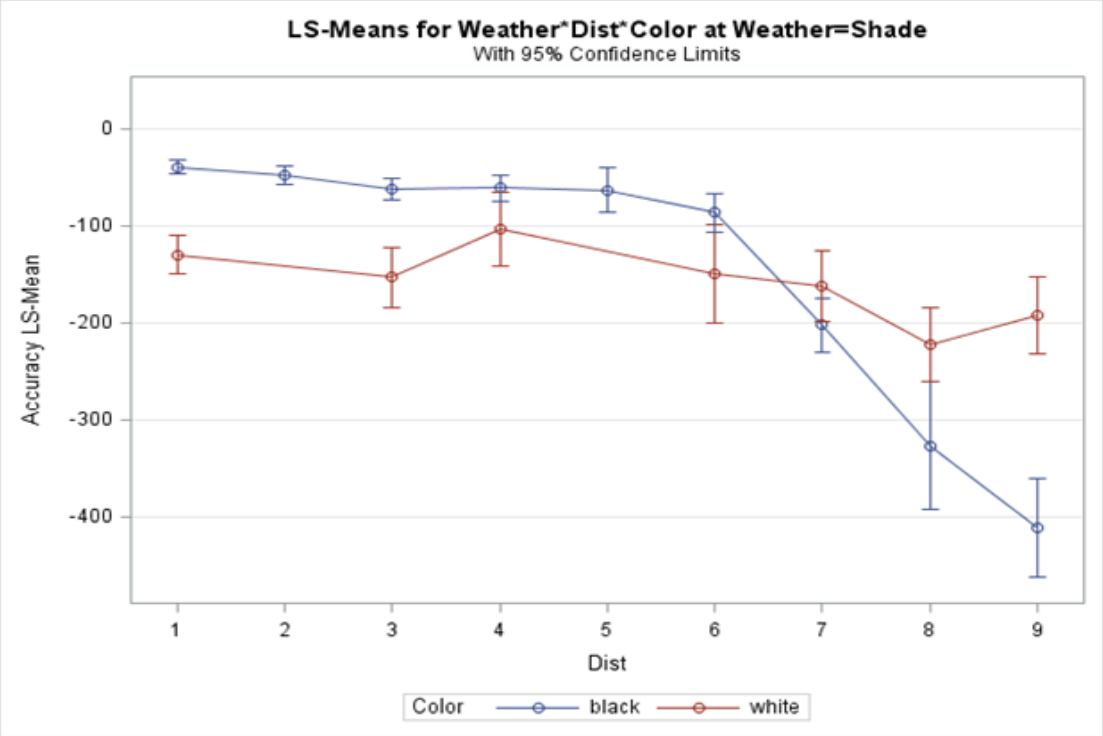


Figure 3.17 A graph showing the LS-Means and 95% confidence interval bounds between the two color obstacles at each distance. It is noticed that at distances < six meters away the brown/black obstacles are more accurate and distances > six white obstacles are more

nine meters away. Much like the previous analysis the brown/black color obstacle is more accurate at closer distances. Brown/black obstacles less than 8 meters give a more accurate reading than white obstacles. A graph comparing the accuracy at each distance

and lighting condition when the RPLIDAR is in sun can be seen in Figure 3.18. The accuracy is only greater than 100 mm at distances greater than eight meters.

Analysis was done to see if there was a factor that affected the accuracy of the sensor.

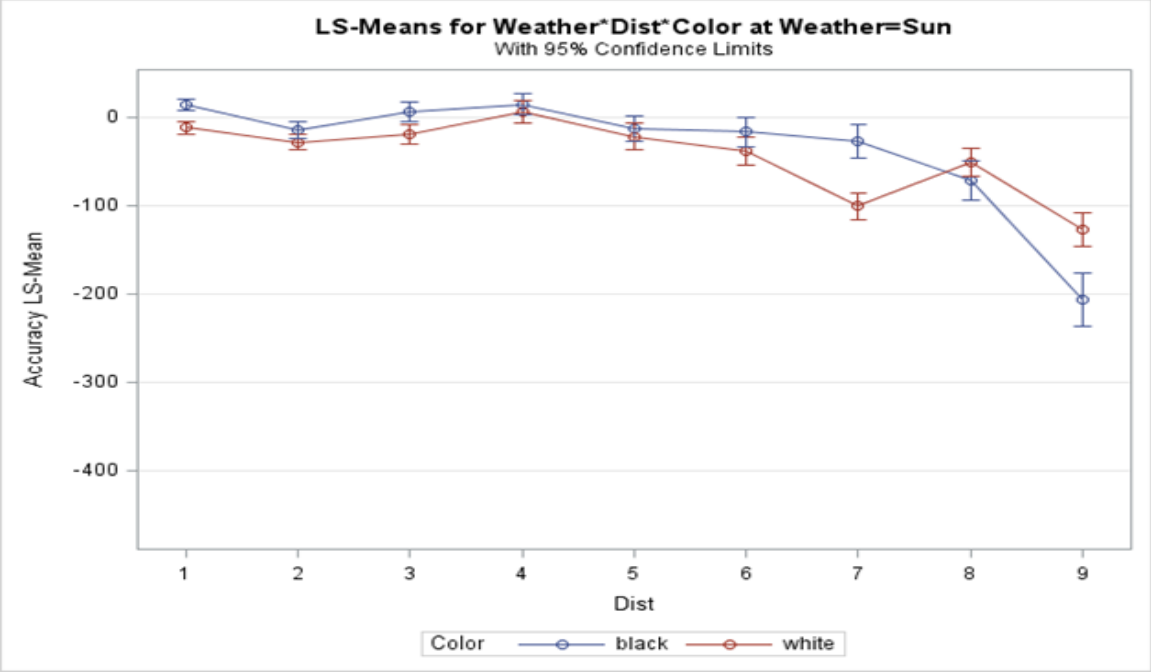


Figure 3.18 A graph showing the LS-Means for accuracy when the RPLIDAR in the sun and both color obstacles. A glance at the graph shows accuracy staying under 100 mm untl it reaches 9000 mm

When distance and color were held constant and the lighting condition was changed between shade and sun, a statistically significant difference was found between every color and distance (all p-values <.0043).

3.4.3 *Hokuyo URG-04LX-UG01 Results and Discussion*

The result of the test was unsatisfactory for detecting fence posts. The Hokuyo URG-04LX-UG01 was unable to detect even the closest obstacle at 141.455 cm away. The calculations from Equation 4…….Linear Distance between two consecutive data points

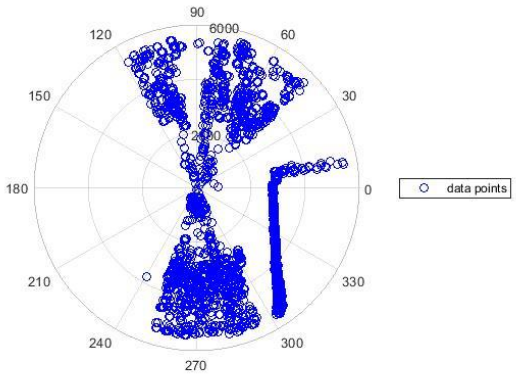


Figure 3.19 Plotted data from Test 6

showed that this sensor could detect obstacles less than .89 mm wide at 141.455 cm away. It was noted in the manual that it was not sunlight resistant, but it was unknown if ambient sunlight was a factor even in the shade. An example dataset from the test can be seen in Figure 3.19. The sensor detected a wall past its range.

3.5LIDAR Conclusions

It was found that LIDARs could possibly be a capable sensor for detecting large solid objects like round bales or tractors but would not be able to detect large truss structures like pivots. Calculations from developed equations from the data, showed that angular resolutions would not be small enough to detect a pivot's truss structure at a distance far enough away to avoid it with off-the-self LIDARs. High Resolution LIDARs do have a small enough angular resolution to detect pivots at 20 meters however, they were out of

the scope of this project to pursue because they were too expensive to obtain. Use of cameras to detect and identify field obstacles was pursued next.

3.5.1 *SCANSE Conclusion*

After performing the calculations to find the angular resolution and analyzing the limited data collected, SCANSE could have been used for obstacle detection but, with limitations. It would only be able to detect large obstacles at closer distances and hence using SCANSE in field condition is not recommended.

3.5.2 *RPLIDAR Test Conclusions*

A total of three unique tests were performed for this sensor. In the Occupancy Map Tests an occupancy map with a path planned around the obstacles was successfully created. This was not in the scope of the study, so it was not pursued further. In the Straight Line Test, the RPLIDAR was not able to identify small changes with error of 12 cm or greater, depending on the color of the obstacle. In the Final RPLIDAR Test the sensor had a proportion of 0.5 in the number of false positives collected to total points collected. This test also revealed that the lighting of obstacles had an impact on how well the sensor detects obstacles.

3.5.3 *Hokuyo URG-04LX-UG01 conclusions*

The sensor had the specifications desired to detect small obstacles at short distances. However, it was evident from the data that it would not work for the requirements needed in the outdoor conditions. It is confirmed that this would only work indoors.

Chapter 4 Obstacle Detection and Identification Algorithms for Camera Images

4.1 Introduction and Objectives

It was shown in the previous chapters that low cost LIDARs had challenges because of the low angular resolution between data points. RGB cameras were explored next for being able to detect agricultural obstacles. This chapter:

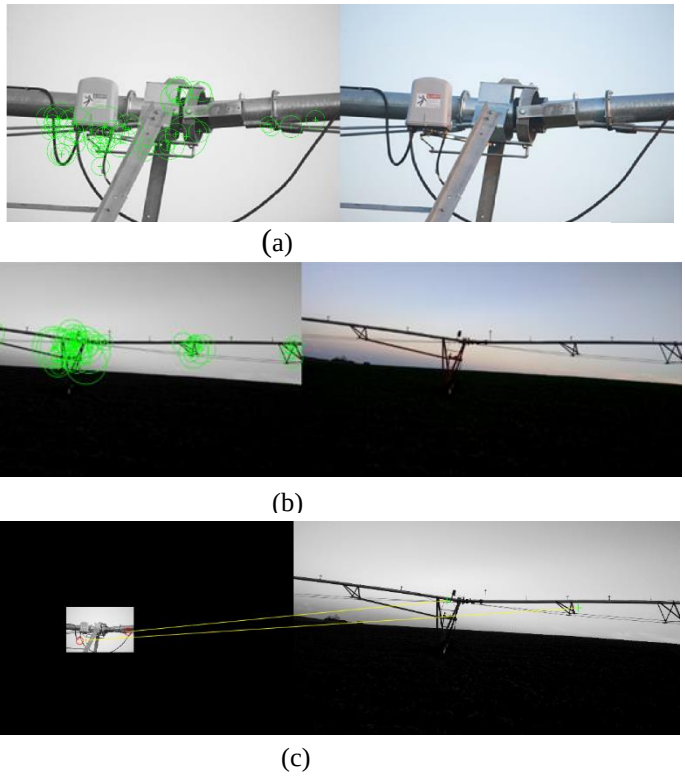
1. Explores different image processing algorithms and techniques using RGB images for agricultural obstacle detection
2. Evaluates multiple object edge detection algorithms for suitability of identifying obstacles common to agricultural fields based on images (RBG and RGB-D)
3. Discusses the applicability of convolutional neural networks for detecting in-field obstacles based on RGB or RBG-D image processing techniques.

4.2 SURF Feature Extraction and Matching

The first approach to detecting objects in camera images was the Speed Up Robust Features (SURF) Extraction and Matching algorithm. This algorithm is designed to detect close similarities between pictures of the same scene or object (Bay et al., 2006). A grayscale image is loaded initially into SURF. Figure 4.1a shows a reference image and the feature points detected in the reference image. These feature points needed to be detected in the test image. Figure 4.1b is the test image in which the feature points detected are displayed. Figure 4.1c depicts both matched outlier and inlier feature points between the images in Figure 4.1a and Figure 4.1b. Inlier points are positive matches

between the two images and outliers are points that match but are not identical to the points in the reference photo. It is assumed that no inlier points existed between the two photos because no image could be produced of matching inlier points.

SURF Feature Extraction and Matching was able to detect a few similar feature points in both images. It was later learned that an almost exact copy of the test image had to be matched with the input image in order to be detected.



(a)

(b)

(c)

Figure 4.1(a) Extracted features out of the grayscale reference image. (b) Extracted features out of the grayscale test image. (c) Matching the extracted features in reference photo to features in the test photo. Only ouliers are displayed because no inliers were preent

Even though this method looked promising, it was quickly eliminated because it did not adapt well to new images. Script can be seen in APPENDIX D.1 SURF Feature Extraction and Matching Script (MATLAB).

4.3 Edge Detection

Edge detection of the structural components and identifying shapes was recognized as another approach to identifying a pivot. Seven unique edge detection algorithms found in MATLAB were tested to find which method gave the most useful image data. The seven

methods were approxcanny, Canny, log, Prewitt, Roberts, Sobel, and zerocross (MATLAB, 2018c). Each method gives a unique result of edge detection. (Script can be seen in APPENDIX D.2 Edge Detection Script (MATLAB))

An algorithm found on MathWorks File Exchange called “Shape Recognition” (Samieh, 2016) was used to identify shapes. Shapes could be identified as a circle, square or other shape. This Script can be seen in APPENDIX D.3 Shape Identification algorithm (MATLAB).

It was forewarned in the documentation that this algorithm would have difficulties recognizing shapes with differing aspect ratios of the object (Samieh, 2016). Many of the shapes found could not be identified as either a circle or square because of the variability of camera angles and how the edge recognition algorithms extracted shapes. Figure 4.2 shows the results from the Sobel edge detection algorithm with the shape recognition algorithm. Outputs from the other edge detection methods can be seen in APPENDIX D SURF Feature Extraction and Matching and Edge Detection. The code for this algorithm can be seen in APPENDIX D.4 Edge detection and Shape Identification Algorithm (MATLAB).

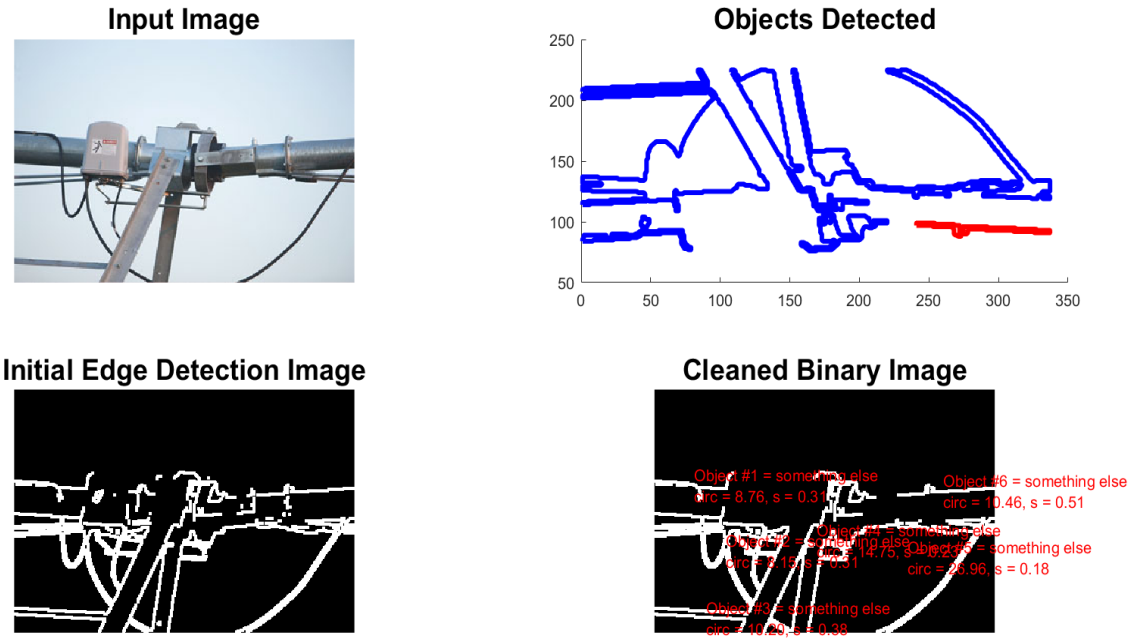


Figure 4.2 Sobel edge detection with the "Shape Recognition" algorithm. The initial image can be seen in the top left picture. The bottom left picture is the result of using the Sobel edge detection method. The top right depicts the objects the algorithm detected based on the input binary image. This image is flipped compared to the input image. The bottom right gives scores on what the shape of the object is .

4.4 Combination of SURF Feature Extraction and Matching and Edge Detection Algorithms

The idea of merging SURF Feature Matching and Extraction and Edge Detection with shape identification algorithm seemed appropriate as a method to help in the identification of obstacles. The idea was to use Edge Detection with SURF Feature Extraction and Matching, by matching edges in pictures. The concern about merging of these two methods was that features extracted would be hard to differentiate in the two images because of the lack of grayscale in the image.

In the first run of merging of the two different algorithms, the volume of positively matched points including outliers suggested that the grayscale concern was not a factor

Figure 4.4). However, the number on inlier points was significantly less. Every inlier point found one feature point that matched (Figure 4.3).

Only having one inlier point proved that the combination of the two algorithms was not an effective way to detect and identify obstacles. The MATLAB script can be seen in APPENDIX D.5 SURF Feature Extraction and Matching and Edge Detection (MATLAB).

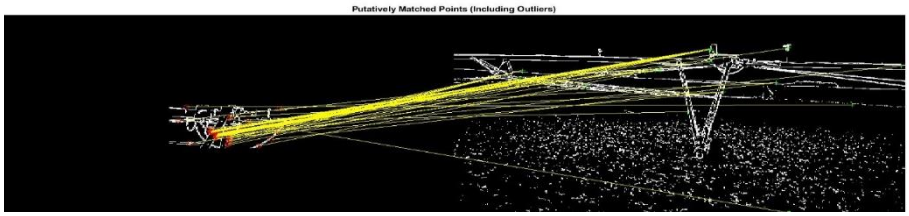


Figure 4.4 Positively matched SURF Points (Including Outliers)

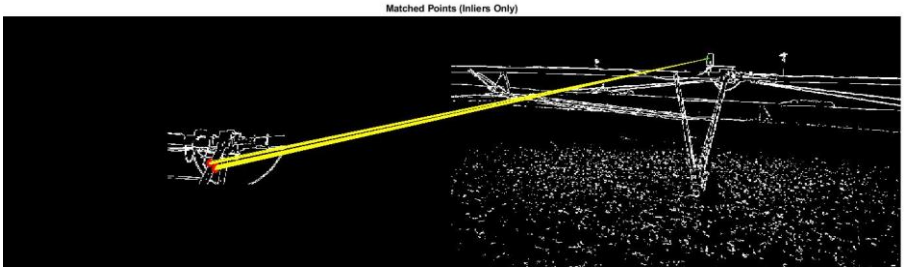


Figure 4.3 Positively matched points (Inliers only)

4.5 Conclusions from SURF Feature Extraction and Matching and Edge Detection

It was concluded that SURF Feature Extraction and Matching, edge detection with a shape identification algorithm would not suffice for pivot obstacle detection. The algorithms could not adjust to the different orientations and aspect ratios of the objects. Object identification using the Viola – Jones algorithm and a Convolutional Neural

Network (CNN) were identified as possible object identification algorithms using machine vision.

4.6 Object Identification

The cascade object detector uses the Viola-Jones algorithm to detect people's mouth, ears, nose, face or upper body (David & Viola, 2001). This algorithm can be trained to identify other things but, the objects cannot have substantial aspect ratio changes. A single detector cannot handle every orientation and cannot handle more than one class of objects so more than one detector has to be used (Mathworks, 2018b).

Tractors and pivots were trained through the cascade object detector that uses the Viola – Jones algorithm. Code for the cascade object detector that uses the Viola – Jones algorithm can be seen in

APPENDIX E Viola-Jones Algorithm Script (MATLAB).

4.6.1 *Cascade Object Detector and the Viola-Jones Algorithm*

4.6.1.1 Tractors

The first obstacle to be trained was a tractor. The picture quality of the training image and quantity of tractor images were unknown. Forty-two pictures of tractors were used. Each image had to be labeled. Labeling is the process of drawing a region of interest (ROI) box around the object to be identified.

4.6.1.2 Pivots

The pivot was trained using video labeling and extracting the images from the video. The video used was recorded in a field located near Lindsay, Nebraska. Picture quality and quantity needed for training was unknown. 67 pictures of pivots were used.

4.6.1.3 Results and Discussion

The Viola – Jones algorithm could identify a tractor. However, the bounding box could not encompass the entire tractor and could not detect a tractor in clutter. The best detection found of a tractor is shown in Figure 4.5 Viola-Jones algorithm detecting a tractor with no clutter. Yellow *bounding box does not surround the entire tractor.* This image features a single tractor with no clutter around it.

A separate detector was trained on the image data of a pivot. After training, the detector was not able to clearly identify a pivot. There was a significant number of false positives in the test image (Figure 4.6). The Viola-Jones research and experimental training objects confirmed that the Viola-Jones algorithm would not be pursued due to its performance and inadaptability to classify an array of objects.

Figure 4.5 Viola-Jones algorithm detecting a tractor with no clutter. Yellow bounding box does not surround the entire tractor.

Another possible solution to object detection and identification is with a CNN.

Figure 4.6 Viola-Jones algorithm detecting a pivot in the middle of a soybean field. Yellow bounding boxes do not surround the entire pivot but, false positives.

4.6.2 *Convolutional neural networks (CNN)*

CNNs are Deep Learning algorithms that can be used for object detection and classification. CNNs were pioneered in 1994 by Yann LeCun (Yann Lecun & Bengio, 1994). LeCun's paper based the neural network on three features that were found in the mammalian visual cortex: local connections, hierarchy of different clusters of neurons, and spatial invariance. LeCun named his network LeNet5 (Lecun et al., 1998). LeNet5 uses a sequence of 3 layers: convolution, pooling and non-linearity

From 1994 to 2012, work with CNNs was non-existent (Ujjwalkarn, 2016). During this time data became increasingly easier to obtain and computing power increased. In 2012, Alex Krizhevsky designed a neural network that won the difficult ImageNet Competition by designing a deeper and wider version of LeNet. In this CNN there are 25 layers compared to three in LeNet (Mathworks, 2018e).

AlexNet is the cause for the widespread application of CNNs today. The layers are a combination of convolution, nonlinearity function and pooling. The number of layers in a network can be as small as tens or as many as hundreds of layers deep (Mathworks, 2018a).

Major companies use neural networks. Facebook uses them in photo recognition algorithms, Google uses neural networks for photo search, and Amazon uses them for product recommendations (Deshpande, 2016). Figure 4.7 illustrates the general flow of a CNN.

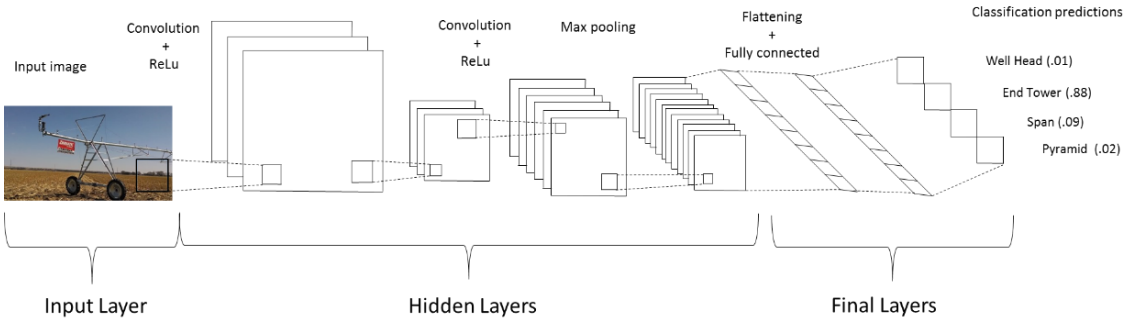


Figure 4.7 Layout of the entirety of a CNN with a pivot as the input image

4.6.2.1 Beginning layer in a CNN

A CNN is made of layers that are stacked on top of each other. Each layer's output is the next layer's input. The first layer is the input layer and followed by the hidden layers (aka: feature learning layers). The hidden layers are composed of three layers:

convolution, an activation function and pooling. (The convolution layer is always followed by an activation function layer so, sometimes they are considered one layer.) The last layers of the convolutional network are called the final layers. The final layers consist of flattening, fully connected and SoftMax (classification). These layers are like parts of a sandwich (Figure 4.8). The input layer is the top bun, hidden layers are the meat, cheese and toppings that you can put into any order and the final layers are the bottom bun.

4.6.2.1.1 Input

The input layer is the image coming into the neural network needing to be classified. The input layer has dimensions [width x height x depth]. If the example was an RGB image the width of 32 pixels, the height of 32 pixels and the depth might be 3 for the three matrices of color. The dimensions would be [32 x 32 x 3]. The input layer needs to be divisible by two, many times (Doukkali, 2017). Different CNNs require different sized inputs. After the input layer, the feature learning layers begin. Each layer consists of neurons, weights and biases.

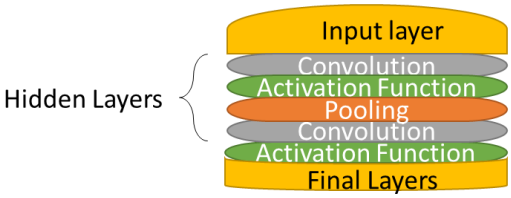


Figure 4.8 Layers in convolutional neural network

4.6.2.1.2 Neurons

Neurons are cells in a matrix. Neurons take in values and the weight assigned to them from the previous layer, sum the values, performs an activation function (discussed later) then outputs the data to the next layer.

The mathematical representation for a neuron is:

Equation 6.....Mathematical Representation of a Neuron

$$\sum_{n=1}^{r}((x_n * w_n) + b)$$

Where: x_n = incoming value

w_n = weight

b = bias

An illustration of a neuron can be seen in Figure 4.9

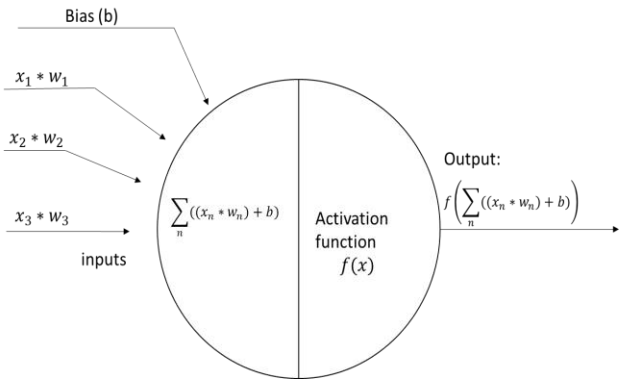


Figure 4.9 Layout of a single neuron in a CNN

4.6.2.2 Hidden Layers (Feature Learning)

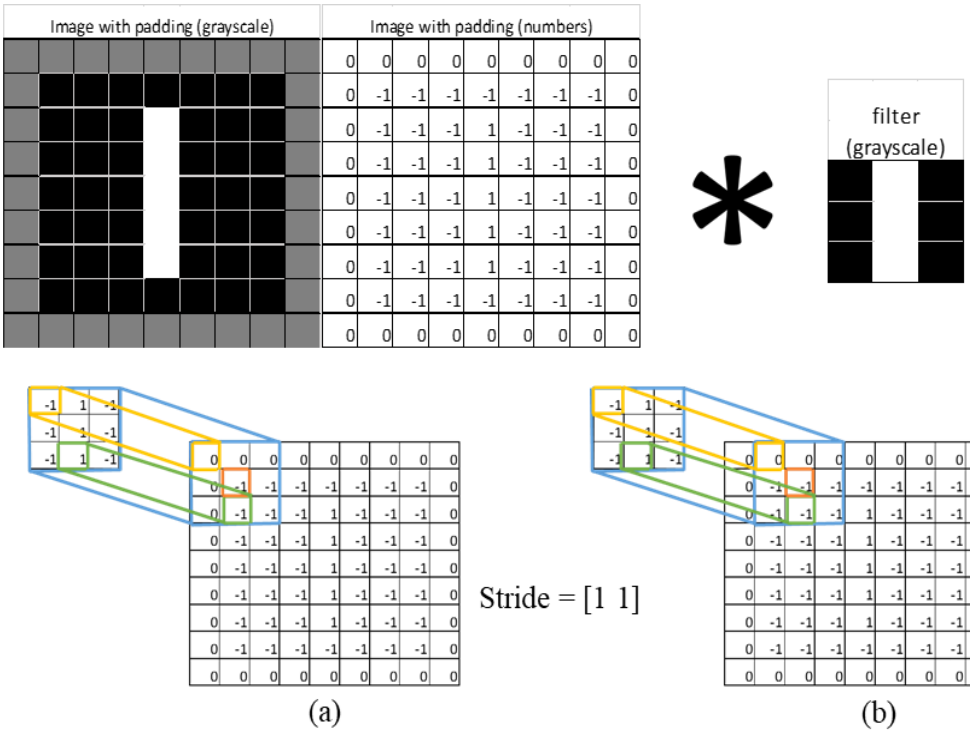
These layers can be put in specific sequences to extract certain features. The layers are convolution, an activation function layer or regularization layer, and pooling. These layers can be repeated as many times as the designer of the CNN would like.

4.6.2.2.1 Convolution

Convolution is the first layer after the input layer and does most of the heavy computation in a CNN. Convolution uses filters to pull out important features in an image. The number of filters adds depth to the convolution layer.

In convolution, a dot product is taken of the filter and every pixel value. Depending on the CNN, the dot product is divided by either the size of the filter for an average value or the product is left untouched. The dot product is the new value after convolution.

An example of convolution is given in Figure 4.10. Convolution starts by comparing the first pixel in the image to the top left value in the filter or kernel (yellow box in filter and yellow box in the image). In the example, the pixel value is not averaged for each convolution, the stride is set to [1 1] ([horizontal vertical]) and padding of [1 1 1 1] [top bottom left right] (Mathworks, 2018d).



Yellow: -1 * 0 = 0

Green: 1 * -1 = -1

Total (Blue) = (0+0+0+0-1+1+0-1+1) =0

Yellow: -1 * -1 = 1

Green: 1 *1 = -1

Total (Blue) = (0+0+0+1-1+1+1-1+1) = 2

Figure 4.10 Example of a convolution. A simple picture of a straight line. With a straight line filter. The input image with [1 1 1 1] is in (a) resulting matrix (b) and stride [1 1].

The output size of a convolution is:

$$W2 = \frac{W1 - F + 2P}{S} + 1$$

$$H2 = \frac{H1 - F + 2P}{S} + 1$$

$$D2 = K$$

Where: [W1 X H1 X D1] = are input volume size

[W2 X H2 X D2] = are output volume size

F = Width of the filter

S = Stride

P = Amount of padding used

K = Depth (number of filters)

The stride is how many pixels the filter is moved between convolutions. (Stanford, 2018). The filter size used during convolution is a hyperparameter or, a parameter that can be controlled by the designer. Strides are commonly 1 or 2 in magnitude (Figure 4.11). Filters can be as small as [1 1] or larger size filters are [7 7] or bigger.

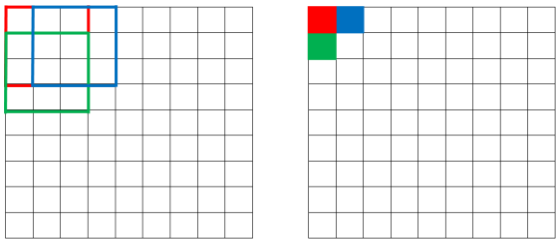


Figure 4.11 Illustration of an image [9 9], filter size [3 3] and a stride of [1 1].

The result of convolution from the example can be seen in Figure 4.12. A color gradient is added to help see the gradients in the numbers. Weights and biases are also a part of the network. For this example, weight is a consistent 1 and bias was -4. For every neuron a weight is assigned to its output for the next layer. These weights and biases are computed during training. Figure 4.13 is the result after weights and biases are added to convolution. The color gradients remain the same, but this bias will help in the next layer.

0	2	0	4	0	2	0
0	3	-1	7	-1	3	0
0	3	-3	9	-3	3	0
0	3	-3	9	-3	3	0
0	3	-3	9	-3	3	0
0	3	-1	7	-1	3	0
0	2	0	4	0	2	0

Figure 4.12 The result of convolution between the test image and the chosen filter

4.6.2.2.2 Activation function

There are multiple activation functions to normalize convolutional data and are nonlinear in nature. The activation function takes the previous layers output, applies a mathematical function to normalize the data, then outputs it to the next layer as an input. The activation functions and corresponding graphed equations are in Figure 4.14. Activation functions can be classified into two types of functions: ReLu and non-ReLu.

-4	-2	-4	0	-4	-2	-4
-4	-1	-5	3	-5	-1	-4
-4	-1	-7	5	-7	-1	-4
-4	-1	-7	5	-7	-1	-4
-4	-1	-7	5	-7	-1	-4
-4	-1	-5	3	-5	-1	-4
-4	-2	-4	0	-4	-2	-4

Figure 4.13 The resulting matrix after a bias of -4 is implemented.

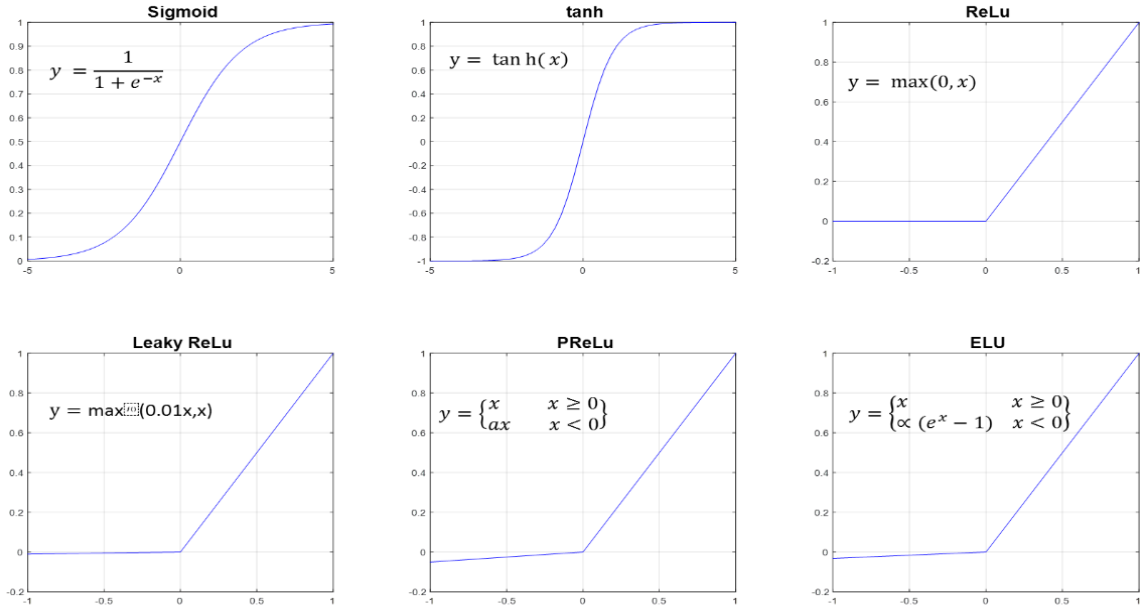


Figure 4.14 Graphical representation of the activation functions

4.6.2.2.2.1 Non-ReLu functions

The Sigmoid function is an example of a non-ReLu activation function. The sigmoid function keeps the number always between 0 and 1. However the Sigmoid can be computationally expensive because of the exponential function (Jadon, 2018). Sigmoid functions have saturated gradients, a vanishing gradient problem and have slow convergence (V, 2017) . Vanishing gradients can cause the algorithm to stop training by not being able to learn or becomes slow. The sigmoid function is defined as:

Equation 7.....Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

The hyperbolic tangent function has a range between -1 and 1 and is centered on zero. Like the Sigmoid function the hyperbolic tangent can develop saturated gradients and vanishing gradients. The hyperbolic tangent is defined:

Equation 8……..Hyperbolic Tangent

$$\tanh(x)$$

The Sigmoid and tanh should not be used because of the vanishing gradient problem; that causes the network to have a lower accuracy and poorer performance.

4.6.2.2.2.2 *Rectified Linear Units (ReLu)*

ReLu is the most used activation function because of its simplicity during backpropagation and is not computationally expensive. The main formula for ReLu is:

Equation 9…….ReLu

$$max(0, x)$$

In ReLu a comparison between the neuron value and 0 is made and the higher value is the new neuron value. Figure 4.15 shows what ReLu does to the example convolution. When a neuron gets a zero value from the previous layer that neuron becomes "dead". This problem is known as the "dying ReLu" problem, meaning no information is passed through that neuron to the next layer. The zeros in Figure 4.15 are considered dead neurons. Alternative equations are proposed to fix the dead neuron problem (Liu, 2017). The first equation is leaky ReLu:

0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.4	0.0	0.0	0.0
0.0	0.0	0.0	0.6	0.0	0.0	0.0
0.0	0.0	0.0	0.6	0.0	0.0	0.0
0.0	0.0	0.0	0.6	0.0	0.0	0.0
0.0	0.0	0.0	0.4	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 4.15 Result of ReLu on Example 1

Equation 10……..Leaky ReLu

$$max(0.01x, x)$$

In Leaky ReLu the max value between 0.01x and x is the value given to that neuron.

Parametric ReLu (PReLu) is like leaky ReLu but it does not have a predetermined slope (Liu, 2017). Instead the slope is a parameter (a) that is set by the CNN during training. The formula for parametric ReLu is the following:

Equation 11……..……...Parametric ReLu

$$\begin{Bmatrix} x & x \geq 0 \\ ax & x < 0 \end{Bmatrix}$$

Another variation to counter the "dying ReLu" problem is Exponential Linear Units (ELU). However, ELU is computationally expensive because of an exponential function in its formula. The equation is:

Equation 12……..……Exponential Linear Units (ELU)

$$\begin{Bmatrix} x & x \geq 0 \\ a(e^x - 1) & x < 0 \end{Bmatrix}$$

4.6.2.2.3 Pooling

The Pooling layer is inserted into a CNN to reduce the spatial size of the data in the network. A 3 x 3 filter with a stride of 3 is an example of pooling parameters in this process. Max pooling uses the maximum value in the filter to use the replacement value. Other pooling methods like average pooling and L2-norm pooling are used less often. L2-norm pooling is a vector norm of the filtered values (Weisstein, 2018).

In the example found in Figure 4.10, the image size is not devisable by the size of the filter. When this occurs "SAME" and "VALID" methods are used in pooling (MiniQuark, 2018).

In the “SAME” method, there is “0” padding added to the image. In this option “SAME” tries to pad the matrix evenly on all sides.

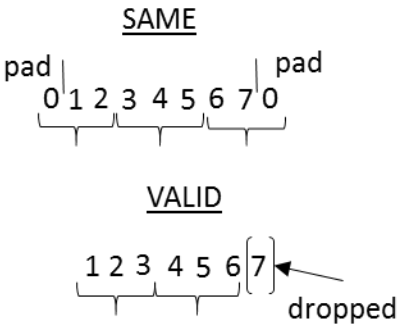


Figure 4.16 Examples of the 2 methods "VALID" and "SAME" filter width= 3 stride = 3

The method called “VALID” is without padding, the right-most columns or the bottom-most rows are dropped and are not used in pooling. Figure 4.16 shows examples of both “SAME” and “VALID” methods.

When max pooling of the output image from Figure 4.10, “VALID” and “SAME” methods are performed on the output of the convolution layer. “VALID” method results can be seen in Figure 4.17 and the “SAME” method results can be seen in Figure 4.18.

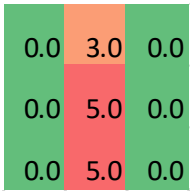


Figure 4.17 Result of "VALID" max pooling in the example

Figure 4.18 Result of "SAME" max pooling

These are the three sub-layers that can be found in the hidden layers: convolution, activation function layers, and pooling. The last layers are the final layers.

4.6.2.3 Final Layers

The final layers are how the CNN relates the hidden layer data to the classification of the image. Several layers can be stacked on top of each other.

4.6.2.3.1 Flattening

This layer only happens once because in this layer it takes the matrices from the last convolution and “flattens” them (Kirill Eremenko, 2018). Figure 4.19 shows an example of flattening. In the example the “VALID” max pooling shown previously was used.

4.6.2.3.2 Fully Connected

The fully connected layer connects the flattening layer to the SoftMax function or to another fully connected layer. Each neuron in a layer is connected to every neuron in the next layer including weights and biases. This is the layer that helps connect numerical data into classification of objects.

Figure 4.19 flattening of the "VALID" max pooling (Figure 4.17)

4.6.2.3.3 SoftMax function

The SoftMax Function is the last step in a CNN and calculates the probability of the object to each class. The highest probability class is then given to the identified object. The sum of all probabilities is equal to one. An example and the SoftMax equation are in Figure 4.20

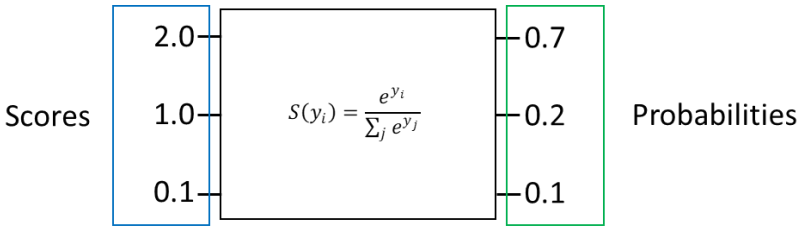


Figure 4.20 SoftMax equation and example

Figure 4.21 shows an example of the final layers of a CNN. Values from the last convolution are first flattened then two additional fully connected layers followed by SoftMax equation to classify the object.

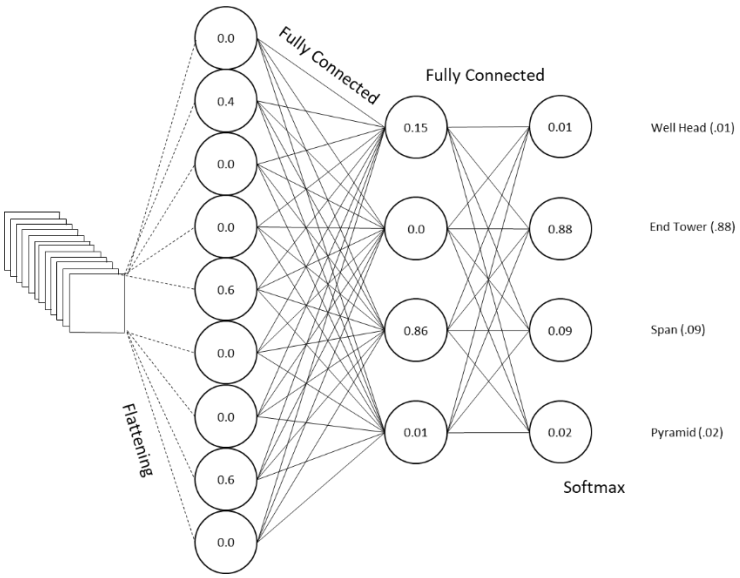


Figure 4.21 The Final Layers of a CNN. Flattening, two fully connected layers and SoftMax to give the probability of that object in the picture

4.7 Conclusions

In this chapter several object detection algorithms were explored but were unsuccessful in clearly detecting the obstacle. This research led to obstacle identification algorithms. The Viola – Jones algorithm was the first algorithm to be used to help identify obstacles. This algorithm was unable to handle different orientations because the algorithm required keeping the aspect ratio the same as the training images. Convolutional Neural Networks were discussed and explained in detail. In the next Chapter, CNNs are trained and used to identify tractors, bales and pivots.

Chapter 5 Convolutional Neural Networks for Tractor, Round Bale, and Center-Pivot Identification

5.1 Introduction and Objectives

Previous chapters experimented with LIDARs and other obstacle identification algorithms. Tractors and bales can possibly be detected by off-the-shelf LIDARs but, are unable to detect pivots because of pivots' truss structure. The angular resolution needed to detect an object the size of the truss was less than provided by the LIDAR sensors tested. Obstacle detection algorithms using machine vision were explored before CNNs because they were less complex. The previously explored machine vision algorithms could detect pivots only under certain conditions. A CNN was previously explained and will be pursued as a method to detect a pivot.

CNNs are used for image classification without localizing where the object is. Object detection is finding the location of objects in an image. In this chapter several CNNs are chosen to act as a feature extractor in the Region based-CNN (R-CNN) architecture to detect and identify a pivot, tractors and bales

The objectives for this chapter are:

1. Compare CNN, R-CNN, and Faster R-CNN architectures for suitability to detect objects common to an agricultural field (e.g., tractors, round bales, and center pivots).
2. Evaluate promising architectures for mean Average Precision and detection time with the goal of real-time obstacle detection using image processing techniques

5.2 R-CNN

CNNs are used for image classification whereas Region based Convolutional Neural Networks (R-CNN) are used for object identification (Girshick et al., 2013). The R-CNN uses a selective search algorithm that proposes 2000 bounding box regions at random and a CNN is performed on each bounding box area (Figure 5.1). The CNN acts like a feature extractor and feeds a support vector machine (SVM) to predict if the object is in that bounding box. The SVM returns four offset values that predict where a better bounding box location may be. It takes roughly 49 seconds for this algorithm to process.

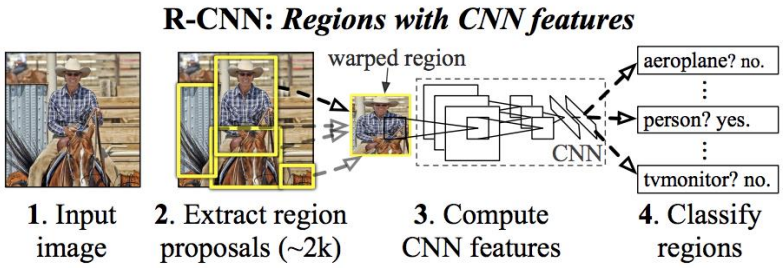


Figure 5.1 Steps of an R-CNN (Lee, 2017)

5.3 Faster R-CNN

The Faster R-CNN is an improvement of R-CNN. It takes about 0.2 seconds per image. This speed is fast enough for real-time object detection. Faster R-CNN contains two networks (Figure 5.2), a region proposal network (RPN) that generates region proposals and a CNN that uses these region proposals to identify objects (Ren et al., 2015). The RPN is used instead of the selective search algorithm which speeds up the processing time (Gao, 2017). The CNN in Figure 5.2, can be any network most appropriate for the application.

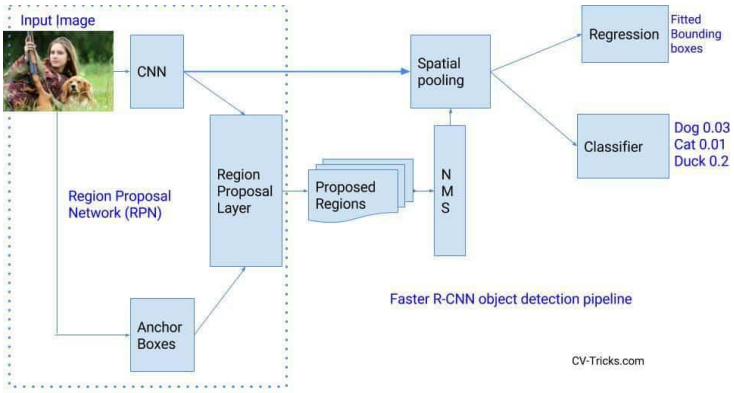


Figure 5.2 Two network layout of a Faster CNN network. The CNN can be replaced with any CNN to best fit the application applied to. (Sinhal & Sachan, 2017)

5.4 Training Faster R-CNN

All training and evaluations were done on a NVIDIA GeForce GTX 1060 6GB GPU in MATLAB 2018b update 2. All code, training outputs and mean Average Precision (mAP) graphs from training are in APPENDIX F Faster R-CNN Training. mAP is a metric used to measure the accuracy of object detectors (Hui, 2018). Values are between zero and one.

Images for tractors and bales were resized to 448 x 448 from various resolution pictures and padded to keep aspect ratios the same (Figure 5.3). After resizing, the images were labeled and augmented. Tractor and bale images were brightened, darkened, sharpened and blurred to create more images for training. After augmentation, 210 images of tractors and 250 images of round bales were available for training.

Figure 5.3 Tractor image 448 x 448 after padding to keep the same aspect ratio. The black part of the image is padding.

Pivot images used for training were taken from three minutes and 24 seconds of pivot videos. MATLAB was able to extract 1,231 images from the pivot videos initially. Pivot images were not augmented to create more images because a satisfactory number of images were already collected.

These pivot videos had an original resolution of 1920 x 1080 but were resized to 448 x 448 pixels and labeled without padding because it was unknown at the time of labeling that keeping the aspect ratio the same was important (Figure 5.4). The four main identifiable components of a pivot are pyramids, towers, spans and end towers (Figure 5.5).

Components of the pivot were labeled because each component is treated as an individual obstacle to be identified.

Four CNNs were tested as feature extractors for the Faster R-CNN: InceptionResNetv2, ResNet 101, Inceptionv3 and GoogLeNet. InceptionResNetv2, ResNet 101 and Inceptionv3 were shown to be the top performing CNNs in Faster R-CNN (Huang et al., 2016, Figure 5.6) and GoogLeNet was chosen at random. The hyperparameters and settings used in Huang's study were not followed because it was unknown how to

(a)

(b)

Figure 5.4 (a) A 1920 x 1080 pivot picture (b) pivot picture resized to 448 x 448 with no padding.

(a)

(b)

(c)

(d)

Figure 5.5 Components of a pivot (a) span (b) pyramid (c) tower (d) end tower

change them which may have made the results unfairly reflect the true capabilities of the networks.

ResNet 101, Inception V3 and GoogLeNet were trained twice in the Faster R-CNN. The best average hyperparameter settings found after experimentation were used for the pivot and combined trainings.

Max Epochs and Initial Learn Rate were the two hyperparameters changed for tractor and round bale training only, Run 1 had Max Epochs set to 1 and the Initial Learn Rate was .01 and Run 2 had Max Epochs set to 2 and the Initial Learn Rate was .005. Max Epochs

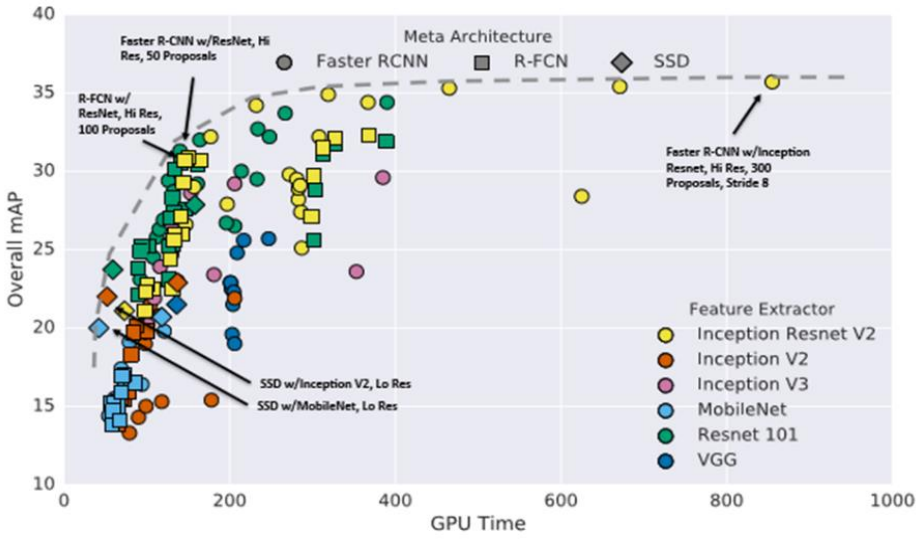


Figure 5.6 Accuracy vs time, the marker shape indicates the meta-architecture and color indficates feature extractor used. (Huang et al., 2016)

is how many times the model trains on a set of images (Anirudh Sharma, 2018). Initial Learn Rate is how fast the algorithm can make changes in weight. Initial Learn Rate will increase both training time and accuracy (Zulkifli, 2018).

The best performing set of hyperparameters from the round bale and tractor training were used in the pivot training. Pivot images were trained twice. Run 1 was the first-time training and Run 2 was the second time training.

The highest mAP hyperparameter settings and CNN were used for the combined image's Faster R-CNN training. The mAP of training was determined to be a higher priority than training time. During training it was noted that 30 pivot images had bounding boxes that extended past the image boundaries. These images were omitted from the training data. It was found through analysis of the original code and output that every fifth image was extracted out of the pivot videos. A small adjustment to the code allowed each video frame to be extracted and used for training for a total of 6,148 or 5x Pivot training data (APPENDIX F.1 Code used to extract images out of video (MATLAB)). Pivotv2 was trained with five times the number of images compared to Pivot Run 1 & Run 2 to observe if more images improved training results. GoogLeNet was used with the hyperparameters used in Run 2 in the round bale and tractor training. Only one detector was trained using the newly extracted images. The goal was to get a mAP of at least 80% on all obstacles, then combined for the final Faster R-CNN training.

5.5 Results of Faster R-CNN training

5.5.1 *Tractor*

Tractor training used GoogLeNet, ResNet 101, Inception ResNet V2 and Inception V3 feature extractors. Each feature extractor was tried with both sets of hyperparameters except that Inception ResNet V2 was only ran once.

Training time and the mAP results for tractor training can be seen in Table 5.1. GoogLeNet and ResNet 101 averaged a mAP of 90% and 49.5%, respectively between the two runs. Inception V3 was able to complete training but, outputted an average mAP

of 0 %. Inception ResNet V2 required more memory than was available and couldn't be run (APPENDIX F.8.1.4 Inception ResNet V2).

Table 5.1 Tractor Faster R-CNN training results

Tractor (210 pictures)			
Network/version	Trainining time (s)	mAP (%)	
GoogLeNetv2	241	87%	Run1
ResNet101v3	450	9%	
Inceptionv3v4	828	0%	
GoogLeNetv5	309	93%	Run2
ResNet101v6	921	90%	
Inceptionv3v7	994	0%	
InceptionResNetv2v8	error		Run1

GoogLeNet was both the most accurate and fastest to train. ResNet 101 had mixed accuracy results by having mAPs of 9% in Run 1 and 90% in Run 2, Inception V3 training had a mAP of 0% in both runs. It was determined that only GoogLeNet, ResNet 101 and Inception V3 would be tried for round bale training. A graphical representation of training time vs. mAP percentage is shown in Figure 5.7.

Run 2 was identified as the set of hyperparameters that worked the best for tractor detection. These hyperparameters returned a 93% in Run 2 compared to an 87% in Run 1 for GoogLeNet and a 90% in Run 2 compared to 9% in Run 1 for ResNet 101.

The training times of the feature extractor from slowest to fastest was Inception V3 at 4.34 seconds/image, ResNet 101 was 3.26 seconds/image

Figure 5.7 Tractor Faster R-CNN training graphed results

and GoogLeNet was the fastest in tractor training at 1.31 seconds/image.

A trained detector “tdgooglenetv2”, was tested on tractor images found on the internet and the results can be seen in APPENDIX F.8.1.1 GoogLeNet.

. In Figure 5.8 (a) the detector labeled the image of the tractor and planter eight different times instead of labeling the tractor once. This showed that more data was needed to clearly define the tractor from other equipment. The tractor in Figure 5.8 (b) is correctly labeled but the bounding box does not enclose the tractor tightly. More results from testing detectors on test tractor images can be seen in APPENDIX F Faster R-CNN Training.

(a) (b)

Figure 5.8 Tractor training results

GoogLeNet as the feature extractor in Faster R-CNN was able to identify tractors at about 644 milliseconds per image. When ResNet 101 was used as the feature extractor it took roughly 966 milliseconds to identify the tractor. Both feature extractors worked well but are still not fast enough for real time obstacle detection. More training test images are in APPENDIX F.8 Tractor images.

5.5.2 *Round Bale*

A round bale detector was trained using GoogLeNet, ResNet 101 and Inception V3 using the same hyperparameters as Run 1 and Run 2. GoogLeNet was the only network to return mAP for round bales. The mAPs for round bale training on GoogLeNet were 77% for Run 1 and 94% on Run 2. Run 2 hyperparameters had a 17% higher mAP compared to Run 1 for training done in GoogLeNet (Figure 5.9). Inception V3 completed training for the first run but was unable to complete the second run because it could not find region proposals to use as positive training samples causing the error.

ResNet 101 was able to complete both training runs but unable to give a mAP. The results can be seen in Table 5.2 Round Bale Faster R-CNN training result

Table 5.2 Round Bale Faster R-CNN training result

Round Bale (250 pictures)			
Network/version	Training time (s)	mAP (%)	
GoogLeNetv2	347	77%	Run1
ResNet101v3	920	0%	
Inceptionv3v4	1073	0%	
GoogLeNetv5	405	94%	Run2
ResNet101v6	1184	0%	
Inceptionv3v7	error		

. It was determined that Inception V3 would not be used and Run 2 hyperparameters with GoogleNet would be used for pivot training.

Inception V3 had the longest training time of 4.29 seconds/image followed by ResNet 101 at 4.21 seconds/image. The fastest training time was GoogLeNet at 1.50 seconds/image.

ResNet 101 was not able to identify round bales for the calculation of mAP. Test images were tried on GoogLeNet.

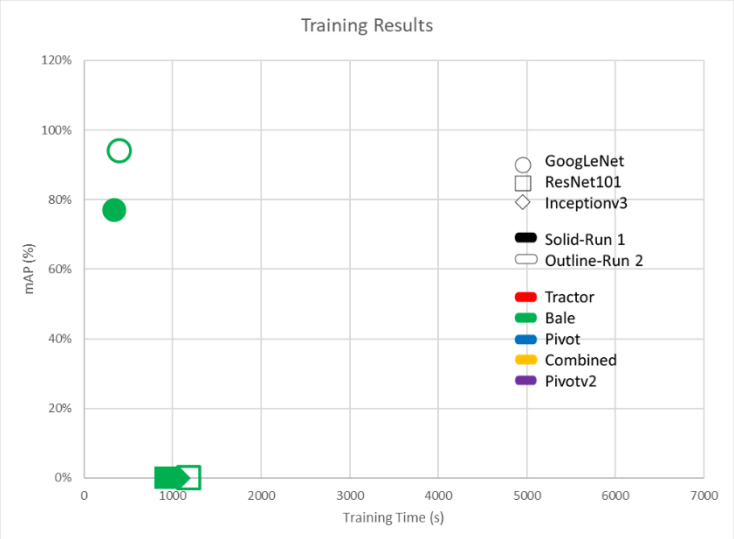
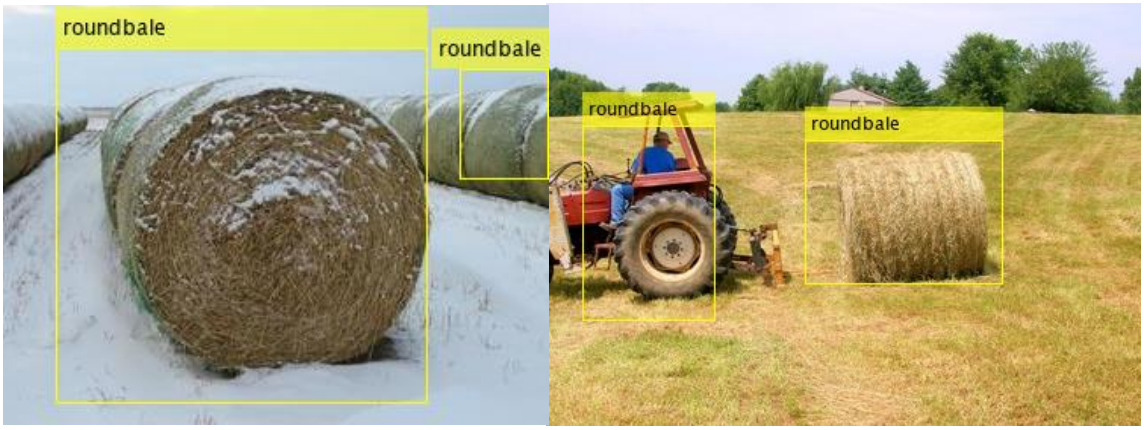


Figure 5.9 Graphical representation of the training results from the round bale Faster R-CNN training.

In Figure 5.10(a) the detector labeled both bounding boxes correctly but, mislabeled Figure 5.10(b), labeling the



(a) (b)

Figure 5.10 Bale detector testing images.

tractor as a round bale. In this detector the tractor was not a class during training. It is fair to assume that the detector mistook the tire as a bale. It was determined that only Run 2 hyperparameters would be used and only ResNet 101 and GoogLeNet would be used for pivot detection. More training test images can be found in APPENDIX F.9 Round Bale. Round bale detection using GoogLeNet averaged 990 milliseconds per image.

5.5.3 *Pivot*

A mAP was created for each component of the pivot when it was trained using GoogLeNet but was unable to produce a mAP for the components when ResNet 101 was used. Two pivot detectors were trained for both GoogLeNet and ResNet 101. When training was done using GoogLeNet, Run 1 had an average mAP for the components of 95% and Run 2 had a mAP of 92%, respectively. ResNet 101 was able to complete the first run of training but unable to complete the second training.

The highest component mAP was the pyramid at 99% for Run 1 and the tower at 98% for Run 2. The lowest was the end tower and span at 92% for Run 1 and the end tower was the lowest 81% for Run 2. Table 5.4 Results from pivot training shows a breakdown of component mAPs for both runs of GoogLeNet and ResNet 101.

Table 5.4 Results from pivot training

Pivot			
Network/version/class	Training time (s)	mAP (%)	
GoogLeNetv1 Avg.	1177	95%	Run1
pyramid		99%	
span		92%	
tower		98%	
end_tower		92%	
ResNet101v2 Avg.	3622	0%	
pyramid		0%	
span		0%	
tower		0%	
end_tower		0%	
GoogLeNetv3 Avg.	1170	92%	Run2
pyramid		96%	
span		91%	
tower		98%	
end_tower		81%	
ResNet101v4 Avg.	N/A		
pyramid			
span			
tower			
end_tower			

The average training time per image was 0.95 seconds/image in GoogLeNet. The training time for the one run of ResNet 101 was 2.94 seconds/image.

Figure 5.11 plots the times and mAPs of training for the GoogLeNet and ResNet 101 against each other. It was determined that GoogLeNet will be selected for the final combined training given the reduced average training time and superior mAP values.

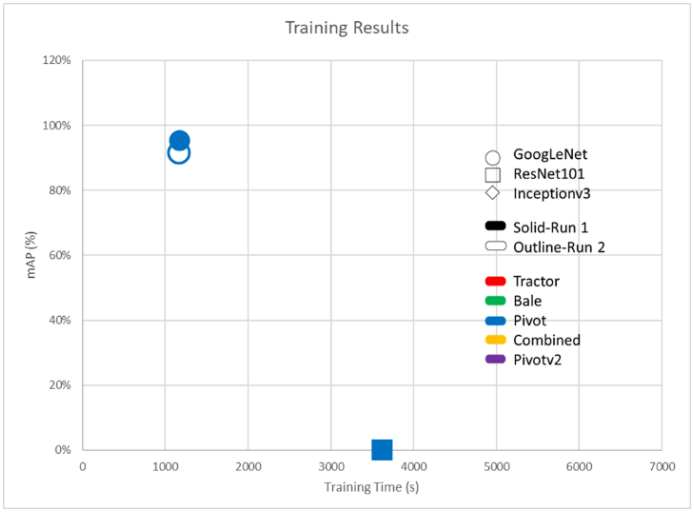


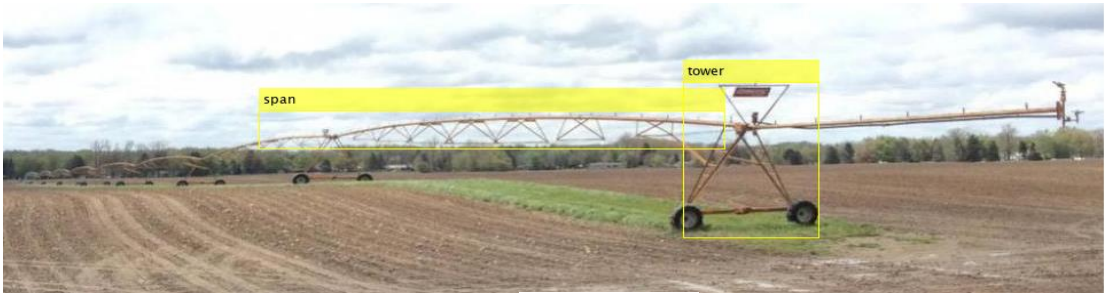
Figure 5.11 Results of pivot training graphed

In Figure 5.12 (a) is identified correctly with the two spans and tower. In Figure 5.12(b), the span was broken into two pieces to fully bind the span and part of the pyramid was wrongly labeled as a span. In Figure 5.12c the end tower is incorrectly identified as a

tower but, the entire span is correctly bound. Only GoogLeNet would be used as a feature extractor. GoogLeNet took approximately 1.125 seconds per image for identification. More training test images are in APPENDIX F.10 Pivot.

(a)

(b)



(c)

Figure 5.12 Pivot detectorv1 labeling: (a) two spans and a tower, (b) pyramid and span, (c) span and end tower

5.5.4 *Combined*

The combined detector used Faster R-CNN with GoogLeNet as the feature extractor. Run 1 is the first time the algorithm was trained and Run 2 is the second time the algorithm was ran with the same hyperparameters in both runs. All the image data were randomized and hyperparameters were the same as Run 2 of tractor and round bale training for both combined detector trainings. The average mAP for the combined pictures was 89% for the Run 1 and 91% for Run 2 (Figure 5.13).

The pyramid class had the highest mAP in both Run 1 and Run 2 at 97% and 99% respectively. The lowest mAPs occurred in end tower class in both Run 1 and Run 2 at 65% and 82%.

A total of 1691 images were used to train GoogLeNet for the combined detector. The time and accuracy are plotted in Figure 5.13.

The detectors were tested by pictures found on the internet. Test results for Run 1 can be seen in Figure 5.14. The results for more images can be seen in APPENDIX F.11 Combined Detector. When examining these images, it was found that some images were labeled incorrectly despite the high mAP. In Figure 5.14 (a) the tractor is identified correctly but labels the round bales as tractors. In Figure 5.14 (b), the detector correctly identifies all three images.

The detector when pictures were collaged together (Figure 5.15), was able to correctly identify round bales, a tractor, the span and tower of a pivot. It incorrectly labeled the end

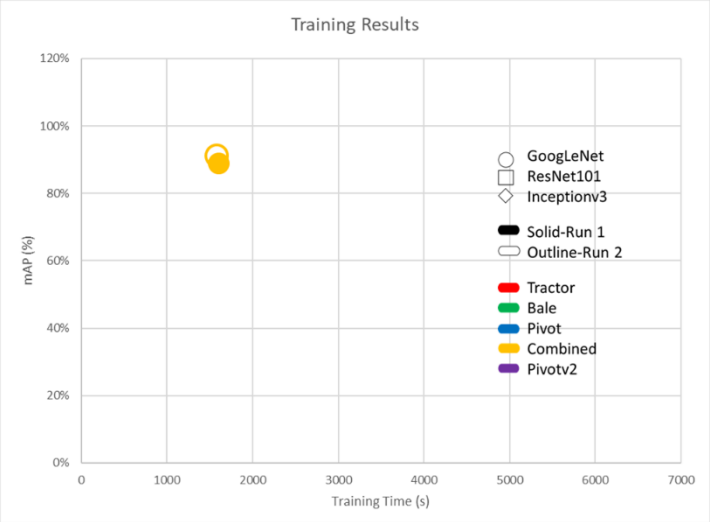


Figure 5.13 Accuracy vs time of combined dedectors.

(a)

(b)

Figure 5.14 Results from running the Run 1 combined detector on pictures from the internet. (a) A tractor and multiple bales and (b) multiple bales in a field.

tower as a tower. It was noticed that the bounding boxes were not tightly bounding the images. The detector was unable to notice an image of a pyramid though it was 99% mAP.

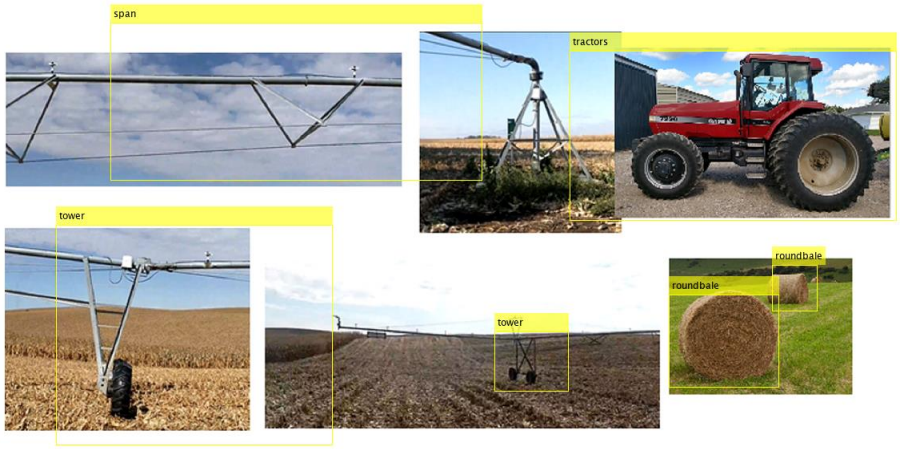


Figure 5.15 Collage of the six classes the combined detector is suppose to be able to identify.

For the combined detector it roughly takes 1.67 seconds per image which is not fast enough for real time detection. More combined test images can be seen APPENDIX F.11 Combined Detector.

5.5.5 *Pivotv2*

It was not known that every 5th image of video was being extracted, for the previous training data. The code can be seen APPENDIX F.1 Code used to extract images out of video (MATLAB). Pivotv2 training data had extracted every image extracted out of the video for a total of 6,148 images. Pivotv2 was the last training and used the same hyperparameters and CNN as the combined training.

Having the new data improved the mAP results over the first detector of pivots. The end tower mAP improved by 18%, the span 5%, pyramid 3% and tower stayed the same

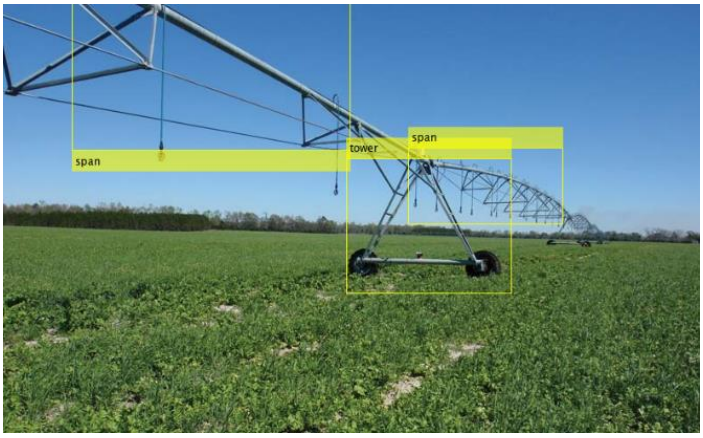
(Table 5.6). Results on test images can be seen in Figure 5.16. More test images can be seen APPENDIX F.12 Pivotv. With more training data it was still unable to identify the end tower correctly. The increase in number of training pictures did not affect the speed Faster R-CNN with GoogLeNet detected pictures. With the additional training images detection was roughly 1.08 seconds per image.

Table 5.6 Results of training with more pivot image data

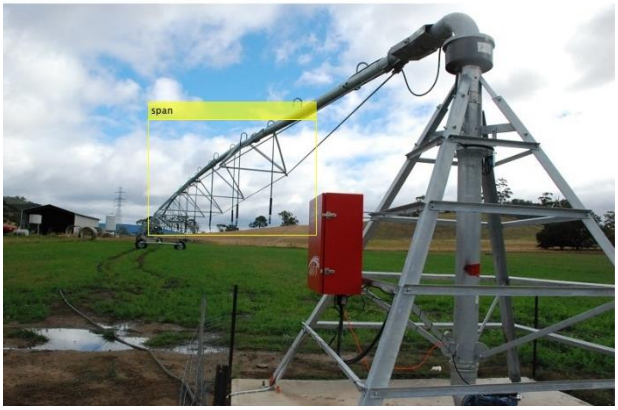
Pivotv2 (6148 pictures)			
Network/version	Training time (s)	mAP (%)	
GoogLeNetv3v2	6093	98%	Run1
pyramid		99%	
span		96%	
tower		98%	
end_tower		99%	

A summary of the feature extractors and training statistics can be found in Table 5.7 and Figure 5.17. GoogLeNet had the fastest training times out of all trained detectors. GoogLeNet had the fastest detection time in all instances those times are in Table 5.8 and can be seen in Figure 5.18. It can be noted that a 0% mAP cannot detect objects. Padded pivot images could have improved pivot detector accuracy. Both training and test images were skewed for the pivot which may have taught the detector wrong.

The trained detectors were tested on various times for speed in detection of objects GoogLeNet was mostly around 1 second. The statistics can be found in Table 5.8. A graphical comparison between the detection times can be seen in Figure 5.18.



(a)



(b)

(c)

Figure 5.16 Pivot detectorv2 labeling: (a) two spans and a tower, (b) pyramid and span, (c) span and end tower

Table 5.7 Summary statistics for each feature extractor and object class in Faster R-*CNN*

Summary Statistics				
Tractor stats			amount of data	210
	GoogLeNet	Resnet 101	Inceptionv3	InceptionResNetv2
training time (time(s)/picture)	1.31	3.26	4.34	N/A
map %	90.0%	49.5%	0.0%	N/A
Round Bale stats			amount of data	250
	GoogLeNet	Resnet 101	Inceptionv3	
training time (time(s)/picture)	1.50	4.21	4.29	
map %	85.5%	0.0%	0.0%	
Pivot stats			amount of data	1231
	GoogLeNet	Resnet 101		
training time (time(s)/picture)	0.95	2.94		
map %	93.4%	0.0%		
combined stats			amount of data	1692
	GoogLeNet			
training time (time(s)/picture)	0.94			
map %	89.9%			
Pivotv2 stats			amount of data	6148
	GoogLeNet			
training time (time(s)/picture)	0.99			
map %	98.0%			

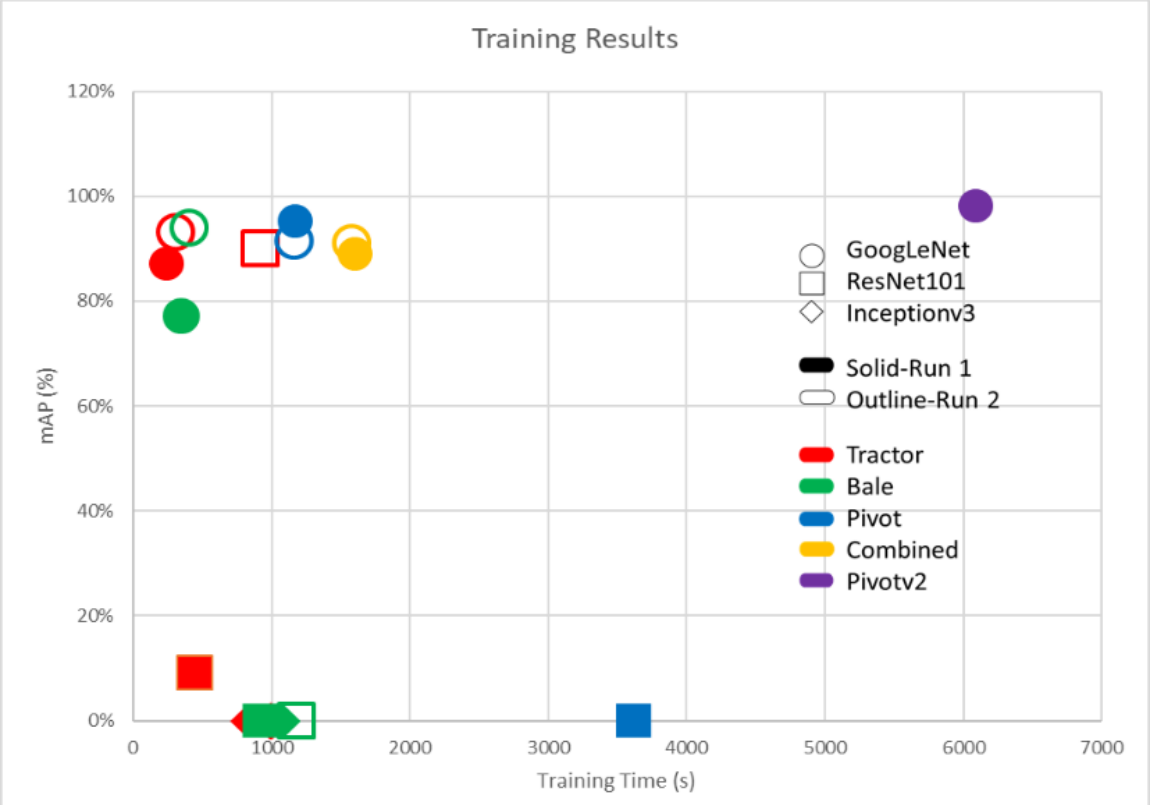


Figure 5.17 A graph comparing all trained detectors

Table 5.8 Detector speed and accuracy

Summary Statistics				
Tractor stats			amount of data	210
	GoogLeNet	Resnet 101	Inceptionv3	InceptionResNetv2
detection time (s)	0.64	0.97	N/A	N/A
map %	90.0%	49.5%	0.0%	N/A
Round Bale stats			amount of data	250
	GoogLeNet	Resnet 101	Inceptionv3	
detection time (s)	0.99	N/A	N/A	
map %	85.5%	0.0%	0.0%	
Pivot stats			amount of data	1231
	GoogLeNet	Resnet 101		
detection time (s)	1.13	N/A		
map %	93.4%	0.0%		
combined stats			amount of data	1692
	GoogLeNet			
detection time (s)	0.95			
map %	89.9%			
Pivotv2 stats			amount of data	6148
	GoogLeNet			
detection time (s)	1.08			
map %	98.0%			

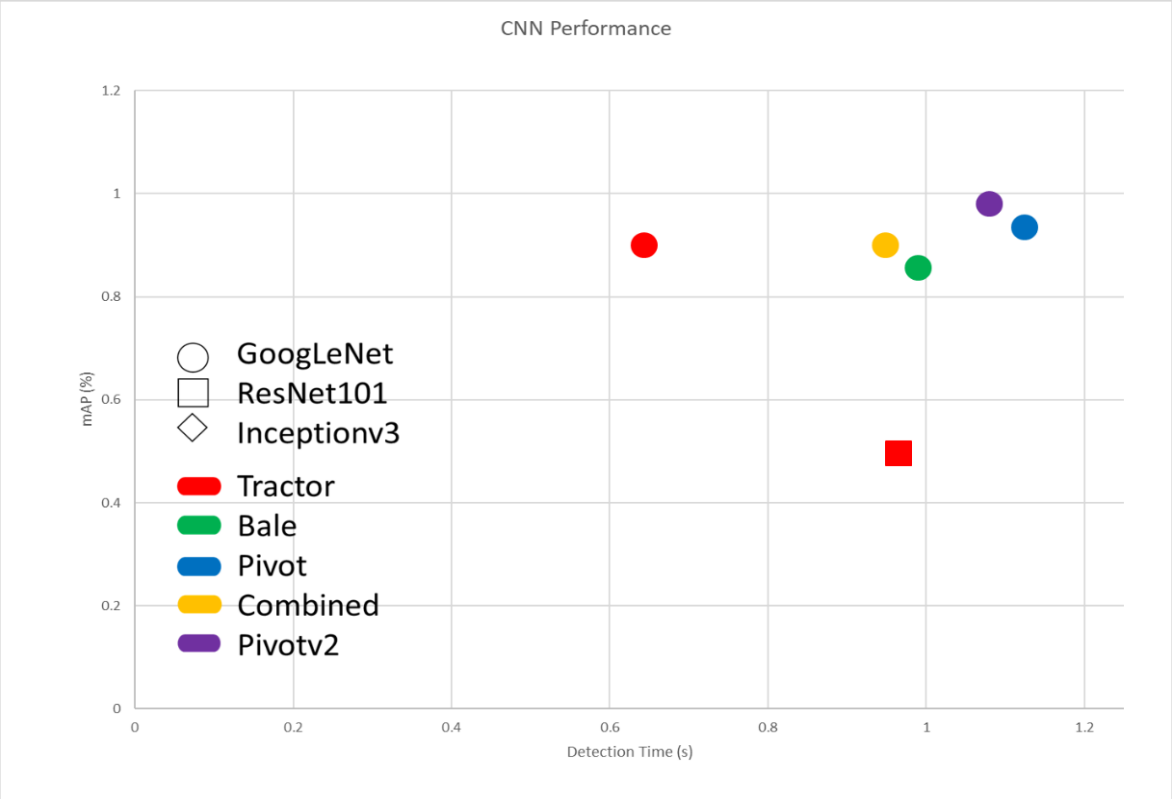


Figure 5.18 A graphical representation of the detector testing times along with the accuracy in detection

5.6 Conclusions of Faster R-CNN Training

It was determined that Faster R-CNN architecture would work best for obstacle detection. A larger data set made up of round bales and tractors may have resulted in better detection because a larger amount of data increased mAP in pivot detection. It was determined that GoogLeNet would work well for agricultural obstacles achieving over an 85% mAP and an increase in the number of images (5x the original amount) increased the mAP of pivots 6%. A drawback to using GoogLeNet as the feature extractor was it did not perform fast enough to reach the 5 Hz processing time needed for real time obstacle detection. Even though no feature extractor could be identified for real-time detection this proves that CNNs can work for pivot and other truss like object detection. It was determined that the size of hardware may have been a limiting factor during training and evaluation of the trained detectors.

Chapter 6 Conclusions and Future Work

6.1 Conclusions

Different sensors and methodologies were experimented and tested for practical use in agricultural object detection. The main objective of detecting obstacles was successful after many different algorithms and methodologies were tried.

Off the shelf LIDARs were thoroughly tested for obstacle detection. The biggest factor in LIDAR obstacle detection was angular resolution. Range, effectiveness in sunlight and sampling rate were also important factors for obstacle detection. It was proven that LIDARs work for obstacle detection for large objects like tractors or bales but, proved ineffective in detecting small truss like structures found in pivots.

The width between two data points would be too large in detecting a pivot at a respectable distance required for obstacle avoidance. Another limitation of LIDARs is identifying the obstacles they detect. Algorithms do exist that are able to tell you what shape the object is by comparing neighboring LIDAR data points but are unable to determine what the obstacle is. Computer vision was tested next to identify truss like structures such as pivots.

Methods in computer vision were tested first. Feature Extracting and Matching was experimented first trying to match a reference image features to an input image. The algorithm was unable to clearly identify the reference pivot image in the input image of a pivot. The algorithm was found not to be adaptable to the incoming image orientations.

Edge detection was the next methodology tried. A shape identifying algorithm was implemented that used the circularity of the identified shape to determine the class of shape. This method proved to be unreliable and could be dependent on camera angles. A combination of edge detection and SURF Feature Matching and Extracting was tried next but, quickly proved to be ineffective when all matching points in the reference image consolidated to one single point in the test image.

A Convolutional Neural Network was determined to be the most effective algorithm for pivot detection and other obstacle identification. It was determined that GoogLeNet with Faster R-CNN would be the algorithms of choice to clearly identify obstacles such as tractors, bales and pivots. It must be noted that even though GoogLeNet used in R-CNN was the most accurate, the time needed to identify the object was unrealistic for real time detection at 1.17 seconds average per image during testing of the detectors.

RGB cameras with CNNs can identify objects a far distance away. Which will help with obstacle avoidance by warning the computer that an obstacle exists, but it would need the help of other sensors to clearly detect the obstacles.

In this study more images are needed to make the CNN more robust. Only a small amount images were collected and used in this study.

6.2 Future Work

A task that can be done is collecting and labeling thousands of more pictures of many farm obstacles. This study focused on tractors, round bales and pivots and laid the foundation for others to build upon. Another challenge would be learning how to use sensor fusion and SLAM to clearly identify where the obstacle is relative to the

autonomous tractor. Future work would be finding and implementing a faster feature detector for the Faster R-CNN than GoogLeNet or a different object detector algorithm such as the Single-Shot Detector (SSD) or the “You Only Look Once” (YOLO) network. The hope of this study was to shed light on what is possible for obstacle detection for autonomous tractors.

References

Agricar. (2019). New Holland T7.210 Tractor at Agricar. Retrieved March 13, 2019, from Agricar website: https://www.agricar.co.uk/item/11525/agricar/New-Holland-T7210-Tractor.html

Anirudh Sharma. (2018, July 20). What is an epoch in deep learning? Retrieved March 12, 2019, from https://www.quora.com/What-is-an-epoch-in-deep-learning

Arduino Library for Scanse Sweep LiDAR [Arduino]. (2017). Retrieved from https://github.com/scanse/sweep-arduino (Original work published 2017)

AuctionTime Blog. (2018). Case IH Autonomous Concept Tractor Wins GOOD DESIGN Award. Retrieved March 15, 2019, from AuctionTime.com website: https://www.auctiontime.com/blog/fun-stuff/2018/01/case-ih-autonomous-concept-tractor-wins-good-design-award

Barbosa, D., Lopes, A., & Araújo, R. E. (2016). Sensor fusion algorithm based on Extended Kalman Filter for estimation of ground vehicle dynamics. *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 1049–1054. https://doi.org/10.1109/IECON.2016.7793145

Bay, H., Tuytelaars, T., & Van Gool, L. (2006). SURF: Speeded Up Robust Features. In A. Leonardis, H. Bischof, & A. Pinz (Eds.), *Computer Vision – ECCV 2006* (Vol. 3951, pp. 404–417). https://doi.org/10.1007/11744023_32

Bedord, L. (2018, February 14). Case IH Moves Autonomous Concept Tractor Forward. *Successful Farming*. Retrieved from

https://www.agriculture.com/news/technology/case-ih-moves-autonomy-project-forward

beefmagazine.com. (2014, October 6). Why You Must Remove Net Wrap On Round Bales Before Feeding To Cattle. Retrieved March 15, 2019, from Beef Magazine website: https://www.beefmagazine.com/beef-cattle-feed/why-you-must-remove-net-wrap-round-bales-feeding-cattle

Bellone, M., Messina, A., & Reina, G. (2013). A new approach for terrain analysis in mobile robot applications. *2013 IEEE International Conference on Mechatronics (ICM)*, 225–230. https://doi.org/10.1109/ICMECH.2013.6518540

Bernini, N., Bertozzi, M., Castangia, L., Patander, M., & Sabbatelli, M. (2014). Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey. *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 873–878. https://doi.org/10.1109/ITSC.2014.6957799

Big Ag. (2018, January 18). Autonomous Tractors- The Future of Farming? Retrieved August 21, 2018, from Big Ag website: http://www.bigag.com/topics/equipment/autonomous-tractors-future-farming/

Case IH. (2016, August 30). CNH Industrial brands reveal concept autonomous tractor development. Retrieved August 21, 2018, from http://www.cnhindustrial.com/en-US/media/press_releases/2016/august/pages/CNH_Industrial_brands_reveal_concept_autonomous_tractor_development_Announcement.aspx

Case IH. (2018, February 14). Case IH Defines Categories of Autonomy and Announces Pilot Program. Retrieved October 9, 2018, from

https://cloudfront.cdn.caseih.com:4495/emea/en-za/News/Pages/2018-02-14-Case-IH-Defines-Categories-of-Autonomy-and-Announces-Pilot-Program.aspx

Chicot Irrigation. (2019). Valley Center Pivots. Retrieved March 15, 2019, from https://www.chicotirrigation.com/valley-center-pivots

David, L., & Viola, P. (2001, April). *The Viola/Jones Face Detector*. Lecture Presentations presented at the University of California-Berkley. Retrieved from https://www.cs.ubc.ca/~lowe/425/slides/13-ViolaJones.pdf

Denise O'Sullivan. (2019). Awesome! Tractor Loader Transformer. Mega Bale Spear and Bale Fork. Retrieved March 15, 2019, from Pinterest website: https://www.pinterest.com/pin/759489924632642728/

Deshpande, A. (2016, July 20). A Beginner's Guide To Understanding Convolutional Neural Networks. Retrieved October 11, 2018, from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Doukkali, F. (2017, September 28). Convolutional Neural Networks (CNN, or ConvNets). Retrieved October 15, 2018, from Firdaouss Doukkali website: https://medium.com/@phidaouss/convolutional-neural-networks-cnn-or-convnets-d7c688b0a207

Dubois, C. (2018). What Is Solid State LiDAR and Is It Faster, Cheaper, Better? *All About Circuits*. Retrieved from https://www.allaboutcircuits.com/news/solid-state-lidar-faster-cheaper-better/

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, *22*(6), 46–57. https://doi.org/10.1109/2.30720

everythingattachments.com. (2019). Bale Spears for Skid Steers. Retrieved March 15, 2019, from https://www.everythingattachments.com/Skid-Steer-Bale-Spears/109.htm

FAO Director's office. (2009, October). High Level Export Forum- How to Feed the World in 2050. Retrieved August 21, 2018, from http://www.fao.org/fileadmin/templates/wsfs/docs/Issues_papers/HLEF2050_Global_Agriculture.pdf

Freitas, G., Hamner, B., Bergerman, M., & Singh, S. (2012). A practical obstacle detection system for autonomous orchard vehicles. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3391–3398. https://doi.org/10.1109/IROS.2012.6385638

Gao, H. (2017, September 27). Faster R-CNN Explained. Retrieved January 18, 2019, from Hao Gao website: https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

Gillespie, K. (2018, February 19). Ultrasonic Sensors: Advantages and Limitations. Retrieved September 27, 2018, from MaxBotix Inc. website: https://www.maxbotix.com/articles/articlesadvantages-limitations-ultrasonic-sensors-htm.htm

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *ArXiv:1311.2524 [Cs]*. Retrieved from http://arxiv.org/abs/1311.2524

Global Auction Guide. (2019). Image Viewer. Retrieved March 15, 2019, from https://www.globalauctionguide.com/image-viewer/54692238

Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., … Murphy, K. (2016). Speed/accuracy trade-offs for modern convolutional object detectors. *ArXiv:1611.10012 [Cs]*. Retrieved from http://arxiv.org/abs/1611.10012

Hui, J. (2018, March 27). Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and…. Retrieved October 2, 2018, from Medium website: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Jadon, S. (2018, March 16). Introduction to Different Activation Functions for Deep Learning. Retrieved November 1, 2018, from Medium website: https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Jo, Youngtae., & Jung, Inbum. (2014). Analysis of Vehicle Detection with WSN-Based Ultrasonic Sensors. *Sensors (Basel, Switzerland)*, *14*(8), 14050–14069. https://doi.org/10.3390/s140814050

John Deere. (2019). Tractors | John Deere US. Retrieved March 13, 2019, from https://www.deere.com/en/tractors/

Keeping It Dutch. (2015). *How to move a round bale of hay without a tractor*. Retrieved from https://www.youtube.com/watch?v=KSAovFiLO7s

Kirill Eremenko. (2018, August). *Deep Learning A-Z™: Convolutional Neural Networks (CNN) - Step 3: Fla…*. Education. Retrieved from https://www.slideshare.net/KirillEremenko/deep-learning-az-convolutional-neural-networks-cnn-step-3-flattening?ref=https://www.superdatascience.com/convolutional-neural-networks-cnn-step-3-flattening-presentation/

Kochhar, R. (2014, February 3). 10 projections for the global population in 2050. Retrieved July 3, 2018, from Pew Research Center website: http://www.pewresearch.org/fact-tank/2014/02/03/10-projections-for-the-global-population-in-2050/

Kubota. (2019). Tractor | Products. Retrieved March 13, 2019, from https://www.kubota.com/products/tractor/

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Lecun, Yann, & Bengio, Y. (1994). Word-level training of a handwritten word recognizer based on convolutional neural networks. *Proceedings of the International Conference on Pattern Recognition, Jerusalem, October 1994*, 88–92.

Lenz, P., Ziegler, J., Geiger, A., & Roser, M. (2011). Sparse scene flow segmentation for moving object detection in urban environments. *2011 IEEE Intelligent Vehicles Symposium (IV)*, 926–932. https://doi.org/10.1109/IVS.2011.5940558

Lindsay Europe. (2019). 9500P - Center irrigation pivot by LINDSAY. Retrieved March 15, 2019, from http://www.agriexpo.online/prod/lindsay-europe-sa/product-179304-85640.html

Litomisky, K. (2012, Spring). Consumer RGB-D Cameras and their Applications. Retrieved November 29, 2017, from http://alumni.cs.ucr.edu/~klitomis/files/RGBD-intro.pdf

Liu, D.-C. (2017, November 30). A Practical Guide to ReLU. Retrieved October 22, 2018, from TinyMind website: https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7

Magneon. (2018, May 11). Robotics [Www.reddit.com]. Retrieved August 29, 2018, from What happen to Scanse LIDAR website: https://www.reddit.com/r/robotics/comments/8fjpql/what_happen_to_the_scanse_lidar/

Mahindra. (2019). 2019 Mahindra 1526 4WD Shuttle. Retrieved March 13, 2019, from https://www.esuperbike.com/Tractors-Mahindra-1526-4WD-Shuttle-2019-Evansville-IN-e6eea2b1-cb2a-4781-aa06-a9d60085c059

Mapanauta. (2018, May 24). Why Solid-State Lidar Is Key to the Future of Self-Driving Cars. Retrieved August 20, 2018, from Medium website:

https://medium.com/@mapanauta/why-solid-state-lidar-is-key-to-the-future-of-self-driving-cars-5e90ea906608

Marybeth Feutz. (2010, October 2). Farm Equipment Friday: Bale spear. Retrieved March 15, 2019, from My Fearless Kitchen website: https://www.myfearlesskitchen.com/farm-equipment-friday-bale-spear/

Mathworks. (2018a). Convolutional Neural Network. Retrieved October 12, 2018, from https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Mathworks. (2018b). Detect objects using the Viola-Jones algorithm. Retrieved December 28, 2018, from https://www.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html?s_tid=doc_ta

Mathworks. (2018c). Find edges in intensity image. Retrieved December 29, 2018, from https://www.mathworks.com/help/images/ref/edge.html?s_tid=srchtitle

Mathworks. (2018d). Max pooling layer. Retrieved October 22, 2018, from https://www.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.maxpooling2dlayer.html

Mathworks. (2018e). Pretrained AlexNet convolutional neural network - MATLAB alexnet. Retrieved October 10, 2018, from https://www.mathworks.com/help/deeplearning/ref/alexnet.html

Mathworks. (2019). Path Planning in Environments of Different Complexity. Retrieved March 21, 2019, from https://www.mathworks.com/help/robotics/examples/path-

planning-in-environments-of-difference-complexity.html;jsessionid=1b8c1fda0df72f1490d9e739d50d

Mesa Irrigation Co in Lamesa , TX. (2019). Mesa Irrigation Co in Lamesa , TX. Retrieved March 15, 2019, from YP.com website: http://www.yellowpages.com/lamesa-tx/mip/mesa-irrigation-co-448942?lid=448942&from=SocSh_Facebook

MiniQuark. (2018, September 7). python - What is the difference between “SAME” and “VALID” padding in tf.nn.max_pool of tensorflow? Retrieved October 23, 2018, from Stack Overflow website: https://stackoverflow.com/questions/37674306/what-is-the-difference-between-same-and-valid-padding-in-tf-nn-max-pool-of-t

Mokey, N. (2018, March 15). A self-driving car in every driveway? Solid-state lidar is the key. Retrieved August 20, 2018, from Digital Trends website: https://www.digitaltrends.com/cars/solid-state-lidar-for-self-driving-cars/

New Holland. (2017). NEW HOLLAND AGRICULTURE UNVEILS METHANE POWERED CONCEPT TRACTOR AND ITS VISION FOR THE SUSTAINABLE FUTURE OF FARMING. Retrieved March 15, 2019, from https://agriculture.newholland.com/apac/en-nz/about-us/whats-up/news-events/2017/new-holland-unveils-methane-powered-concept

newwayirrigation.com. (2019). Pivots. Retrieved March 15, 2019, from New Way Irrigation website: https://newwayirrigation.com/pivots/

Ohio's Country Journal. (2019). Alternative Uses for Round Bales. Retrieved March 15, 2019, from https://www.ocj.com/2010/09/alternative-uses-for-round-bales/

Oniga, F., & Nedevschi, S. (2010). Processing Dense Stereo Data Using Elevation Maps: Road Surface, Traffic Isle, and Obstacle Detection. *IEEE Transactions on Vehicular Technology*, *59*(3), 1172–1182. https://doi.org/10.1109/TVT.2009.2039718

pivotsplus.com. (2019). Image: Used Zimmatic Pivot 0148. Retrieved March 15, 2019, from https://www.google.com/imgres?imgurl=http://www.pivotsplus.com/Images/listings/pivots/zim148/083.jpg&imgrefurl=http://www.pivotsplus.com/pivots/used-zimmatic-pivot-0148&h=612&w=816&tbnid=2rj2b6IxxjhJ3M&tbnh=194&tbnw=259&usg=K_nm34EtP1JKrxtGHQAjKdgK_mTfo=&hl=en&docid=boSJoNxUOwABcM

RainFine Irrigation Solution. (2019). CENTER PIVOT FIXED. Retrieved March 15, 2019, from http://www.rainfineaustralia.com/?page_id=8873

Reina, G. (2013, February). Unevenness Point Descriptor for Terrain Analysis in Mobile Robot Applications. Retrieved March 19, 2019, from ResearchGate website: https://www.researchgate.net/publication/260024583_Unevenness_Point_Descriptor_for_Terrain_Analysis_in_Mobile_Robot_Applications

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information*

Processing Systems 28 (pp. 91–99). Retrieved from http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

Repos, R. P. (2019). *RoboPeak RPLIDAR driver for Arduino and Arduino-compatible devices: robopeak/rplidar_arduino* [C++]. Retrieved from https://github.com/robopeak/rplidar_arduino (Original work published 2014)

Ritchie Bros. (2019). New and used tractors for sale. Retrieved March 13, 2019, from https://www.rbauction.com/agriculture-tractors?cid=23572767093

Sabale, A. S., & Vaidya, Y. M. (2016). Accuracy measurement of depth using Kinect sensor. *2016 Conference on Advances in Signal Processing (CASP)*, 155–159. https://doi.org/10.1109/CASP.2016.7746156

Samieh, A. (2016, March 31). Shape Recognition [File Exchange - MATLAB Central]. Retrieved December 28, 2018, from https://www.mathworks.com/matlabcentral/fileexchange/15491

Satofumi Kamimura. (2013, January 31). URG Helper. Retrieved May 31, 2019, from SourceForge website: https://sourceforge.net/projects/urgwidget/

Senix Corporation. (2018). Ultrasonic Sensors - Object Detection. Retrieved March 18, 2019, from https://senix.com/object-detection/

Sheridan Reality & Auction Co. (2017). 2016 Zimmatic Center Pivot, 8- tower. Retrieved March 18, 2019, from https://bid.sheridanauctionservice.com/m/lot-details/index/catalog/15357/lot/2085832/Zimmatic-7-tower-Center-Pivot-1-center-section-is-damaged

Southern Plains Photography. (2019). Hay Bale and Storm Photography Print. Retrieved March 15, 2019, from https://www.amazon.com/Hay-Bale-Storm-Photography-Print/dp/B01E7VGGRS

Spielmaker, D. M. (2018, March 21). Growing a Nation Historical Timeline. Retrieved June 22, 2018, from https://www.agclassroom.org/gan/timeline/index.htm

Stanford. (2018, October 5). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved October 5, 2018, from http://cs231n.github.io/convolutional-networks/#conv

Sullivan, A. (2014, May 24). Fact Check: Reynolds says one Iowa farmer feeds 155 people worldwide. Retrieved June 22, 2018, from The Gazette website: http://thegazette.com/subject/news/government/fact-check/fact-check-reynolds-says-one-iowa-farmer-feeds-155-people-worldwide-20140524

Tracey Media. (2013). *Tractor Making Hay Bales*. Retrieved from https://www.youtube.com/watch?v=B2dmW2MWfGw

Ujjwalkarn. (2016, August 11). An Intuitive Explanation of Convolutional Neural Networks. Retrieved October 16, 2018, from the data science blog website: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

V, A. S. (2017, March 30). Understanding Activation Functions in Neural Networks. Retrieved November 2, 2018, from Medium website: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Vishal Agro Industries. (2019). Vishal Agro Industries. Retrieved March 15, 2019, from https://www.vishalagroindustries.com/

Vodar Co., Ltd. (2019). Vodar Center Pivot Irrigation System For Alfalfa Irrigation. Retrieved March 15, 2019, from www.alibaba.com website: //www.alibaba.com/product-detail/VODAR-Center-Pivot-Irrigation-System-for_60675652116.html

Vogt, W. (2018, March 7). Defining the levels of automation. Retrieved October 9, 2018, from Wallaces Farmer website: https://www.wallacesfarmer.com/technology/defining-levels-automation

Weisstein, E. W. (2018, October 22). L^2-Norm [Text]. Retrieved October 23, 2018, from http://mathworld.wolfram.com/L2-Norm.html

Wiring Schematic Design. (2019). Valley Center Pivot Irrigation Systems. Retrieved March 15, 2019, from Wiring Schematic Diagram website: http://157.twizer.co

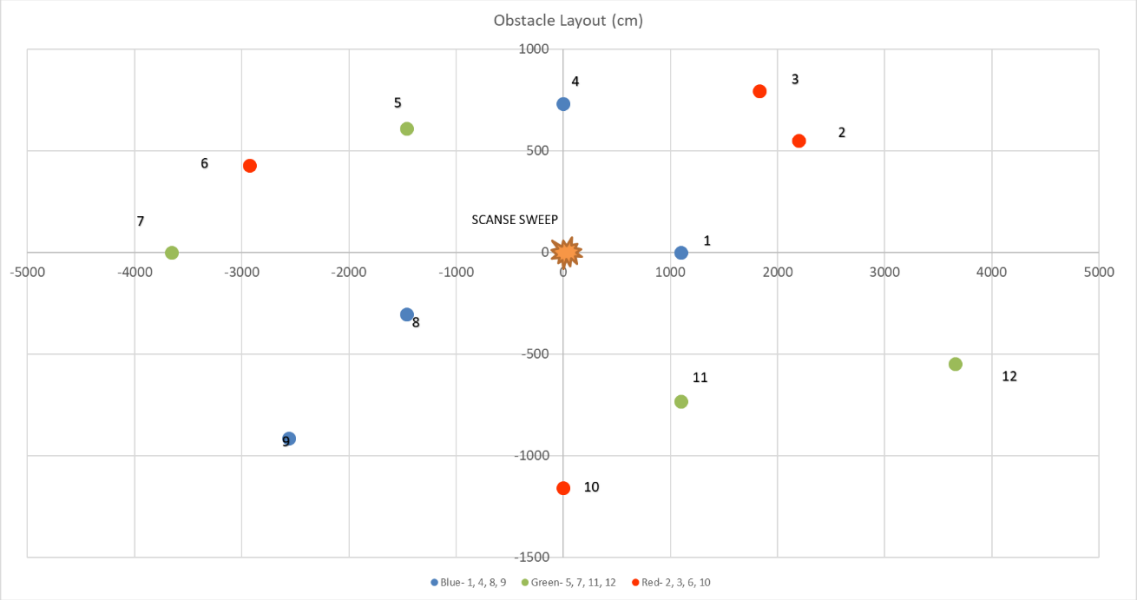
WorldAgNetwork. (2016, February 15). 50-plus Years of Center-pivot Irrigation. Retrieved March 15, 2019, from World Agriculture Network website: https://worldagnetwork.com/50-plus-years-of-center-pivot-irrigation/

Zulkifli, H. (2018, January 21). Understanding Learning Rates and How It Improves Performance in Deep Learning. Retrieved March 13, 2019, from Towards Data Science website: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10

APPENDIX A SCANSE

APPENDIX A.1 Test Setup

Obstacle number	X Coordinate columns	Y Coordinate rows	X Coordinate feet	Y Coordinate feet	X Coordinate cm	Y Coordinate cm	radial distance feet	radial distance cm	azimuth degrees
1	3	0	36	0	1097.28	0	36	1097.28	0.00
2	6	9	72	18	2194.56	548.64	74.21590126	2262.10067	14.04
3	5	13	60	26	1828.8	792.48	65.3911309	1993.12167	23.43
4	0	12	0	24	0	731.52	24	731.52	90.00
5	4	10	48	20	-1463.04	609.6	52	1584.96	157.38
6	8	7	96	14	-2926.08	426.72	97.01546269	2957.031303	171.70
7	10	0	120	0	-3657.6	0	120	3657.6	180.00
8	4	5	48	10	-1463.04	-304.8	49.03060269	1494.45277	191.77
9	7	15	84	30	-2560.32	-914.4	89.19641248	2718.706653	199.65
10	0	19	0	38	0	-1158.24	38	1158.24	270.00
11	3	12	36	24	1097.28	-731.52	43.26661531	1318.766435	326.31
12	10	9	120	18	3657.6	-548.64	121.3424905	3698.51911	351.47



APPENDIX A.2 Data Collection code (Arduino)

```
/*
 Scanse Sweep Arduino Library Examples

 MegaSerialPrinter:
    - Example sketch for using the Scanse Sweep with the Arduino Mega 2560.
      Collects 3 complete scans, and then prints the sensor readings
    - Assumes Sweep sensor is physically connected to Serial #1 (RX1 & TX1)
      - For the sweep's power, ground, RX & TX pins, follow the connector
```

```
    pinouts in the sweep user manual located here:
    http://scanse.io/downloads
  - Be sure to connect RX_device -> TX_arduino & TX_device -> RX_arduino
  - For best results, you should provide dedicated external 5V power to the Sweep
    rather than using power from the Arduino. Just be sure to connect the ground
    from the power source and the arduino. If you are just experimenting, you can
    run the sweep off the 5V power from the Arduino with the Arduino receiving power
    over USB. However this has only been tested with an external powered USB hub.
    It is possible that using a low power USB port (ex: laptop) to power the
    arduino & sweep together will result in unexpected behavior.
  - Note that running off of USB power is not entirely adequate for the sweep,
    so the quantity and quality of sensor readings will drop. This is OK for
    this example, as it is only meant to provide some visual feedback over
    the serial monitor.
  - In your own projects, be sure to use dedicated power instead of the USB.

 Created by Scanse LLC, February 21, 2017.
 Released into the public domain.
*/

#include <Sweep.h>
#include "SD.h"
#include <Wire.h>
#include "RTClib.h"

int record = 39;
int recLED = 52;

// Create a Sweep device using Serial #1 (RX1 & TX1)
Sweep device(Serial1);
// Scan packet struct, used to store info for a single reading
ScanPacket reading;

// keeps track of how many scans have been collected
uint8_t scanCount = 0;
// keeps track of how many samples have been collected
uint16_t sampleCount = 0;

// Arrays to store attributes of collected scans
bool syncValues[500];       // 1 -> first reading of new scan, 0 otherwise
float angles[500];          // in degrees (accurate to the millidegree)
uint16_t distances[500];    // in cm
uint8_t signalStrengths[500]; // 0:255, higher is better

// Finite States for the program sequence
const uint8_t STATE_WAIT_FOR_USER_INPUT = 0;
```

```
const uint8_t STATE_ADJUST_DEVICE_SETTINGS = 1;
const uint8_t STATE_VERIFY_CURRENT_DEVICE_SETTINGS = 2;
const uint8_t STATE_BEGIN_DATA_ACQUISITION = 3;
const uint8_t STATE_GATHER_DATA = 4;
const uint8_t STATE_STOP_DATA_ACQUISITION = 5;
const uint8_t STATE_REPORT_COLLECTED_DATA = 6;
const uint8_t STATE_RESET = 7;
const uint8_t STATE_ERROR = 8;

// Current state in the program sequence
uint8_t currentState;

// String to collect user input over serial
String userInput = "";

// A simple data logger for the Arduino analog pins
#define LOG_INTERVAL  1000 // mills between entries
#define ECHO_TO_SERIAL   1 // echo data to serial port
#define WAIT_TO_START    0 // Wait for serial input in setup()


// the digital pins that connect to the LEDs
#define redLEDpin 3
#define greenLEDpin 4

RTC_DS1307 RTC; // define the Real Time Clock object


// for the data logging shield, we use digital pin 10 for the SD cs line
const int chipSelect = 10;

// the logging file
File logfile;

void error(char *str)
{
  Serial.print("error: ");
  Serial.println(str);

  // red LED indicates error
  digitalWrite(redLEDpin, HIGH);

  while(1);
}

void setup()
```

```
{
  pinMode(record, INPUT);
  pinMode(recLED, OUTPUT);
  digitalWrite(recLED, LOW);

  // Initialize serial
  Serial1.begin(115200); // sweep device

  // reserve space to accumulate user message
  userInput.reserve(50);

  // initialize counter variables and reset the current state
  reset();

  Serial.begin(9600);
  Serial.println();

#if WAIT_TO_START
  Serial.println("Type any character to start");
  while (!Serial.available());
#endif //WAIT_TO_START
}

// Loop functions as an FSM (finite state machine)
void loop()
{
  switch (currentState)
  {
  case STATE_WAIT_FOR_USER_INPUT:
    if (listenForUserInput())
      currentState = STATE_ADJUST_DEVICE_SETTINGS;
    break;
  case STATE_ADJUST_DEVICE_SETTINGS:
    currentState = adjustDeviceSettings() ? STATE_VERIFY_CURRENT_DEVICE_SETTINGS :
STATE_ERROR;
    break;
  case STATE_VERIFY_CURRENT_DEVICE_SETTINGS:
    currentState = verifyCurrentDeviceSettings() ? STATE_BEGIN_DATA_ACQUISITION :
STATE_ERROR;
    break;
  case STATE_BEGIN_DATA_ACQUISITION:
    currentState = beginDataCollectionPhase() ? STATE_GATHER_DATA : STATE_ERROR;
    break;
  case STATE_GATHER_DATA:
    gatherSensorReading();
    if (scanCount > 3)
```

```
    currentState = STATE_STOP_DATA_ACQUISITION;
   break;
  case STATE_STOP_DATA_ACQUISITION:
   currentState = stopDataCollectionPhase() ? STATE_REPORT_COLLECTED_DATA :
STATE_ERROR;
   break;
  case STATE_REPORT_COLLECTED_DATA:
   printCollectedData();
   currentState = STATE_RESET;
   break;
  case STATE_RESET:
   Serial.println("\n\nAttempting to reset and run the program again...");
   reset();
   currentState = STATE_WAIT_FOR_USER_INPUT;
   break;
  default: // there was some error
   Serial.println("\n\nAn error occured. Attempting to reset and run program again...");
   reset();
   currentState = STATE_WAIT_FOR_USER_INPUT;
   break;
  }
}

// checks if the user has communicated anything over serial
// looks for the user to send "start"
bool listenForUserInput()
{
  while (Serial.available())
  {
   userInput += (char)Serial.read();
  }
  if (userInput.indexOf("start") != -1)
  {
   Serial.println("Registered user start.");
   return true;
  }
  return false;
}

// Adjusts the device settings
bool adjustDeviceSettings()
{
  // Set the motor speed to 5HZ (codes available from 1->10 HZ)
  bool bSuccess = device.setMotorSpeed(MOTOR_SPEED_CODE_5_HZ);
  Serial.println(bSuccess ? "\nSuccessfully set motor speed." : "\nFailed to set motor speed");
```

```
  /*
  // Device will always default to 500HZ scan rate when it is powered on.
  // Snippet below is left for reference.
  // Set the sample rate to 500HZ (codes available for 500, 750 and 1000 HZ)
  bool bSuccess = device.setSampleRate(SAMPLE_RATE_CODE_500_HZ);
  Serial.println(bSuccess ? "\nSuccessfully set sample rate." : "\nFailed to set sample rate.");
*/
  return bSuccess;
}

// Querries the current device settings (motor speed and sample rate)
// and prints them to the console
bool verifyCurrentDeviceSettings()
{
  // Read the current motor speed and sample rate
  int32_t currentMotorSpeed = device.getMotorSpeed();
  if (currentMotorSpeed < 0)
  {
    Serial.println("\nFailed to get current motor speed");
    return false;
  }
  int32_t currentSampleRate = device.getSampleRate();
  if (currentSampleRate < 0)
  {
    Serial.println("\nFailed to get current sample rate");
    return false;
  }

  // Report the motor speed and sample rate to the computer terminal
  Serial.println("\nMotor Speed Setting: " + String(currentMotorSpeed) + " HZ");
  Serial.println("Sample Rate Setting: " + String(currentSampleRate) + " HZ");

  return true;
}

// Initiates the data collection phase (begins scanning)
bool beginDataCollectionPhase()
{
  // Attempt to start scanning
  Serial.println("\nWaiting for motor speed to stabilize and calibration routine to complete...");
  bool bSuccess = device.startScanning();
  Serial.println(bSuccess ? "\nSuccessfully initiated scanning..." : "\nFailed to start scanning.");
  if (bSuccess)
    Serial.println("\nGathering 3 scans...");
  return bSuccess;
}
```

```
// Gathers individual sensor readings until 3 complete scans have been collected
void gatherSensorReading()
{
  // attempt to get the next scan packet
  // Note: getReading() will write values into the "reading" variable
  if (device.getReading(reading))
  {
    // check if this reading was the very first reading of a new 360 degree scan
    if (reading.bIsSync)
      scanCount++;

    // don't collect more than 3 scans
    if (scanCount > 3)
      return;

    // store the info for this sample
    syncValues[sampleCount] = reading.bIsSync;
    angles[sampleCount] = reading.angle;
    distances[sampleCount] = reading.distance;
    signalStrengths[sampleCount] = reading.signalStrength;

    // increment sample count
    sampleCount++;
  }
}

// Terminates the data collection phase (stops scanning)
bool stopDataCollectionPhase()
{
  // Attempt to stop scanning
  bool bSuccess = device.stopScanning();

  Serial.println(bSuccess ? "\nSuccessfully stopped scanning." : "\nFailed to stop scanning.");
  return bSuccess;
}

// Prints the collected data to the console
// (only prints the complete scans, ignores the first partial)
void printCollectedData()
{
  Serial.println("\nPrinting info for the collected scans (NOT REAL-TIME):");

  int indexOfFirstSyncReading = 0;
  // don't print the trailing readings from the first partial scan
  while (!syncValues[indexOfFirstSyncReading])
```

```
 {
  indexOfFirstSyncReading++;
 }
 // print the readings for all the complete scans
 for (int i = indexOfFirstSyncReading; i < sampleCount; i++)
 {
  if (syncValues[i])
  {
   Serial.println("\n----------------------NEW SCAN----------------------");
  }
  Serial.println("Angle: " + String(angles[i], 3) + ", Distance: " + String(distances[i]) + ", Signal
Strength: " + String(signalStrengths[i]));
 }
}

// Resets the variables and state so the sequence can be repeated
void reset()
{
 scanCount = 0;
 sampleCount = 0;
 // reset the sensor
 device.reset();
 delay(50);
 Serial.flush();
 userInput = "";
 Serial.println("\n\nWhenever you are ready, type \"start\" to to begin the sequence...");
 currentState = 0;
}
```

APPENDIX A.3 Plotting Data code (MATLAB)

```
filename = 'SCANSE12.CSV';
ScanseData = xlsread(filename);

wdist = 0; %distance threshold for an obstacle---furthest distance that
it will detect an obstacle
lowangle = 0;% lowest angle that you want to see in FOV
highangle = 360; % highest angle that you want to see in FOV

num = length(ScanseData); %measures the length of the Scanse data
thetad = ScanseData(:,1); %pulls the azimuth data from the Scanse data
r = ScanseData(:,3); %pulls the distance data from the Scanse data

wtheta=[];
wr=[];

%-------------------------------------------------------------------------
----------------------%
```

```
for k=1:num % compares the measured angle to see if ots the FOV it was
given
    if (thetad(k) >= lowangle) && (thetad(k) <= highangle)
        thetad(k) = thetad(k);
    end
    if r(k)> 4000
        r(k) = 0;
    end
end

xdist=r.*cosd(thetad);%produces the x axis distance
ydist=r.*sind(thetad);%produces the y axis distance

thetar = deg2rad(thetad); %converts scanse theta to radians from
degrees

for k =1:num %compares every data point to the warning distance

    if r(k) < wdist && r(k) ~= -1
        wtheta(k) = thetar(k);
        wr(k) = r(k);
    end
end
[pathstr,name,ext] = fileparts (filename); %added these next two lines
for sure. (25 & 26)
figure ('name', name, 'numbertitle', 'off');
polarscatter(thetar, r, 'bo') %plots the Scanse data in a polar graph
hold on
polarscatter(wtheta, wr, 'r*')
hold off
polarscatter(thetar, r, 'bo') %plots the Scanse data in a polar graph
hold on
polarscatter(wtheta, wr, 'r*')
hold off

legend('data points','obstacle')
```

APPENDIX A.4 Sample Data

----------NEW SCAN----------

Angle	3	Distance	1089	Signal Strength	45
Angle	4	Distance	1096	Signal Strength	183
Angle	12	Distance	1105	Signal Strength	40
Angle	19	Distance	1	Signal Strength	20
Angle	26	Distance	1	Signal Strength	20
Angle	34	Distance	1	Signal Strength	25
Angle	41	Distance	1	Signal Strength	20

Angle	48	Distance	1	Signal Strength	30
Angle	56	Distance	1	Signal Strength	20
Angle	63	Distance	1	Signal Strength	25
Angle	71	Distance	1	Signal Strength	20
Angle	78	Distance	1	Signal Strength	45
Angle	86	Distance	3247	Signal Strength	45
Angle	90	Distance	717	Signal Strength	98
Angle	92	Distance	722	Signal Strength	191
Angle	93	Distance	724	Signal Strength	191
Angle	95	Distance	721	Signal Strength	191
Angle	102	Distance	1	Signal Strength	35
Angle	110	Distance	1	Signal Strength	40
Angle	117	Distance	134	Signal Strength	55
Angle	119	Distance	128	Signal Strength	199
Angle	120	Distance	127	Signal Strength	199
Angle	122	Distance	134	Signal Strength	199
Angle	129	Distance	1	Signal Strength	35
Angle	136	Distance	1	Signal Strength	30
Angle	144	Distance	1	Signal Strength	25
Angle	151	Distance	1	Signal Strength	15
Angle	159	Distance	1	Signal Strength	20
Angle	166	Distance	1	Signal Strength	25
Angle	173	Distance	1	Signal Strength	20
Angle	181	Distance	1	Signal Strength	25
Angle	188	Distance	1	Signal Strength	20
Angle	196	Distance	1487	Signal Strength	40
Angle	203	Distance	1	Signal Strength	30
Angle	210	Distance	1	Signal Strength	25
Angle	218	Distance	1	Signal Strength	20
Angle	226	Distance	1	Signal Strength	25
Angle	233	Distance	1	Signal Strength	20
Angle	240	Distance	1	Signal Strength	20
Angle	248	Distance	1	Signal Strength	20
Angle	255	Distance	1	Signal Strength	30
Angle	263	Distance	1	Signal Strength	25
Angle	270	Distance	5	Signal Strength	25
Angle	277	Distance	1	Signal Strength	25
Angle	285	Distance	1	Signal Strength	20
Angle	292	Distance	1	Signal Strength	25
Angle	300	Distance	1	Signal Strength	30
Angle	307	Distance	1	Signal Strength	25

Angle	314	Distance	1	Signal Strength	35
Angle	322	Distance	1	Signal Strength	20
Angle	329	Distance	1	Signal Strength	20
Angle	337	Distance	1	Signal Strength	25
Angle	344	Distance	3294	Signal Strength	20
Angle	351	Distance	1	Signal Strength	20
Angle	359	Distance	1	Signal Strength	20

----------NEW SCAN----------

Angle	4	Distance	1091	Signal Strength	84
Angle	11	Distance	1087	Signal Strength	30
Angle	19	Distance	1	Signal Strength	25
Angle	26	Distance	1	Signal Strength	25
Angle	34	Distance	1	Signal Strength	40
Angle	41	Distance	1	Signal Strength	25
Angle	48	Distance	1	Signal Strength	40
Angle	56	Distance	579	Signal Strength	15
Angle	63	Distance	1	Signal Strength	25
Angle	71	Distance	1	Signal Strength	35
Angle	78	Distance	1	Signal Strength	40
Angle	86	Distance	3244	Signal Strength	50
Angle	90	Distance	716	Signal Strength	98
Angle	92	Distance	723	Signal Strength	191
Angle	93	Distance	721	Signal Strength	191
Angle	101	Distance	719	Signal Strength	45
Angle	108	Distance	1	Signal Strength	20
Angle	116	Distance	143	Signal Strength	60
Angle	117	Distance	129	Signal Strength	199
Angle	118	Distance	128	Signal Strength	199
Angle	120	Distance	126	Signal Strength	199
Angle	121	Distance	135	Signal Strength	199
Angle	129	Distance	1	Signal Strength	25
Angle	136	Distance	1	Signal Strength	30
Angle	144	Distance	1	Signal Strength	25
Angle	151	Distance	1	Signal Strength	35
Angle	159	Distance	1	Signal Strength	25
Angle	166	Distance	6681	Signal Strength	25
Angle	173	Distance	1	Signal Strength	25
Angle	181	Distance	1	Signal Strength	35
Angle	188	Distance	1	Signal Strength	20
Angle	195	Distance	1493	Signal Strength	35

Angle	203	Distance	1	Signal Strength	45
Angle	210	Distance	1	Signal Strength	40
Angle	218	Distance	1	Signal Strength	20
Angle	225	Distance	1	Signal Strength	15
Angle	233	Distance	1	Signal Strength	20
Angle	240	Distance	172	Signal Strength	20
Angle	247	Distance	1	Signal Strength	20
Angle	255	Distance	1	Signal Strength	35
Angle	262	Distance	1	Signal Strength	20
Angle	270	Distance	1	Signal Strength	20
Angle	277	Distance	1	Signal Strength	20
Angle	284	Distance	1	Signal Strength	20
Angle	292	Distance	1	Signal Strength	25
Angle	299	Distance	1	Signal Strength	25
Angle	307	Distance	1	Signal Strength	20
Angle	314	Distance	5348	Signal Strength	20
Angle	321	Distance	1	Signal Strength	25
Angle	329	Distance	1	Signal Strength	30
Angle	336	Distance	1	Signal Strength	25
Angle	344	Distance	1	Signal Strength	20
Angle	351	Distance	1	Signal Strength	25
Angle	358	Distance	1	Signal Strength	20

APPENDIX A.5 Sensor Calculations

		DATA POINTS PER REVOLUTION									
Motor speed (Hz)	1	2	3	4	5	6	7	8	9	10	
samples (n)											
500	500.00	250.00	166.67	125.00	100.00	83.33	71.43	62.50	55.56	50.00	
750	750.00	375.00	250.00	187.50	150.00	125.00	107.14	93.75	83.33	75.00	
1000	1000.00	500.00	333.33	250.00	200.00	166.67	142.86	125.00	111.11	100.00	
	AVERAGE DEGREES PER REVOLUTION										
500	0.72	1.44	2.16	2.88	3.60	4.32	5.04	5.76	6.48	7.20	
750	0.48	0.96	1.44	1.92	2.40	2.88	3.36	3.84	4.32	4.80	
1000	0.36	0.72	1.08	1.44	1.80	2.16	2.52	2.88	3.24	3.60	

		Spinning frequency of the LIDAR (Hz)								
distance to obstacle (cm)	1	2	3	4	5	6	7	8	9	10
	ARCH LENGTH (CENTIMETERS) BETWEEN TWO DATA POINTS (500 Hz)									
500	6.28	12.57	18.85	25.13	31.42	37.70	43.98	50.27	56.55	62.83

1000	12.57	25.13	37.70	50.27	62.83	75.40	87.96	100.53	113.10	125.66
1500	18.85	37.70	56.55	75.40	94.25	113.10	131.95	150.80	169.65	188.50
2000	25.13	50.27	75.40	100.53	125.66	150.80	175.93	201.06	226.19	251.33
2500	31.42	62.83	94.25	125.66	157.08	188.50	219.91	251.33	282.74	314.16
3000	37.70	75.40	113.10	150.80	188.50	226.19	263.89	301.59	339.29	376.99
3500	43.98	87.96	131.95	175.93	219.91	263.89	307.88	351.86	395.84	439.82
4000	50.27	100.53	150.80	201.06	251.33	301.59	351.86	402.12	452.39	502.65

ARCH LENGTH (CENTIMETERS) BETWEEN TWO DATA POINTS (750 Hz)

500	4.19	8.38	12.57	16.76	20.94	25.13	29.32	33.51	37.70	41.89
1000	8.38	16.76	25.13	33.51	41.89	50.27	58.64	67.02	75.40	83.78
1500	12.57	25.13	37.70	50.27	62.83	75.40	87.96	100.53	113.10	125.66
2000	16.76	33.51	50.27	67.02	83.78	100.53	117.29	134.04	150.80	167.55
2500	20.94	41.89	62.83	83.78	104.72	125.66	146.61	167.55	188.50	209.44
3000	25.13	50.27	75.40	100.53	125.66	150.80	175.93	201.06	226.19	251.33
3500	29.32	58.64	87.96	117.29	146.61	175.93	205.25	234.57	263.89	293.22
4000	33.51	67.02	100.53	134.04	167.55	201.06	234.57	268.08	301.59	335.10

ARCH LENGTH (CENTIMETERS) BETWEEN TWO DATA POINTS (1000 Hz)

500	3.14	6.28	9.42	12.57	15.71	18.85	21.99	25.13	28.27	31.42
1000	6.28	12.57	18.85	25.13	31.42	37.70	43.98	50.27	56.55	62.83
1500	9.42	18.85	28.27	37.70	47.12	56.55	65.97	75.40	84.82	94.25
2000	12.57	25.13	37.70	50.27	62.83	75.40	87.96	100.53	113.10	125.66
2500	15.71	31.42	47.12	62.83	78.54	94.25	109.96	125.66	141.37	157.08
3000	18.85	37.70	56.55	75.40	94.25	113.10	131.95	150.80	169.65	188.50
3500	21.99	43.98	65.97	87.96	109.96	131.95	153.94	175.93	197.92	219.91
4000	25.13	50.27	75.40	100.53	125.66	150.80	175.93	201.06	226.19	251.33

Spinning frequency of the LIDAR (Hz)

distance to obstacle (cm)	1	2	3	4	5	6	7	8	9	10

LINEAR DISTANCE (CENTIMETERS) BETWEEN TWO DATA POINTS (500 Hz)

500	6.28	12.57	18.85	25.13	31.41	37.69	43.97	50.24	56.52	62.79
1000	12.57	25.13	37.70	50.26	62.82	75.38	87.94	100.49	113.04	125.58
1500	18.85	37.70	56.55	75.39	94.23	113.07	131.90	150.73	169.56	188.37
2000	25.13	50.26	75.39	100.52	125.64	150.76	175.87	200.98	226.07	251.16
2500	31.42	62.83	94.24	125.65	157.05	188.45	219.84	251.22	282.59	313.95
3000	37.70	75.40	113.09	150.78	188.46	226.14	263.81	301.47	339.11	376.74
3500	43.98	87.96	131.94	175.91	219.88	263.83	307.78	351.71	395.63	439.53
4000	50.27	100.53	150.79	201.04	251.29	301.52	351.74	401.95	452.15	502.32

LINEAR DISTANCE (CENTIMETERS) BETWEEN TWO DATA POINTS (750 Hz)

500	4.19	8.38	12.57	16.75	20.94	25.13	29.32	33.50	37.69	41.88
1000	8.38	16.75	25.13	33.51	41.88	50.26	58.63	67.01	75.38	83.75

1500	12.57	25.13	37.70	50.26	62.83	75.39	87.95	100.51	113.07	125.63
2000	16.76	33.51	50.26	67.02	83.77	100.52	117.27	134.02	150.76	167.50
2500	20.94	41.89	62.83	83.77	104.71	125.65	146.59	167.52	188.45	209.38
3000	25.13	50.26	75.40	100.53	125.65	150.78	175.90	201.02	226.14	251.25
3500	29.32	58.64	87.96	117.28	146.60	175.91	205.22	234.53	263.83	293.13
4000	33.51	67.02	100.53	134.04	167.54	201.04	234.54	268.03	301.52	335.01

LINEAR DISTANCE (CENTIMETERS) BETWEEN TWO DATA POINTS (1000 Hz)

500	3.14	6.28	9.42	12.57	15.71	18.85	21.99	25.13	28.27	31.41
1000	6.28	12.57	18.85	25.13	31.41	37.70	43.98	50.26	56.54	62.82
1500	9.42	18.85	28.27	37.70	47.12	56.55	65.97	75.39	84.81	94.23
2000	12.57	25.13	37.70	50.26	62.83	75.39	87.96	100.52	113.08	125.64
2500	15.71	31.42	47.12	62.83	78.54	94.24	109.95	125.65	141.35	157.05
3000	18.85	37.70	56.55	75.40	94.24	113.09	131.94	150.78	169.62	188.46
3500	21.99	43.98	65.97	87.96	109.95	131.94	153.93	175.91	197.89	219.88
4000	25.13	50.27	75.40	100.53	125.66	150.79	175.92	201.04	226.16	251.29

AVERAGE DATA POINTS FOR OBSTACLES (RPLIDAR)

DATA POINTS PER REVOLUTION

Motor speed (Hz)	1	2	3	4	5	6	7	8	9	10
samples (n)										
1000	1000.00	500.00	333.33	250.00	200.00	166.67	142.86	125.00	111.11	100.00
750	750.00	375.00	250.00	187.50	150.00	125.00	107.14	93.75	83.33	75.00
500	500.00	250.00	166.67	125.00	100.00	83.33	71.43	62.50	55.56	50.00

CM TO FEET 30.48

AVERAGE NUMBER OF DATA POINTS PER FOOT **1**

Spinning frequency of the LIDAR (500Hz)

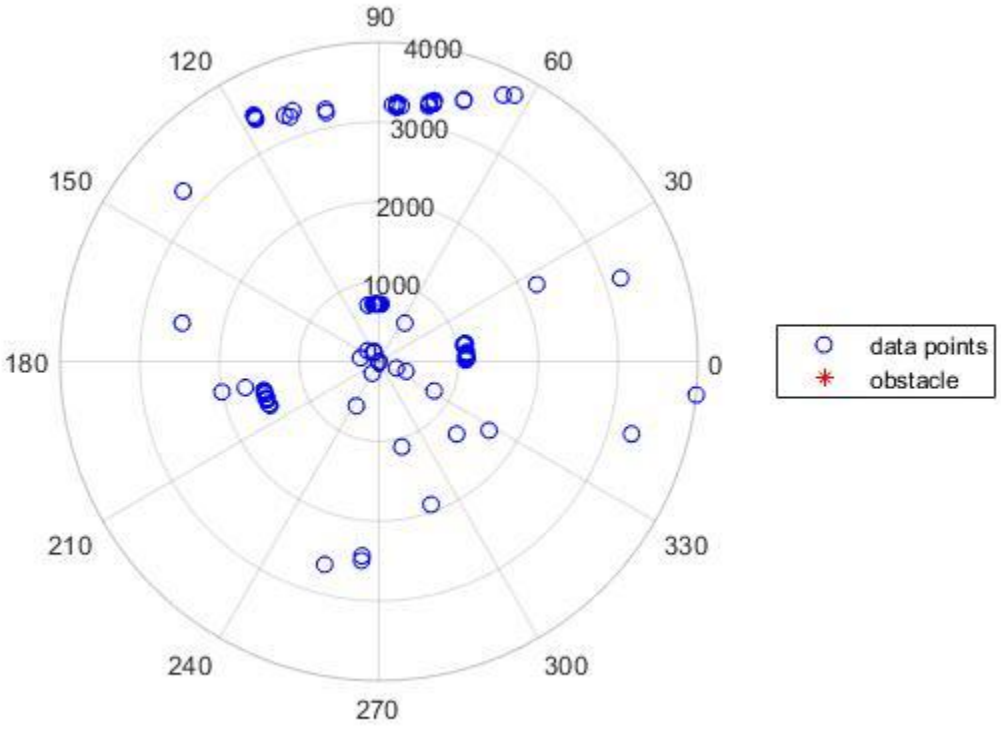
distance to obstacle (cm)	1	2	3	4	5	6	7	8	9	10
500	4.8511	2.4256	1.6171	1.2129	0.9704	0.8087	0.6932	0.6066	0.5393	0.4854
1000	2.4255	1.2128	0.8086	0.6064	0.4852	0.4043	0.3466	0.3033	0.2696	0.2427
1500	1.617	0.8085	0.539	0.4043	0.3235	0.2696	0.2311	0.2022	0.1798	0.1618
2000	1.2128	0.6064	0.4043	0.3032	0.2426	0.2022	0.1733	0.1517	0.1348	0.1214
2500	0.9702	0.4851	0.3234	0.2426	0.1941	0.1617	0.1386	0.1213	0.1079	0.0971
3000	0.8085	0.4043	0.2695	0.2021	0.1617	0.1348	0.1155	0.1011	0.0899	0.0809
3500	0.693	0.3465	0.231	0.1733	0.1386	0.1155	0.099	0.0867	0.077	0.0693
4000	0.6064	0.3032	0.2021	0.1516	0.1213	0.1011	0.0867	0.0758	0.0674	0.0607

Spinning frequency of the LIDAR (750Hz)

500	7.2766	3.6383	2.4256	1.8192	1.4554	1.2129	1.0397	0.9097	0.8087	0.7279
1000	3.6383	1.8192	1.2128	0.9096	0.7277	0.6064	0.5198	0.4549	0.4043	0.3639
1500	2.4255	1.2128	0.8085	0.6064	0.4851	0.4043	0.3466	0.3032	0.2696	0.2426

2000	1.8191	0.9096	0.6064	0.4548	0.3639	0.3032	0.2599	0.2274	0.2022	0.182
2500	1.4553	0.7277	0.4851	0.3638	0.2911	0.2426	0.2079	0.1819	0.1617	0.1456
3000	1.2128	0.6064	0.4043	0.3032	0.2426	0.2021	0.1733	0.1516	0.1348	0.1213
3500	1.0395	0.5198	0.3465	0.2599	0.2079	0.1733	0.1485	0.13	0.1155	0.104
4000	0.9096	0.4548	0.3032	0.2274	0.1819	0.1516	0.13	0.1137	0.1011	0.091
Spinning frequency of the LIDAR (1000Hz)										
500	9.7021	4.8511	3.2341	2.4256	1.9405	1.6171	1.3861	1.2129	1.0782	0.9704
1000	4.8511	2.4255	1.617	1.2128	0.9702	0.8086	0.6931	0.6064	0.5391	0.4852
1500	3.234	1.617	1.078	0.8085	0.6468	0.539	0.462	0.4043	0.3594	0.3235
2000	2.4255	1.2128	0.8085	0.6064	0.4851	0.4043	0.3465	0.3032	0.2695	0.2426
2500	1.9404	0.9702	0.6468	0.4851	0.3881	0.3234	0.2772	0.2426	0.2156	0.1941
3000	1.617	0.8085	0.539	0.4043	0.3234	0.2695	0.231	0.2021	0.1797	0.1617
3500	1.386	0.693	0.462	0.3465	0.2772	0.231	0.198	0.1733	0.154	0.1386
4000	1.2128	0.6064	0.4043	0.3032	0.2426	0.2021	0.1733	0.1516	0.1348	0.1213

APPENDIX A.6 Sample Data Plot



APPENDIX B RPLIDAR

RPLIDAR

Motor speed (Hz)	DATA POINTS PER REVOLUTION										
samples (n)	5	6	7	8	9	10	11	12	13	14	15
4000	800.00	666.67	571.43	500.00	444.44	400.00	363.64	333.33	307.69	285.71	266.67
	AVERAGE DEGREES PER REVOLUTION										
4000	0.45	0.54	0.63	0.72	0.81	0.90	0.99	1.08	1.17	1.26	1.35

distance to obstacle (mm)	Spinning frequency of the LIDAR (Hz)										
	5	6	7	8	9	10	11	12	13	14	15
	ARCH LENGTH (MILLIMETERS) BETWEEN TWO DATA POINTS (4000 Hz)										
1000	7.85	9.42	11.00	12.57	14.14	15.71	17.28	18.85	20.42	21.99	23.56
2000	15.71	18.85	21.99	25.13	28.27	31.42	34.56	37.70	40.84	43.98	47.12
3000	23.56	28.27	32.99	37.70	42.41	47.12	51.84	56.55	61.26	65.97	70.69
4000	31.42	37.70	43.98	50.27	56.55	62.83	69.12	75.40	81.68	87.96	94.25
5000	39.27	47.12	54.98	62.83	70.69	78.54	86.39	94.25	102.10	109.96	117.81
6000	47.12	56.55	65.97	75.40	84.82	94.25	103.67	113.10	122.52	131.95	141.37
7000	54.98	65.97	76.97	87.96	98.96	109.96	120.95	131.95	142.94	153.94	164.93
8000	62.83	75.40	87.96	100.53	113.10	125.66	138.23	150.80	163.36	175.93	188.50

distance to obstacle (cm)	Spinning frequency of the LIDAR (Hz)										
	5	6	7	8	9	10	11	12	13	14	15
	LINEAR DISTANCE (MILLIMETERS) BETWEEN TWO DATA POINTS (4000 Hz)										
1000	7.85	9.42	11.00	12.57	14.14	15.71	17.28	18.85	20.42	21.99	23.56
2000	15.71	18.85	21.99	25.13	28.27	31.42	34.56	37.70	40.84	43.98	47.12
3000	23.56	28.27	32.99	37.70	42.41	47.12	51.84	56.55	61.26	65.97	70.68
4000	31.42	37.70	43.98	50.27	56.55	62.83	69.11	75.40	81.68	87.96	94.25
5000	39.27	47.12	54.98	62.83	70.69	78.54	86.39	94.25	102.10	109.95	117.81
6000	47.12	56.55	65.97	75.40	84.82	94.25	103.67	113.10	122.52	131.94	141.37
7000	54.98	65.97	76.97	87.96	98.96	109.95	120.95	131.94	142.94	153.93	164.93
8000	62.83	75.40	87.96	100.53	113.10	125.66	138.23	150.79	163.36	175.93	188.49

	Spinning frequency of the LIDAR (Hz)										
distance to obstacle (cm)	1	2	3	4	5	6	7	8	9	10	11
	DIFFERENCE (MILLIMETERS) IN THE TWO DIFFERENT METHODS (4000 Hz)										
1000	0.0000	0.0000	0.0001	0.0001	0.0001	0.0002	0.0002	0.0003	0.0004	0.0004	0.0005
2000	0.0000	0.0001	0.0001	0.0002	0.0002	0.0003	0.0004	0.0006	0.0007	0.0009	0.0011
3000	0.0001	0.0001	0.0002	0.0002	0.0004	0.0005	0.0006	0.0008	0.0011	0.0013	0.0016
4000	0.0001	0.0001	0.0002	0.0003	0.0005	0.0006	0.0009	0.0011	0.0014	0.0018	0.0022
5000	0.0001	0.0002	0.0003	0.0004	0.0006	0.0008	0.0011	0.0014	0.0018	0.0022	0.0027
6000	0.0001	0.0002	0.0003	0.0005	0.0007	0.0010	0.0013	0.0017	0.0021	0.0027	0.0033
7000	0.0001	0.0002	0.0004	0.0006	0.0008	0.0011	0.0015	0.0020	0.0025	0.0031	0.0038
8000	0.0002	0.0003	0.0004	0.0007	0.0009	0.0013	0.0017	0.0022	0.0028	0.0035	0.0044

AVERAGE DATA POINTS FOR OBSTACLES (RPLIDAR)

	DATA POINTS PER REVOLUTION										
Motor speed (Hz)	5	6	7	8	9	10	11	12	13	14	15
samples (n)											
4000	800.00	666.67	571.43	500.00	444.44	400.00	363.64	333.33	307.69	285.71	266.67

MM TO FEET 304.8

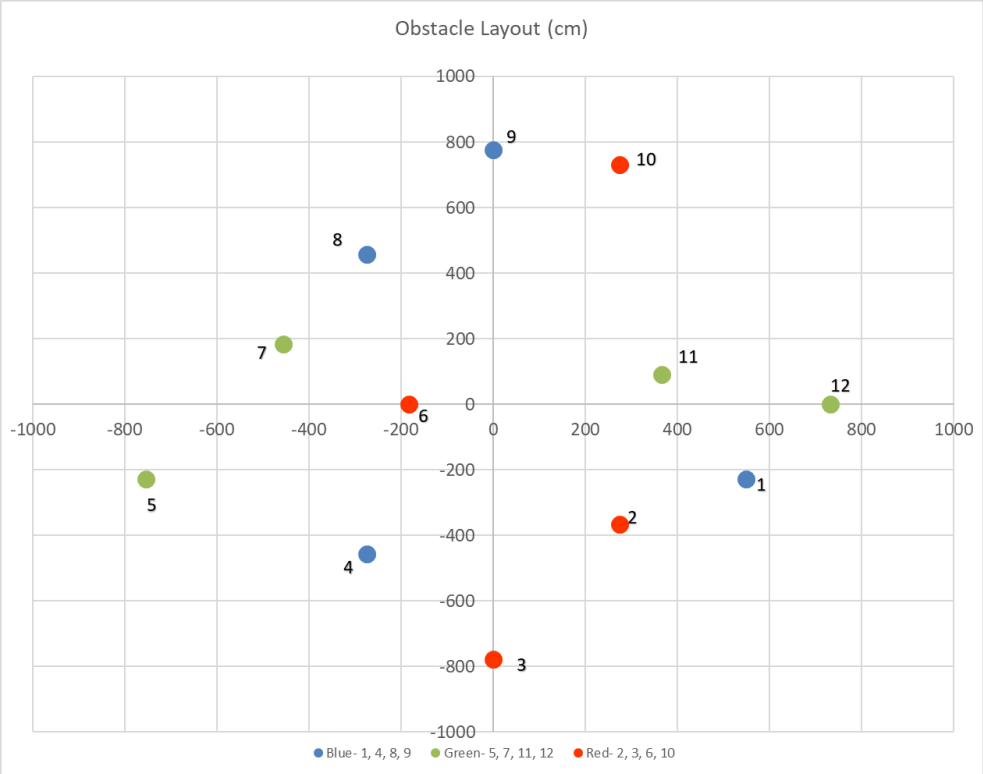
AVERAGE NUMBER OF DATA POINTS PER FOOT **1**

	Spinning frequency of the LIDAR (Hz)										
distance to obstacle (mm)	5	6	7	8	9	10	11	12	13	14	15
1000	38.808	32.34	27.72	24.255	21.56	19.404	17.64	16.17	14.927	13.86	12.936
2000	19.404	16.17	13.86	12.128	10.78	9.7022	8.8202	8.0852	7.4633	6.9302	6.4682
3000	12.936	10.78	9.2401	8.0851	7.1868	6.4681	5.8801	5.3901	4.9755	4.6201	4.3121
4000	9.7021	8.0851	6.9301	6.0638	5.3901	4.8511	4.4101	4.0426	3.7316	3.4651	3.2341
5000	7.7617	6.4681	5.5441	4.8511	4.3121	3.8809	3.5281	3.2341	2.9853	2.7721	2.5873
6000	6.4681	5.3901	4.6201	4.0426	3.5934	3.2341	2.9401	2.6951	2.4878	2.3101	2.1561
7000	5.5441	4.6201	3.9601	3.4651	3.0801	2.7721	2.5201	2.3101	2.1324	1.9801	1.8481
8000	4.8511	4.0426	3.465	3.0319	2.695	2.4255	2.205	2.0213	1.8658	1.7326	1.6171

APPENDIX B.1 Test Setup

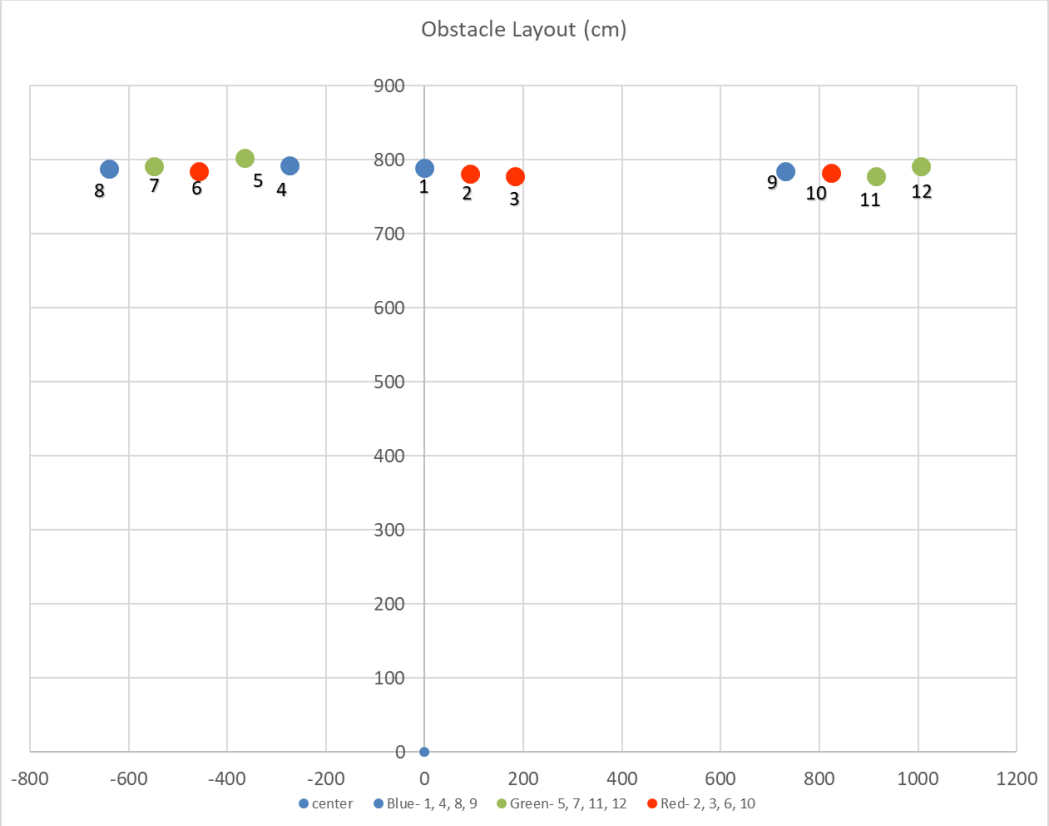
APPENDIX B.1.1 Occupancy Map Setup

Obstacle number	X Coordinate columns	Y Coordinate rows	X Coordinate feet	Y Coordinate feet	X Coordinate cm	Y Coordinate cm	radial distance feet	radial distance cm	azimuth degrees	Measured
12	8	0	24	0	731.52	0	24	731.52	0.00	19.4
11	4	2	12	3	365.76	91.44	12.36931688	377.0167784	14.04	14.6
10	3	16	9	24	274.32	731.52	25.63201124	781.2637025	69.44	25.4
9	0	17	0	25.5	0	777.24	25.5	777.24	90.00	17.6
8	3	10	9	15	-274.32	457.2	17.49285568	533.1822413	120.96	26.1
7	5	4	15	6	-457.2	182.88	16.15549442	492.41947	158.20	6
6	2	0	6	0	-182.88	0	6	182.88	180.00	16.1
5	8.25	5	24.75	7.5	-754.38	-228.6	25.86140947	788.2557608	196.86	17.3
4	3	10	9	15	-274.32	-457.2	17.49285568	533.1822413	239.04	25.6
3	0	17	0	25.5	0	-777.24	25.5	777.24	270.00	26.1
2	3	8	9	12	274.32	-365.76	15	457.2	306.87	12.4
1	6	5	18	7.5	548.64	-228.6	19.5	594.36	337.38	24



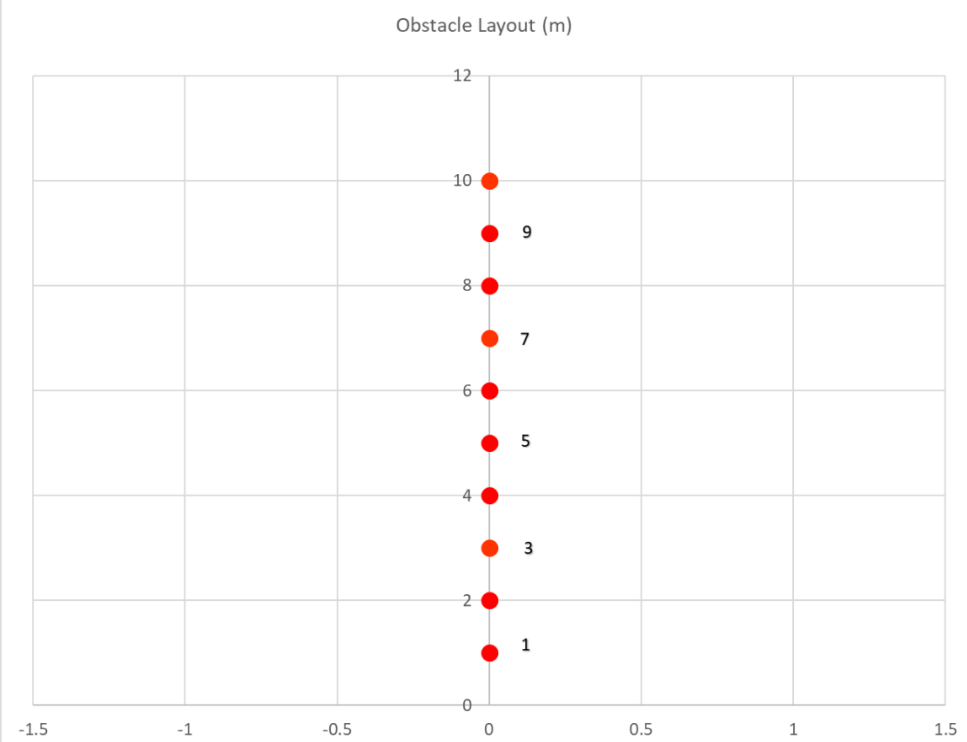
APPENDIX B.1.2 Straight Line Setup

Obstacle number	X Coordinate columns	Y Coordinate rows	X Coordinate feet	Y Coordinate feet	X Coordinate cm	Y Coordinate cm	radial distance feet	radial distance cm	azimuth degrees	Measured
12	11	17.3	33	25.95	1005.84	790.956	41.98097784	1279.580204	38.18	25.87
11	10	17	30	25.5	914.4	777.24	39.37321425	1200.09557	40.36	25.83
10	9	17.12	27	25.68	822.96	782.7264	37.26207724	1135.748114	43.56	26.22
9	8	17.16	24	25.74	731.52	784.5552	35.19300499	1072.682792	47.00	27.55
8	7	17.23	21	25.845	-640.08	787.7556	33.30111147	1015.017878	129.10	28.97
7	6	17.3	18	25.95	-548.64	790.956	31.58167982	962.6096008	124.75	29.9
6	5	17.17	15	25.755	-457.2	785.0124	29.80469804	908.4471961	120.22	31.65
5	4	17.56	12	26.34	-365.76	802.8432	28.944699	882.2344254	114.49	33.45
4	3	17.32	9	25.98	-274.32	791.8704	27.49473404	838.0394936	109.11	35.18
3	2	17	6	25.5	182.88	777.24	26.19637379	798.4654733	76.76	37.18
2	1	17.08	3	25.62	91.44	780.8976	25.79504604	786.2330032	83.32	39.34
1	0	17.25	0	25.875	0	788.67	25.875	788.67	90.00	41.92



APPENDIX B.1.3 Final RPLIDAR Test Setup

Obstacle number	X Coordinate m	Y Coordinate m	radial distance m	azimuth degrees
1	0	1	1	90.00
2	0	2	2	90.00
3	0	3	3	90.00
4	0	4	4	90.00
5	0	5	5	90.00
6	0	6	6	90.00
7	0	7	7	90.00
8	0	8	8	90.00
9	0	9	9	90.00
10	0	10	10	90.00



APPENDIX B.2 Data Collection code (Arduino)

```
/*
 *
 * USAGE:
 * --------------------------------
 * 1. Download this sketch code to your Arduino board
 * 2. Connect the RPLIDAR's serial port (RX/TX/GND) to your Arduino Com port 2
 *                    Yellow ----------------> pin 17
 *                    Green  ----------------> pin 16
 *                    GND    ----------------> GND
 * 3. Connect the RPLIDAR's motor ctrl pin to the Arduino board pin 3
 *                    Blue   ----------------> Pin 3
 *
 * 4. Connect Leds to pins 24, 26, 28
 * 5. connect Button to pin 22
 * 6. Plug in SD card Shield
 */

/*
 * Copyright (c) 2014, RoboPeak
 * All rights reserved.
 * RoboPeak.com
 */

// This sketch code is based on the RPLIDAR driver library provided by RoboPeak
#include <RPLidar.h>
#include <SD.h>

// create an driver instance
RPLidar RPlidar;

#define RPLIDAR_MOTOR 3 // The PWM pin for control the speed of RPLIDAR's motor.
                        // This pin should connected with the RPLIDAR's MOTOCTRL signal

// LED and button pin convigurations
const byte red_LED = 24;
const byte clear_LED = 28;
const byte button = 22;

uint16_t sampleCount = 0;   // keeps track of how many samples have been collected
bool firstRun = true;

//Timing variables
unsigned long sampleTime = 10000; // time to sample in milliseconds
```

```
unsigned long startTime;


// Arrays to store attributes of collected scans
#define numberOfSamples 600          //  = (sample rate / motor speed) * number pf scans
float distances[numberOfSamples];     // in mm
float angles[numberOfSamples];        // in degrees
bool startBit[numberOfSamples];       // 1 -> first reading of new scan, 0 otherwise (syncValues)
uint8_t quality[numberOfSamples];     // 0:255, higher is better (signal Strength)

// Finite States for the program sequence
const byte STATE_WAIT_FOR_USER_INPUT = 0;
const byte STATE_BEGIN_DATA_ACQUISITION = 1;
const byte STATE_GATHER_DATA = 2;
const byte STATE_STOP_DATA_ACQUISITION = 3;
const byte STATE_REPORT_COLLECTED_DATA = 4;
const byte STATE_RESET = 5;
const byte STATE_ERROR = 6;

// Current state in the program sequence
byte currentState;

// String to collect user input over serial
String userInput = "";

//SD stuff
File logfile;
const byte chipSelect = 10;


void setup() {
  // bind the RPLIDAR driver to the arduino hardware serial
  RPlidar.begin(Serial2);

  Serial.begin(250000);

  // set pin modes
  pinMode(RPLIDAR_MOTOR, OUTPUT);

  //------------ SD setup ---------------------
// initialize the SD card
  // make sure that the default chip select pin is set to
  // output, even if you don't use it:
  pinMode(chipSelect, OUTPUT);

  Serial.print("Initializing SD card.... ");
```

```
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
    // don't do anything more:
  }
  else{
    Serial.println("Card initialized");
    // don't do anything more:
  }

  // initialize counter variables and reset the current state
  reset();
}

void loop() {
  switch (currentState)
  {
  case STATE_WAIT_FOR_USER_INPUT:
     while (Serial.available()){
       userInput += (char)Serial.read();
     }
     // create a new file with an original name (include date first)
     char filename[] = "529RPL00.CSV";
     if (userInput.indexOf("start") != -1){
       Serial.println(F("Registered user start."));

       Serial.print("Attempting to create logFile... ");
       for (uint8_t i = 0; i < 100; i++) {
         filename[6] = i/10 + '0';
         filename[7] = i%10 + '0';
         if (! SD.exists(filename)) {
           // only open a new file if it doesn't exist
           logfile = SD.open(filename, FILE_WRITE);
           break;  // leave the loop!
         }
       }

       if (! logfile) {
         Serial.println("Couldn't initialize logfile");
         // don't do anything more:
       }
       else{
          Serial.println("Scucessfully created " + String(filename));
         // don't do anything more:
       }
       currentState = STATE_BEGIN_DATA_ACQUISITION;
```

```
      startTime = millis();
     }
     if (digitalRead(button) == HIGH){
      Serial.println(F("Registered user start."));

      // blink LED to confirm
      digitalWrite(red_LED, HIGH);


      Serial.print("Attempting to create logFile... ");
      for (uint8_t i = 0; i < 100; i++) {
       filename[6] = i/10 + '0';
       filename[7] = i%10 + '0';
       if (! SD.exists(filename)) {
         // only open a new file if it doesn't exist
         logfile = SD.open(filename, FILE_WRITE);
         break;  // leave the loop!
       }
      }

      if (! logfile) {
       Serial.println("Couldn't initialize logfile");
       // don't do anything more:
      }
      else{
         Serial.println("Successfully created " + String(filename));
       // don't do anything more:
      }
      currentState = STATE_BEGIN_DATA_ACQUISITION;
      startTime = millis();
     }

   break;
  case STATE_BEGIN_DATA_ACQUISITION:
   currentState = beginDataCollectionPhase() ? STATE_GATHER_DATA : STATE_ERROR;
   break;

  case STATE_GATHER_DATA:
   gatherSensorReading();
   if (sampleCount >= numberOfSamples)
    currentState = STATE_STOP_DATA_ACQUISITION;
   break;

  case STATE_STOP_DATA_ACQUISITION:
   currentState = stopDataCollectionPhase() ? STATE_REPORT_COLLECTED_DATA :
STATE_ERROR;
```

```
    break;

  case STATE_REPORT_COLLECTED_DATA:
    printCollectedData();
    if(startTime <= millis()- sampleTime){
      currentState = STATE_RESET;
    }
    else{
     // Serial.println(F("\nGathering more data\n"));
      sampleCount = 0;
      currentState = STATE_BEGIN_DATA_ACQUISITION;
    }
    break;

  case STATE_RESET:
    Serial.println(F("\n\nAttempting to reset and run the program again..."));
    reset();
    currentState = STATE_WAIT_FOR_USER_INPUT;
    break;

  default: // there was some error
    Serial.println(F("\n\nAn error occured. Attempting to reset and run program again..."));
    reset();
    currentState = STATE_WAIT_FOR_USER_INPUT;
    break;
  }

}

// checks if the user has communicated anything over serial
// looks for the user to send "start"
bool listenForUserInput(){
  while (Serial.available())
  {
    userInput += (char)Serial.read();
  }
  if (userInput.indexOf("start") != -1){
    Serial.println(F("Registered user start."));



    return true;
  }
  return false;
}
```

```
// Initiates the data collection phase (begins scanning)
bool beginDataCollectionPhase(){



 if(firstRun)
  Serial.println(F("Attempting to start the motor."));

 // start motor rotating at max allowed speed
  analogWrite(RPLIDAR_MOTOR, 255);
  if(firstRun == true){
   delay(10000);
   firstRun = false;
   Serial.println(F("firstRun = false."));
  }
  else {
   Serial.println(F("Failed to start motor."));
  }
  delay(50);


 // Attempt to start scanning
 // try to detect RPLIDAR...
   rplidar_response_device_info_t info;
   if (IS_OK(RPlidar.getDeviceInfo(info, 100))) {
      // detected...
      if(firstRun)
        Serial.println(F("Motor successfully started.\n\n Starting data collection.\n"));
      RPlidar.startScan();
       return true;
   }
   else {
   analogWrite(RPLIDAR_MOTOR, 0); //stop the rplidar motor

   // try to detect RPLIDAR...
   rplidar_response_device_info_t info;
   if (IS_OK(RPlidar.getDeviceInfo(info, 100))) {
      // detected...
      RPlidar.startScan();

      // start motor rotating at max speed
      analogWrite(RPLIDAR_MOTOR, 255);
      delay(1000);
   }
 }
```

```
}

// Gathers individual sensor readings until 3 complete scans have been collected
void gatherSensorReading(){
  digitalWrite(red_LED, LOW);
  if (IS_OK(RPlidar.waitPoint())) {
    //if(RPlidar.getCurrentPoint().quality != 0){
    if(0 == 0){
      // store the info for this sample
      startBit[sampleCount] = RPlidar.getCurrentPoint().startBit; //whether this point is belong to a
new scan
      angles[sampleCount] = RPlidar.getCurrentPoint().angle; //anglue value in degree
      distances[sampleCount] = RPlidar.getCurrentPoint().distance; //distance value in mm unit
      quality[sampleCount] = RPlidar.getCurrentPoint().quality; //quality of the current
measurement

  //    Serial.println(sampleCount);

  //Serial.println("A:" + String(angles[sampleCount]) + ":D:" + String(distances[sampleCount]));
//Uncomment to see data real time

      // increment sample count
      if(sampleCount < numberOfSamples){
        sampleCount++;
      }
    }
  }
}

// Terminates the data collection phase (stops scanning)
bool stopDataCollectionPhase(){
  // Attempt to stop scanning
  bool bSuccess = true;
  RPlidar.stop();

  if(firstRun)
    Serial.println(bSuccess ? F("\nSuccessfully stopped scanning.") : F("\nFailed to stop
scanning."));

  digitalWrite(clear_LED, HIGH);
  delay(250);
  digitalWrite(clear_LED, LOW);

  return bSuccess;
}
```

```
// Prints the collected data to the console
// (only prints the complete scans, ignores the first partial)
void printCollectedData(){
 Serial.println(F("\nPrinting info for the collected scans (NOT REAL-TIME):"));

 digitalWrite(red_LED, HIGH);

 int indexOfFirstSyncReading = 0;
 // don't print the trailing readings from the first partial scan
 while (!startBit[indexOfFirstSyncReading])
 {
  indexOfFirstSyncReading++;
 }
 // print the readings for all the complete scans
 for (int i = indexOfFirstSyncReading; i < sampleCount; i++)
 {
  Serial.println("Angle: " + String(angles[i], 3) + ", Distance: " + String(distances[i]) + ", Signal
Strength: " + String(quality[i]));
  logfile.println("Angle," + String(angles[i], 3) + ",Distance," + String(distances[i]) + ",Signal
Strength," + String(quality[i]));

  digitalWrite(red_LED, !digitalRead(red_LED)); //blink the red led while writing data
 }
 logfile.flush();
 digitalWrite(red_LED, LOW);
}

// Resets the variables and state so the sequence can be repeated
void reset(){
 sampleCount = 0;
 // reset the sensor
//  RPlidar.reset();
 delay(50);
 Serial.flush();
 //logfile.close();
 userInput = "";
 Serial.println(F("\n\nWhenever you are ready, type \"start\" to to begin the sequence..."));
 currentState = 0;
}
```

APPENDIX B.3 Plotting Data code (MATLAB)

```
RPLIDARdata = xlsread('RLIDAR05.CSV');%reads in all Scanse data

wdist = 100; %distance threshold for an obstacle---furthest distance
that it will detect an obstacle
lowangle = 90;% lowest angle that you want to see in FOV
```

```
highangle = 270; % highest angle that you want to see in FOV

num = length(RPLIDARdata); %measures the length of the Scanse data
thetad = RPLIDARdata(:,1); %pulls the azimuth data from the Scanse data
r = RPLIDARdata(:,3); %pulls the distance data from the Scanse data

wtheta=[];
wr=[];

%-------------------------------------------------------------------------
-------------------------------------------------%

for k=1:num % compares the measured angle to see if ots the FOV it was
given
    if (thetad(k) >= lowangle) && (thetad(k) <= highangle)
        thetad(k) = thetad(k);
    end
    if r(k)> 15000
        r(k) = 0;
    end
end

xdist=r.*cosd(thetad);%produces the x axis distance
ydist=r.*sind(thetad);%produces the y axis distance

thetar = deg2rad(thetad); %converts scanse theta to radians from
degrees

for k =1:num %compares every data point to the warning distance

    if r(k) < wdist && r(k) ~= -1
        wtheta(k) = thetar(k);
        wr(k) = r(k);
    end
end

polarscatter(thetar, r, 'bo') %plots the Scanse data in a polar graph
hold on
polarscatter(wtheta, wr, 'r*')
hold off

legend('data points','possible obstacle')
```

APPENDIX B.4 Sample Data

Angle:	357.484	Distance:	0	Signal	Strength:	0
Angle:	1.438	Distance:	0	Signal	Strength:	0
Angle:	5.391	Distance:	0	Signal	Strength:	0

Angle:	9.328	Distance:	0	Signal	Strength:	0
Angle:	5.844	Distance:	2569.25	Signal	Strength:	8
Angle:	17.234	Distance:	0	Signal	Strength:	0
Angle:	21.094	Distance:	0	Signal	Strength:	0
Angle:	25.031	Distance:	0	Signal	Strength:	0
Angle:	28.969	Distance:	0	Signal	Strength:	0
Angle:	32.891	Distance:	0	Signal	Strength:	0
Angle:	36.844	Distance:	0	Signal	Strength:	0
Angle:	40.766	Distance:	0	Signal	Strength:	0
Angle:	44.766	Distance:	0	Signal	Strength:	0
Angle:	48.688	Distance:	0	Signal	Strength:	0
Angle:	52.641	Distance:	0	Signal	Strength:	0
Angle:	56.563	Distance:	0	Signal	Strength:	0
Angle:	60.531	Distance:	0	Signal	Strength:	0
Angle:	64.469	Distance:	0	Signal	Strength:	0
Angle:	68.313	Distance:	0	Signal	Strength:	0
Angle:	72.234	Distance:	0	Signal	Strength:	0
Angle:	76.188	Distance:	0	Signal	Strength:	0
Angle:	80.109	Distance:	0	Signal	Strength:	0
Angle:	84.047	Distance:	0	Signal	Strength:	0
Angle:	87.969	Distance:	0	Signal	Strength:	0
Angle:	91.984	Distance:	0	Signal	Strength:	0
Angle:	95.938	Distance:	0	Signal	Strength:	0
Angle:	99.891	Distance:	0	Signal	Strength:	0
Angle:	103.844	Distance:	0	Signal	Strength:	0
Angle:	107.766	Distance:	0	Signal	Strength:	0
Angle:	111.703	Distance:	0	Signal	Strength:	0
Angle:	115.594	Distance:	0	Signal	Strength:	0
Angle:	119.531	Distance:	0	Signal	Strength:	0
Angle:	123.453	Distance:	0	Signal	Strength:	0
Angle:	127.391	Distance:	0	Signal	Strength:	0
Angle:	131.328	Distance:	0	Signal	Strength:	0
Angle:	128.109	Distance:	1834.5	Signal	Strength:	15
Angle:	139.156	Distance:	0	Signal	Strength:	0
Angle:	143.078	Distance:	0	Signal	Strength:	0
Angle:	146.984	Distance:	0	Signal	Strength:	0
Angle:	150.922	Distance:	0	Signal	Strength:	0
Angle:	147.156	Distance:	5032.5	Signal	Strength:	14
Angle:	158.766	Distance:	0	Signal	Strength:	0
Angle:	162.766	Distance:	0	Signal	Strength:	0
Angle:	159.063	Distance:	4003.5	Signal	Strength:	12

Angle:	163.141	Distance:	2852.5	Signal	Strength:	9
Angle:	174.563	Distance:	0	Signal	Strength:	0
Angle:	178.484	Distance:	0	Signal	Strength:	0
Angle:	182.438	Distance:	0	Signal	Strength:	0
Angle:	186.375	Distance:	0	Signal	Strength:	0
Angle:	182.422	Distance:	11001.5	Signal	Strength:	8
Angle:	186.375	Distance:	10990.5	Signal	Strength:	9
Angle:	190.297	Distance:	11023.75	Signal	Strength:	10
Angle:	194.234	Distance:	11388.5	Signal	Strength:	9
Angle:	206.031	Distance:	0	Signal	Strength:	0
Angle:	202.156	Distance:	11879.75	Signal	Strength:	9
Angle:	213.906	Distance:	0	Signal	Strength:	0
Angle:	217.875	Distance:	0	Signal	Strength:	0
Angle:	221.781	Distance:	0	Signal	Strength:	0
Angle:	225.734	Distance:	0	Signal	Strength:	0
Angle:	229.656	Distance:	0	Signal	Strength:	0
Angle:	233.625	Distance:	0	Signal	Strength:	0
Angle:	237.609	Distance:	0	Signal	Strength:	0
Angle:	241.563	Distance:	0	Signal	Strength:	0
Angle:	245.516	Distance:	0	Signal	Strength:	0
Angle:	249.453	Distance:	0	Signal	Strength:	0
Angle:	253.375	Distance:	0	Signal	Strength:	0
Angle:	257.344	Distance:	0	Signal	Strength:	0
Angle:	261.125	Distance:	0	Signal	Strength:	0
Angle:	265.047	Distance:	0	Signal	Strength:	0
Angle:	269	Distance:	0	Signal	Strength:	0
Angle:	265.453	Distance:	2799.25	Signal	Strength:	7
Angle:	269.328	Distance:	2818.25	Signal	Strength:	6
Angle:	280.75	Distance:	0	Signal	Strength:	0
Angle:	284.859	Distance:	0	Signal	Strength:	0
Angle:	288.828	Distance:	0	Signal	Strength:	0
Angle:	292.797	Distance:	0	Signal	Strength:	0
Angle:	296.719	Distance:	0	Signal	Strength:	0
Angle:	300.672	Distance:	0	Signal	Strength:	0
Angle:	304.641	Distance:	0	Signal	Strength:	0
Angle:	308.422	Distance:	0	Signal	Strength:	0
Angle:	312.359	Distance:	0	Signal	Strength:	0
Angle:	316.281	Distance:	0	Signal	Strength:	0
Angle:	320.219	Distance:	0	Signal	Strength:	0
Angle:	324.172	Distance:	0	Signal	Strength:	0
Angle:	328.094	Distance:	0	Signal	Strength:	0

Angle:	332.078	Distance:	0	Signal	Strength:	0
Angle:	336.031	Distance:	0	Signal	Strength:	0
Angle:	339.953	Distance:	0	Signal	Strength:	0
Angle:	343.891	Distance:	0	Signal	Strength:	0
Angle:	347.828	Distance:	0	Signal	Strength:	0
Angle:	351.781	Distance:	0	Signal	Strength:	0
Angle:	355.766	Distance:	0	Signal	Strength:	0
Angle:	359.719	Distance:	0	Signal	Strength:	0
Angle:	3.672	Distance:	0	Signal	Strength:	0
Angle:	7.625	Distance:	0	Signal	Strength:	0
Angle:	11.578	Distance:	0	Signal	Strength:	0
Angle:	15.516	Distance:	0	Signal	Strength:	0
Angle:	19.375	Distance:	0	Signal	Strength:	0
Angle:	23.328	Distance:	0	Signal	Strength:	0
Angle:	27.266	Distance:	0	Signal	Strength:	0

APPENDIX B.5 Sensor Calculations

				DATA POINTS PER REVOLUTION							
Motor speed (Hz)	5	6	7	8	9	10	11	12	13	14	15
samples (n)											
4000	800.00	666.67	571.43	500.00	444.44	400.00	363.64	333.33	307.69	285.71	266.67
				AVERAGE DEGREES PER REVOLUTION							
4000	0.45	0.54	0.63	0.72	0.81	0.90	0.99	1.08	1.17	1.26	1.35

				Spinning frequency of the LIDAR (Hz)							
distance to obstacle (mm)	5	6	7	8	9	10	11	12	13	14	15
				ARCH LENGTH (MILLIMETERS) BETWEEN TWO DATA POINTS (4000 Hz)							
1000	7.85	9.42	11.00	12.57	14.14	15.71	17.28	18.85	20.42	21.99	23.56
2000	15.71	18.85	21.99	25.13	28.27	31.42	34.56	37.70	40.84	43.98	47.12
3000	23.56	28.27	32.99	37.70	42.41	47.12	51.84	56.55	61.26	65.97	70.69
4000	31.42	37.70	43.98	50.27	56.55	62.83	69.12	75.40	81.68	87.96	94.25
5000	39.27	47.12	54.98	62.83	70.69	78.54	86.39	94.25	102.10	109.96	117.81
6000	47.12	56.55	65.97	75.40	84.82	94.25	103.67	113.10	122.52	131.95	141.37
7000	54.98	65.97	76.97	87.96	98.96	109.96	120.95	131.95	142.94	153.94	164.93
8000	62.83	75.40	87.96	100.53	113.10	125.66	138.23	150.80	163.36	175.93	188.50

				Spinning frequency of the LIDAR (Hz)							
distance to obstacle (cm)	5	6	7	8	9	10	11	12	13	14	15
				LINEAR DISTANCE (MILLIMETERS) BETWEEN TWO DATA POINTS (4000 Hz)							
1000	7.85	9.42	11.00	12.57	14.14	15.71	17.28	18.85	20.42	21.99	23.56

2000	15.71	18.85	21.99	25.13	28.27	31.42	34.56	37.70	40.84	43.98	47.12
3000	23.56	28.27	32.99	37.70	42.41	47.12	51.84	56.55	61.26	65.97	70.68
4000	31.42	37.70	43.98	50.27	56.55	62.83	69.11	75.40	81.68	87.96	94.25
5000	39.27	47.12	54.98	62.83	70.69	78.54	86.39	94.25	102.10	109.95	117.81
6000	47.12	56.55	65.97	75.40	84.82	94.25	103.67	113.10	122.52	131.94	141.37
7000	54.98	65.97	76.97	87.96	98.96	109.95	120.95	131.94	142.94	153.93	164.93
8000	62.83	75.40	87.96	100.53	113.10	125.66	138.23	150.79	163.36	175.93	188.49

Spinning frequency of the LIDAR (Hz)

distance to obstacle (cm)	1	2	3	4	5	6	7	8	9	10	11
DIFFERENCE (MILLIMETERS) IN THE TWO DIFFERENT METHODS (4000 Hz)											
1000	0.0000	0.0000	0.0001	0.0001	0.0001	0.0002	0.0002	0.0003	0.0004	0.0004	0.0005
2000	0.0000	0.0001	0.0001	0.0002	0.0002	0.0003	0.0004	0.0006	0.0007	0.0009	0.0011
3000	0.0001	0.0001	0.0002	0.0002	0.0004	0.0005	0.0006	0.0008	0.0011	0.0013	0.0016
4000	0.0001	0.0001	0.0002	0.0003	0.0005	0.0006	0.0009	0.0011	0.0014	0.0018	0.0022
5000	0.0001	0.0002	0.0003	0.0004	0.0006	0.0008	0.0011	0.0014	0.0018	0.0022	0.0027
6000	0.0001	0.0002	0.0003	0.0005	0.0007	0.0010	0.0013	0.0017	0.0021	0.0027	0.0033
7000	0.0001	0.0002	0.0004	0.0006	0.0008	0.0011	0.0015	0.0020	0.0025	0.0031	0.0038
8000	0.0002	0.0003	0.0004	0.0007	0.0009	0.0013	0.0017	0.0022	0.0028	0.0035	0.0044

AVERAGE DATA POINTS FOR OBSTACLES (RPLIDAR)

DATA POINTS PER REVOLUTION

Motor speed (Hz)	5	6	7	8	9	10	11	12	13	14	15
samples (n)											
4000	800.00	666.67	571.43	500.00	444.44	400.00	363.64	333.33	307.69	285.71	266.67

MM TO FEET 304.8

AVERAGE NUMBER OF DATA POINTS PER FOOT **1**

Spinning frequency of the LIDAR (Hz)

distance to obstacle (mm)	5	6	7	8	9	10	11	12	13	14	15
1000	38.808	32.34	27.72	24.255	21.56	19.404	17.64	16.17	14.927	13.86	12.936
2000	19.404	16.17	13.86	12.128	10.78	9.7022	8.8202	8.0852	7.4633	6.9302	6.4682
3000	12.936	10.78	9.2401	8.0851	7.1868	6.4681	5.8801	5.3901	4.9755	4.6201	4.3121
4000	9.7021	8.0851	6.9301	6.0638	5.3901	4.8511	4.4101	4.0426	3.7316	3.4651	3.2341
5000	7.7617	6.4681	5.5441	4.8511	4.3121	3.8809	3.5281	3.2341	2.9853	2.7721	2.5873
6000	6.4681	5.3901	4.6201	4.0426	3.5934	3.2341	2.9401	2.6951	2.4878	2.3101	2.1561
7000	5.5441	4.6201	3.9601	3.4651	3.0801	2.7721	2.5201	2.3101	2.1324	1.9801	1.8481
8000	4.8511	4.0426	3.465	3.0319	2.695	2.4255	2.205	2.0213	1.8658	1.7326	1.6171

APPENDIX B.6 Occupancy Map Generation Code (MATLAB)

```
tic
filename = 'RPLIDAR17.CSV';
RPLIDARdata = xlsread(filename);
numclust = 15; %leave at twelve to identify little changes in
elevation.
options = [2 10 1e-4 false];
maxwidth=3;
robotRadius = 2; %sets width of the Robot


num = length(RPLIDARdata); %measures the length of the Scanse data
thetad = RPLIDARdata(:,1); %pulls the azimuth data from the Scanse data
sgnlStrngth = RPLIDARdata(:,5);

r = RPLIDARdata(:,3);
bob = RPLIDARdata(r > 1, :);
toc
%% remove NaN from the angle and distance columns and delete zero rows
tic
for i= 1:num
    if thetad(isnan(thetad(i)))
        thetad(i) = (thetad(i-1));
    end
    if r(isnan(r(i)))
        r(i) = 1;
    end
end

bobangler = deg2rad(bob(:,1));
bobr = bob(:,3);

%% convert the polar coordinates to rectangular
xdist=bobr.*cos(bobangler);%produces the x axis distance
ydist=bobr.*sin(bobangler);%produces the y axis distance

scandata = [xdist ydist];
occenters=((scandata+15000)/1000);%scale the points to match the grid
%%
map = robotics.BinaryOccupancyGrid(30,30,1);
setOccupancy(map,occenters,1); %replace scandata with occenters


show(map);

mapInflated = copy(map);
```

```
inflate(mapInflated,robotRadius);%inflates objects to allow the robot
to pass

% prm = robotics.PRM(mapInflated);
planner = robotics.PRM(mapInflated); %creates a path planner object for
the occupancy map


xy= findpath(planner, [15 0], [15 30]); %developes  a path connecting
given datapoints


[pathstr,name,ext] = fileparts (filename); %pulls apart the filename
figure ('name', name, 'numbertitle', 'off');%I cannot find how to
calculate the total pixels

show(map); %shows the binary occupancy map
hold on
show(planner)
plot(occenters(:,1),occenters(:,2),'or','LineWidth',3)%plots the
obstacles on the occupancy grid
plot(15,15,'+g','MarkerSize',10,'LineWidth',3)
grid on%creates grid on the binary  occupancy grid
set(gca,'XTick',0:1:30,'YTick',0:1:30)

plot(xy(:,1),xy(:,2),'-x')%plot path found
toc
```

APPENDIX B.7 Sample Point Summaries (no false positives)

Weather=Shade Color=black Dist=1

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	283	-39.64	12.14	-75.00	-15.50
MeasuredDistance	MeasuredDistance	283	1039.64	12.14	1015.50	1075.00

Weather=Shade Color=black Dist=2

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	148	-48.16	19.54	-97.00	52.00
MeasuredDistance	MeasuredDistance	148	2048.16	19.54	1948.00	2097.00

Weather=Shade Color=black Dist=3

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	102	-61.96	26.50	-201.75	0.25
MeasuredDistance	MeasuredDistance	102	3061.96	26.50	2999.75	3201.75

Weather=Shade Color=black Dist=4

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	73	-61.26	26.71	-116.25	5.25
MeasuredDistance	MeasuredDistance	73	4061.26	26.71	3994.75	4116.25

Weather=Shade Color=black Dist=5

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	24	-63.89	61.25	-140.25	84.50
MeasuredDistance	MeasuredDistance	24	5063.89	61.25	4915.50	5140.25

Weather=Shade Color=black Dist=6

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	34	-86.99	84.44	-318.00	109.50
MeasuredDistance	MeasuredDistance	34	6086.99	84.44	5890.50	6318.00

Weather=Shade Color=black Dist=7

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	17	-202.79	127.86	-362.50	23.50
MeasuredDistance	MeasuredDistance	17	7202.79	127.86	6976.50	7362.50

Weather=Shade Color=black Dist=8

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	3	-326.50	102.62	-445.00	-267.25
MeasuredDistance	MeasuredDistance	3	8326.50	102.62	8267.25	8445.00

Weather=Shade Color=black Dist=9

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	5	-411.75	388.83	-1084.75	-146.50
MeasuredDistance	MeasuredDistance	5	9411.75	388.83	9146.50	10084.75

Weather=Shade Color=white Dist=1

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	32	-130.22	18.97	-189.25	-96.75
MeasuredDistance	MeasuredDistance	32	1130.22	18.97	1096.75	1189.25

Weather=Shade Color=white Dist=3

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	13	-153.48	31.72	-195.25	-105.00
MeasuredDistance	MeasuredDistance	13	3153.48	31.72	3105.00	3195.25

Weather=Shade Color=white Dist=4

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	9	-103.81	184.02	-430.75	251.75
MeasuredDistance	MeasuredDistance	9	4103.81	184.02	3748.25	4430.75

Weather=Shade Color=white Dist=6

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	5	-149.15	187.46	-310.75	162.50
MeasuredDistance	MeasuredDistance	5	6149.15	187.46	5837.50	6310.75

Weather=Shade Color=white Dist=7

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	10	-162.48	299.85	-854.75	127.75
MeasuredDistance	MeasuredDistance	10	7162.48	299.85	6872.25	7854.75

Weather=Shade Color=white Dist=8

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	9	-222.89	180.05	-490.50	54.75
MeasuredDistance	MeasuredDistance	9	8222.89	180.05	7945.25	8490.50

Weather=Shade Color=white Dist=9

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	8	-192.09	133.36	-364.25	32.25
MeasuredDistance	MeasuredDistance	8	9192.09	133.36	8967.75	9364.25

Weather=Shade Color=white Dist=10

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	9	-80.31	177.18	-389.25	123.25
MeasuredDistance	MeasuredDistance	9	10080.31	177.18	9876.75	10389.25

Weather=Sun Color=black Dist=1

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	311	13.20	9.1181527	-8.25	33.75
MeasuredDistance	MeasuredDistance	311	986.79	9.1181527	966.25	1008.25

Weather=Sun Color=black Dist=2

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	143	-14.63	11.35	-49.25	38.50
MeasuredDistance	MeasuredDistance	143	2014.63	11.35	1961.50	2049.25

Weather=Sun Color=black Dist=3

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	107	5.67	32.21	-179.00	72.75
MeasuredDistance	MeasuredDistance	107	2994.33	32.21	2927.25	3179.00

Weather=Sun Color=black Dist=4

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	84	14.07	34.96	-97.50	90.50
MeasuredDistance	MeasuredDistance	84	3985.93	34.96	3909.50	4097.50

Weather=Sun Color=black Dist=5

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	60	-13.02	41.55	-102.25	130.50
MeasuredDistance	MeasuredDistance	60	5013.02	41.55	4869.50	5102.25

Weather=Sun Color=black Dist=6

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	46	-16.79	60.53	-128.75	125.25
MeasuredDistance	MeasuredDistance	46	6016.79	60.53	5874.75	6128.75

Weather=Sun Color=black Dist=7

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	35	-28.06	68.65	-171.25	127.75
MeasuredDistance	MeasuredDistance	35	7028.06	68.65	6872.25	7171.25

Weather=Sun Color=black Dist=8

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	26	-72.14	110.87	-311.00	88.75
MeasuredDistance	MeasuredDistance	26	8072.14	110.87	7911.25	8311.00

Weather=Sun Color=black Dist=9

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	14	-206.21	134.03	-501.75	32.25
MeasuredDistance	MeasuredDistance	14	9206.21	134.03	8967.75	9501.75

Weather=Sun Color=white Dist=1

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	296	-12.51	13.91	-78.75	9.75
MeasuredDistance	MeasuredDistance	296	1012.51	13.91	990.25	1078.75

Weather=Sun Color=white Dist=2

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	158	-28.88	9.96	-85.00	-12.50
MeasuredDistance	MeasuredDistance	158	2028.88	9.96	2012.50	2085.00

Weather=Sun Color=white Dist=3

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	106	-20.14	17.92	-91.50	28.75
MeasuredDistance	MeasuredDistance	106	3020.14	17.92	2971.25	3091.50

Weather=Sun Color=white Dist=4

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	84	5.24	39.54	-233.50	79.25
MeasuredDistance	MeasuredDistance	84	3994.76	39.54	3920.75	4233.50

Weather=Sun Color=white Dist=5

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	56	-22.63	41.83	-104.50	77.75
MeasuredDistance	MeasuredDistance	56	5022.63	41.83	4922.25	5104.50

Weather=Sun Color=white Dist=6

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	52	-38.81	75.70	-339.75	109.50
MeasuredDistance	MeasuredDistance	52	6038.81	75.70	5890.50	6339.75

Weather=Sun Color=white Dist=7

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	55	-101.12	59.65	-241.75	140.50
MeasuredDistance	MeasuredDistance	55	7101.12	59.65	6859.50	7241.75

Weather=Sun Color=white Dist=8

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	51	-51.09	162.30	-698.50	184.00
MeasuredDistance	MeasuredDistance	51	8051.09	162.30	7816.00	8698.50

Weather=Sun Color=white Dist=9

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	37	-127.80	221.95	-711.25	363.00
MeasuredDistance	MeasuredDistance	37	9127.80	221.95	8637.00	9711.25

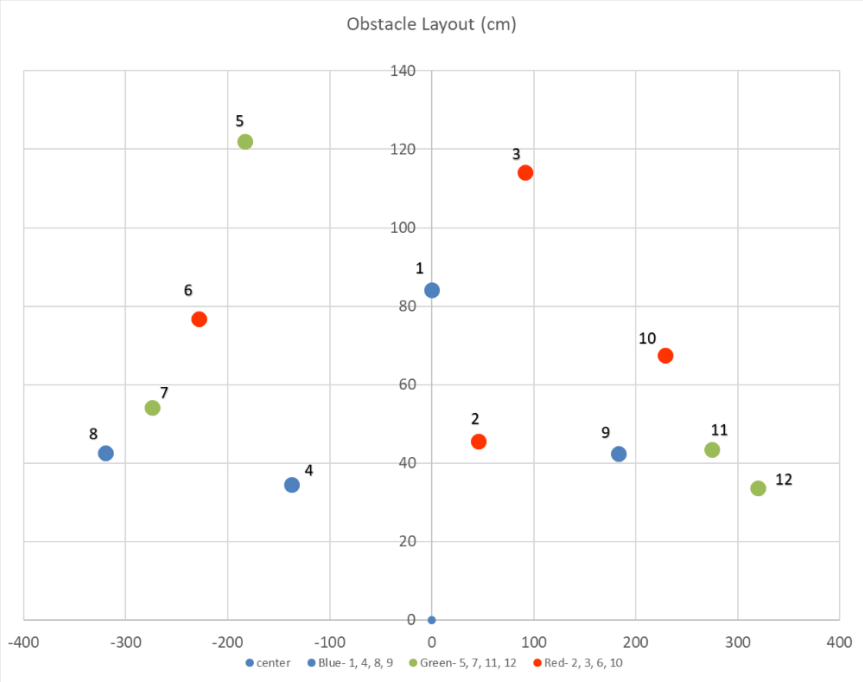
Weather=Sun Color=white Dist=10

Variable	Label	N	Mean	Std Dev	Minimum	Maximum
Accuracy	Accuracy	31	-243.69	213.71	-744.00	331.50
MeasuredDistance	MeasuredDistance	31	10243.69	213.71	9668.50	10744.00

APPENDIX C Hokuyo URG-04LX-UG01

APPENDIX C.1 Test Setup

Obstacle number	X Coordinate columns	Y Coordinate rows	X Coordinate feet	Y Coordinate feet	X Coordinate cm	Y Coordinate cm	radial distance feet	radial distance cm	azimuth degrees
12	3.5	0.736266314	10.5	1.104399471	320.04	33.66209587	10.55792111	321.8054355	6.004345
11	3	0.951008892	9	1.426513338	274.32	43.48012653	9.112350976	277.7444577	9.006542
10	2.5	1.477913982	7.5	2.216870973	228.6	67.57022724	7.8207747	238.3772129	16.46677
9	2	0.926921941	6	1.390382912	182.88	42.37887114	6.158990554	187.7260321	13.0469
8	3.5	0.932531414	10.5	1.398797121	-320.04	42.63533624	10.59276326	322.867424	172.4118
7	3	1.185282266	9	1.777923399	-274.32	54.19110519	9.173931088	279.6214196	168.8253
6	2.5	1.678478045	7.5	2.517717067	-228.6	76.74001621	7.911314633	241.13687	161.4433
5	2	2.666694823	6	4.000042235	-182.88	121.9212873	7.211125979	219.7951198	146.3097
4	1.5	0.756618095	4.5	1.134927142	-137.16	34.5925793	4.640911507	141.4549827	165.8449
3	1	2.496225428	3	3.744338141	91.44	114.1274265	4.797923313	146.2407026	51.29795
2	0.5	0.997409338	1.5	1.496114008	45.72	45.60155495	2.118574314	64.57414509	44.92569
1	0	1.841387998	0	2.762081997	0	84.18825928	2.762081997	84.18825928	90



APPENDIX C.2 Data Collection code (C)

```
/*!
  \example calculate_xy.c Calculates X-Y coordinates

  Having the X axis aligned to the front step of the sensor, calculates the
coordinates for measurement data
  \author Satofumi KAMIMURA

  $Id$
*/

#include "urg_sensor.h"
#include "urg_utils.h"
#include "open_urg_sensor.h"
#include <time.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>



int main(int argc, char *argv[])
{
//Makes a file to write to and opens it. Will eventually be replaced with code to
speak to other devices.
    char filename[50];
    sprintf(filename, "/media/sf_Hokuyo_Data/%s.csv", argv[1]);
    FILE *fp = NULL;
    fp = fopen (filename, "w");
```

```
    if (fp == NULL) {
        printf ("File not created okay");
       return 1;
    }
    else {
    printf ("File created okay.\n");
    }
//set Variables
    urg_t urg;
    long *data;
    long max_distance;
    long min_distance;
    long time_stamp;
    int i;
    int n;
    int count;
    int scantime = atoi(argv[2]);

//Create a timer to run the program for a specified time period
    time_t start,end;
    double elapsed;
    time(&start); //starts the timer

    if (open_urg_sensor(&urg, argc, argv) < 0) {
        return 1;
    }
while (elapsed <= scantime) {
    data = (long *)malloc(urg_max_data_size(&urg) * sizeof(data[0]));
    if (!data) {
        perror("urg_max_index()");
        return 1;
    }

    // Gets measurement data
    urg_start_measurement(&urg, URG_DISTANCE, 1, 0);
    n = urg_get_distance(&urg, data, &time_stamp);
    if (n < 0) {
        printf("urg_get_distance: %s\n", urg_error(&urg));
        urg_close(&urg);
        return 1;
    }

    // Outputs X-Y coordinates
    urg_distance_min_max(&urg, &min_distance, &max_distance);
    for (i = 0; i < n; ++i) {
        long distance = data[i];
        double radian;
        long x;
        long y;

        if ((distance < min_distance) || (distance > max_distance)) {
            continue;
        }

        radian = urg_index2rad(&urg, i);
```

```
        x = (long)(distance * cos(radian));
        y = (long)(distance * sin(radian));

        printf("X: %ld Y: %ld radian: %lf distance(r): %ld \n", x, y,radian,
distance);
        fprintf(fp, "X: %ld Y: %ld radian: %lf distance(r): %ld \n", x, y,radian,
distance);

    }
    printf("\n");

    //Change the time elapsed
    time(&end);
    elapsed = difftime(end,start); //use difftime() for time_t structs
}

    fclose(fp);

    // Disconnects
    free(data);
    urg_close(&urg);

#if defined(URG_MSC)
    getchar();
#endif
    return 0;
}
```

APPENDIX C.3 Plotting Data code (MATLAB)

```
filename = 'hokR03.CSV';
hokuyo = xlsread(filename);

wdist =0; %distance threshold for an obstacle---furthest distance that
it will detect an obstacle
lowangle = 30;% lowest angle that you want to get rid of in FOV
highangle = 3; % highest angle that you want to get rid in FOV

num = length(hokuyo); %measures the length of the Scanse data
thetar = hokuyo(:,3); %pulls the azimuth data from the Scanse data
r = hokuyo(:,4); %pulls the distance data from the Scanse data

wtheta=[];
wr=[];

%-------------------------------------------------------------------------
-------------------------------------------------%

xdist=r.*cos(thetar);%produces the x axis distance
ydist=r.*sin(thetar);%produces the y axis distance
```

```
[pathstr,name,ext] = fileparts (filename); %added these next two lines
for sure. (25 & 26)
figure ('name', name, 'numbertitle', 'off');
polarscatter(thetar, r, 'bo') %plots the Scanse data in a polar graph

legend('data points')
```

APPENDIX C.4 Sample Data.

X:	Y:	Radian:	Distance:	Timestamp:
-203	-359	-2.08621	413	392329
-204	-365	-2.08008	419	392329
-199	-361	-2.07394	413	392329
-196	-363	-2.06781	413	392329
-189	-353	-2.06167	401	392329
-186	-353	-2.05553	400	392329
-179	-346	-2.0494	390	392329
-177	-347	-2.04326	390	392329
-178	-354	-2.03713	397	392329
-177	-358	-2.03099	400	392329
-174	-356	-2.02486	397	392329
-173	-360	-2.01872	400	392329
-171	-361	-2.01258	400	392329
-169	-364	-2.00645	402	392329
-169	-369	-2.00031	406	392329
-166	-370	-1.99418	406	392329
-162	-367	-1.98804	402	392329
-160	-368	-1.9819	402	392329
-158	-369	-1.97577	402	392329
-156	-370	-1.96963	402	392329
-153	-371	-1.9635	402	392329
-151	-372	-1.95736	402	392329
-152	-381	-1.95122	411	392329
-150	-382	-1.94509	411	392329
-144	-375	-1.93895	402	392329
-146	-386	-1.93282	413	392329
-144	-389	-1.92668	416	392329
-142	-390	-1.92054	416	392329
-140	-393	-1.91441	418	392329
-140	-399	-1.90827	423	392329

-137	-399	-1.90214	423	392329
-138	-411	-1.896	434	392329
-137	-416	-1.88986	439	392329
-181	-560	-1.88373	589	392329
-179	-567	-1.87759	595	392329
-318	-4312	-1.64443	4324	392329
-291	-4314	-1.63829	4324	392329
-49	-4029	-1.58307	4030	392329
-24	-4029	-1.57693	4030	392329
0	-4030	-1.5708	4030	392329
27	-4480	-1.56466	4481	392329
17	-473	-1.53398	474	392329
20	-473	-1.52785	474	392329
337	-4218	-1.49103	4232	392329
363	-4216	-1.48489	4232	392329
739	-4595	-1.41126	4655	392329
599	-3588	-1.40513	3638	392329
621	-3584	-1.39899	3638	392329
597	-3322	-1.39286	3376	392329
665	-3576	-1.38672	3638	392329
1149	-4977	-1.34377	5109	392329
563	-2373	-1.33763	2440	392329
578	-2370	-1.3315	2440	392329
459	-1835	-1.32536	1892	392329
607	-2363	-1.31922	2440	392329
1228	-4662	-1.31309	4822	392329
3014	-4452	-0.97561	5377	392329
3041	-4433	-0.96948	5377	392329
2977	-4282	-0.96334	5216	392329
3003	-4264	-0.9572	5216	392329
3028	-4244	-0.95107	5214	392329
3052	-4223	-0.94493	5211	392329
2962	-4045	-0.9388	5014	392329
2986	-4027	-0.93266	5014	392329
3011	-4008	-0.92652	5014	392329
3055	-4015	-0.92039	5046	392329
3079	-3996	-0.91425	5046	392329
3104	-3977	-0.90812	5046	392329
3076	-3892	-0.90198	4962	392329
2970	-3711	-0.89585	4754	392329
2969	-3663	-0.88971	4716	392329

2988	-3641	-0.88357	4711	392329
2993	-3602	-0.87744	4684	392329
3011	-3578	-0.8713	4677	392329
3022	-3547	-0.86517	4660	392329
3001	-3480	-0.85903	4596	392329
2993	-3427	-0.85289	4551	392329
2988	-3380	-0.84676	4512	392329
2954	-3299	-0.84062	4429	392329
2972	-3280	-0.83449	4427	392329
2941	-3205	-0.82835	4350	392329
2946	-3171	-0.82221	4329	392329
2899	-3083	-0.81608	4233	392329
2917	-3064	-0.80994	4231	392329
2936	-3046	-0.80381	4231	392329
2938	-3011	-0.79767	4208	392329
2933	-2969	-0.79153	4174	392329
2931	-2931	-0.7854	4146	392329
2905	-2870	-0.77926	4084	392329
2915	-2845	-0.77313	4074	392329
2933	-2827	-0.76699	4074	392329
2920	-2780	-0.76085	4033	392329
2924	-2749	-0.75472	4014	392329
2880	-2675	-0.74858	3931	392329
2896	-2657	-0.74245	3931	392329
2888	-2618	-0.73631	3899	392329
2898	-2594	-0.73018	3890	392329

APPENDIX D SURF Feature Extraction and Matching and Edge Detection

APPENDIX D.1 SURF Feature Extraction and Matching Script (MATLAB)

```
% learning image
pivotImage = rgb2gray(imread(%image name));
figure ('name', 'Learning Image');
imshow(pivotImage);
title('Learning Image');
```

```
%trying to identify this image
sceneImage = rgb2gray(imread(%image name));
figure ('name', 'Image to be Classified');
imshow(sceneImage);
title('Image to be Classified');

%detect points in the images
pivotPoints = detectSURFFeatures(pivotImage);
scenePoints = detectSURFFeatures(sceneImage);

%Extract feature descriptors at the interest points in both images.
[pivotFeatures, pivotPoints] = extractFeatures(pivotImage,
pivotPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage,
scenePoints);

%match the features
pivotPairs = matchFeatures(pivotFeatures, sceneFeatures);

%display positively matched features
matchedPivotPoints = pivotPoints(pivotPairs(:, 1), :);
matchedScenePoints = scenePoints(pivotPairs(:, 2), :);
figure ('name','Outlier');
showMatchedFeatures(pivotImage, sceneImage, matchedPivotPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');

%locate and show the object in the scene
[tform, inlierPivotPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedPivotPoints, matchedScenePoints,
'affine');
figure;
showMatchedFeatures(pivotImage, sceneImage, inlierPivotPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```

APPENDIX D.2 Edge Detection Script (MATLAB)

```
img = %File location
method = %edge detection method

%reads in an image
pivotImage =imread(img);
gray = rgb2gray(pivotImage);
figure
% ('name', method);
% subplot(1,2,1)
imshow(pivotImage);
title('Image of a Pivot');

%finds the edges in the picture and filters out noise
```

```
BW = wiener2(edge(gray,method));
% subplot(1,2,2)
imshow(BW);
title (method)
```

APPENDIX D.3 Shape Identification algorithm (MATLAB)

```
% Demo to find certain shapes in an image based on their shape.
clc;    % Clear the command window.
close all;  % Close all figures (except those of imtool.)
imtool close all;  % Close all imtool figures.
clear;  % Erase all existing variables.
workspace;  % Make sure the workspace panel is showing.
fontSize = 20;

% Read in a standard MATLAB gray scale demo image.
folder = fullfile(matlabroot, '\toolbox\images\imdemos');
baseFileName = 'pillsetc.png';
% Get the full filename, with path prepended.
fullFileName = fullfile(folder, baseFileName);
% Check if file exists.
if ~exist(fullFileName, 'file')
    % File doesn't exist -- didn't find it there.  Check the search
path for it.
    fullFileName = baseFileName; % No path this time.
    if ~exist(fullFileName, 'file')
        % Still didn't find it.  Alert user.
        errorMessage = sprintf('Error: %s does not exist in the search
path folders.', fullFileName);
        uiwait(warndlg(errorMessage));
        return;
    end
end

% Read in image into an array.
rgbImage = imread(fullFileName);
[rows, columns, numberOfColorBands] = size(rgbImage);
% Display it.
subplot(2, 2, 1);
imshow(rgbImage, []);
title('Input Image', 'FontSize', fontSize);
% Enlarge figure to full screen.
set(gcf, 'units','normalized','outerposition',[0 0 1 1]);
% Give a name to the title bar.
set(gcf,'name','Shape Recognition Demo','numbertitle','off')

% If it's monochrome (indexed), convert it to color.
if numberOfColorBands > 1
    grayImage = rgbImage(:,:,2);
else
    % It's already a gray scale image.
```

```
    grayImage = rgbImage;
end

% Make a triangle on it.
triangleXCoordinates = [360 420 480];
triangleYCoordinates = [350 252 350];
traiangleBinaryImage = poly2mask(triangleXCoordinates,
triangleYCoordinates, rows, columns);
% Burn it into the gray scale image.
grayImage(traiangleBinaryImage) = 255;

% Display it.
subplot(2, 2, 2);
imshow(grayImage, []);
title('Grayscale Image', 'FontSize', fontSize);

% Binarize the image.
binaryImage = grayImage > 120;
% Display it.
subplot(2, 2, 3);
imshow(binaryImage, []);
title('Initial (Noisy) Binary Image', 'FontSize', fontSize);

% Remove small objects.
binaryImage = bwareaopen(binaryImage, 300);
% Display it.
subplot(2, 2, 4);
imshow(binaryImage, []);
title('Cleaned Binary Image', 'FontSize', fontSize);

[labeledImage numberOfObjects] = bwlabel(binaryImage);
blobMeasurements = regionprops(labeledImage,...
    'Perimeter', 'Area', 'FilledArea', 'Solidity', 'Centroid');

% Get the outermost boundaries of the objects, just for fun.
filledImage = imfill(binaryImage, 'holes');
boundaries = bwboundaries(filledImage);

% Collect some of the measurements into individual arrays.
perimeters = [blobMeasurements.Perimeter];
areas = [blobMeasurements.Area];
filledAreas = [blobMeasurements.FilledArea];
solidities = [blobMeasurements.Solidity];
% Calculate circularities:
circularities = perimeters .^2 ./ (4 * pi * filledAreas);
% Print to command window.
fprintf('#, Perimeter,        Area, Filled Area, Solidity,
Circularity\n');
for blobNumber = 1 : numberOfObjects
    fprintf('%d, %9.3f, %11.3f, %11.3f, %8.3f, %11.3f\n', ...
        blobNumber, perimeters(blobNumber), areas(blobNumber), ...
        filledAreas(blobNumber), solidities(blobNumber),
circularities(blobNumber));
```

```
end

% Say what they are.
% IMPORTANT NOTE: depending on the aspect ratio of the rectangel or
triangle
for blobNumber = 1 : numberOfObjects
    % Outline the object so the user can see it.
    thisBoundary = boundaries{blobNumber};
    subplot(2, 2, 2); % Switch to upper right image.
    hold on;
    % Display prior boundaries in blue
    for k = 1 : blobNumber-1
        thisBoundary = boundaries{k};
        plot(thisBoundary(:,2), thisBoundary(:,1), 'b', 'LineWidth',
3);
    end
    % Display this bounary in red.
    thisBoundary = boundaries{blobNumber};
    plot(thisBoundary(:,2), thisBoundary(:,1), 'r', 'LineWidth', 3);
    subplot(2, 2, 4); % Switch to lower right image.

    % Determine the shape.
    if circularities(blobNumber) < 1.2
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a circle',...
            blobNumber, circularities(blobNumber));
        shape = 'circle';
    elseif circularities(blobNumber) < 1.6
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a square',...
            blobNumber, circularities(blobNumber));
        shape = 'square';
    elseif circularities(blobNumber) > 1.6 && circularities(blobNumber)
< 1.8
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is an isocoles triangle',...
            blobNumber, circularities(blobNumber));
        shape = 'triangle';
    else
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is something else.',...
            blobNumber, circularities(blobNumber));
        shape = 'something else';
    end
    % Display in overlay above the object
    overlayMessage = sprintf('Object #%d = %s\ncirc = %.2f, s = %.2f',
...
        blobNumber, shape, circularities(blobNumber),
solidities(blobNumber));
    text(blobMeasurements(blobNumber).Centroid(1),
blobMeasurements(blobNumber).Centroid(2), ...
        overlayMessage, 'Color', 'r');
    button = questdlg(message, 'Continue', 'Continue', 'Cancel',
'Continue');
```

```
    if strcmp(button, 'Cancel')
        break;
    end
end
```

APPENDIX D.4 Edge detection and Shape Identification Algorithm (MATLAB)

```
clc;    % Clear the command window.
close all;  % Close all figures (except those of imtool.)
imtool close all;  % Close all imtool figures.
clear;  % Erase all existing variables.
workspace;  % Make sure the workspace panel is showing.
fontSize = 18;

img = %image name
method = % =edge detection method

 %reads in an image
pivotImage =imread(img);
gray = rgb2gray(pivotImage);
figure ('name', method);

%finds the edges in the picture and filters out noise
BW = wiener2(edge(gray,method));

subplot(2, 2, 1);
imshow(pivotImage, []);
title('Input Image', 'FontSize', fontSize);

 % Display it.
 subplot(2, 2, 2);
% imshow(grayImage, []);
 title('Objects Detected', 'FontSize', fontSize);

subplot(2, 2, 3);
%imshow(binaryImage, []);
imshow(BW, [])
title('Initial Edge Detection Image', 'FontSize', fontSize);

binaryImage = bwareaopen(BW, 300);
% Display it.
subplot(2, 2, 4);
imshow(binaryImage, []);
title('Cleaned Binary Image', 'FontSize', fontSize);

[labeledImage numberOfObjects] = bwlabel(binaryImage);
blobMeasurements = regionprops(labeledImage,...
    'Perimeter', 'Area', 'FilledArea', 'Solidity', 'Centroid');
```

```
% Get the outermost boundaries of the objects, just for fun.
filledImage = imfill(binaryImage, 'holes');
boundaries = bwboundaries(filledImage);

% Collect some of the measurements into individual arrays.
perimeters = [blobMeasurements.Perimeter];
areas = [blobMeasurements.Area];
filledAreas = [blobMeasurements.FilledArea];
solidities = [blobMeasurements.Solidity];
% Calculate circularities:
circularities = perimeters .^2 ./ (4 * pi * filledAreas);
% Print to command window.
fprintf('#, Perimeter,        Area, Filled Area, Solidity,
Circularity\n');
for blobNumber = 1 : numberOfObjects
    fprintf('%d, %9.3f, %11.3f, %11.3f, %8.3f, %11.3f\n', ...
        blobNumber, perimeters(blobNumber), areas(blobNumber), ...
        filledAreas(blobNumber), solidities(blobNumber),
circularities(blobNumber));
end

% Say what they are.
for blobNumber = 1 : numberOfObjects
    % Outline the object so the user can see it.
    thisBoundary = boundaries{blobNumber};
    subplot(2, 2, 2); % Switch to upper right image.
    hold on;
    % Display prior boundaries in blue
    for k = 1 : blobNumber-1
        thisBoundary = boundaries{k};
        plot(thisBoundary(:,2), thisBoundary(:,1), 'b', 'LineWidth',
3);
    end
    % Display this bounary in red.
    thisBoundary = boundaries{blobNumber};
    plot(thisBoundary(:,2), thisBoundary(:,1), 'r', 'LineWidth', 3);
    subplot(2, 2, 4); % Switch to lower right image.

    % Determine the shape.
    if circularities(blobNumber) < 1.2
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a circle',...
            blobNumber, circularities(blobNumber));
        shape = 'circle';
    elseif circularities(blobNumber) < 1.6
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is a square',...
            blobNumber, circularities(blobNumber));
        shape = 'square';
    elseif circularities(blobNumber) > 1.6 && circularities(blobNumber)
< 1.8
```

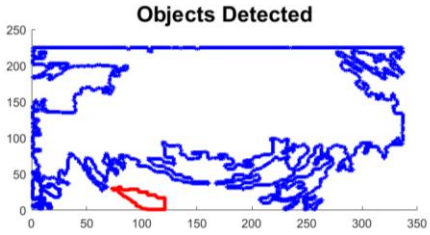
```
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is an isocoles triangle',...
            blobNumber, circularities(blobNumber));
        shape = 'triangle';
    else
        message = sprintf('The circularity of object #%d is %.3f,\nso
the object is something else.',...
            blobNumber, circularities(blobNumber));
        shape = 'something else';
    end
    % Display in overlay above the object
    overlayMessage = sprintf('Object #%d = %s\ncirc = %.2f, s = %.2f',
...
        blobNumber, shape, circularities(blobNumber),
solidities(blobNumber));
    text(blobMeasurements(blobNumber).Centroid(1),
blobMeasurements(blobNumber).Centroid(2), ...
        overlayMessage, 'Color', 'r');
    button = questdlg(message, 'Continue', 'Continue', 'Cancel',
'Continue');
    if strcmp(button, 'Cancel')
        break;
    end
end
```

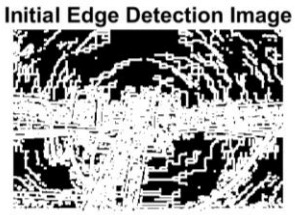
APPENDIX D.5 SURF Feature Extraction and Matching and Edge Detection (MATLAB)

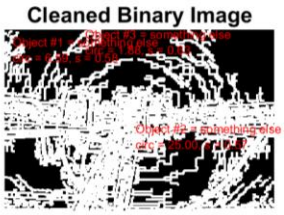
```
img1 = reference image
img2 = test image

% reference image (img1)
pivotImage = edge_detection_method_3(img1);
figure;
imshow(pivotImage);
title('Image of a known obstacle');

%test image (img2)
sceneImage = edge_detection_method_3 (img2);
figure;
imshow(sceneImage);
title('Image of a Cluttered Scene');

%detect points in the images
pivotPoints = detectSURFFeatures(pivotImage);
scenePoints = detectSURFFeatures(sceneImage);

%Extract feature descriptors at the interest points in both images.
```

```
[pivotFeatures, pivotPoints] = extractFeatures(pivotImage,
pivotPoints);
[sceneFeatures, scenePoints] = extractFeatures(sceneImage,
scenePoints);

%match the features
pivotPairs = matchFeatures(pivotFeatures, sceneFeatures);

%display positively matched features
matchedPivotPoints = pivotPoints(pivotPairs(:, 1), :);
matchedScenePoints = scenePoints(pivotPairs(:, 2), :);
figure;
showMatchedFeatures(pivotImage, sceneImage, matchedPivotPoints, ...
    matchedScenePoints, 'montage');
title('Putatively Matched Points (Including Outliers)');

%locate and show the object in the scene
[tform, inlierPivotPoints, inlierScenePoints] = ...
    estimateGeometricTransform(matchedPivotPoints, matchedScenePoints,
'affine');
figure;
showMatchedFeatures(pivotImage, sceneImage, inlierPivotPoints, ...
    inlierScenePoints, 'montage');
title('Matched Points (Inliers Only)');
```

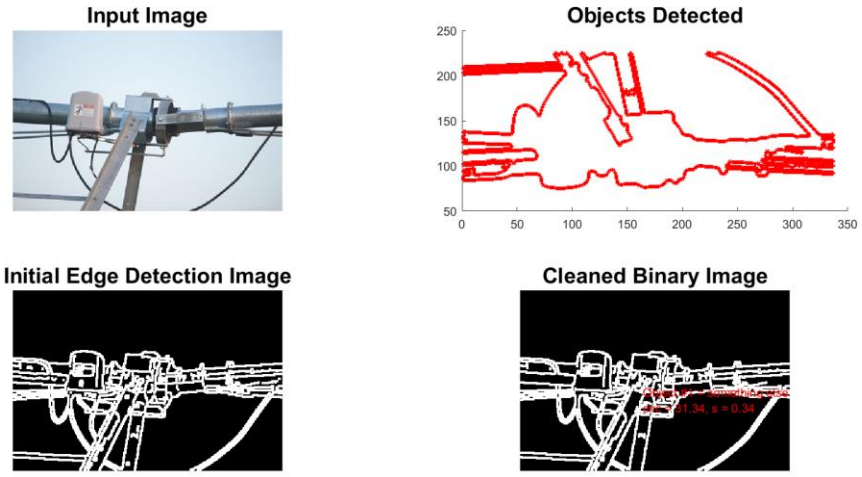
APPENDIX D.6 approxcanny



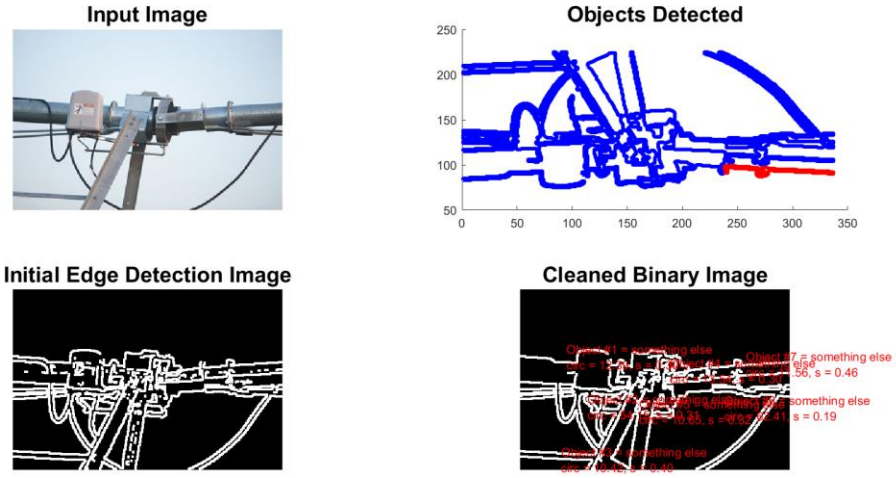




APPENDIX D.7 Canny



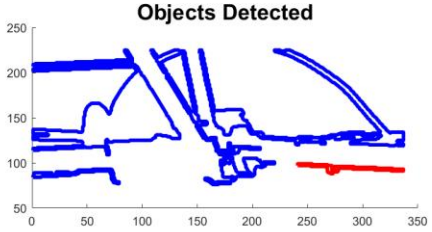
APPENDIX D.8 log



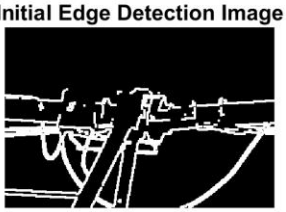
APPENDIX D.9 Prewitts

Input Image

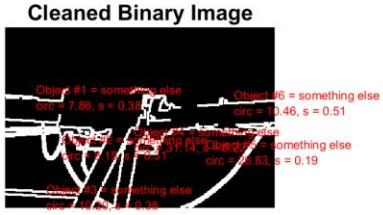
Objects Detected



Initial Edge Detection Image



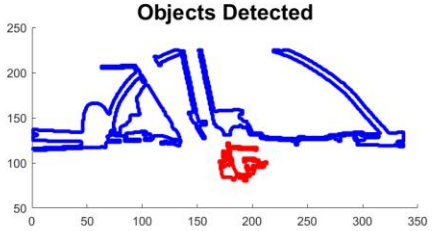
Cleaned Binary Image



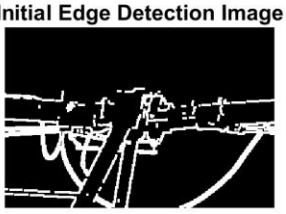
APPENDIX D.10 Roberts

Input Image

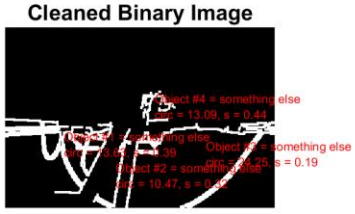
Objects Detected



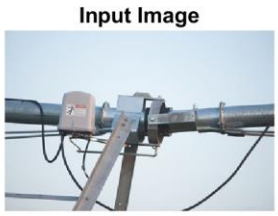
Initial Edge Detection Image

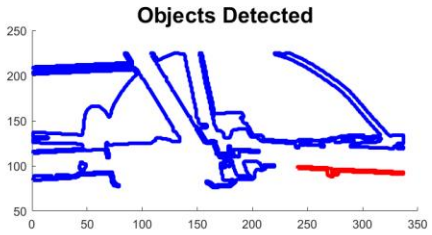


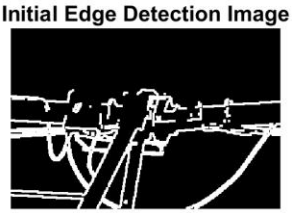
Cleaned Binary Image

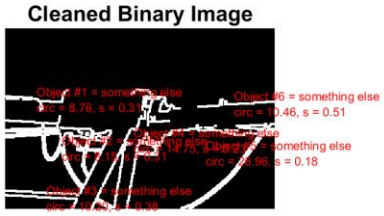


APPENDIX D.11 Sobel

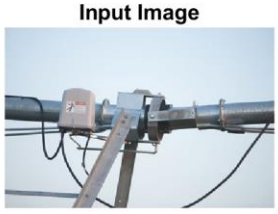


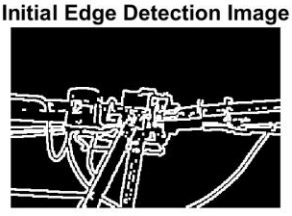


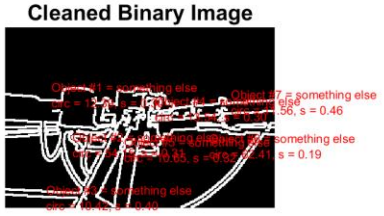




APPENDIX D.12 zerocross







APPENDIX E Viola-Jones Algorithm Script (MATLAB)

```
gTruth = load(%ground Truth data);

trainingData = objectDetectorTrainingData(gTruth);

positiveInstances = trainingData(:,1:2);

negativeFolder = fullfile(%location of negative images) ;

negativeImages = imageDatastore(negativeFolder);

trainCascadeObjectDetector(%name of trained detector positiveInstances,
negativeFolder,...
    'FalseAlarmRate',0.025,'NumCascadeStages',3);

detector = vision.CascadeObjectDetector(%name of trained detector);

img = imread(%name of test image);

bbox = step(detector,img);

detectedImg = insertObjectAnnotation(img,'rectangle',bbox, %name of
bounding box);

figure; imshow(detectedImg);
```

APPENDIX F Faster R-CNN Training

APPENDIX F.1 Code used to extract images out of video (MATLAB)

```
gTruth = load(%ground truth data fom video);
trainingData = objectDetectorTrainingData(gTruth.gTruth);
%Default 'SamplingFactor' is set to 5 to sample every 5th image
%This extracts evey single image
%('SamplingFactor',1) at end of gTruth video data to sample every
picture
```

APPENDIX F.2 Augmentation Code (MATLAB)

```
tic
%% blurring, sharpening, and change the brightness of the color images
srcFiles = dir(%location of original photos);
for K= 1:50
cd(srcFiles(1).folder)
filename = strcat(srcFiles(K).name);
image = imread(filename);
Blurred_Image= imgaussfilt(image);
Sharped_Image=imsharpen(image);
Brighter_Image=image+50;
Darker_image=image-50;
cd(%location for blurred images)
imwrite(Blurred_Image,filename)
cd(%location for sharpened images)
imwrite(Sharped_Image,filename)
cd(%location for brighter images)
imwrite(Brighter_Image,filename)
cd(%location for darker images)
imwrite(Darker_image,filename)
end
toc
```

APPENDIX F.3 Training detector Code (MATLAB)

```
tic
% Load vehicle data set
data = load(%load groundtruth data);
vehicleDataset = %location of the data inside the data variable;
label = %label locations inside the data variable
net = 'googlenet'; % load in any pretrained network
% can replace 'googlenet' with any of the pretrained networks. must
download the support package...
% home-> addons->search
%   'alexnet'
%   'vgg16'
%   'vgg19'
%   'resnet50'
%   'resnet101'
%   'inceptionv3'
%   'googlenet'
%   'inceptionresnetv2'
%   'squeezenet'

%%
% Split data into a training and test set.
idx = floor(0.6 * height(vehicleDataset));
trainingData = vehicleDataset(1:idx,:);
testData = vehicleDataset(idx:end,:);
```

```
%% Change hyperparameters
% Options for step 1.
optionsStage1 = trainingOptions('sgdm', ...
    'MaxEpochs', 2, ... %How many time does your model train on usually
range f% from 1 -> 5. First, usaually use 1 at first.
    'MiniBatchSize', 1, ...
    'InitialLearnRate', .005, ... %%  1 -> 1e-3. Boost this up %% tune
this to increase accuracy + trianing time
    'CheckpointPath', tempdir)

% Options for step 2.
optionsStage2 = trainingOptions('sgdm', ...
    'MaxEpochs', 2, ... %How many time does your model train on usually
range f% from 1 -> 5. First, usaually use 1 at first.
    'MiniBatchSize', 1, ...
    'InitialLearnRate', .005, ... %%  1 -> 1e-3. Boost this up %% tune
this to increase accuracy + trianing time
    'CheckpointPath', tempdir)

% Options for step 3.
optionsStage3 = trainingOptions('sgdm', ...
    'MaxEpochs', 2, ... %How many time does your model train on usually
range f% from 1 -> 5. First, usaually use 1 at first.
    'MiniBatchSize', 1, ...
    'InitialLearnRate', .005, ... %%  1 -> 1e-3. Boost this up %% tune
this to increase accuracy + trianing time
    'CheckpointPath', tempdir)

% Options for step 4.
optionsStage4 = trainingOptions('sgdm', ...
    'MaxEpochs', 3, ... %How many time does your model train on usually
range f% from 1 -> 5. First, usaually use 1 at first.
    'MiniBatchSize', 1, ...
    'InitialLearnRate', .005, ... %%  1 -> 1e-3. Boost this up %% tune
this to increase accuracy + trianing time
    'CheckpointPath', tempdir)


options = [
    optionsStage1
    optionsStage2
    optionsStage3
    optionsStage4
    ];
%%
% train Faster R-CNN object detector
trainedDetector = trainFasterRCNNObjectDetector(trainingData,
net,options)

doTrainingAndEval = true;

toc
tic
```

```
%%
if doTrainingAndEval
    % Run detector on each image in the test set and collect results.
    resultsStruct = struct([]);
    for i = 1:height(testData)

        % Read the image.
        I = imread(testData.imageFilename{i});

        % Run the detector.
        [bboxes, scores, labels] = detect(trainedDetector, I);

        % Collect the results.
        resultsStruct(i).Boxes = bboxes;
        resultsStruct(i).Scores = scores;
        resultsStruct(i).Labels = labels;
    end

    % Convert the results into a table.
    results = struct2table(resultsStruct);
else
    % Load results from disk.
    results = data.results;
end

% Extract expected bounding box locations from test data.
expectedResults = testData(:, 2:end);

% Evaluate the object detector using Average Precision metric.
[ap, recall, precision] = evaluateDetectionPrecision(results,
expectedResults);
%%

% Plot precision/recall curve
for k = 1:length(label)
figure(k)
plot(recall{k},precision{k})
xlabel('Recall')
ylabel('Precision')
grid on
title(sprintf('Average Precision = %.2f\n%s', ap(k), label{k}))
end
toc
```

APPENDIX F.4 Code for detecting objects using a trained detector for a single image (MATLAB)

```
%load detector

detector = load(detectorFile);

% Read a test image.
I = imread(testImage);

% Run the detector.
    [bboxes,scores,labels] = detect(detector.trainedDetector,I);


% Annotate detections in the image.
I = insertObjectAnnotation(I,'rectangle',bboxes,cellstr(labels));
figure
imshow(I)
```

APPENDIX F.5 Code for detecting objects using a trained detector for a multiple images (MATLAB)

```
srcFiles = dir(%image file directory);%load images
detector = load(%detector file); %load detector

for n= 1:length(srcFiles)
    cd (srcFiles(1).folder)
    filename = strcat(srcFiles(n).name);
    image = imread(filename);
    % Read a test image.
    I = imread(filename);

    % Run the detector.
    [bboxes,scores,labels] = detect(detector.trainedDetector,I);
    bob{n} = [scores];

    % Annotate detections in the image.
    I = insertObjectAnnotation(I,'rectangle',bboxes,cellstr(labels));
    figure ('name', filename);
    imshow(I)
end
```

APPENDIX F.6 Shuffling code (MATLAB)

```
load %load a ground truth table of your image files

shuffledArray = orderedArray(randperm(size(orderedArray,1)),:);
%this line sfuffles the images
```

APPENDIX F.7 Padding code (MATLAB)

```
srcFiles = dir(%directory image files are located);


for n=1:length(srcFiles)
    cd (srcFiles(1).folder)
    filename = strcat(srcFiles(n).name);
    YourImage = imread(filename);
if isinteger(YourImage)
   pad = intmax(class(YourImage));
else
   pad = 1;   %white for floating point is 1.0
end
%figure out which dimension is longer and rescale that to be the 256
%and pad the shorter one to 256
[r, c, ~] = size(YourImage);
if r > c
  NewImage = imresize(YourImage, 'new image height' / r);
  NewImage(:, end+1 : 'new image height', :) = pad;
elseif c > r
  NewImage = imresize(YourImage, 'new image width' / c);
  NewImage(end+1 : 'new image width', :, :) = pad;
else
  NewImage = imresize(YourImage, ['new image width' 'new image
height']);
end
    cd (%new image file location)
    imwrite (NewImage, filename)
end
```

APPENDIX F.8 Tractor images

Image 1 – (John Deere, 2019)

Image 2 – (Mahindra, 2019)

Image 3 – (Agricar, 2019)

Image 4 – (Ritchie Bros., 2019)

Image 5 – (Kubota, 2019)

APPENDIX F.8.1 Run 1

```
'MaxEpochs', 1,
'MiniBatchSize', 1,
'InitialLearnRate', .01,
'CheckpointPath', tempdir)
```

APPENDIX F.8.1.1 GoogLeNet

```
>> TractorDetectorgooglenetv2
optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
        GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 1
              MiniBatchSize: 1
                    Verbose: 1
            VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                  Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
           SequenceLength: 'longest'
       SequencePaddingValue: 0
```

```
optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
```

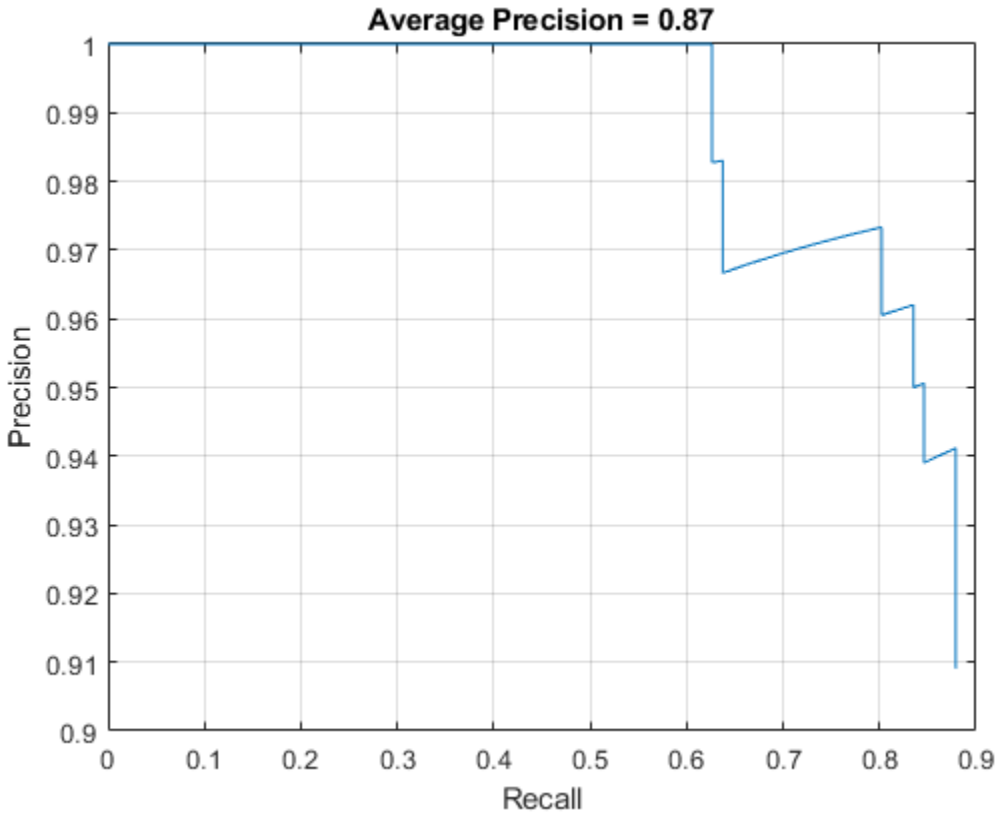
```
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 3
                MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0

*********************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors
```

```
Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy   |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.9906 |     35.16% |       1.56 |        0.0100 |
|     1 |        50 |     00:00:04 |     0.9211 |    100.00% |       0.88 |        0.0100 |
|     1 |       100 |     00:00:09 |     0.9586 |     97.64% |       1.05 |        0.0100 |
|     1 |       126 |     00:00:12 |     0.5319 |    100.00% |       0.83 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy   |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.0248 |     17.97% |       1.86 |        0.0100 |
|     1 |        50 |     00:00:12 |     0.1775 |     95.70% |       0.72 |        0.0100 |
|     1 |       100 |     00:00:25 |     0.1493 |     96.92% |       0.90 |        0.0100 |
|     1 |       126 |     00:00:31 |     0.0726 |     96.10% |       0.48 |        0.0100 |
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy   |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.6817 |    100.00% |       0.85 |        0.0100 |
```

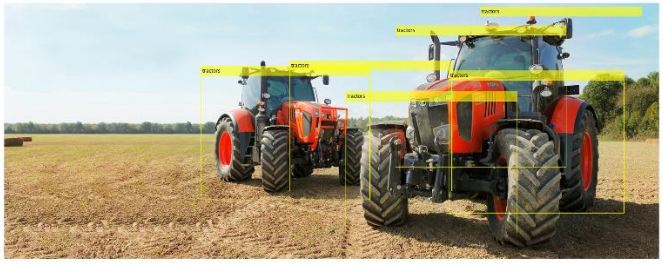
```
|      1 |        50 |      00:00:02 |      0.5786 |     100.00% |       0.86 |       0.0100 |
|      1 |       100 |      00:00:05 |      0.6043 |     100.00% |       0.83 |       0.0100 |
|      1 |       126 |      00:00:06 |      0.6100 |     100.00% |       0.83 |       0.0100 |
|=========================================================================
=========================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=========================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |      Rate      |
|=========================================================================
=========================================|
|      1 |         1 |      00:00:00 |      0.0916 |     100.00% |       0.81 |       0.0050 |
|      1 |        50 |      00:00:09 |      0.0969 |      95.70% |       0.59 |       0.0050 |
|      1 |       100 |      00:00:18 |      0.1413 |      98.96% |       1.05 |       0.0050 |
|      2 |       150 |      00:00:30 |      0.0313 |     100.00% |       0.43 |       0.0050 |
|      2 |       200 |      00:00:40 |      0.0283 |     100.00% |       0.46 |       0.0050 |
|      2 |       250 |      00:00:49 |      0.0685 |      97.73% |       0.56 |       0.0050 |
|      3 |       300 |      00:01:02 |      0.0355 |     100.00% |       1.11 |       0.0050 |
|      3 |       350 |      00:01:11 |      0.0261 |     100.00% |       0.45 |       0.0050 |
|      3 |       378 |      00:01:16 |      0.0316 |      98.86% |       0.39 |       0.0050 |
|=========================================================================
=========================================|

Detector training complete.
**************************************************************************



trainedDetector =
  fasterRCNNObjectDetector with properties:


       ModelName: 'tractors'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [4×2 double]
      ClassNames: {'tractors'  'Background'}
    MinObjectSize: [16 16]
```

Elapsed time is 241.028839 seconds.
Elapsed time is 26.416004 seconds.

APPENDIX F.8.1.1.1 Result



APPENDIX F.8.1.1.2 Results on test pictures



APPENDIX F.8.1.2 ResNet 101

```
>> TractorDetectorresnet101v3

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
                MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
          ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
```

```
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                    Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                    MaxEpochs: 1
                MiniBatchSize: 1
                    Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
          ValidationFrequency: 50
           ValidationPatience: Inf
                    Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                    Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
```

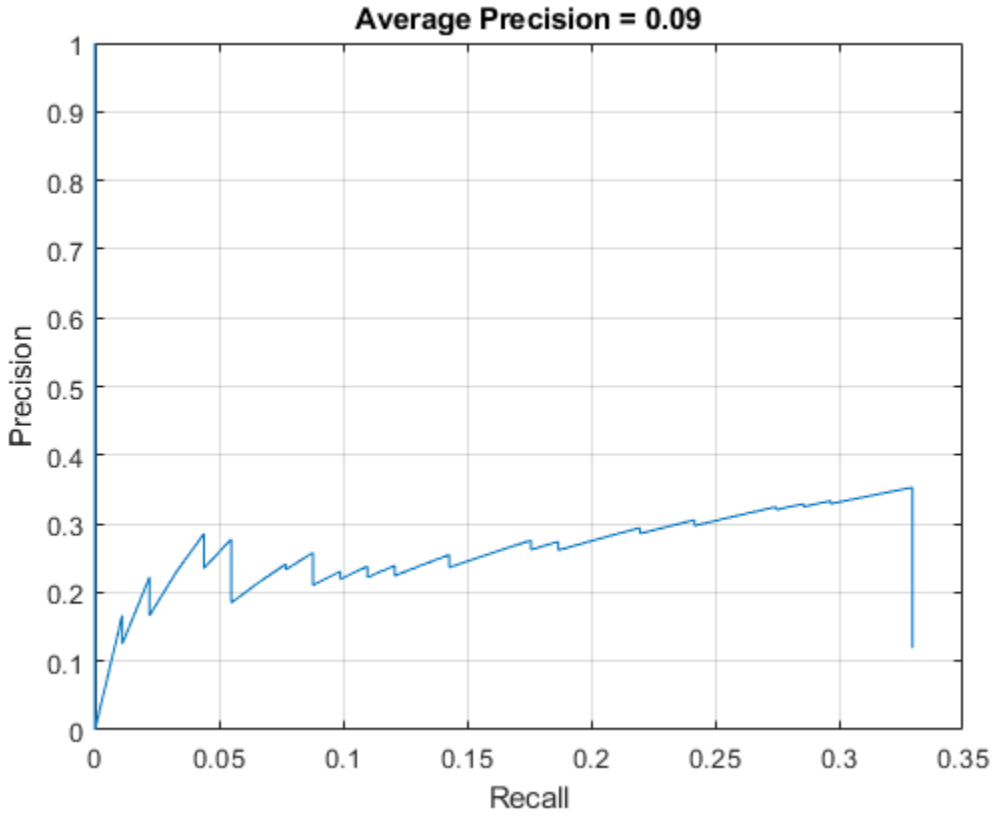
```
          GradientThreshold: Inf
                MaxEpochs: 1
             MiniBatchSize: 1
                  Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                 Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
     LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                MaxEpochs: 3
             MiniBatchSize: 1
                  Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                 Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0
```

```
********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.4758 |     73.23% |       1.14 |        0.0100 |
|     1 |        50 |     00:00:20 |     0.5206 |    100.00% |       0.69 |        0.0100 |
|     1 |       100 |     00:00:40 |     1.3325 |    100.00% |       1.32 |        0.0100 |
|     1 |       126 |     00:00:50 |     0.4061 |    100.00% |       0.74 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.6645 |     72.73% |       0.73 |        0.0100 |
|     1 |        50 |     00:00:38 |     0.2433 |     95.45% |       0.79 |        0.0100 |
|     1 |       100 |     00:01:17 |     0.1771 |     96.91% |       0.93 |        0.0100 |
|     1 |       123 |     00:01:35 |     0.1694 |     95.45% |       0.51 |        0.0100 |
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================================
=======================================|
```

```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.4140 |    100.00% |       0.74 |        0.0100 |
|     1 |        50 |     00:00:11 |     0.4995 |    100.00% |       0.80 |        0.0100 |
|     1 |       100 |     00:00:22 |     0.6177 |    100.00% |       0.84 |        0.0100 |
|     1 |       126 |     00:00:27 |     0.5852 |    100.00% |       0.82 |        0.0100 |
|============================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.1196 |     96.10% |       0.60 |        0.0050 |
|     1 |        50 |     00:00:24 |     0.0572 |     97.73% |       0.49 |        0.0050 |
|     1 |       100 |     00:00:47 |     0.0373 |    100.00% |       0.51 |        0.0050 |
|     2 |       150 |     00:01:28 |     0.1439 |     98.10% |       1.29 |        0.0050 |
|     2 |       200 |     00:01:52 |     0.0696 |    100.00% |       1.01 |        0.0050 |
|     2 |       250 |     00:02:15 |     0.3809 |     96.19% |       1.01 |        0.0050 |
|     3 |       300 |     00:02:56 |     0.0937 |    100.00% |       1.36 |        0.0050 |
|     3 |       350 |     00:03:20 |     0.1092 |     98.85% |       1.08 |        0.0050 |
|     3 |       378 |     00:03:33 |     0.0527 |     98.86% |       0.64 |        0.0050 |
|============================================================================
=======================================|

Detector training complete.
**********************************************************************



trainedDetector =

  fasterRCNNObjectDetector with properties:
```

```
       ModelName: 'tractors'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [4×2 double]
      ClassNames: {'tractors'  'Background'}
   MinObjectSize: [16 16]

Elapsed time is 749.969774 seconds.
Elapsed time is 45.415633 seconds.
```

APPENDIX F.8.1.2.1 Result



APPENDIX F.8.1.2.2 Results on test pictures


tractors
tractors
tractors


tractors

APPENDIX F.8.1.3 Inception V3

```
>> TractorDetectorinceptionv3v4

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
        GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 1
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
          ExecutionEnvironment: 'auto'
```

```
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0100
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 1
            MiniBatchSize: 1
                  Verbose: 1
          VerboseFrequency: 50
           ValidationData: []
      ValidationFrequency: 50
       ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0100
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
```

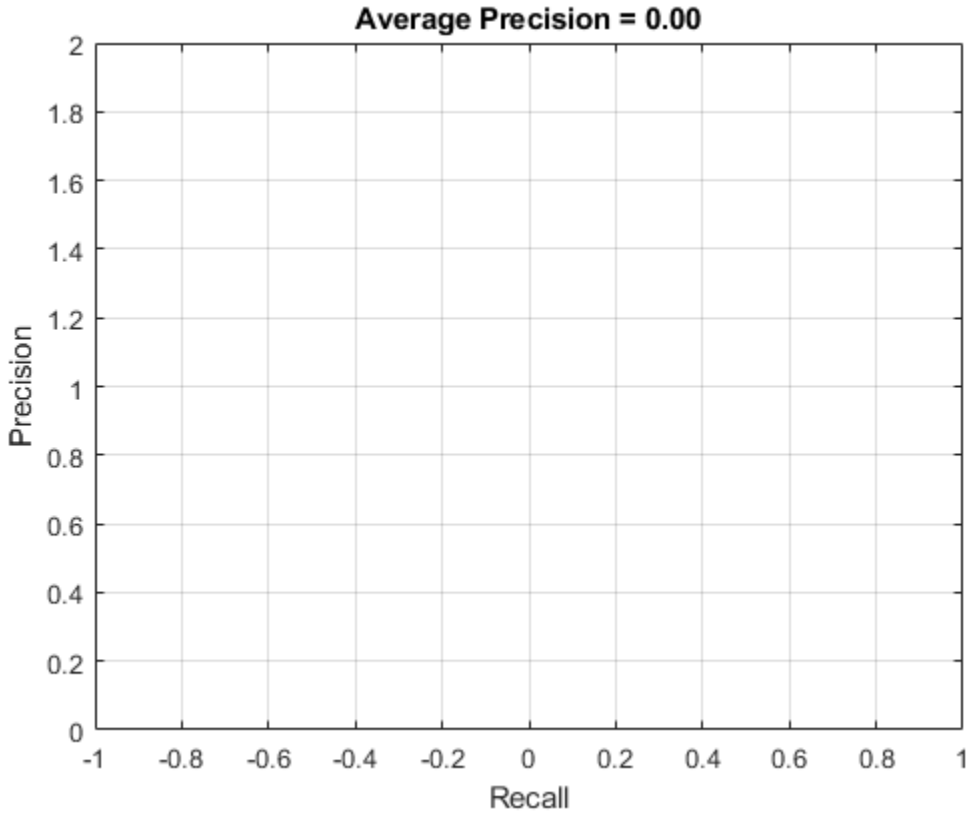
```
              MaxEpochs: 1
          MiniBatchSize: 1
                Verbose: 1
       VerboseFrequency: 50
         ValidationData: []
    ValidationFrequency: 50
     ValidationPatience: Inf
                Shuffle: 'once'
         CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
              MaxEpochs: 3
          MiniBatchSize: 1
                Verbose: 1
       VerboseFrequency: 50
         ValidationData: []
    ValidationFrequency: 50
     ValidationPatience: Inf
                Shuffle: 'once'
         CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0
```

```
********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:01 |     1.1939 |     61.72% |       0.90 |        0.0100 |
|     1 |        50 |     00:00:13 |     1.0419 |    100.00% |       1.21 |        0.0100 |
|     1 |       100 |     00:00:26 |     0.4007 |    100.00% |       0.75 |        0.0100 |
|     1 |       126 |     00:00:32 |     0.4841 |    100.00% |       0.78 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:01 |     0.8771 |      2.97% |       0.71 |        0.0100 |
|     1 |        50 |     00:00:48 |        NaN |      2.94% |        NaN |        0.0100 |
|     1 |       100 |     00:01:38 |        NaN |      1.56% |        NaN |        0.0100 |
|     1 |       126 |     00:02:03 |        NaN |      2.56% |        NaN |        0.0100 |
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================================
=======================================|
```

```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.5095 |    100.00% |       0.80 |        0.0100 |
|     1 |        50 |     00:00:09 |     0.5382 |    100.00% |       0.85 |        0.0100 |
|     1 |       100 |     00:00:18 |     0.3757 |    100.00% |       0.78 |        0.0100 |
|     1 |       126 |     00:00:23 |     0.5029 |    100.00% |       0.95 |        0.0100 |
|============================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |        NaN |      1.56% |        NaN |        0.0050 |
|     1 |        50 |     00:00:37 |        NaN |      0.78% |        NaN |        0.0050 |
|     1 |       100 |     00:01:15 |        NaN |      1.71% |        NaN |        0.0050 |
|     2 |       150 |     00:02:06 |        NaN |      1.19% |        NaN |        0.0050 |
|     2 |       200 |     00:02:43 |        NaN |      2.44% |        NaN |        0.0050 |
|     2 |       250 |     00:03:21 |        NaN |      1.96% |        NaN |        0.0050 |
|     3 |       300 |     00:04:10 |        NaN |      9.38% |        NaN |        0.0050 |
|     3 |       350 |     00:04:47 |        NaN |      1.19% |        NaN |        0.0050 |
|     3 |       378 |     00:05:09 |        NaN |      1.56% |        NaN |        0.0050 |
|============================================================================
=======================================|

Detector training complete.
**********************************************************************



trainedDetector =

  fasterRCNNObjectDetector with properties:
```

```
       ModelName: 'tractors'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [4×2 double]
      ClassNames: {'tractors'  'Background'}
   MinObjectSize: [18 18]

Elapsed time is 827.975924 seconds.
Elapsed time is 91.680995 seconds.
```

APPENDIX F.8.1.3.1 Result



APPENDIX F.8.1.4 Inception ResNet V2

```
>> TractorDetectorinceptionvresnetv2v8

optionsStage1 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
```

```
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
```

```
           SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                 Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 2
           MiniBatchSize: 1
                 Verbose: 1
         VerboseFrequency: 50
           ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                 Shuffle: 'once'
          CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
      ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                   Plots: 'none'
          SequenceLength: 'longest'
      SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                 Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 3
           MiniBatchSize: 1
                 Verbose: 1
```

```
          VerboseFrequency: 50
            ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                   Shuffle: 'once'
          CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0

********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:01 |     1.1699 |     95.31% |       0.89 |        0.0050 |
|     1 |        50 |     00:00:53 |     0.3512 |    100.00% |       0.83 |        0.0050 |
|     1 |       100 |     00:01:46 |     0.7950 |    100.00% |       1.25 |        0.0050 |
|     2 |       150 |     00:03:11 |     0.2683 |    100.00% |       0.82 |        0.0050 |
|     2 |       200 |     00:04:04 |     0.3442 |    100.00% |       0.86 |        0.0050 |
|     2 |       250 |     00:04:56 |     0.3262 |    100.00% |       1.09 |        0.0050 |
|     2 |       252 |     00:04:58 |     0.3461 |    100.00% |       0.82 |        0.0050 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
```

```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |      Rate      |
|=========================================================================
=======================================|
Warning: GPU is low on memory, which can slow performance due to additional data
transfers with main memory. Try reducing the
'MiniBatchSize' training option. This warning will not appear again unless you run the
command:
warning('on','nnet_cnn:warning:GPULowOnMemory').
|       1 |         1 |       00:00:10 |       0.9738 |       14.06% |         1.20 |          0.0050 |
Warning: While copying object of class 'gpuArray':
'Out of memory on device. To view more detail about available memory on the GPU, use
'gpuDevice()'. If the problem persists, reset
the GPU by calling 'gpuDevice(1)'.'
> In nnet.internal.cnn.DAGNetwork/forwardPropagationWithMemory (line 330)
  In nnet.internal.cnn.DAGNetwork/computeGradientsForTraining (line 562)
  In nnet.internal.cnn.Trainer/computeGradients (line 184)
  In nnet.internal.cnn.Trainer/train (line 85)
  In vision.internal.cnn.trainNetwork (line 47)
  In fastRCNNObjectDetector.train (line 190)
  In trainFasterRCNNObjectDetector (line 410)
  In TractorDetectorinceptionvresnetv2v8 (line 61)
Warning: While copying object of class 'gpuArray':
'Out of memory on device. To view more detail about available memory on the GPU, use
'gpuDevice()'. If the problem persists, reset
the GPU by calling 'gpuDevice(1)'.'
> In nnet.internal.cnn.DAGNetwork/forwardPropagationWithMemory (line 286)
  In nnet.internal.cnn.DAGNetwork/computeGradientsForTraining (line 562)
  In nnet.internal.cnn.Trainer/computeGradients (line 184)
  In nnet.internal.cnn.Trainer/train (line 85)
  In vision.internal.cnn.trainNetwork (line 47)
  In fastRCNNObjectDetector.train (line 190)
  In trainFasterRCNNObjectDetector (line 410)
  In TractorDetectorinceptionvresnetv2v8 (line 61)
Warning: While copying object of class 'gpuArray':
'Out of memory on device. To view more detail about available memory on the GPU, use
'gpuDevice()'. If the problem persists, reset
the GPU by calling 'gpuDevice(1)'.'
> In nnet.internal.cnn.DAGNetwork/forwardPropagationWithMemory (line 286)
  In nnet.internal.cnn.DAGNetwork/computeGradientsForTraining (line 562)
  In nnet.internal.cnn.Trainer/computeGradients (line 184)
  In nnet.internal.cnn.Trainer/train (line 85)
  In vision.internal.cnn.trainNetwork (line 47)
```

```
  In fastRCNNObjectDetector.train (line 190)
  In trainFasterRCNNObjectDetector (line 410)
  In TractorDetectorinceptionvresnetv2v8 (line 61)
Error using nnet.internal.cnn.layer.CustomLayer/forward (line 103)
Error using 'predict' in Layer nnet.inceptionresnetv2.layer.ScalingFactorLayer. The
function threw an error and could not be
executed.

Error in nnet.internal.cnn.DAGNetwork>@()this.Layers{i}.forward(XForThisLayer)
(line 330)
                @() this.Layers{i}.forward( XForThisLayer ), ...

Error in nnet.internal.cnn.util.executeWithStagedGPUOOMRecovery (line 11)
      [ varargout{1:nOutputs} ] = computeFun();

Error in nnet.internal.cnn.DAGNetwork>iExecuteWithStagedGPUOOMRecovery (line
1195)
[varargout{1:nargout}] =
nnet.internal.cnn.util.executeWithStagedGPUOOMRecovery(varargin{:});

Error in nnet.internal.cnn.DAGNetwork/forwardPropagationWithMemory (line 329)
            [outputActivations, memory] = iExecuteWithStagedGPUOOMRecovery( ...

Error in nnet.internal.cnn.DAGNetwork/computeGradientsForTraining (line 562)
        [activationsBuffer, memoryBuffer, layerIsLearning] =
this.forwardPropagationWithMemory(X);

Error in nnet.internal.cnn.Trainer/computeGradients (line 184)
        [gradients, predictions, states] = net.computeGradientsForTraining(X, Y,
needsStatefulTraining, propagateState);

Error in nnet.internal.cnn.Trainer/train (line 85)
                [gradients, predictions, states] = this.computeGradients(net, X, response,
needsStatefulTraining,
                propagateState);

Error in vision.internal.cnn.trainNetwork (line 47)
trainedNet = trainer.train(trainedNet, trainingDispatcher);

Error in fastRCNNObjectDetector.train (line 190)
        [network, info] = vision.internal.cnn.trainNetwork(ds, lgraph, opts, mapping,
checkpointSaver);

Error in trainFasterRCNNObjectDetector (line 410)
```

```
   [stage2Detector, fastRCNN, ~, info(2)] = fastRCNNObjectDetector.train(trainingData,
fastRCNN, options(2),
   iStageTwoParams(params), checkpointSaver);

Error in TractorDetectorinceptionvresnetv2v8 (line 61)
trainedDetector = trainFasterRCNNObjectDetector(trainingData, net,options)

Caused by:
   Error using  *
   The data no longer exists on the device.

>>
```

APPENDIX F.8.2 Run 2

```
    'MaxEpochs', 2,
    'MiniBatchSize', 1,
    'InitialLearnRate', .005,
    'CheckpointPath', tempdir
```

APPENDIX F.8.2.1 GoogLeNet

```
>> TractorDetectorgooglenetv5

optionsStage1 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
         L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
         GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                   Verbose: 1
           VerboseFrequency: 50
            ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                   Shuffle: 'once'
          CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
```

```
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 2
            MiniBatchSize: 1
                  Verbose: 1
         VerboseFrequency: 50
          ValidationData: []
      ValidationFrequency: 50
       ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
```

```
              MaxEpochs: 2
          MiniBatchSize: 1
                Verbose: 1
       VerboseFrequency: 50
         ValidationData: []
    ValidationFrequency: 50
     ValidationPatience: Inf
                Shuffle: 'once'
         CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
              MaxEpochs: 3
          MiniBatchSize: 1
                Verbose: 1
       VerboseFrequency: 50
         ValidationData: []
    ValidationFrequency: 50
     ValidationPatience: Inf
                Shuffle: 'once'
         CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0
```

```
**********************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.9025 |     57.81% |       1.80 |        0.0050 |
|     1 |        50 |     00:00:05 |     1.0249 |    100.00% |       1.10 |        0.0050 |
|     1 |       100 |     00:00:10 |     0.6775 |    100.00% |       0.88 |        0.0050 |
|     2 |       150 |     00:00:17 |     1.0044 |    100.00% |       1.35 |        0.0050 |
|     2 |       200 |     00:00:22 |     0.6509 |    100.00% |       0.88 |        0.0050 |
|     2 |       250 |     00:00:27 |     0.6383 |    100.00% |       0.89 |        0.0050 |
|     2 |       252 |     00:00:27 |     0.5738 |    100.00% |       0.83 |        0.0050 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.7994 |     33.33% |       0.59 |        0.0050 |
|     1 |        50 |     00:00:12 |     0.0503 |     99.22% |       1.76 |        0.0050 |
|     1 |       100 |     00:00:24 |     0.0575 |     99.22% |       1.03 |        0.0050 |
|     2 |       150 |     00:00:40 |     0.0431 |     98.85% |       0.46 |        0.0050 |
|     2 |       200 |     00:00:52 |     0.0616 |     97.44% |       0.60 |        0.0050 |
|     2 |       244 |     00:01:03 |     0.0468 |    100.00% |       0.77 |        0.0050 |
|=========================================================================
=======================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Mini-batch	Base Learning
		(hh:mm:ss)	Loss	Accuracy	RMSE	Rate
1	1	00:00:00	3.7144	100.00%	3.84	0.0050
1	50	00:00:02	0.4621	100.00%	0.74	0.0050
1	100	00:00:05	0.6576	100.00%	1.18	0.0050
2	150	00:00:10	0.6290	100.00%	0.86	0.0050
2	200	00:00:13	0.2639	99.22%	1.16	0.0050
2	250	00:00:15	1.3167	100.00%	1.63	0.0050
2	252	00:00:15	1.7197	100.00%	1.99	0.0050

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.

Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Mini-batch	Base Learning
		(hh:mm:ss)	Loss	Accuracy	RMSE	Rate
1	1	00:00:00	0.0421	97.30%	0.56	0.0050
1	50	00:00:09	0.0167	100.00%	0.41	0.0050
1	100	00:00:18	0.0096	100.00%	0.47	0.0050
2	150	00:00:31	0.1151	100.00%	1.45	0.0050
2	200	00:00:40	0.1554	96.88%	0.82	0.0050
2	250	00:00:49	0.0536	100.00%	0.95	0.0050
3	300	00:01:02	0.0105	100.00%	0.48	0.0050
3	350	00:01:11	0.0658	97.37%	0.32	0.0050
3	378	00:01:16	0.0110	100.00%	0.37	0.0050

Detector training complete.

****

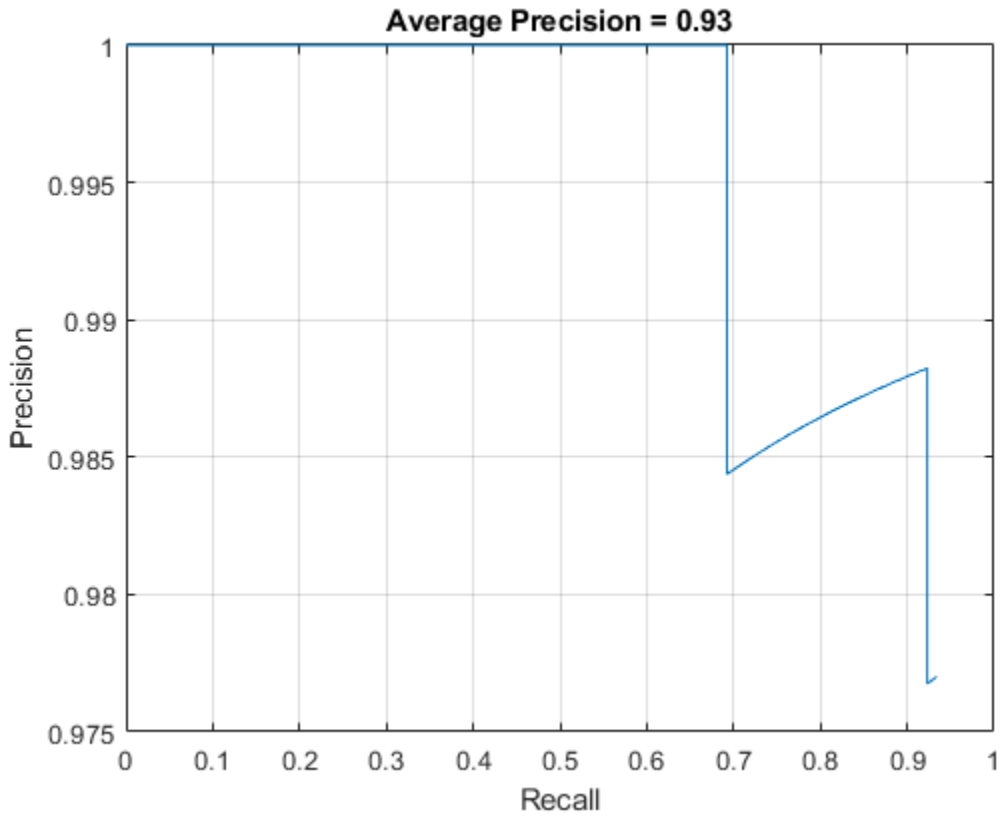
trainedDetector =

  fasterRCNNObjectDetector with properties:

        ModelName: 'tractors'
          Network: [1×1 DAGNetwork]
      AnchorBoxes: [4×2 double]
       ClassNames: {'tractors' 'Background'}
    MinObjectSize: [16 16]

Elapsed time is 309.450047 seconds.
Elapsed time is 30.981631 seconds.

APPENDIX F.8.2.1.1 Result



APPENDIX F.8.2.1.2 Results on test pictures


tractors

APPENDIX F.8.2.2 ResNet 101

```
>> TractorDetectorresnet101v6

optionsStage1 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
                MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
              SequenceLength: 'longest'
        SequencePaddingValue: 0
```

```
optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 2
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 2
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
```

```
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 3
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0

**********************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors
```

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	1.4758	73.23%	1.14	0.0050
1	50	00:00:19	0.5570	100.00%	0.96	0.0050
1	100	00:00:39	1.0036	100.00%	1.16	0.0050
2	150	00:01:13	0.3985	100.00%	0.72	0.0050
2	200	00:01:33	0.6009	100.00%	0.86	0.0050
2	250	00:01:53	0.1861	100.00%	1.03	0.0050
2	252	00:01:54	0.2884	100.00%	0.68	0.0050

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	0.7019	79.69%	1.66	0.0050
1	50	00:00:36	0.4044	95.31%	1.09	0.0050
1	100	00:01:12	0.3798	92.96%	0.63	0.0050
2	150	00:02:04	0.3179	93.24%	0.83	0.0050
2	200	00:02:41	0.1131	98.25%	0.60	0.0050
2	250	00:03:17	0.1955	97.25%	1.05	0.0050
2	252	00:03:18	0.2595	97.33%	1.28	0.0050

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.6969 |    100.00% |       1.83 |        0.0050 |
|     1 |        50 |     00:00:10 |     0.4697 |    100.00% |       0.76 |        0.0050 |
|     1 |       100 |     00:00:21 |     0.6912 |    100.00% |       1.13 |        0.0050 |
|     2 |       150 |     00:00:47 |     0.3111 |    100.00% |       0.65 |        0.0050 |
|     2 |       200 |     00:00:58 |     0.3705 |    100.00% |       0.68 |        0.0050 |
|     2 |       250 |     00:01:09 |     0.3554 |    100.00% |       0.64 |        0.0050 |
|     2 |       252 |     00:01:09 |     0.2847 |    100.00% |       0.57 |        0.0050 |
|============================================================================
=======================================|
```

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.

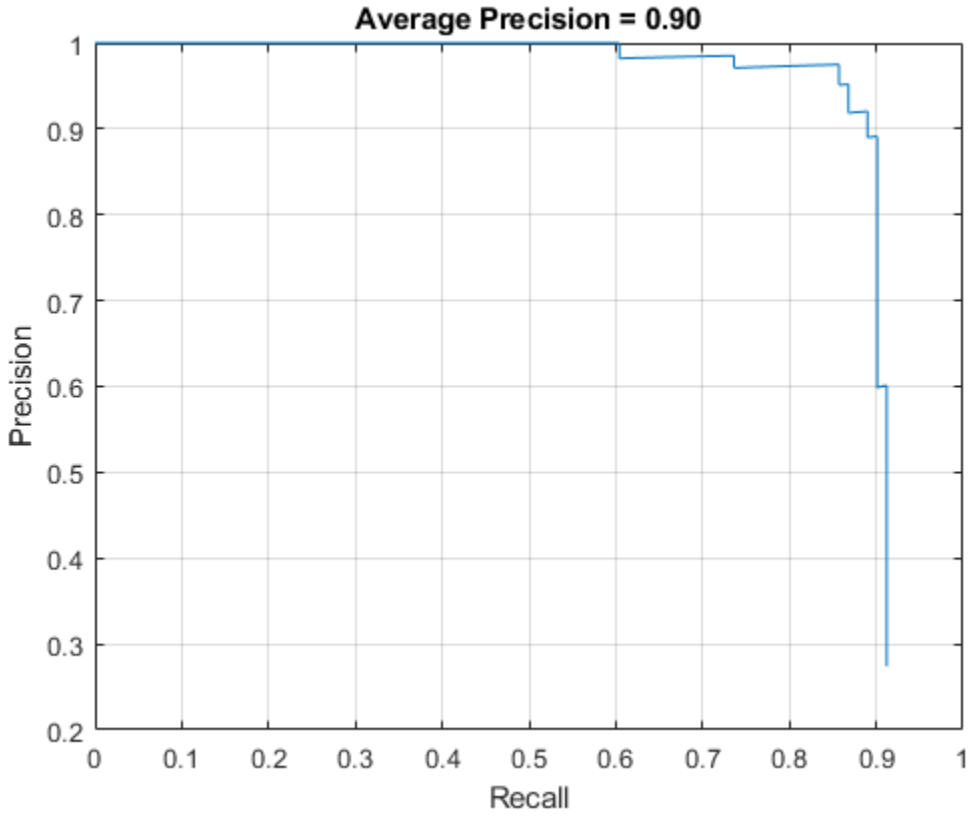
```
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.0755 |     99.22% |       1.18 |        0.0050 |
|     1 |        50 |     00:00:22 |     0.0366 |     99.22% |       1.32 |        0.0050 |
|     1 |       100 |     00:00:45 |     0.1023 |     97.53% |       0.46 |        0.0050 |
|     2 |       150 |     00:01:23 |     0.0066 |    100.00% |       0.27 |        0.0050 |
|     2 |       200 |     00:01:46 |     0.2911 |     92.19% |       1.03 |        0.0050 |
|     3 |       250 |     00:02:25 |     0.0050 |    100.00% |       0.33 |        0.0050 |
|     3 |       300 |     00:02:48 |     0.0245 |    100.00% |       0.38 |        0.0050 |
|     3 |       350 |     00:03:11 |     0.0172 |    100.00% |       0.70 |        0.0050 |
|     3 |       372 |     00:03:20 |     0.0098 |    100.00% |       0.27 |        0.0050 |
|============================================================================
=======================================|
```

Detector training complete.
****

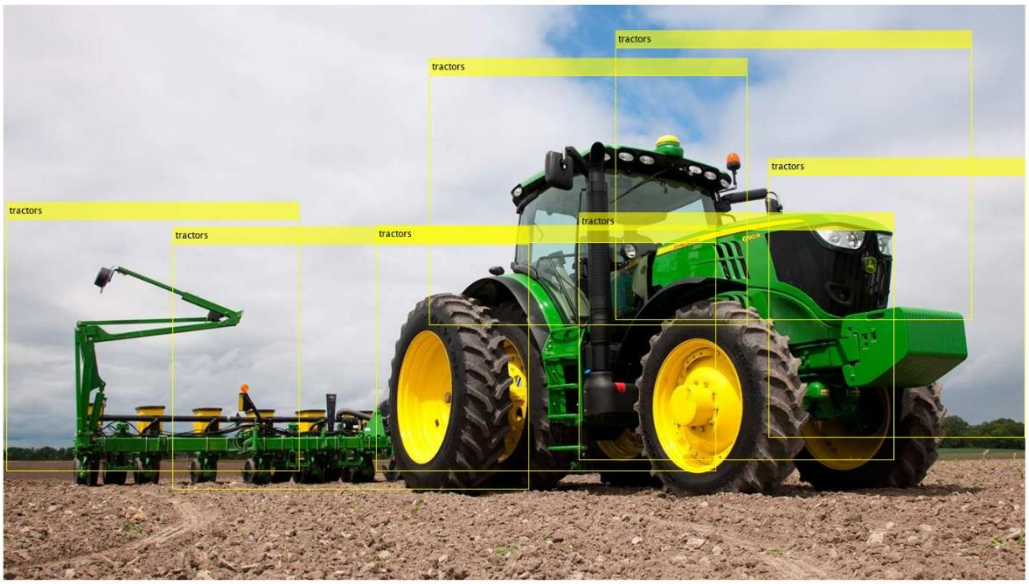
trainedDetector =

```
  fasterRCNNObjectDetector with properties:

        ModelName: 'tractors'
          Network: [1×1 DAGNetwork]
      AnchorBoxes: [4×2 double]
       ClassNames: {'tractors'  'Background'}
    MinObjectSize: [16 16]

Elapsed time is 921.760705 seconds.
Elapsed time is 49.185925 seconds.
```

APPENDIX F.8.2.2.1 Result

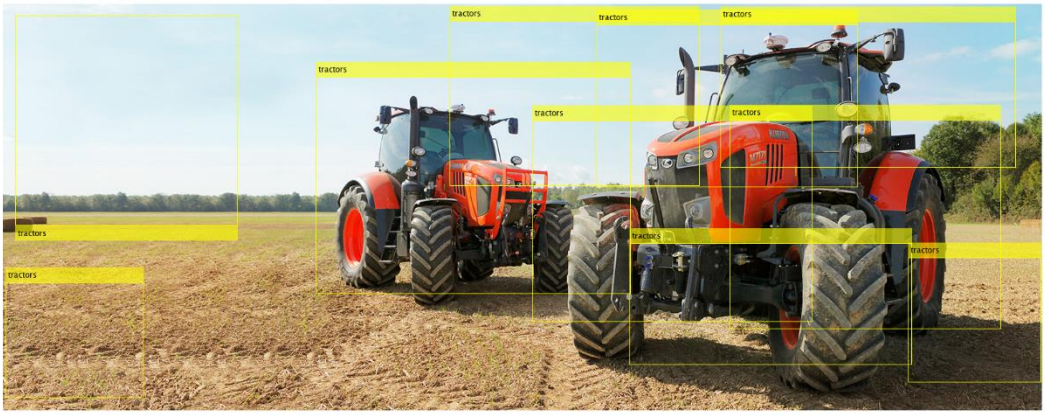


APPENDIX F.8.2.2.2 Results on test pictures




tractors


tractors



APPENDIX F.8.2.3 Inception V3

```
>> TractorDetectorinceptionv3v7

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
              SequenceLength: 'longest'
        SequencePaddingValue: 0
```

```
optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
```

```
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                   Momentum: 0.9000
             InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                   MaxEpochs: 3
                MiniBatchSize: 1
                     Verbose: 1
              VerboseFrequency: 50
                ValidationData: []
           ValidationFrequency: 50
            ValidationPatience: Inf
                    Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0

*************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors
```

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	1.1914	78.13%	0.91	0.0050
1	50	00:00:12	0.7551	100.00%	0.90	0.0050
1	100	00:00:25	0.5370	100.00%	0.73	0.0050
2	150	00:00:46	0.7253	97.64%	0.99	0.0050
2	200	00:00:59	0.9790	100.00%	1.36	0.0050
2	250	00:01:11	0.5327	100.00%	0.84	0.0050
2	252	00:01:12	0.4809	100.00%	0.89	0.0050

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.

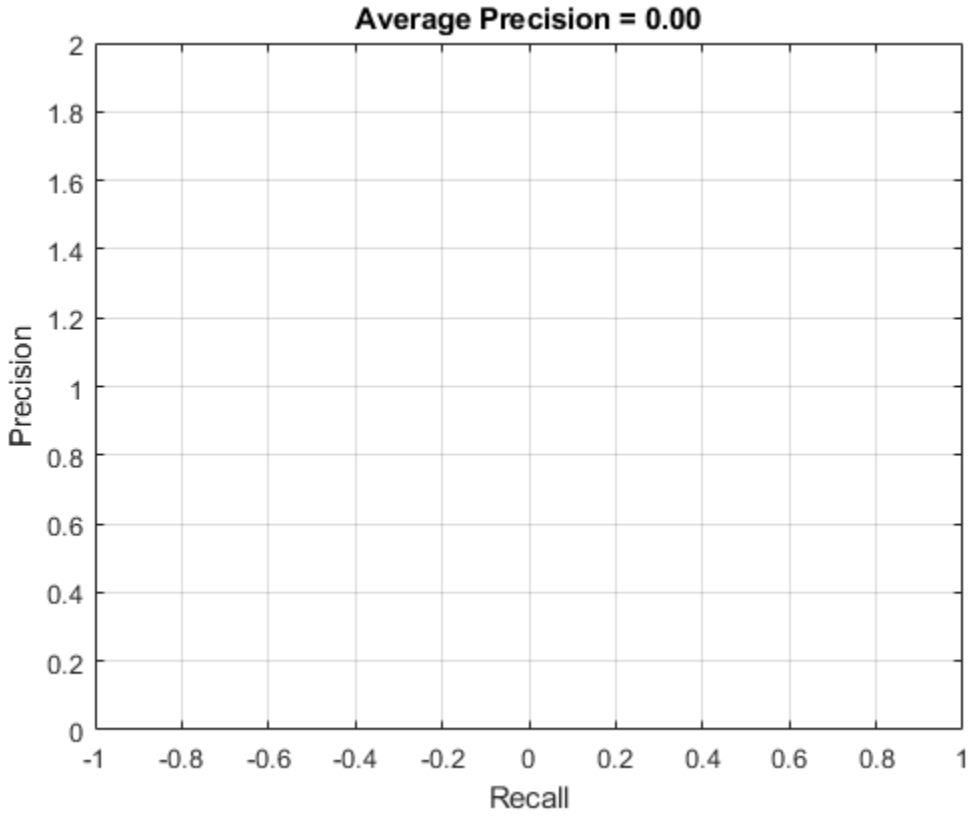
Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	0.7994	2.13%	0.72	0.0050
1	50	00:00:47	0.1704	97.65%	1.03	0.0050
1	100	00:01:34	NaN	1.56%	NaN	0.0050
2	150	00:02:34	NaN	2.53%	NaN	0.0050
2	200	00:03:21	NaN	3.19%	NaN	0.0050
2	250	00:04:09	NaN	2.56%	NaN	0.0050
2	252	00:04:11	NaN	0.78%	NaN	0.0050

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.6811 |    100.00% |       1.01 |        0.0050 |
|     1 |        50 |     00:00:09 |     0.5409 |    100.00% |       0.86 |        0.0050 |
|     1 |       100 |     00:00:18 |     0.3212 |    100.00% |       0.71 |        0.0050 |
|     2 |       150 |     00:00:35 |     0.4968 |    100.00% |       0.94 |        0.0050 |
|     2 |       200 |     00:00:44 |     0.7720 |    100.00% |       1.35 |        0.0050 |
|     2 |       250 |     00:00:53 |     0.4652 |    100.00% |       0.82 |        0.0050 |
|     2 |       252 |     00:00:54 |     0.2684 |    100.00% |       0.74 |        0.0050 |
|============================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 126 training images...done.

Training on single GPU.
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |        NaN |      2.53% |        NaN |        0.0050 |
|     1 |        50 |     00:00:36 |        NaN |      2.44% |        NaN |        0.0050 |
|     1 |       100 |     00:01:12 |        NaN |      2.56% |        NaN |        0.0050 |
|     2 |       150 |     00:02:01 |        NaN |      2.56% |        NaN |        0.0050 |
|     2 |       200 |     00:02:38 |        NaN |      2.56% |        NaN |        0.0050 |
|     3 |       250 |     00:03:26 |        NaN |      2.75% |        NaN |        0.0050 |
|     3 |       300 |     00:04:03 |        NaN |      1.56% |        NaN |        0.0050 |
|     3 |       350 |     00:04:40 |        NaN |      2.56% |        NaN |        0.0050 |
|     3 |       369 |     00:04:54 |        NaN |      2.56% |        NaN |        0.0050 |
|============================================================================
=======================================|

Detector training complete.
**********************************************************************


trainedDetector =
```

```
  fasterRCNNObjectDetector with properties:

        ModelName: 'tractors'
          Network: [1×1 DAGNetwork]
      AnchorBoxes: [4×2 double]
       ClassNames: {'tractors'  'Background'}
    MinObjectSize: [18 18]

Elapsed time is 994.296007 seconds.
Elapsed time is 88.889685 seconds.
```

APPENDIX F.8.2.3.1 Result



APPENDIX F.9 Round Bale

Image 1 – (everythingattachments.com, 2019)

Image 2 – (Southern Plains Photography, 2019)

Image 3 – (Global Auction Guide, 2019)

Image 4 – (Ohio's Country Journal, 2019)

Image 5 – (Marybeth Feutz, 2010)

APPENDIX F.9.1 Run 1

```
'MaxEpochs', 1,
'MiniBatchSize', 1,
'InitialLearnRate', .01,
'CheckpointPath', tempdir)
```

APPENDIX F.9.1.1 GoogLeNet

RBDetectorgooglenetv2

optionsStage1 =

TrainingOptionsSGDM with properties:

```
                    Momentum: 0.9000
           InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
       SequencePaddingValue: 0
```

optionsStage2 =

```
  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0100
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 1
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
                  CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
             ExecutionEnvironment: 'auto'
                      WorkerLoad: []
                       OutputFcn: []
                           Plots: 'none'
                  SequenceLength: 'longest'
             SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0100
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 1
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
```

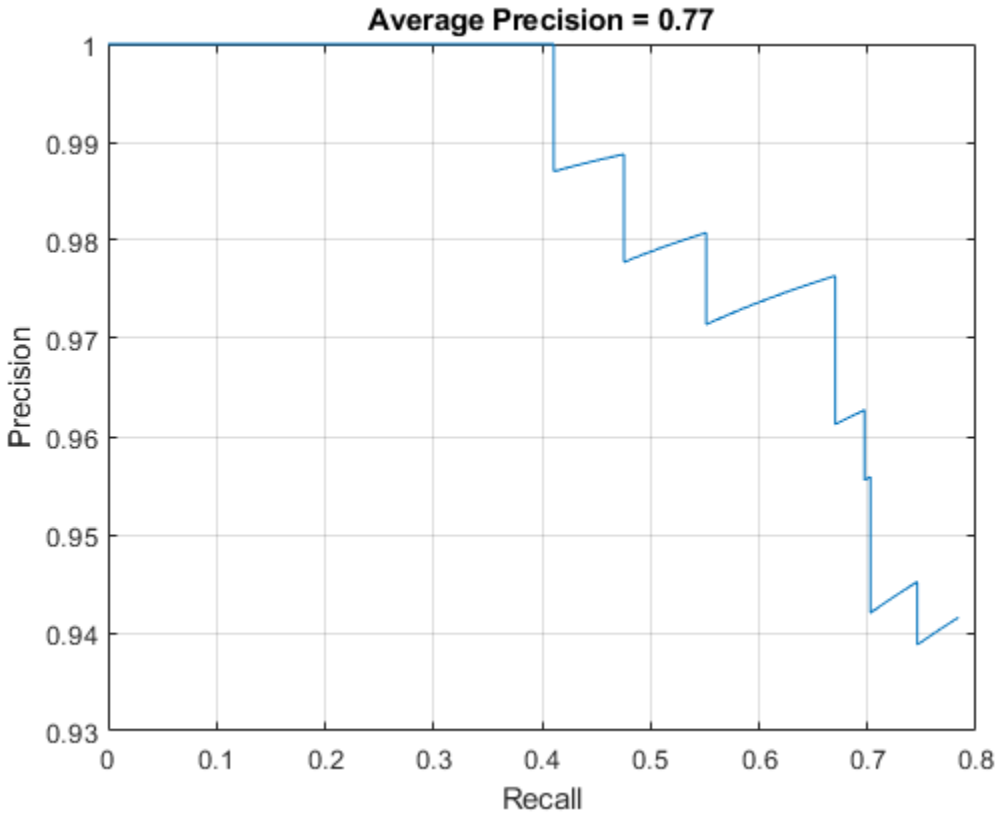
```
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0

Starting parallel pool (parpool) using the 'local' profile ...
connected to 6 workers.
************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
```

```
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:02 |     0.9764 |     85.94% |       1.37 |        0.0100 |
|     1 |        50 |     00:00:06 |     0.4098 |    100.00% |       0.91 |        0.0100 |
|     1 |       100 |     00:00:11 |     0.4329 |    100.00% |       0.75 |        0.0100 |
|     1 |       150 |     00:00:15 |     0.4199 |    100.00% |       0.71 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.2682 |     17.19% |       0.94 |        0.0100 |
|     1 |        50 |     00:00:12 |     0.4803 |     89.06% |       1.00 |        0.0100 |
|     1 |       100 |     00:00:24 |     0.2796 |     94.83% |       0.93 |        0.0100 |
|     1 |       138 |     00:00:33 |     0.2329 |     96.09% |       0.97 |        0.0100 |
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.4178 |    100.00% |       0.73 |        0.0100 |
|     1 |        50 |     00:00:02 |     0.4448 |    100.00% |       0.77 |        0.0100 |
```

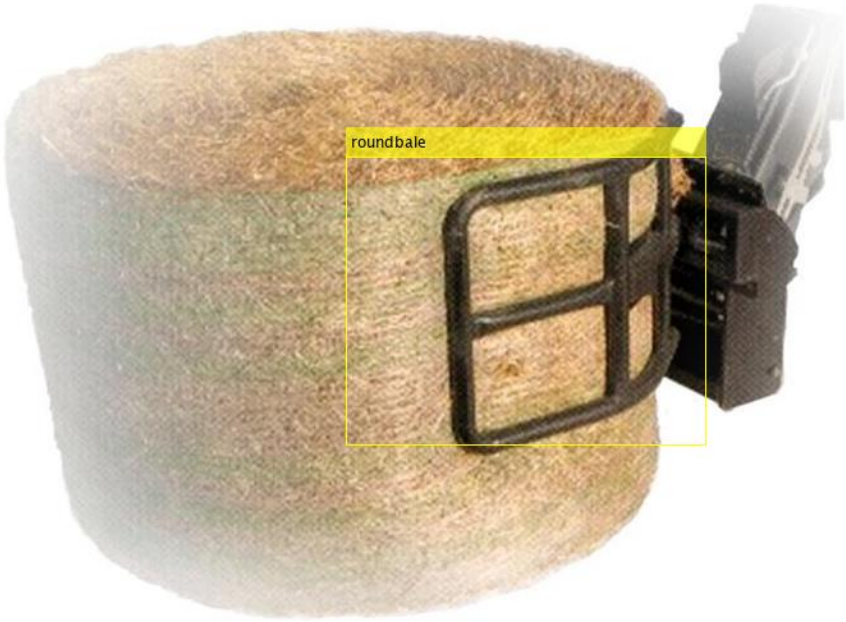
```
|      1 |        100 |      00:00:04 |      0.3046 |     100.00% |        0.79 |        0.0100 |
|      1 |        150 |      00:00:07 |      0.1029 |     100.00% |        0.88 |        0.0100 |
|========================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|========================================================================
=======================================|
|      1 |         1 |      00:00:00 |      0.2399 |      92.97% |        0.69 |        0.0050 |
|      1 |        50 |      00:00:09 |      0.1202 |      98.44% |        1.08 |        0.0050 |
|      1 |       100 |      00:00:18 |      0.1023 |      99.22% |        0.61 |        0.0050 |
|      2 |       150 |      00:00:31 |      0.1301 |      96.09% |        0.67 |        0.0050 |
|      2 |       200 |      00:00:40 |      0.2102 |      96.09% |        0.79 |        0.0050 |
|      2 |       250 |      00:00:49 |      0.2343 |      95.31% |        0.81 |        0.0050 |
|      3 |       300 |      00:01:01 |      0.1169 |      97.66% |        0.83 |        0.0050 |
|      3 |       350 |      00:01:11 |      0.0729 |      97.66% |        0.67 |        0.0050 |
|      3 |       400 |      00:01:20 |      0.1662 |      99.22% |        0.75 |        0.0050 |
|      3 |       414 |      00:01:23 |      0.1650 |      95.31% |        0.74 |        0.0050 |
|========================================================================
=======================================|

Detector training complete.
**************************************************************************



trainedDetector =

  fasterRCNNObjectDetector with properties:

        ModelName: 'roundbale'
          Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'roundbale'  'Background'}
    MinObjectSize: [16 16]
```

Elapsed time is 347.412311 seconds.
Elapsed time is 92.124512 seconds.

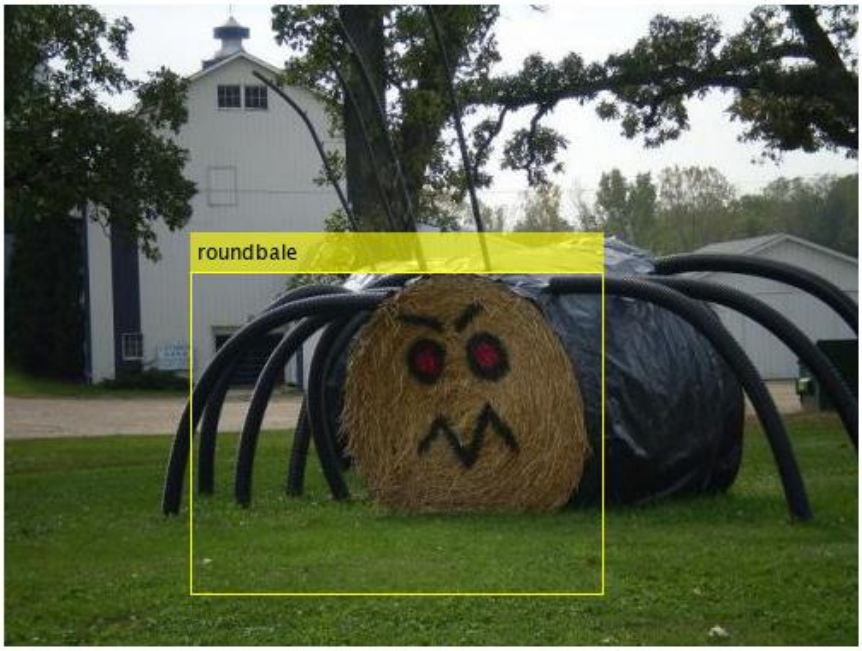
APPENDIX F.9.1.1.1 Result

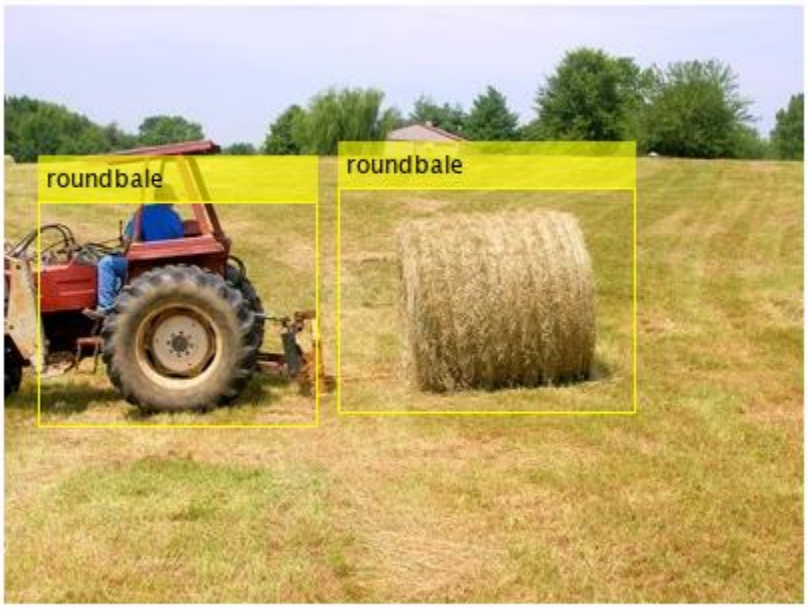


APPENDIX F.9.1.1.2 Results on test pictures




roundbale


roundbale



APPENDIX F.9.1.2 ResNet 101

```
>> RBDetectorresnet101v3

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
```

```
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0100
     LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 1
                MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0100
```

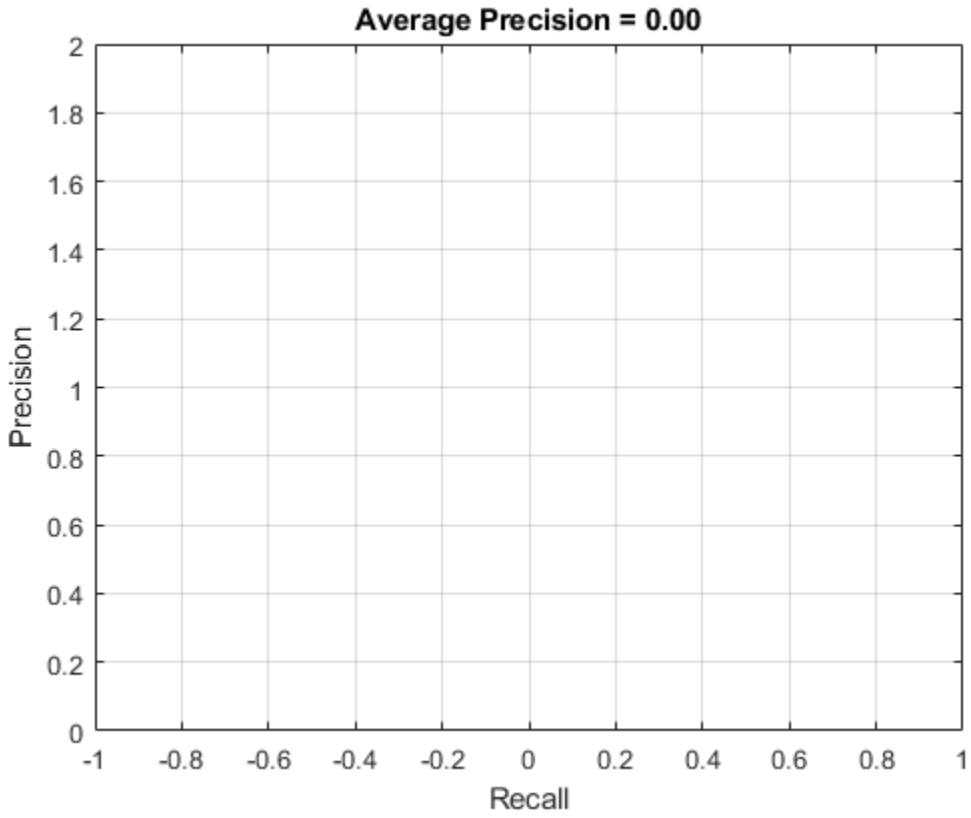
```
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 1
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
```

```
           SequenceLength: 'longest'
        SequencePaddingValue: 0

********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.8881 |     85.94% |       0.72 |        0.0100 |
|     1 |        50 |     00:00:19 |     0.6631 |    100.00% |       0.80 |        0.0100 |
|     1 |       100 |     00:00:39 |     0.4379 |    100.00% |       0.84 |        0.0100 |
|     1 |       150 |     00:00:59 |     0.5513 |    100.00% |       0.94 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:01 |     0.9438 |     35.94% |       1.16 |        0.0100 |
|     1 |        50 |     00:00:41 |        NaN |     10.94% |        NaN |        0.0100 |
|     1 |       100 |     00:01:23 |        NaN |      6.96% |        NaN |        0.0100 |
|     1 |       144 |     00:01:59 |        NaN |      3.91% |        NaN |        0.0100 |
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
```

```
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate     |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.3600 |    100.00% |       0.87 |       0.0100 |
|     1 |        50 |     00:00:10 |     0.2833 |    100.00% |       1.21 |       0.0100 |
|     1 |       100 |     00:00:21 |     0.2528 |    100.00% |       0.91 |       0.0100 |
|     1 |       150 |     00:00:32 |     2.2767 |    100.00% |       1.69 |       0.0100 |
|=========================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate     |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |        NaN |     12.50% |        NaN |       0.0050 |
|     1 |        50 |     00:00:25 |        NaN |      9.38% |        NaN |       0.0050 |
|     1 |       100 |     00:00:50 |        NaN |      4.69% |        NaN |       0.0050 |
|     2 |       150 |     00:01:29 |        NaN |      3.91% |        NaN |       0.0050 |
|     2 |       200 |     00:01:54 |        NaN |      5.47% |        NaN |       0.0050 |
|     2 |       250 |     00:02:20 |        NaN |      7.03% |        NaN |       0.0050 |
|     3 |       300 |     00:02:58 |        NaN |      2.34% |        NaN |       0.0050 |
|     3 |       350 |     00:03:24 |        NaN |     11.72% |        NaN |       0.0050 |
|     3 |       400 |     00:03:49 |        NaN |      5.47% |        NaN |       0.0050 |
|     3 |       432 |     00:04:05 |        NaN |      3.91% |        NaN |       0.0050 |
|=========================================================================
=======================================|

Detector training complete.
**************************************************************************
```

```
trainedDetector =

  fasterRCNNObjectDetector with properties:

        ModelName: 'roundbale'
          Network: [1×1 DAGNetwork]
      AnchorBoxes: [6×2 double]
       ClassNames: {'roundbale'  'Background'}
    MinObjectSize: [16 16]

Elapsed time is 920.846822 seconds.
Elapsed time is 168.548623 seconds.
```

APPENDIX F.9.1.2.1 Result



APPENDIX F.9.1.3 Inception V3

```
>> RBDetectorinceptionv3v4

optionsStage1 =
```

```
  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0100
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 1
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
                CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
            ExecutionEnvironment: 'auto'
                      WorkerLoad: []
                       OutputFcn: []
                           Plots: 'none'
                SequenceLength: 'longest'
            SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0100
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 1
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
```

```
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0100
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 1
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
```

```
    GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
              SequenceLength: 'longest'
        SequencePaddingValue: 0

*************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |      00:00:00 |      1.1171 |      92.19% |       1.09 |        0.0100 |
|     1 |        50 |      00:00:12 |      0.6419 |     100.00% |       0.78 |        0.0100 |
|     1 |       100 |      00:00:24 |      0.1943 |     100.00% |       0.96 |        0.0100 |
|     1 |       150 |      00:00:37 |      0.3896 |     100.00% |       0.90 |        0.0100 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
```

```
|=========================================================================
=====================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |     Rate      |
|=========================================================================
=====================================|
|     1 |         1 |    00:00:01 |    0.8499 |   10.16% |       1.03 |     0.0100 |
|     1 |        50 |    00:00:48 |       NaN |    7.81% |        NaN |     0.0100 |
|     1 |       100 |    00:01:36 |       NaN |    3.91% |        NaN |     0.0100 |
|     1 |       144 |    00:02:19 |       NaN |    7.14% |        NaN |     0.0100 |
|=========================================================================
=====================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
|=========================================================================
=====================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |     Rate      |
|=========================================================================
=====================================|
|     1 |         1 |    00:00:00 |    0.0801 |  100.00% |       0.86 |     0.0100 |
|     1 |        50 |    00:00:09 |    0.3153 |  100.00% |       1.01 |     0.0100 |
|     1 |       100 |    00:00:18 |    0.2837 |  100.00% |       0.85 |     0.0100 |
|     1 |       150 |    00:00:27 |    0.5945 |  100.00% |       1.00 |     0.0100 |
|=========================================================================
=====================================|
```

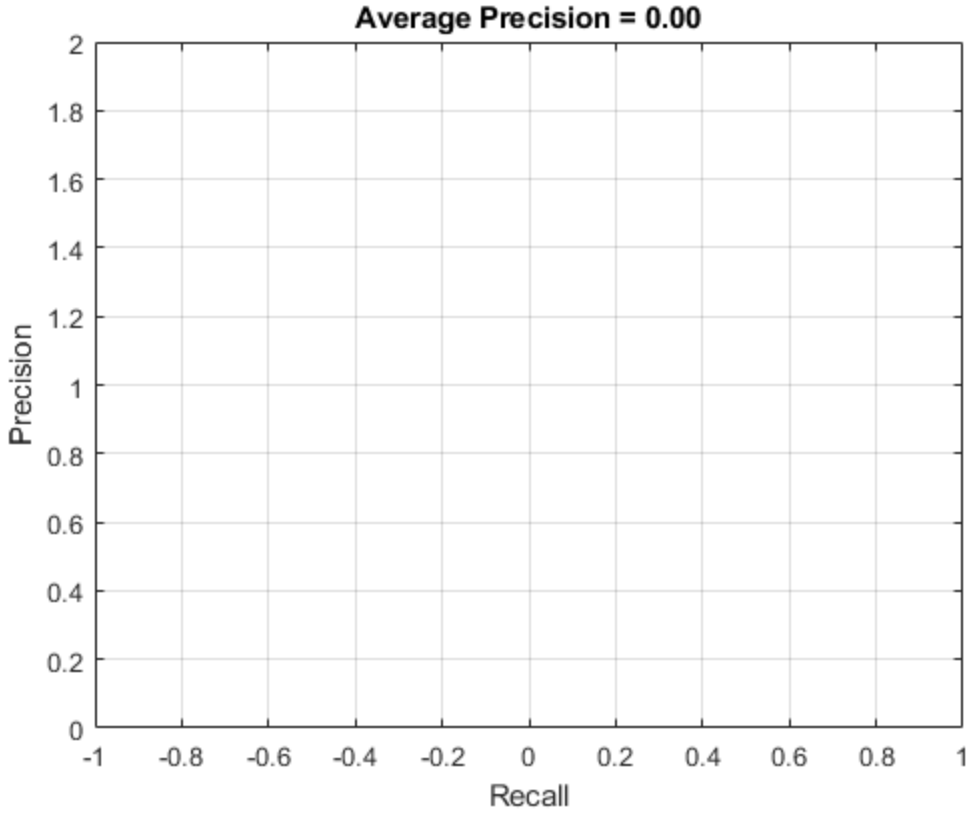
Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.

```
|=========================================================================
=====================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |     Rate      |
|=========================================================================
=====================================|
|     1 |         1 |    00:00:00 |       NaN |    6.25% |        NaN |     0.0050 |
|     1 |        50 |    00:00:36 |       NaN |    3.13% |        NaN |     0.0050 |
|     1 |       100 |    00:01:13 |       NaN |    3.60% |        NaN |     0.0050 |
```

```
|     2 |       150 |   00:02:02 |       NaN |    3.91% |       NaN |    0.0050 |
|     2 |       200 |   00:02:39 |       NaN |    6.25% |       NaN |    0.0050 |
|     2 |       250 |   00:03:17 |       NaN |    2.42% |       NaN |    0.0050 |
|     3 |       300 |   00:04:06 |       NaN |    3.91% |       NaN |    0.0050 |
|     3 |       350 |   00:04:43 |       NaN |    3.91% |       NaN |    0.0050 |
|     3 |       400 |   00:05:21 |       NaN |    7.81% |       NaN |    0.0050 |
|     3 |       432 |   00:05:45 |       NaN |    2.36% |       NaN |    0.0050 |
|=========================================================================
=======================================|

Detector training complete.
**************************************************************************


trainedDetector =

  fasterRCNNObjectDetector with properties:

        ModelName: 'roundbale'
          Network: [1×1 DAGNetwork]
      AnchorBoxes: [6×2 double]
       ClassNames: {'roundbale'  'Background'}
    MinObjectSize: [18 18]

Elapsed time is 1073.125178 seconds.
Elapsed time is 278.413001 seconds.
```

APPENDIX F.9.1.3.1 Result



APPENDIX F.9.2 Run 2

```
'MaxEpochs', 2,
'MiniBatchSize', 1,
'InitialLearnRate', .005,
'CheckpointPath', tempdir
```

APPENDIX F.9.2.1 GoogLeNet

>> RBDetectorgooglenetv5

optionsStage1 =

TrainingOptionsSGDM with properties:

Momentum: 0.9000
InitialLearnRate: 0.0050
LearnRateScheduleSettings: [1×1 struct]

```
            L2Regularization: 1.0000e-04
        GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                MaxEpochs: 2
            MiniBatchSize: 1
                Verbose: 1
            VerboseFrequency: 50
            ValidationData: []
        ValidationFrequency: 50
        ValidationPatience: Inf
                Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                    Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
            L2Regularization: 1.0000e-04
        GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                MaxEpochs: 2
            MiniBatchSize: 1
                Verbose: 1
            VerboseFrequency: 50
            ValidationData: []
        ValidationFrequency: 50
        ValidationPatience: Inf
                Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                    Plots: 'none'
            SequenceLength: 'longest'
```

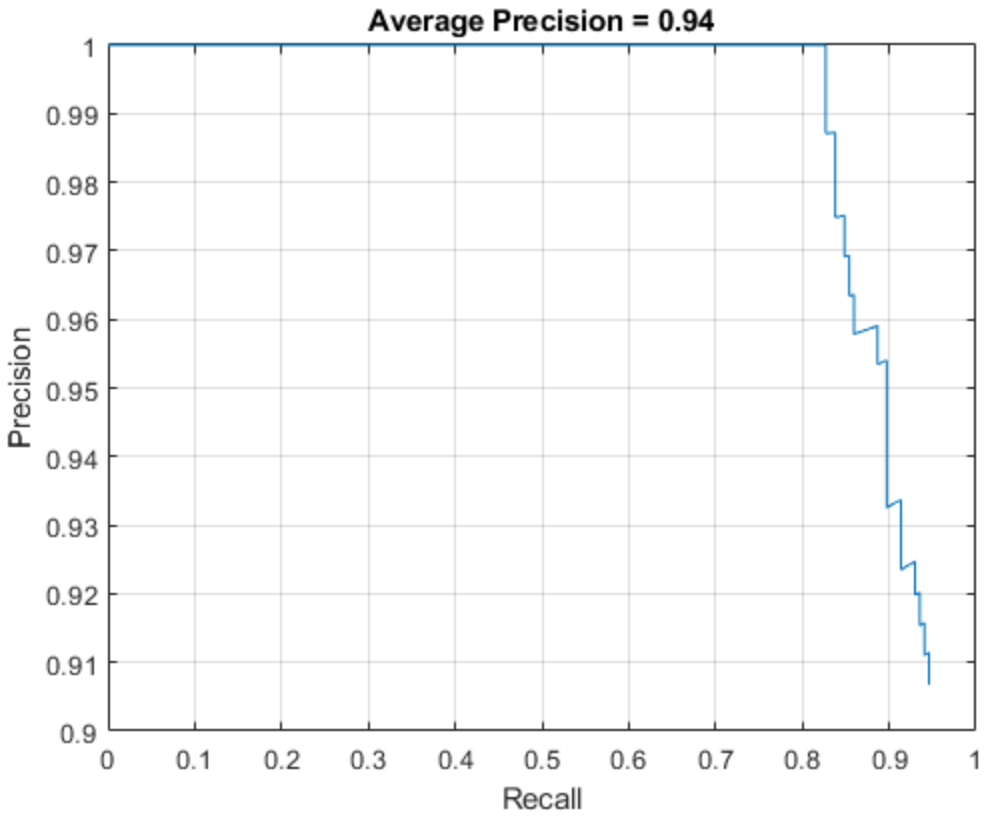
```
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                  Plots: 'none'
           SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 3
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
```

```
          ValidationData: []
      ValidationFrequency: 50
       ValidationPatience: Inf
                  Shuffle: 'once'
         CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0

*************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.9764 |     85.94% |       1.37 |        0.0050 |
|     1 |        50 |     00:00:04 |     0.0788 |    100.00% |       0.78 |        0.0050 |
|     1 |       100 |     00:00:09 |     0.3363 |    100.00% |       0.80 |        0.0050 |
|     1 |       150 |     00:00:14 |     0.3335 |    100.00% |       0.72 |        0.0050 |
|     2 |       200 |     00:00:21 |     0.1540 |    100.00% |       1.15 |        0.0050 |
|     2 |       250 |     00:00:26 |     0.4405 |    100.00% |       0.95 |        0.0050 |
|     2 |       300 |     00:00:30 |     0.2091 |    100.00% |       0.55 |        0.0050 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
```

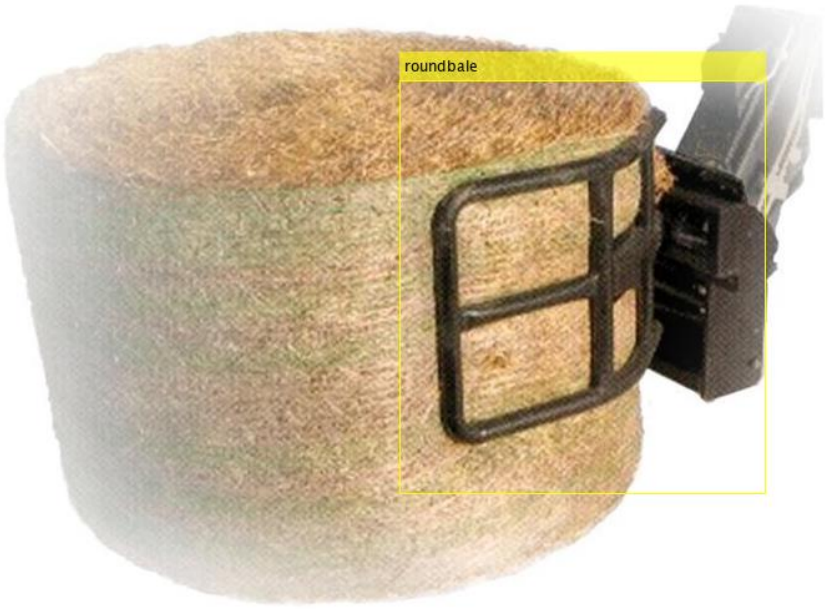
```
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     1.2814 |     13.28% |       0.96 |        0.0050 |
|     1 |        50 |     00:00:12 |     0.2622 |     95.50% |       0.98 |        0.0050 |
|     1 |       100 |     00:00:26 |     0.3400 |     95.90% |       1.05 |        0.0050 |
|     1 |       150 |     00:00:39 |     0.3979 |     89.84% |       1.00 |        0.0050 |
|     2 |       200 |     00:00:55 |     0.1793 |     95.50% |       0.91 |        0.0050 |
|     2 |       250 |     00:01:08 |     0.1960 |     95.90% |       1.05 |        0.0050 |
|     2 |       300 |     00:01:21 |     0.3175 |     96.09% |       1.08 |        0.0050 |
|============================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|============================================================================
=======================================|
|     1 |         1 |     00:00:00 |     2.5358 |    100.00% |       2.34 |        0.0050 |
|     1 |        50 |     00:00:02 |     0.0533 |    100.00% |       0.80 |        0.0050 |
|     1 |       100 |     00:00:05 |     0.7800 |    100.00% |       1.12 |        0.0050 |
|     1 |       150 |     00:00:07 |     0.5989 |    100.00% |       0.99 |        0.0050 |
|     2 |       200 |     00:00:12 |     0.0752 |    100.00% |       0.85 |        0.0050 |
|     2 |       250 |     00:00:15 |     0.3513 |    100.00% |       0.68 |        0.0050 |
|     2 |       300 |     00:00:17 |     0.6784 |    100.00% |       1.01 |        0.0050 |
|============================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|============================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
```

```
|=========================================================================
=======================================|
|     1 |       1 |    00:00:00 |     0.5144 |    85.94% |       0.93 |       0.0050 |
|     1 |      50 |    00:00:08 |     0.5818 |    85.16% |       0.81 |       0.0050 |
|     1 |     100 |    00:00:17 |     0.1804 |    94.90% |       0.73 |       0.0050 |
|     1 |     150 |    00:00:27 |     0.3859 |    93.75% |       0.83 |       0.0050 |
|     2 |     200 |    00:00:39 |     0.4119 |    89.06% |       0.69 |       0.0050 |
|     2 |     250 |    00:00:48 |     0.1163 |    96.94% |       0.57 |       0.0050 |
|     2 |     300 |    00:00:57 |     0.3409 |    92.97% |       0.71 |       0.0050 |
|     3 |     350 |    00:01:09 |     0.3053 |    95.31% |       0.63 |       0.0050 |
|     3 |     400 |    00:01:18 |     0.1272 |    94.90% |       0.48 |       0.0050 |
|     3 |     450 |    00:01:27 |     0.2393 |    96.09% |       0.61 |       0.0050 |
|=========================================================================
=======================================|

Detector training complete.
**************************************************************************


trainedDetector =

  fasterRCNNObjectDetector with properties:

       ModelName: 'roundbale'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'roundbale'  'Background'}
    MinObjectSize: [16 16]

Elapsed time is 404.969511 seconds.
Elapsed time is 95.141504 seconds.
```

APPENDIX F.9.2.1.1 Result



APPENDIX F.9.2.1.2 Results on test pictures




roundbale
roundbale


roundbale
roundbale

APPENDIX F.9.2.2 ResNet 101

```
>> RBDetectorresnet101v6

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
```

```
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
     LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
           VerboseFrequency: 50
            ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
              CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
             SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
```

```
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
```

```
            SequenceLength: 'longest'
         SequencePaddingValue: 0

********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.8881 |     85.94% |       0.72 |        0.0050 |
|     1 |        50 |     00:00:20 |     0.3295 |    100.00% |       0.77 |        0.0050 |
|     1 |       100 |     00:00:40 |     0.4481 |    100.00% |       1.02 |        0.0050 |
|     1 |       150 |     00:01:00 |     0.8593 |    100.00% |       1.40 |        0.0050 |
|     2 |       200 |     00:01:35 |     0.9569 |    100.00% |       1.33 |        0.0050 |
|     2 |       250 |     00:01:55 |     0.5256 |    100.00% |       1.10 |        0.0050 |
|     2 |       300 |     00:02:15 |     0.2866 |    100.00% |       0.69 |        0.0050 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:01 |     1.0823 |     42.19% |       1.13 |        0.0050 |
|     1 |        50 |     00:00:41 |        NaN |     10.16% |        NaN |        0.0050 |
|     1 |       100 |     00:01:22 |        NaN |      2.34% |        NaN |        0.0050 |
|     1 |       150 |     00:02:04 |        NaN |      6.03% |        NaN |        0.0050 |
|     2 |       200 |     00:03:01 |        NaN |     10.16% |        NaN |        0.0050 |
```

```
|     2 |       250 |      00:03:42 |         NaN |       2.34% |         NaN |       0.0050 |
|     2 |       300 |      00:04:25 |         NaN |       6.03% |         NaN |       0.0050 |
|=========================================================================
=======================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate     |
|=========================================================================
=======================================|
|     1 |         1 |      00:00:00 |      1.4161 |     100.00% |       1.14 |       0.0050 |
|     1 |        50 |      00:00:10 |      0.5600 |     100.00% |       0.74 |       0.0050 |
|     1 |       100 |      00:00:21 |      0.4805 |     100.00% |       0.80 |       0.0050 |
|     1 |       150 |      00:00:32 |      0.5961 |     100.00% |       1.05 |       0.0050 |
|     2 |       200 |      00:00:56 |      0.3635 |     100.00% |       0.72 |       0.0050 |
|     2 |       250 |      00:01:07 |      0.3745 |     100.00% |       0.77 |       0.0050 |
|     2 |       300 |      00:01:18 |      0.5419 |     100.00% |       1.03 |       0.0050 |
|=========================================================================
=======================================|
```

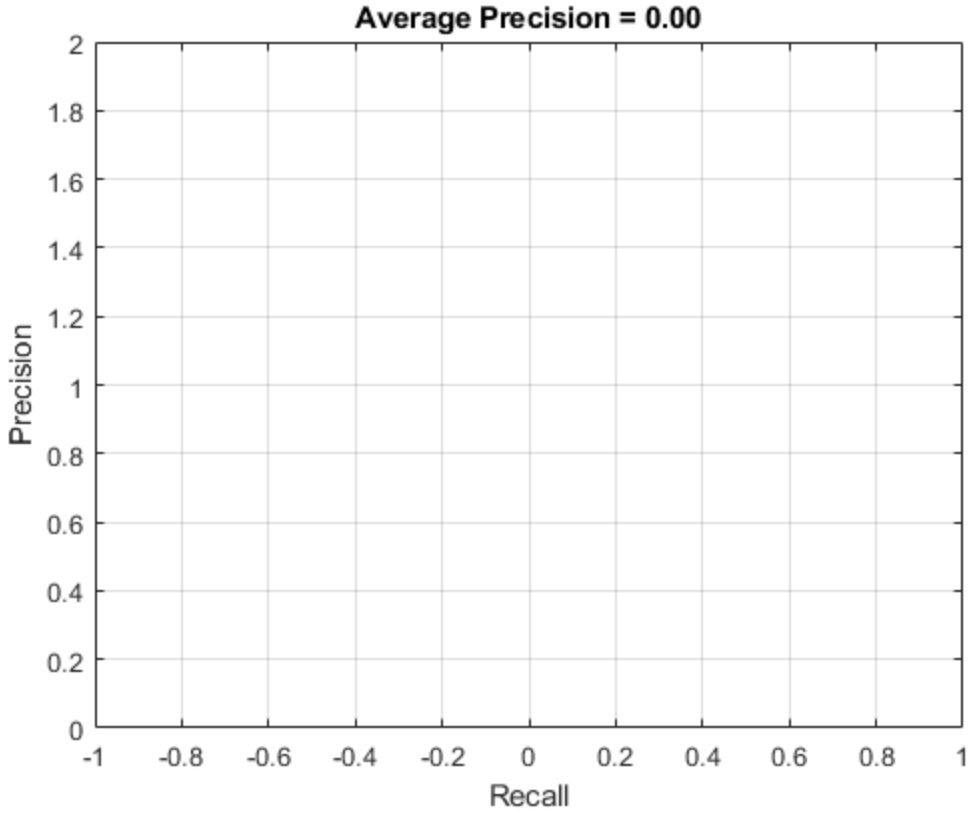
Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 150 training images...done.

Training on single GPU.

```
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate     |
|=========================================================================
=======================================|
|     1 |         1 |      00:00:00 |         NaN |      10.16% |         NaN |       0.0050 |
|     1 |        50 |      00:00:25 |         NaN |      10.16% |         NaN |       0.0050 |
|     1 |       100 |      00:00:50 |         NaN |       3.13% |         NaN |       0.0050 |
|     2 |       150 |      00:01:31 |         NaN |      11.72% |         NaN |       0.0050 |
|     2 |       200 |      00:01:56 |         NaN |       7.81% |         NaN |       0.0050 |
|     2 |       250 |      00:02:21 |         NaN |       5.36% |         NaN |       0.0050 |
|     3 |       300 |      00:03:02 |         NaN |       5.47% |         NaN |       0.0050 |
|     3 |       350 |      00:03:28 |         NaN |       3.06% |         NaN |       0.0050 |
|     3 |       400 |      00:03:53 |         NaN |      10.16% |         NaN |       0.0050 |
```

```
|      3 |        432 |       00:04:10 |         NaN |       10.16% |         NaN |          0.0050 |
|========================================================================
========================================|

Detector training complete.
**********************************************************************



trainedDetector =

  fasterRCNNObjectDetector with properties:

       ModelName: 'roundbale'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'roundbale'  'Background'}
   MinObjectSize: [16 16]

Elapsed time is 1183.976877 seconds.
Elapsed time is 159.679516 seconds.
```

APPENDIX F.9.2.2.1 Result



APPENDIX F.9.2.3 Inception V3

```
>> RBDetectorinceptionv3v7

optionsStage1 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
           VerboseFrequency: 50
```

```
           ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                   Shuffle: 'once'
          CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
        SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
         L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
            ValidationData: []
       ValidationFrequency: 50
        ValidationPatience: Inf
                   Shuffle: 'once'
          CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
             WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
        SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:
```

```
                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                    Plots: 'none'
           SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 3
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
```

```
            OutputFcn: []
                Plots: 'none'
       SequenceLength: 'longest'
    SequencePaddingValue: 0

********************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* roundbale

Step 1 of 4: Training a Region Proposal Network (RPN).
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    | Accuracy   |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.8979 |     39.84% |       0.99 |        0.0050 |
|     1 |        50 |     00:00:12 |     0.6468 |    100.00% |       1.30 |        0.0050 |
|     1 |       100 |     00:00:24 |        NaN |     32.03% |        NaN |        0.0050 |
|     1 |       150 |     00:00:37 |        NaN |      8.59% |        NaN |        0.0050 |
|     2 |       200 |     00:00:57 |        NaN |     21.09% |        NaN |        0.0050 |
|     2 |       250 |     00:01:09 |        NaN |     32.03% |        NaN |        0.0050 |
|     2 |       300 |     00:01:22 |        NaN |      8.59% |        NaN |        0.0050 |
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 150 training images...done.

Error using vision.internal.cnn.fastrcnn.imageCentricRegionDatastore (line 153)
Unable to find any region proposals to use as positive training samples. Lower the first
value of PositiveOverlapRange to increase
the number of positive region proposals.

Error in fastRCNNObjectDetector/createTrainingDatastore (line 1212)
        ds = vision.internal.cnn.fastrcnn.imageCentricRegionDatastore(...

Error in fastRCNNObjectDetector.train (line 174)
        ds = fastRCNNObjectDetector.createTrainingDatastore(...
```

Error in trainFasterRCNNObjectDetector (line 410)
[stage2Detector, fastRCNN, ~, info(2)] = fastRCNNObjectDetector.train(trainingData, fastRCNN, options(2),
iStageTwoParams(params), checkpointSaver);

Error in RBDetectorinceptionv3v7 (line 61)
trainedDetector = trainFasterRCNNObjectDetector(trainingData, net,options)

APPENDIX F.9.2.3.1 Result

Error caused no training to take place

APPENDIX F.10 Pivot

Image 1 – (Mesa Irrigation Co in Lamesa , TX, 2019)

Image 2 – (WorldAgNetwork, 2016)

Image 3 – (Sheridan Reality & Auction Co., 2017)

Image 4 – (newwayirrigation.com, 2019)

Image 5 – (pivotsplus.com, 2019)

Image 6 – (Chicot Irrigation, 2019)

Image 7 – (RainFine Irrigation Solution, 2019)

APPENDIX F.10.1 Run 1

APPENDIX F.10.1.1 GoogLeNet

>> pivotDetectorGoogLeNetv1

optionsStage1 =

TrainingOptionsSGDM with properties:

Momentum: 0.9000
InitialLearnRate: 0.0050
LearnRateScheduleSettings: [1×1 struct]
L2Regularization: 1.0000e-04
GradientThresholdMethod: 'l2norm'
GradientThreshold: Inf
MaxEpochs: 2

```
            MiniBatchSize: 1
                  Verbose: 1
            VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                  Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                   Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
            MiniBatchSize: 1
                  Verbose: 1
            VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                  Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                   Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =
```

```
  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0050
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 2
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
              ValidationFrequency: 50
               ValidationPatience: Inf
                         Shuffle: 'once'
                  CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
             ExecutionEnvironment: 'auto'
                      WorkerLoad: []
                       OutputFcn: []
                           Plots: 'none'
                  SequenceLength: 'longest'
             SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0050
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 3
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
              ValidationFrequency: 50
               ValidationPatience: Inf
                         Shuffle: 'once'
```

```
        CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
            WorkerLoad: []
             OutputFcn: []
                 Plots: 'none'
        SequenceLength: 'longest'
    SequencePaddingValue: 0
```

****
*
Training a Faster R-CNN Object Detector for the following object classes:

* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The following rows in trainingData have invalid
bounding box data:

Invalid Rows
_____

20
49
60
65
82
103
131
145
166
178
179
189
218
232
270
271
275
294
298

322
361
367
388
389
438
514
540
601
617
619
637
647
654
685
697
731

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.

Epoch	Iteration	Time Elapsed (hh:mm:ss)	Mini-batch Loss	Mini-batch Accuracy	Mini-batch RMSE	Base Learning Rate
1	1	00:00:00	1.1908	31.25%	1.60	0.0050
1	50	00:00:05	0.2303	100.00%	0.84	0.0050
1	100	00:00:10	0.0889	100.00%	0.72	0.0050
1	150	00:00:14	0.0823	100.00%	1.17	0.0050
1	200	00:00:19	0.5794	100.00%	1.07	0.0050
1	250	00:00:24	0.1718	100.00%	0.61	0.0050
1	300	00:00:29	0.0431	100.00%	0.58	0.0050
1	350	00:00:34	0.1655	100.00%	0.76	0.0050
1	400	00:00:39	0.1071	100.00%	0.72	0.0050
1	450	00:00:43	0.0809	100.00%	0.75	0.0050
1	500	00:00:48	0.1267	100.00%	0.65	0.0050
1	550	00:00:53	0.0911	100.00%	0.78	0.0050
1	600	00:00:58	0.0666	100.00%	0.57	0.0050
1	650	00:01:03	0.0693	100.00%	0.44	0.0050
1	700	00:01:08	0.0312	100.00%	0.31	0.0050
2	750	00:01:15	0.0035	100.00%	0.30	0.0050

```
|     2 |       800 |      00:01:20 |      0.0274 |     100.00% |       0.65 |       0.0050 |
|     2 |       850 |      00:01:25 |      0.0314 |     100.00% |       0.42 |       0.0050 |
|     2 |       900 |      00:01:30 |      0.4175 |     100.00% |       1.40 |       0.0050 |
|     2 |       950 |      00:01:35 |      0.0888 |     100.00% |       0.44 |       0.0050 |
|     2 |      1000 |      00:01:39 |      0.5475 |     100.00% |       0.95 |       0.0050 |
|     2 |      1050 |      00:01:44 |      0.0390 |     100.00% |       0.49 |       0.0050 |
|     2 |      1100 |      00:01:49 |      0.0161 |     100.00% |       0.71 |       0.0050 |
|     2 |      1150 |      00:01:54 |      0.0080 |     100.00% |       0.24 |       0.0050 |
|     2 |      1200 |      00:01:59 |      0.0345 |     100.00% |       0.33 |       0.0050 |
|     2 |      1250 |      00:02:04 |      0.0721 |     100.00% |       0.88 |       0.0050 |
|     2 |      1300 |      00:02:09 |      0.0348 |     100.00% |       0.31 |       0.0050 |
|     2 |      1350 |      00:02:13 |      0.0289 |     100.00% |       0.51 |       0.0050 |
|     2 |      1400 |      00:02:18 |      0.6019 |     100.00% |       1.93 |       0.0050 |
|     2 |      1450 |      00:02:23 |      0.0507 |     100.00% |       0.50 |       0.0050 |
|     2 |      1472 |      00:02:25 |      0.1358 |     100.00% |       0.63 |       0.0050 |
|=========================================================================
=======================================|
```

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.

```
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |      00:00:00 |      1.9079 |      8.59% |       0.44 |       0.0050 |
|     1 |        50 |      00:00:12 |      0.3323 |     92.97% |       0.48 |       0.0050 |
|     1 |       100 |      00:00:25 |      0.6256 |     84.38% |       0.43 |       0.0050 |
|     1 |       150 |      00:00:37 |      0.1128 |     98.51% |       0.38 |       0.0050 |
|     1 |       200 |      00:00:50 |      0.2967 |     92.16% |       0.32 |       0.0050 |
|     1 |       250 |      00:01:02 |      0.1984 |     95.24% |       0.36 |       0.0050 |
|     1 |       300 |      00:01:15 |      0.2357 |     94.53% |       0.37 |       0.0050 |
|     1 |       350 |      00:01:27 |      0.2655 |     93.75% |       0.35 |       0.0050 |
|     1 |       400 |      00:01:40 |      0.3230 |     90.63% |       0.43 |       0.0050 |
|     1 |       450 |      00:01:52 |      0.4677 |     86.72% |       0.44 |       0.0050 |
|     1 |       500 |      00:02:05 |      0.3455 |     93.97% |       0.45 |       0.0050 |
|     1 |       550 |      00:02:17 |      0.2663 |     93.75% |       0.49 |       0.0050 |
|     1 |       600 |      00:02:29 |      0.3608 |     90.63% |       0.39 |       0.0050 |
|     1 |       650 |      00:02:42 |      0.2770 |     92.19% |       0.39 |       0.0050 |
|     1 |       700 |      00:02:54 |      0.1447 |     95.31% |       0.26 |       0.0050 |
```

```
|     2 |        750 |      00:03:09 |      0.1995 |      95.31% |       0.49 |       0.0050 |
|     2 |        800 |      00:03:22 |      0.2103 |      95.31% |       0.30 |       0.0050 |
|     2 |        850 |      00:03:34 |      0.1183 |      96.88% |       0.33 |       0.0050 |
|     2 |        900 |      00:03:47 |      0.2664 |      96.09% |       0.43 |       0.0050 |
|     2 |        950 |      00:04:00 |      0.1525 |      96.09% |       0.29 |       0.0050 |
|     2 |       1000 |      00:04:12 |      0.1741 |      97.66% |       0.25 |       0.0050 |
|     2 |       1050 |      00:04:25 |      0.1611 |      95.24% |       0.30 |       0.0050 |
|     2 |       1100 |      00:04:38 |      0.2815 |      95.31% |       0.41 |       0.0050 |
|     2 |       1150 |      00:04:50 |      0.1346 |      97.66% |       0.26 |       0.0050 |
|     2 |       1200 |      00:05:03 |      0.6938 |      91.41% |       0.58 |       0.0050 |
|     2 |       1250 |      00:05:15 |      0.1353 |      98.44% |       0.33 |       0.0050 |
|     2 |       1300 |      00:05:27 |      0.0605 |     100.00% |       0.27 |       0.0050 |
|     2 |       1350 |      00:05:40 |      0.3171 |      92.97% |       0.33 |       0.0050 |
|     2 |       1400 |      00:05:53 |      0.0782 |      98.44% |       0.26 |       0.0050 |
|     2 |       1450 |      00:06:05 |      0.1727 |      96.09% |       0.24 |       0.0050 |
|     2 |       1466 |      00:06:09 |      0.1023 |      97.66% |       0.23 |       0.0050 |
|=======================================================================
=======================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
|=======================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=======================================================================
=======================================|
|     1 |         1 |      00:00:00 |      0.5346 |     100.00% |       1.48 |       0.0050 |
|     1 |        50 |      00:00:02 |      0.9156 |     100.00% |       1.27 |       0.0050 |
|     1 |       100 |      00:00:05 |      0.0186 |     100.00% |       0.41 |       0.0050 |
|     1 |       150 |      00:00:08 |      0.1738 |     100.00% |       0.83 |       0.0050 |
|     1 |       200 |      00:00:11 |      0.3531 |     100.00% |       0.88 |       0.0050 |
|     1 |       250 |      00:00:13 |      0.3582 |     100.00% |       0.95 |       0.0050 |
|     1 |       300 |      00:00:16 |      0.1939 |     100.00% |       0.65 |       0.0050 |
|     1 |       350 |      00:00:19 |      0.8208 |     100.00% |       1.71 |       0.0050 |
|     1 |       400 |      00:00:22 |      0.1159 |     100.00% |       0.61 |       0.0050 |
|     1 |       450 |      00:00:24 |      0.0247 |     100.00% |       0.59 |       0.0050 |
|     1 |       500 |      00:00:27 |      0.0197 |     100.00% |       0.49 |       0.0050 |
|     1 |       550 |      00:00:30 |      0.0161 |     100.00% |       0.49 |       0.0050 |
|     1 |       600 |      00:00:33 |      1.0538 |     100.00% |       2.17 |       0.0050 |
|     1 |       650 |      00:00:35 |      0.0939 |     100.00% |       0.74 |       0.0050 |
|     1 |       700 |      00:00:38 |      0.0100 |     100.00% |       0.38 |       0.0050 |
|     2 |       750 |      00:00:43 |      0.2990 |     100.00% |       0.90 |       0.0050 |
```

2	800	00:00:46	0.1959	100.00%	0.70	0.0050
2	850	00:00:49	0.0095	100.00%	0.42	0.0050
2	900	00:00:52	0.1408	100.00%	0.58	0.0050
2	950	00:00:54	0.0808	100.00%	0.47	0.0050
2	1000	00:00:57	0.0103	100.00%	0.52	0.0050
2	1050	00:01:00	0.0460	100.00%	0.56	0.0050
2	1100	00:01:03	0.1228	100.00%	0.73	0.0050
2	1150	00:01:06	0.2649	100.00%	1.06	0.0050
2	1200	00:01:08	0.0144	100.00%	0.41	0.0050
2	1250	00:01:11	0.1458	100.00%	0.54	0.0050
2	1300	00:01:14	0.1317	100.00%	0.59	0.0050
2	1350	00:01:17	0.2456	100.00%	0.65	0.0050
2	1400	00:01:19	0.2157	100.00%	0.92	0.0050
2	1450	00:01:22	0.0945	100.00%	0.60	0.0050
2	1472	00:01:23	0.0854	100.00%	0.45	0.0050

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.

| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |

1	1	00:00:00	0.3201	92.19%	0.48	0.0050
1	50	00:00:09	0.2732	92.97%	0.30	0.0050
1	100	00:00:18	0.1906	96.09%	0.36	0.0050
1	150	00:00:27	0.2854	87.50%	0.32	0.0050
1	200	00:00:36	0.2149	96.88%	0.37	0.0050
1	250	00:00:46	0.1117	95.83%	0.30	0.0050
1	300	00:00:55	0.1404	96.09%	0.24	0.0050
1	350	00:01:05	0.1555	95.73%	0.36	0.0050
1	400	00:01:14	0.1542	96.88%	0.31	0.0050
1	450	00:01:23	0.2406	96.09%	0.23	0.0050
1	500	00:01:33	0.2164	92.97%	0.26	0.0050
1	550	00:01:42	0.1667	96.09%	0.24	0.0050
1	600	00:01:52	0.1604	94.53%	0.26	0.0050
1	650	00:02:01	0.2190	94.53%	0.30	0.0050
1	700	00:02:11	0.2615	94.29%	0.53	0.0050

2	750	00:02:23	0.1665	96.88%	0.37	0.0050
2	800	00:02:32	0.1847	94.53%	0.28	0.0050
2	850	00:02:41	0.2056	96.88%	0.24	0.0050
2	900	00:02:51	0.1138	96.25%	0.24	0.0050
2	950	00:03:00	0.0190	100.00%	0.29	0.0050
2	1000	00:03:09	0.1074	95.70%	0.29	0.0050
2	1050	00:03:19	0.0920	98.44%	0.27	0.0050
2	1100	00:03:28	0.1547	96.09%	0.24	0.0050
2	1150	00:03:37	0.1460	95.31%	0.20	0.0050
2	1200	00:03:47	0.1846	96.88%	0.27	0.0050
2	1250	00:03:56	0.1339	95.31%	0.23	0.0050
2	1300	00:04:05	0.1927	93.75%	0.23	0.0050
2	1350	00:04:15	0.0730	99.22%	0.20	0.0050
2	1400	00:04:24	0.0994	94.55%	0.24	0.0050
2	1450	00:04:34	0.1426	95.31%	0.16	0.0050
3	1500	00:04:46	0.1497	94.53%	0.19	0.0050
3	1550	00:04:55	0.1485	93.75%	0.27	0.0050
3	1600	00:05:05	0.1995	95.31%	0.27	0.0050
3	1650	00:05:14	0.0553	100.00%	0.19	0.0050
3	1700	00:05:23	0.1359	96.09%	0.23	0.0050
3	1750	00:05:33	0.0795	97.66%	0.19	0.0050
3	1800	00:05:42	0.0811	97.66%	0.18	0.0050
3	1850	00:05:51	0.1497	95.31%	0.34	0.0050
3	1900	00:06:00	0.3133	92.19%	0.34	0.0050
3	1950	00:06:10	0.1670	94.53%	0.22	0.0050
3	2000	00:06:19	0.1671	96.09%	0.20	0.0050
3	2050	00:06:29	0.0817	98.44%	0.21	0.0050
3	2100	00:06:38	0.0613	98.86%	0.38	0.0050
3	2150	00:06:48	0.0705	96.88%	0.16	0.0050
3	2200	00:06:57	0.1243	96.88%	0.22	0.0050
3	2205	00:06:58	0.1876	94.53%	0.21	0.0050

Detector training complete (with warnings):
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The following rows in trainingData have invalid
bounding box data:

Invalid Rows

_____

20
49

60
65
82
103
131
145
166
178
179
189
218
232
270
271
275
294
298
322
361
367
388
389
438
514
540
601
617
619
637
647
654
685
697
731

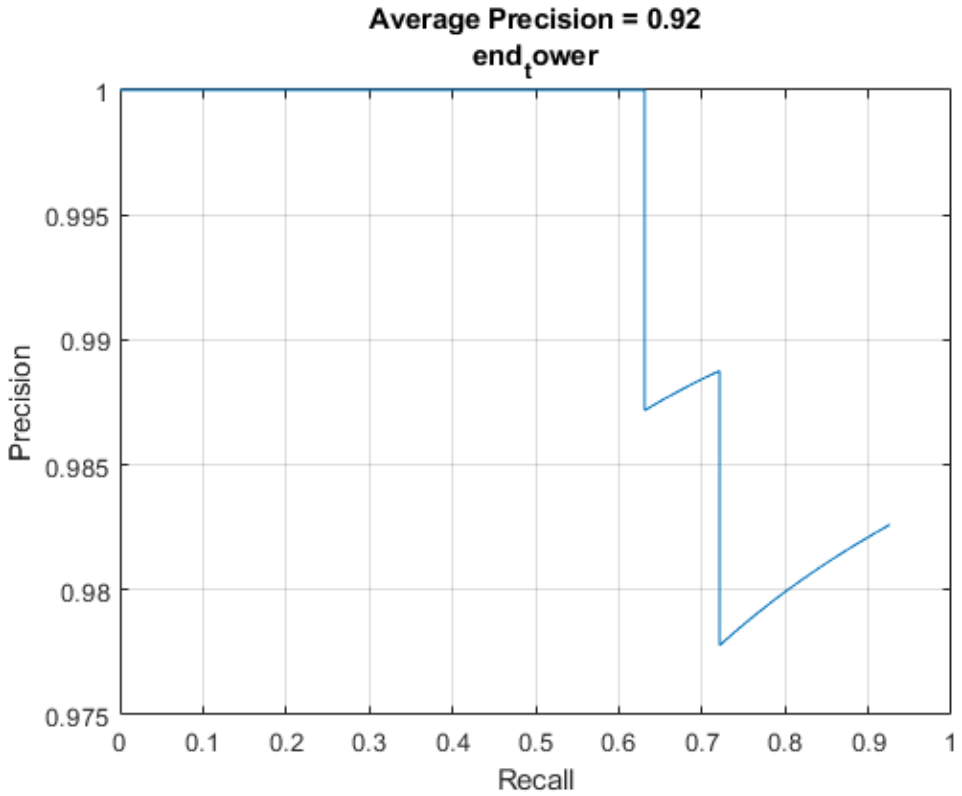
Bounding boxes must be fully contained within their associated image and must have positive width and height.

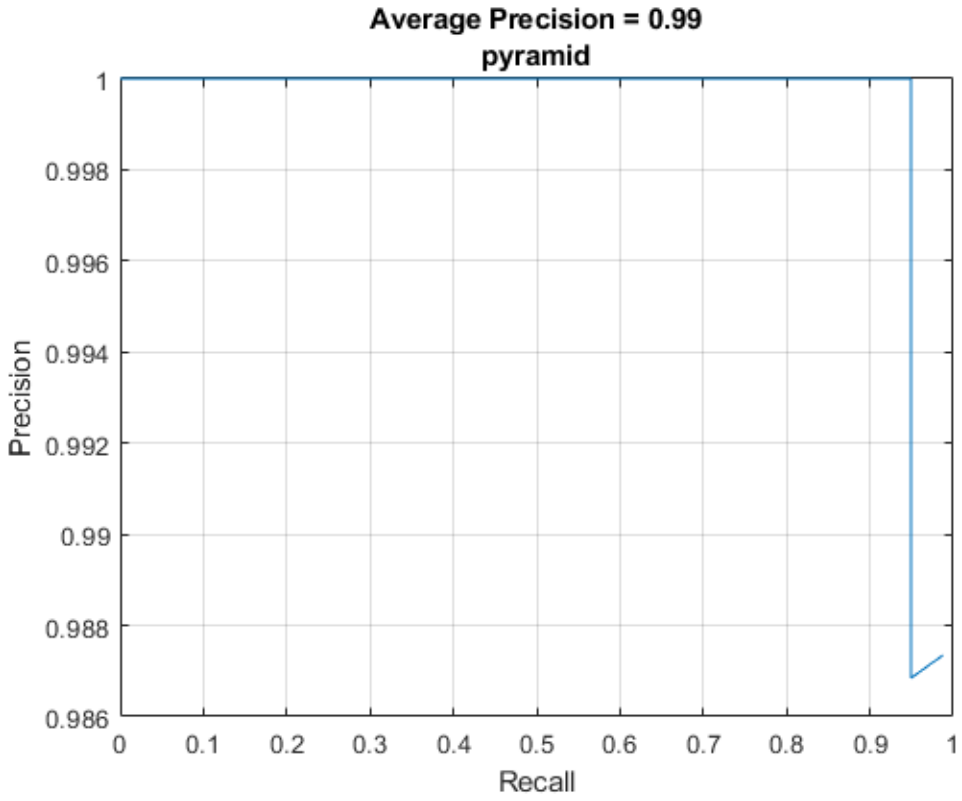
****

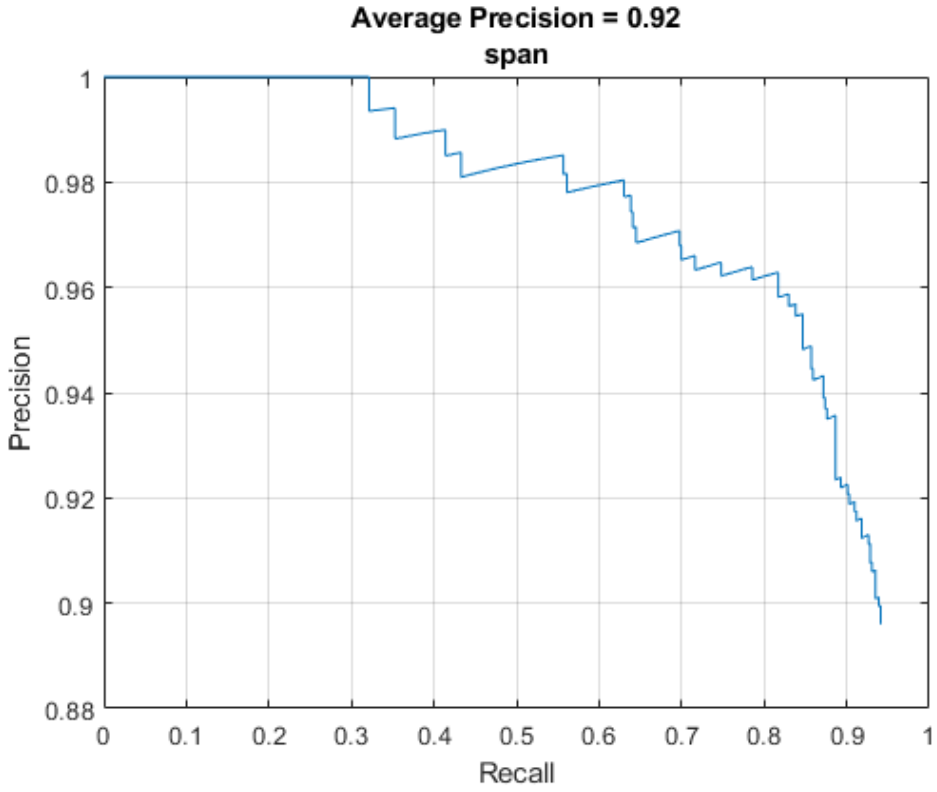
trainedDetector =

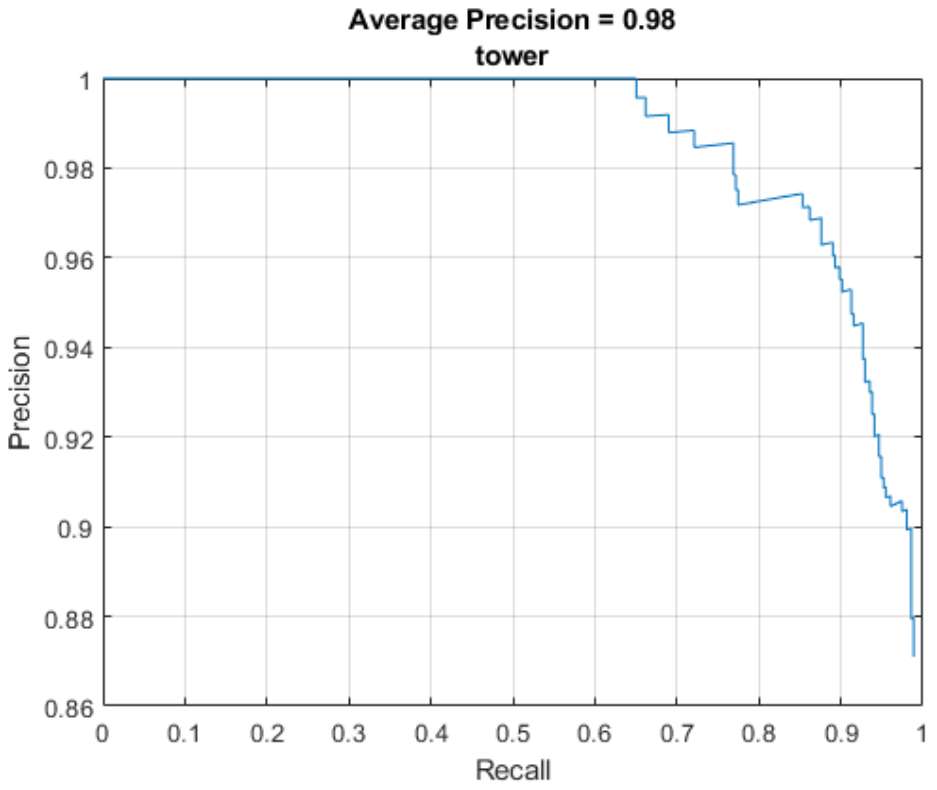
```
  fasterRCNNObjectDetector with properties:

      ModelName: 'end_tower'
        Network: [1×1 DAGNetwork]
    AnchorBoxes: [6×2 double]
     ClassNames: {'end_tower'  'pyramid'  'span'  'tower'  'Background'}
  MinObjectSize: [16 16]

Elapsed time is 1177.149515 seconds.
```

APPENDIX F.10.1.1.1 Result

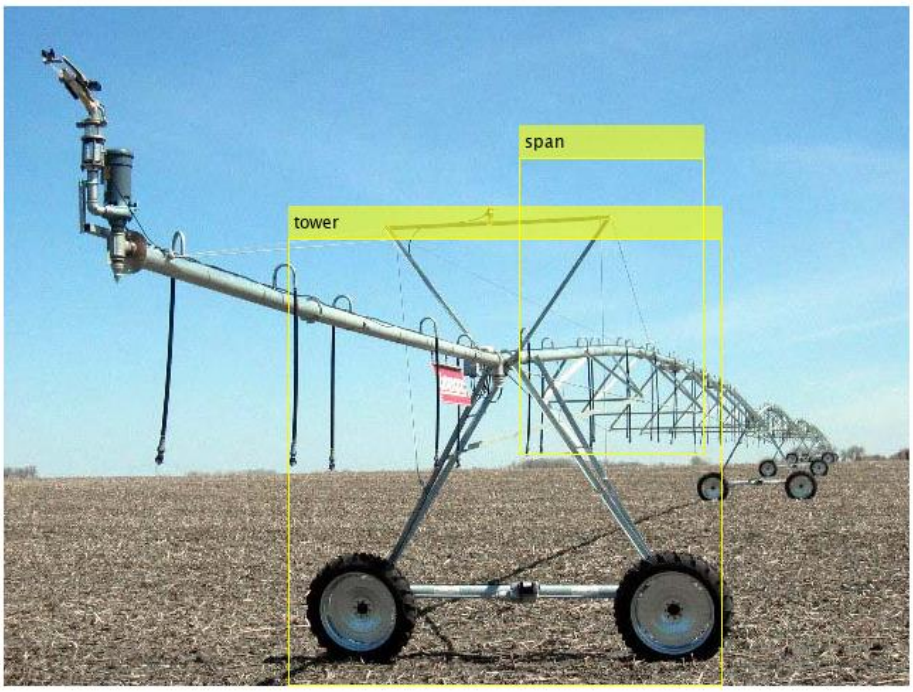





Average Precision = 0.92
span
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

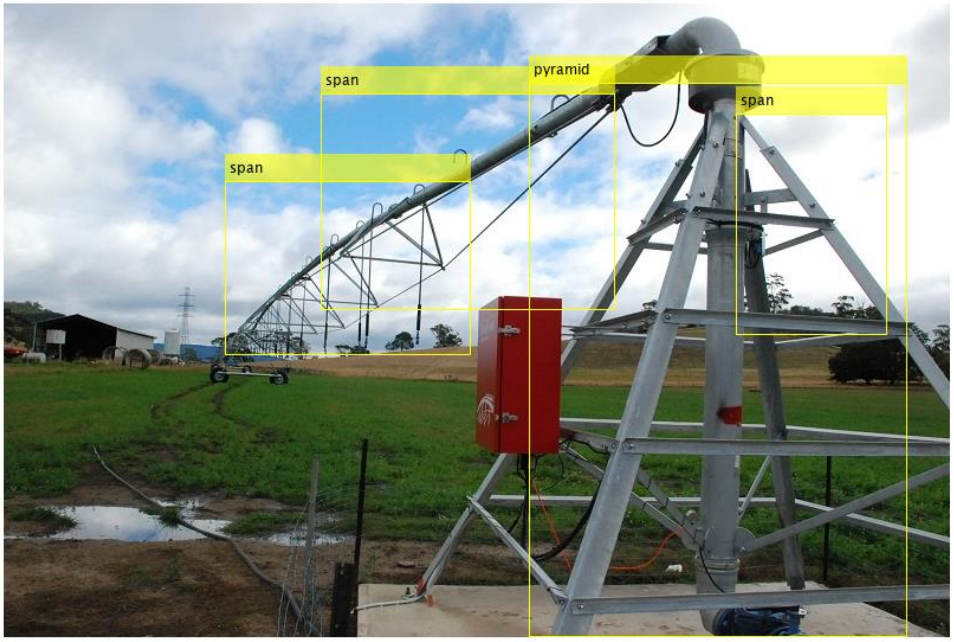

Average Precision = 0.98
tower
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

APPENDIX F.10.1.1.2 Results on test pictures




tower
tower


span



APPENDIX F.10.1.2 Resnet101

```
>> pivotDetectorResNet101v2

optionsStage1 =

  TrainingOptionsSGDM with properties:
```

```
                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                  Plots: 'none'
           SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
```

```
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
                MaxEpochs: 2
            MiniBatchSize: 1
                  Verbose: 1
         VerboseFrequency: 50
          ValidationData: []
      ValidationFrequency: 50
       ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
    ExecutionEnvironment: 'auto'
              WorkerLoad: []
              OutputFcn: []
                  Plots: 'none'
         SequenceLength: 'longest'
    SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
        InitialLearnRate: 0.0050
   LearnRateScheduleSettings: [1×1 struct]
        L2Regularization: 1.0000e-04
    GradientThresholdMethod: 'l2norm'
        GradientThreshold: Inf
```

```
            MaxEpochs: 3
        MiniBatchSize: 1
              Verbose: 1
     VerboseFrequency: 50
       ValidationData: []
  ValidationFrequency: 50
   ValidationPatience: Inf
              Shuffle: 'once'
       CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
 ExecutionEnvironment: 'auto'
           WorkerLoad: []
            OutputFcn: []
                Plots: 'none'
       SequenceLength: 'longest'
 SequencePaddingValue: 0
```

****
*
Training a Faster R-CNN Object Detector for the following object classes:

* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The following rows in trainingData have invalid
bounding box data:

Invalid Rows
_____

20
49
60
65
82
103
131
145
166
178
179

189
218
232
270
271
275
294
298
322
361
367
388
389
438
514
540
601
617
619
637
647
654
685
697
731

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.

Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Mini-batch	Base Learning
		(hh:mm:ss)	Loss	Accuracy	RMSE	Rate
1	1	00:00:00	1.2775	19.53%	0.86	0.0050
1	50	00:00:20	0.2552	100.00%	0.91	0.0050
1	100	00:00:39	0.1866	100.00%	0.64	0.0050
1	150	00:00:59	0.0626	100.00%	0.66	0.0050
1	200	00:01:19	0.0674	100.00%	1.02	0.0050
1	250	00:01:39	0.0211	100.00%	0.45	0.0050
1	300	00:01:58	0.2734	100.00%	0.90	0.0050
1	350	00:02:18	0.0318	100.00%	1.74	0.0050

1	400	00:02:38	0.0257	100.00%	0.61	0.0050
1	450	00:02:58	0.0997	100.00%	0.74	0.0050
1	500	00:03:17	0.0700	100.00%	0.90	0.0050
1	550	00:03:37	0.1365	100.00%	0.59	0.0050
1	600	00:03:57	0.0115	100.00%	0.33	0.0050
1	650	00:04:17	0.1751	100.00%	1.13	0.0050
1	700	00:04:36	0.0624	100.00%	0.46	0.0050
2	750	00:05:11	0.0868	100.00%	0.44	0.0050
2	800	00:05:31	0.0143	100.00%	0.47	0.0050
2	850	00:05:51	0.0745	100.00%	0.38	0.0050
2	900	00:06:10	0.1022	100.00%	0.51	0.0050
2	950	00:06:30	0.1577	100.00%	0.60	0.0050
2	1000	00:06:50	0.0795	100.00%	0.53	0.0050
2	1050	00:07:10	0.2771	100.00%	1.14	0.0050
2	1100	00:07:29	0.1340	100.00%	0.94	0.0050
2	1150	00:07:49	0.0729	100.00%	0.40	0.0050
2	1200	00:08:09	0.0753	100.00%	0.39	0.0050
2	1250	00:08:29	0.0658	100.00%	0.50	0.0050
2	1300	00:08:48	0.0416	100.00%	0.90	0.0050
2	1350	00:09:08	0.1429	100.00%	0.53	0.0050
2	1400	00:09:28	0.0033	100.00%	0.18	0.0050
2	1450	00:09:48	0.0075	100.00%	0.34	0.0050
2	1472	00:09:57	0.0538	100.00%	0.37	0.0050
===
===|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.
|===
===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===
===|
1	1	00:00:01	1.7373	15.38%	0.52	0.0050
1	50	00:00:41	NaN	0.00%	NaN	0.0050
1	100	00:01:22	NaN	0.00%	NaN	0.0050
1	150	00:02:03	NaN	1.56%	NaN	0.0050
1	200	00:02:44	NaN	0.00%	NaN	0.0050
1	250	00:03:25	NaN	0.00%	NaN	0.0050
1	300	00:04:07	NaN	4.65%	NaN	0.0050

```
|     1 |         350 |     00:04:48 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         400 |     00:05:29 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         450 |     00:06:11 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         500 |     00:06:52 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         550 |     00:07:33 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         600 |     00:08:14 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         650 |     00:08:56 |         NaN |       0.00% |         NaN |       0.0050 |
|     1 |         700 |     00:09:37 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |         750 |     00:10:34 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |         800 |     00:11:15 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |         850 |     00:11:56 |         NaN |       1.56% |         NaN |       0.0050 |
|     2 |         900 |     00:12:37 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |         950 |     00:13:19 |         NaN |       1.56% |         NaN |       0.0050 |
|     2 |        1000 |     00:14:00 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1050 |     00:14:42 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1100 |     00:15:23 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1150 |     00:16:04 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1200 |     00:16:45 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1250 |     00:17:27 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1300 |     00:18:08 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1350 |     00:18:49 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1400 |     00:19:31 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1450 |     00:20:12 |         NaN |       0.00% |         NaN |       0.0050 |
|     2 |        1470 |     00:20:28 |         NaN |       6.31% |         NaN |       0.0050 |
|=========================================================================
=======================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.7405 |     100.00% |       0.71 |       0.0050 |
|     1 |        50 |     00:00:11 |     0.3752 |     100.00% |       0.69 |       0.0050 |
|     1 |       100 |     00:00:21 |     0.2460 |     100.00% |       0.69 |       0.0050 |
|     1 |       150 |     00:00:33 |     0.5888 |     100.00% |       1.47 |       0.0050 |
|     1 |       200 |     00:00:44 |     0.1934 |     100.00% |       0.80 |       0.0050 |
|     1 |       250 |     00:00:55 |     0.1820 |     100.00% |       0.70 |       0.0050 |
|     1 |       300 |     00:01:07 |     0.0672 |     100.00% |       0.88 |       0.0050 |
|     1 |       350 |     00:01:18 |     0.2913 |     100.00% |       0.80 |       0.0050 |
```

1	400	00:01:29	0.1006	100.00%	0.93	0.0050
1	450	00:01:40	0.0643	100.00%	0.83	0.0050
1	500	00:01:51	0.1859	100.00%	0.82	0.0050
1	550	00:02:02	0.0540	100.00%	0.67	0.0050
1	600	00:02:13	0.0811	100.00%	0.95	0.0050
1	650	00:02:24	0.0446	100.00%	0.93	0.0050
1	700	00:02:35	0.2030	100.00%	0.71	0.0050
2	750	00:02:59	0.2437	100.00%	1.11	0.0050
2	800	00:03:10	0.1709	100.00%	0.73	0.0050
2	850	00:03:21	1.0493	100.00%	2.20	0.0050
2	900	00:03:32	0.4017	100.00%	0.73	0.0050
2	950	00:03:43	0.1087	100.00%	0.59	0.0050
2	1000	00:03:54	0.1086	100.00%	0.81	0.0050
2	1050	00:04:05	0.0155	100.00%	0.94	0.0050
2	1100	00:04:16	0.5464	100.00%	1.66	0.0050
2	1150	00:04:27	0.0582	100.00%	0.85	0.0050
2	1200	00:04:38	0.2967	100.00%	0.66	0.0050
2	1250	00:04:49	0.1426	100.00%	0.85	0.0050
2	1300	00:05:00	0.0708	100.00%	0.88	0.0050
2	1350	00:05:11	0.0531	100.00%	0.92	0.0050
2	1400	00:05:22	0.1185	100.00%	1.28	0.0050
2	1450	00:05:33	0.0608	100.00%	0.67	0.0050
2	1472	00:05:37	0.0950	100.00%	0.91	0.0050
===
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.
|===
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===
=======================================|
1	1	00:00:00	NaN	1.56%	NaN	0.0050
1	50	00:00:25	NaN	0.00%	NaN	0.0050
1	100	00:00:50	NaN	1.56%	NaN	0.0050
1	150	00:01:15	NaN	3.13%	NaN	0.0050
1	200	00:01:40	NaN	0.00%	NaN	0.0050
1	250	00:02:06	NaN	1.61%	NaN	0.0050
1	300	00:02:32	NaN	1.56%	NaN	0.0050

1	350	00:02:57	NaN	0.00%	NaN	0.0050
1	400	00:03:22	NaN	5.47%	NaN	0.0050
1	450	00:03:48	NaN	5.69%	NaN	0.0050
1	500	00:04:13	NaN	0.00%	NaN	0.0050
1	550	00:04:38	NaN	0.00%	NaN	0.0050
1	600	00:05:03	NaN	0.00%	NaN	0.0050
1	650	00:05:29	NaN	1.56%	NaN	0.0050
2	700	00:06:09	NaN	0.00%	NaN	0.0050
2	750	00:06:33	NaN	0.00%	NaN	0.0050
2	800	00:06:59	NaN	0.00%	NaN	0.0050
2	850	00:07:24	NaN	1.57%	NaN	0.0050
2	900	00:07:50	NaN	0.00%	NaN	0.0050
2	950	00:08:16	NaN	0.00%	NaN	0.0050
2	1000	00:08:41	NaN	0.00%	NaN	0.0050
2	1050	00:09:06	NaN	4.48%	NaN	0.0050
2	1100	00:09:32	NaN	0.00%	NaN	0.0050
2	1150	00:09:57	NaN	0.00%	NaN	0.0050
2	1200	00:10:22	NaN	0.00%	NaN	0.0050
2	1250	00:10:48	NaN	0.00%	NaN	0.0050
2	1300	00:11:13	NaN	0.00%	NaN	0.0050
3	1350	00:11:53	NaN	0.00%	NaN	0.0050
3	1400	00:12:18	NaN	0.00%	NaN	0.0050
3	1450	00:12:44	NaN	1.56%	NaN	0.0050
3	1500	00:13:09	NaN	0.00%	NaN	0.0050
3	1550	00:13:35	NaN	0.00%	NaN	0.0050
3	1600	00:14:01	NaN	0.00%	NaN	0.0050
3	1650	00:14:26	NaN	0.00%	NaN	0.0050
3	1700	00:14:51	NaN	0.00%	NaN	0.0050
3	1750	00:15:16	NaN	0.00%	NaN	0.0050
3	1800	00:15:42	NaN	0.00%	NaN	0.0050
3	1850	00:16:07	NaN	0.00%	NaN	0.0050
3	1900	00:16:32	NaN	0.78%	NaN	0.0050
3	1950	00:16:58	NaN	10.98%	NaN	0.0050
3	1986	00:17:16	NaN	0.00%	NaN	0.0050

|==|

Detector training complete (with warnings):
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The following rows in trainingData have invalid
bounding box data:

Invalid Rows

_____

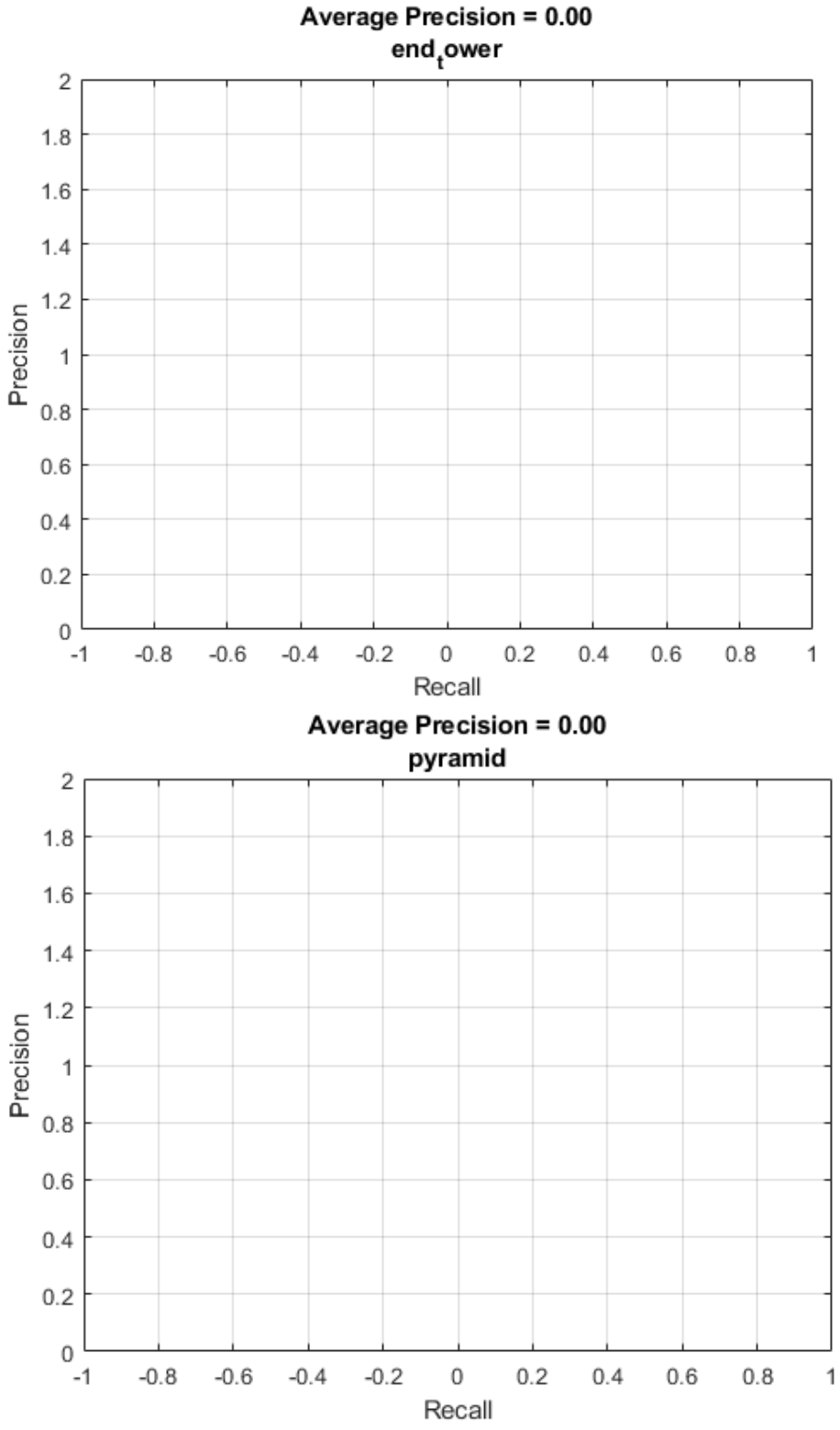
20
49
60
65
82
103
131
145
166
178
179
189
218
232
270
271
275
294
298
322
361
367
388
389
438
514
540
601
617
619
637
647
654
685
697
731

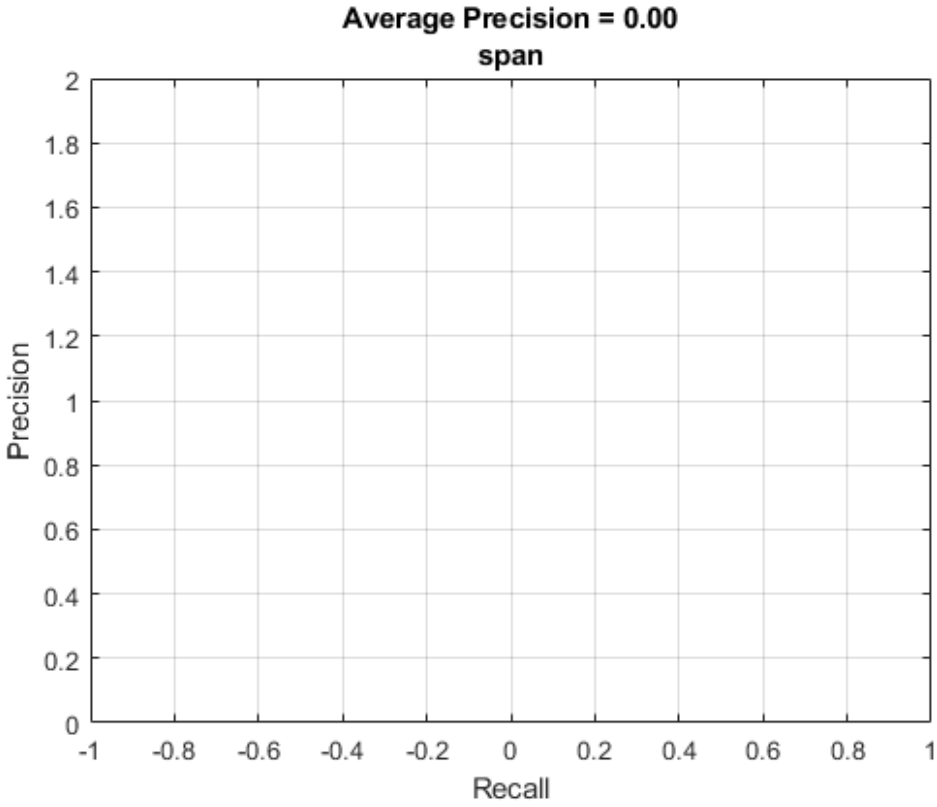
Bounding boxes must be fully contained within their associated image and must have positive width and height.

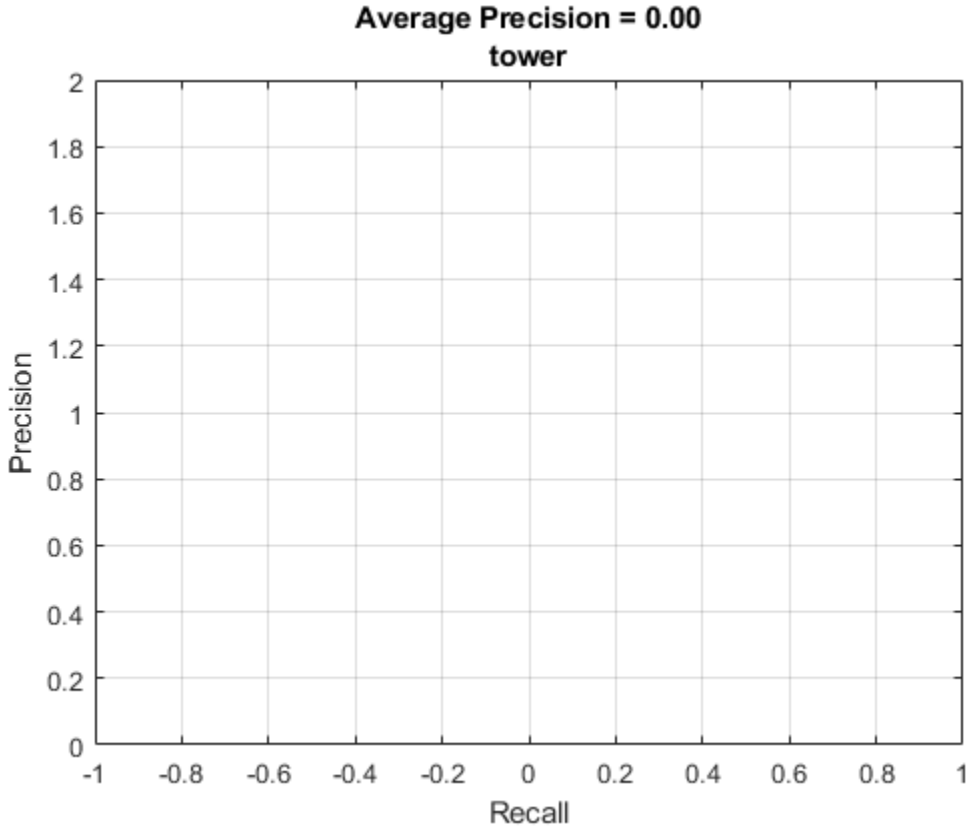
****

```
trainedDetector =

  fasterRCNNObjectDetector with properties:

       ModelName: 'end_tower'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'end_tower'  'pyramid'  'span'  'tower'  'Background'}
    MinObjectSize: [16 16]

Elapsed time is 3622.019200 seconds.
```

APPENDIX F.10.1.2.1 Result




Average Precision = 0.00
span
Precision
Recall
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


Average Precision = 0.00
tower
Precision
Recall
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

APPENDIX F.10.2 Run 2

APPENDIX F.10.2.1 GoogLeNet

```
>> pivotDetectorGoogLeNetv3

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                    MaxEpochs: 2
                MiniBatchSize: 1
                      Verbose: 1
             VerboseFrequency: 50
              ValidationData: []
          ValidationFrequency: 50
           ValidationPatience: Inf
                      Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
            GradientThreshold: Inf
                    MaxEpochs: 2
```

```
            MiniBatchSize: 1
                  Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                   Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
            MiniBatchSize: 1
                  Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                  Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
              WorkerLoad: []
               OutputFcn: []
                   Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =
```

```
  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                MaxEpochs: 3
             MiniBatchSize: 1
                   Verbose: 1
           VerboseFrequency: 50
            ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                   Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0

*************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The
following rows in trainingData have invalid
bounding box data:

Invalid Rows
____________

20
49
```

60
65
82
103
131
145
166
178
179
189
218
232
270
271
275
294
298
322
361
367
388
389
438
514
540
601
617
619
637
647
654
685
697
731

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.
|===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |

```
|=========================================================================
=======================================|
|     1 |       1 |   00:00:00 |     1.1823 |    23.44% |       2.38 |       0.0050 |
|     1 |      50 |   00:00:04 |     0.3574 |   100.00% |       0.92 |       0.0050 |
|     1 |     100 |   00:00:09 |     0.1485 |   100.00% |       0.99 |       0.0050 |
|     1 |     150 |   00:00:14 |     0.2256 |   100.00% |       0.68 |       0.0050 |
|     1 |     200 |   00:00:19 |     0.4887 |   100.00% |       1.87 |       0.0050 |
|     1 |     250 |   00:00:24 |     0.1242 |   100.00% |       0.63 |       0.0050 |
|     1 |     300 |   00:00:29 |     0.1256 |   100.00% |       0.63 |       0.0050 |
|     1 |     350 |   00:00:34 |     0.2036 |   100.00% |       0.89 |       0.0050 |
|     1 |     400 |   00:00:39 |     0.0728 |   100.00% |       0.89 |       0.0050 |
|     1 |     450 |   00:00:44 |     0.1027 |   100.00% |       0.62 |       0.0050 |
|     1 |     500 |   00:00:48 |     0.1459 |   100.00% |       0.61 |       0.0050 |
|     1 |     550 |   00:00:53 |     0.0594 |   100.00% |       0.42 |       0.0050 |
|     1 |     600 |   00:00:58 |     0.0295 |   100.00% |       0.39 |       0.0050 |
|     1 |     650 |   00:01:03 |     0.1691 |   100.00% |       0.69 |       0.0050 |
|     1 |     700 |   00:01:08 |     0.1186 |   100.00% |       0.54 |       0.0050 |
|     2 |     750 |   00:01:15 |     0.0840 |   100.00% |       0.61 |       0.0050 |
|     2 |     800 |   00:01:20 |     0.0100 |   100.00% |       0.24 |       0.0050 |
|     2 |     850 |   00:01:25 |     0.1041 |   100.00% |       0.75 |       0.0050 |
|     2 |     900 |   00:01:30 |     0.0658 |   100.00% |       0.59 |       0.0050 |
|     2 |     950 |   00:01:35 |     0.2962 |   100.00% |       1.13 |       0.0050 |
|     2 |    1000 |   00:01:40 |     0.1418 |   100.00% |       0.56 |       0.0050 |
|     2 |    1050 |   00:01:45 |     0.0117 |   100.00% |       0.49 |       0.0050 |
|     2 |    1100 |   00:01:50 |     0.0128 |   100.00% |       0.28 |       0.0050 |
|     2 |    1150 |   00:01:55 |     0.0977 |   100.00% |       0.84 |       0.0050 |
|     2 |    1200 |   00:02:00 |     0.0242 |   100.00% |       0.88 |       0.0050 |
|     2 |    1250 |   00:02:04 |     0.0064 |   100.00% |       0.29 |       0.0050 |
|     2 |    1300 |   00:02:09 |     0.0501 |   100.00% |       0.40 |       0.0050 |
|     2 |    1350 |   00:02:14 |     0.0741 |   100.00% |       0.38 |       0.0050 |
|     2 |    1400 |   00:02:19 |     0.0353 |   100.00% |       0.28 |       0.0050 |
|     2 |    1450 |   00:02:24 |     0.0500 |   100.00% |       0.65 |       0.0050 |
|     2 |    1472 |   00:02:26 |     0.0068 |   100.00% |       0.21 |       0.0050 |
|=========================================================================
=======================================|
```

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.

```
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
```

| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===
===|
1	1	00:00:00	1.7669	14.84%	0.54	0.0050
1	50	00:00:12	0.1832	97.66%	0.60	0.0050
1	100	00:00:24	0.7543	83.59%	0.46	0.0050
1	150	00:00:36	0.2363	95.69%	0.48	0.0050
1	200	00:00:48	0.2684	94.53%	0.40	0.0050
1	250	00:01:01	0.3447	92.11%	0.54	0.0050
1	300	00:01:13	0.2733	90.63%	0.27	0.0050
1	350	00:01:25	0.3854	91.41%	0.40	0.0050
1	400	00:01:37	0.2960	92.19%	0.39	0.0050
1	450	00:01:50	0.2223	93.75%	0.39	0.0050
1	500	00:02:02	0.1925	95.31%	0.35	0.0050
1	550	00:02:15	0.1724	96.09%	0.30	0.0050
1	600	00:02:27	0.1915	93.75%	0.26	0.0050
1	650	00:02:39	0.1623	94.53%	0.36	0.0050
1	700	00:02:52	0.1298	96.43%	0.34	0.0050
2	750	00:03:08	0.1871	97.26%	0.49	0.0050
2	800	00:03:20	0.3179	93.64%	0.59	0.0050
2	850	00:03:32	0.1845	93.69%	0.37	0.0050
2	900	00:03:45	0.1539	95.31%	0.28	0.0050
2	950	00:03:57	0.1061	96.88%	0.26	0.0050
2	1000	00:04:10	0.2084	96.88%	0.28	0.0050
2	1050	00:04:22	0.1239	97.66%	0.23	0.0050
2	1100	00:04:34	0.1504	96.88%	0.27	0.0050
2	1150	00:04:47	0.1735	96.09%	0.34	0.0050
2	1200	00:04:59	0.0720	98.44%	0.23	0.0050
2	1250	00:05:12	0.1345	97.56%	0.38	0.0050
2	1300	00:05:25	0.0682	99.21%	0.24	0.0050
load('E:\MERGED DETECTOR\shuffleMergedLabels.mat')
2	1350	00:05:37	0.3480	88.28%	0.31	0.0050
2	1400	00:05:50	0.2593	95.56%	0.59	0.0050
2	1450	00:06:03	0.1586	96.88%	0.30	0.0050
2	1466	00:06:07	0.1684	92.19%	0.21	0.0050
===
===|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|===
===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |

```
|      |         | (hh:mm:ss)  |    Loss    | Accuracy  |    RMSE    |     Rate     |
|=========================================================================
=======================================|
|     1 |       1 |    00:00:00 |    0.3273 |   100.00% |       0.95 |       0.0050 |
|     1 |      50 |    00:00:02 |    0.3519 |   100.00% |       1.07 |       0.0050 |
|     1 |     100 |    00:00:05 |    0.1038 |   100.00% |       0.50 |       0.0050 |
|     1 |     150 |    00:00:08 |    0.2377 |   100.00% |       0.84 |       0.0050 |
|     1 |     200 |    00:00:11 |    0.1271 |   100.00% |       0.57 |       0.0050 |
|     1 |     250 |    00:00:14 |    0.0782 |   100.00% |       0.48 |       0.0050 |
|     1 |     300 |    00:00:16 |    0.0791 |   100.00% |       0.51 |       0.0050 |
|     1 |     350 |    00:00:19 |    0.3088 |   100.00% |       0.85 |       0.0050 |
|     1 |     400 |    00:00:22 |    0.0160 |   100.00% |       1.00 |       0.0050 |
|     1 |     450 |    00:00:25 |    0.0605 |   100.00% |       1.40 |       0.0050 |
|     1 |     500 |    00:00:28 |    0.1742 |   100.00% |       1.23 |       0.0050 |
|     1 |     550 |    00:00:30 |    0.2190 |   100.00% |       0.61 |       0.0050 |
|     1 |     600 |    00:00:33 |    0.1348 |   100.00% |       0.80 |       0.0050 |
|     1 |     650 |    00:00:36 |    0.1170 |   100.00% |       0.49 |       0.0050 |
|     1 |     700 |    00:00:39 |    0.1860 |   100.00% |       0.75 |       0.0050 |
|     2 |     750 |    00:00:44 |    0.4218 |   100.00% |       1.31 |       0.0050 |
|     2 |     800 |    00:00:47 |    0.0065 |   100.00% |       0.26 |       0.0050 |
|     2 |     850 |    00:00:50 |    0.0292 |   100.00% |       0.35 |       0.0050 |
|     2 |     900 |    00:00:52 |    0.0523 |   100.00% |       0.44 |       0.0050 |
|     2 |     950 |    00:00:55 |    0.0158 |   100.00% |       0.48 |       0.0050 |
|     2 |    1000 |    00:00:58 |    0.2794 |   100.00% |       0.69 |       0.0050 |
|     2 |    1050 |    00:01:01 |    0.0849 |   100.00% |       0.50 |       0.0050 |
|     2 |    1100 |    00:01:04 |    0.0805 |   100.00% |       0.50 |       0.0050 |
|     2 |    1150 |    00:01:06 |    0.0076 |   100.00% |       0.38 |       0.0050 |
|     2 |    1200 |    00:01:09 |    0.0091 |   100.00% |       0.36 |       0.0050 |
|     2 |    1250 |    00:01:12 |    0.0872 |   100.00% |       0.69 |       0.0050 |
|     2 |    1300 |    00:01:15 |    0.1124 |   100.00% |       0.53 |       0.0050 |
|     2 |    1350 |    00:01:18 |    0.0277 |   100.00% |       0.39 |       0.0050 |
|     2 |    1400 |    00:01:20 |    0.0788 |   100.00% |       0.51 |       0.0050 |
|     2 |    1450 |    00:01:23 |    0.0956 |   100.00% |       0.50 |       0.0050 |
|     2 |    1472 |    00:01:24 |    0.0073 |   100.00% |       0.45 |       0.0050 |
|=========================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 736 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
```

| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===|
1	1	00:00:00	0.4129	88.28%	0.30	0.0050
1	50	00:00:09	0.1602	96.09%	0.24	0.0050
1	100	00:00:18	0.2589	92.19%	0.33	0.0050
1	150	00:00:27	0.2009	96.77%	0.30	0.0050
1	200	00:00:37	0.0439	99.22%	0.30	0.0050
1	250	00:00:46	0.1498	97.66%	0.21	0.0050
1	300	00:00:55	0.3130	92.19%	0.27	0.0050
1	350	00:01:05	0.3609	93.75%	0.47	0.0050
1	400	00:01:14	0.0831	98.44%	0.37	0.0050
1	450	00:01:23	0.1504	94.53%	0.22	0.0050
1	500	00:01:33	0.0795	98.44%	0.25	0.0050
1	550	00:01:42	0.2080	92.97%	0.22	0.0050
1	600	00:01:51	0.0682	98.44%	0.26	0.0050
1	650	00:02:00	0.2193	92.97%	0.24	0.0050
1	700	00:02:10	0.1804	92.97%	0.31	0.0050
2	750	00:02:23	0.2360	94.53%	0.29	0.0050
2	800	00:02:32	0.1425	97.66%	0.50	0.0050
2	850	00:02:41	0.1540	96.09%	0.21	0.0050
2	900	00:02:51	0.0908	99.22%	0.25	0.0050
2	950	00:03:00	0.3021	90.63%	0.25	0.0050
2	1000	00:03:10	0.1644	96.88%	0.26	0.0050
2	1050	00:03:19	0.2113	92.97%	0.27	0.0050
2	1100	00:03:28	0.0690	98.44%	0.29	0.0050
2	1150	00:03:38	0.2397	95.31%	0.37	0.0050
2	1200	00:03:47	0.0985	98.44%	0.34	0.0050
2	1250	00:03:57	0.1410	96.09%	0.22	0.0050
2	1300	00:04:06	0.3255	89.84%	0.22	0.0050
2	1350	00:04:16	0.0885	97.66%	0.21	0.0050
2	1400	00:04:25	0.1506	94.53%	0.22	0.0050
2	1450	00:04:35	0.1794	95.31%	0.34	0.0050
3	1500	00:04:47	0.2395	94.53%	0.36	0.0050
3	1550	00:04:56	0.1395	96.88%	0.29	0.0050
3	1600	00:05:06	0.0794	97.66%	0.19	0.0050
3	1650	00:05:15	0.1622	96.09%	0.24	0.0050
3	1700	00:05:25	0.0919	98.44%	0.23	0.0050
3	1750	00:05:34	0.1105	96.55%	0.35	0.0050
3	1800	00:05:43	0.1625	96.09%	0.29	0.0050
3	1850	00:05:53	0.0967	98.44%	0.22	0.0050
3	1900	00:06:02	0.1704	96.09%	0.25	0.0050

3	1950	00:06:12	0.2008	95.31%	0.32	0.0050
3	2000	00:06:21	0.0934	98.44%	0.19	0.0050
3	2050	00:06:30	0.1384	97.66%	0.20	0.0050
3	2100	00:06:40	0.1627	96.88%	0.23	0.0050
3	2150	00:06:49	0.2234	92.19%	0.20	0.0050
3	2196	00:06:58	0.0872	99.22%	0.19	0.0050

Detector training complete (with warnings):
Warning: Invalid bounding boxes from 36 out of 738 training images were removed. The following rows in trainingData have invalid
bounding box data:

Invalid Rows

_____

20
49
60
65
82
103
131
145
166
178
179
189
218
232
270
271
275
294
298
322
361
367
388
389
438
514
540

601
617
619
637
647
654
685
697
731

Bounding boxes must be fully contained within their associated image and must have positive width and height.

****

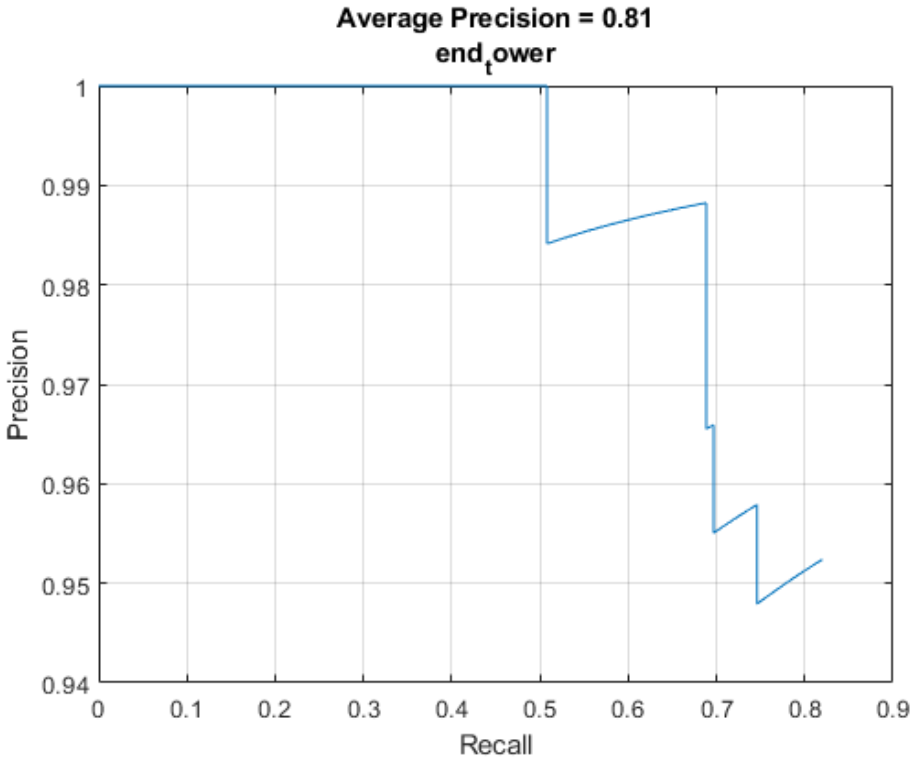
trainedDetector =

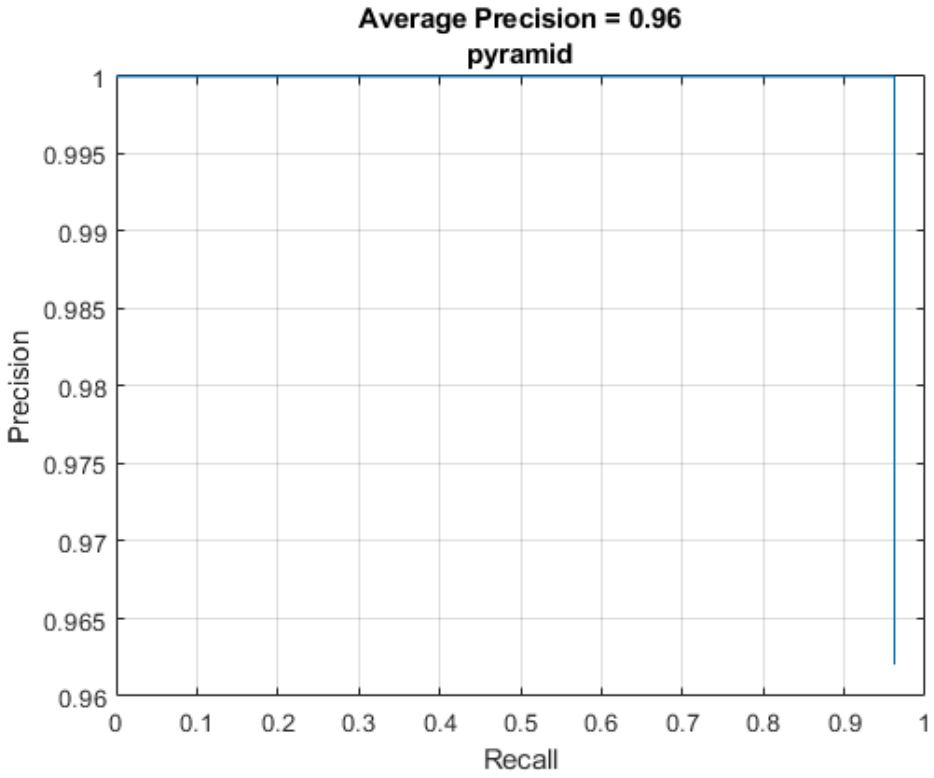
  fasterRCNNObjectDetector with properties:

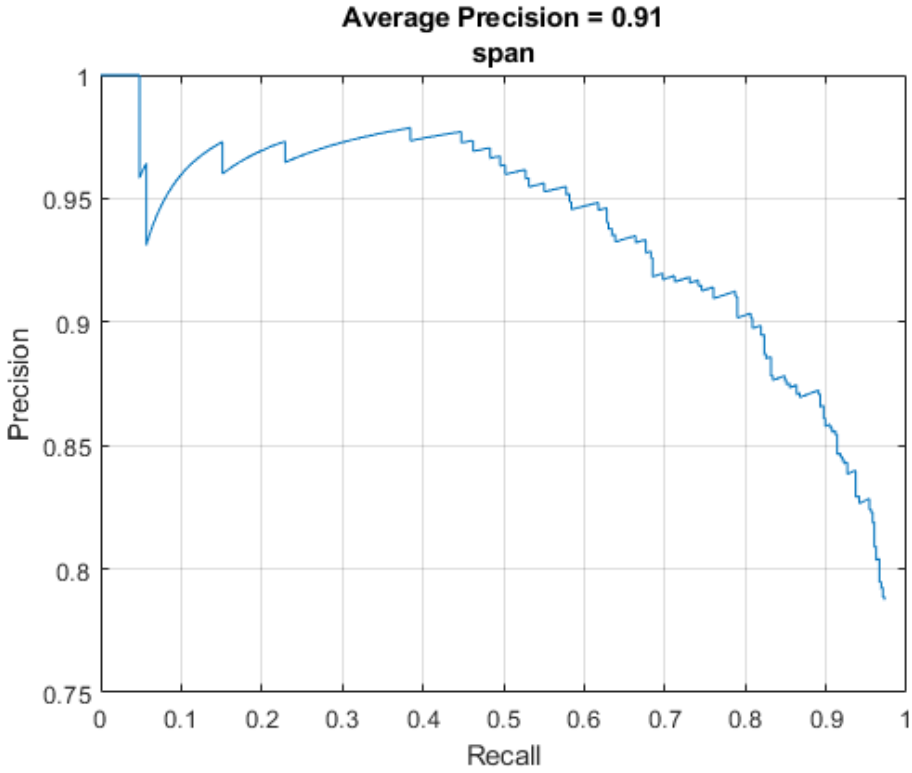
       ModelName: 'end_tower'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'end_tower' 'pyramid' 'span' 'tower' 'Background'}
    MinObjectSize: [16 16]

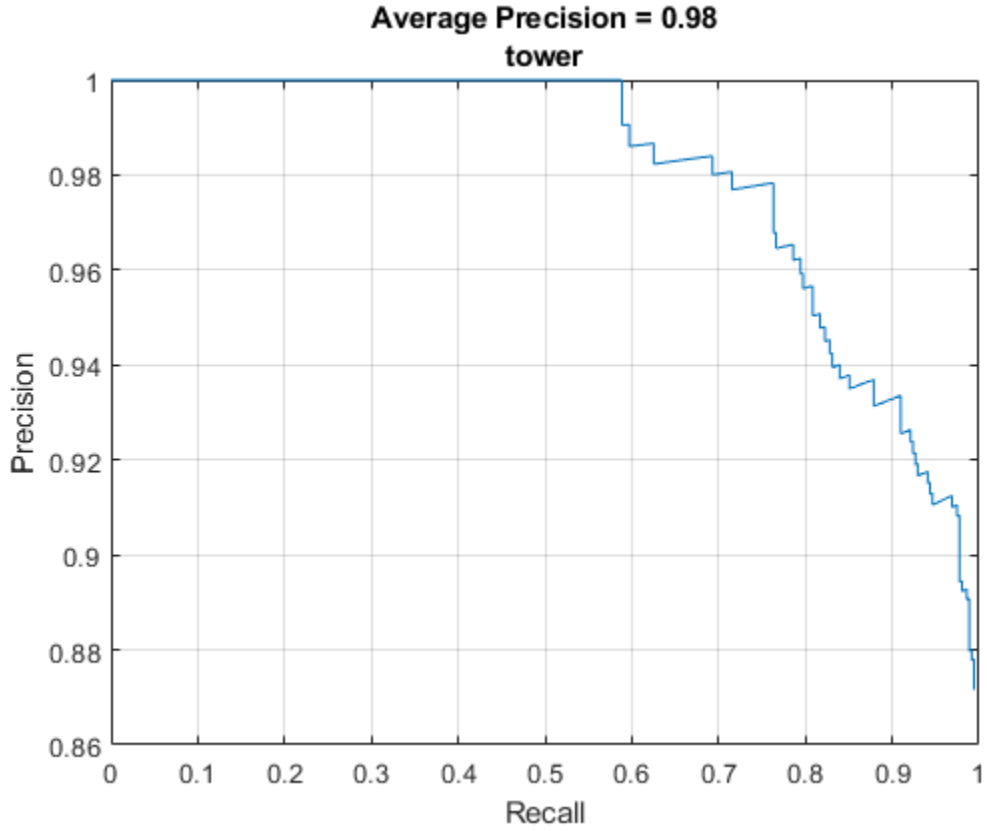
Elapsed time is 1169.963024 seconds.

APPENDIX F.10.2.1.1 Result

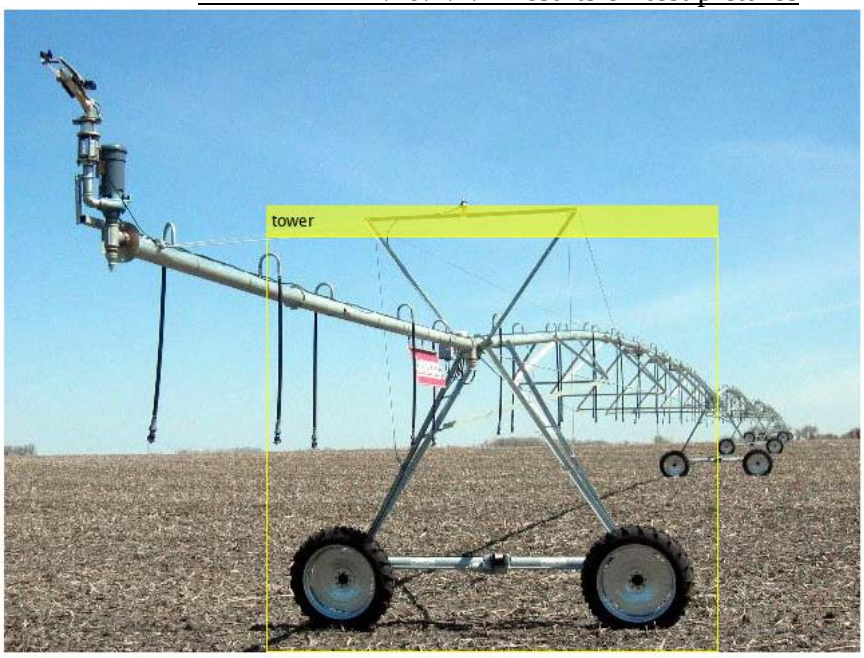


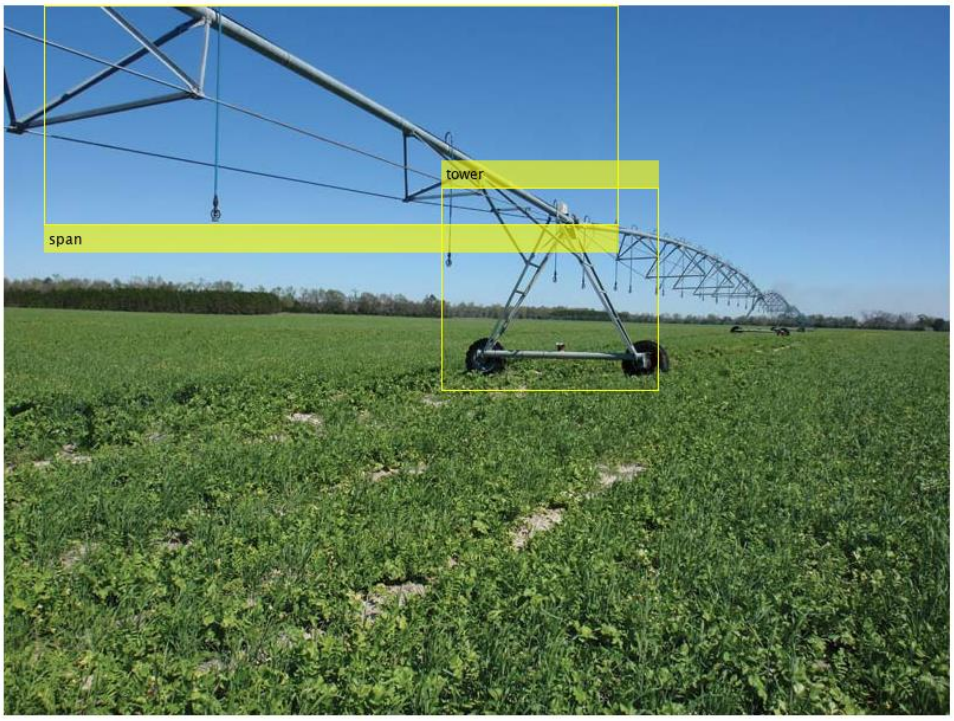



Average Precision = 0.91
span
Precision
Recall
1
0.95
0.9
0.85
0.8
0.75
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.98
tower
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

APPENDIX F.10.2.1.2 Results on test pictures






tower
span


tower


span
span


span

APPENDIX F.10.2.2 Resnet101

N/A

APPENDIX F.10.2.2.1 Result

N/A

APPENDIX F.11 Combined Detector

Image 1 – (beefmagazine.com, 2014)

Image 2 – (Keeping It Dutch, 2015)

Image 3 – (AuctionTime Blog, 2018)

Image 4 – (Vishal Agro Industries, 2019)

Image 5 – (New Holland, 2017)

Image 6 – (Wiring Schematic Design, 2019)

Image 7 – (Vodar Co., Ltd., 2019)

Image 8 – (Lindsay Europe, 2019)

Image 9 – (Denise O’Sullivan, 2019)

Image 10 – (Tracey Media, 2013)

APPENDIX F.11.1 GoogLeNet Run 1

```
>> combinedMergedDetectorv1

optionsStage1 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
            CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                   OutputFcn: []
                       Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                    Momentum: 0.9000
           InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
```

```
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                  Plots: 'none'
           SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                MaxEpochs: 2
             MiniBatchSize: 1
                Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                OutputFcn: []
                  Plots: 'none'
           SequenceLength: 'longest'
```

```
        SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
              MiniBatchSize: 1
                    Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
       SequencePaddingValue: 0

*************************************************************************
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors
* roundbale
* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 31 out of 1014 training images were removed.
The following rows in trainingData have invalid
bounding box data:
```

Invalid Rows
_____

17
75
194
199
202
249
253
261
290
309
369
395
399
498
617
652
678
679
703
717
731
752
767
806
808
811
816
864
897
920
937

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.
|===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |

	1	1	00:00:00	0.9553	41.41%	1.89	0.0050
	1	50	00:00:05	0.4520	100.00%	1.02	0.0050
	1	100	00:00:10	1.4376	100.00%	2.92	0.0050
	1	150	00:00:15	0.4824	100.00%	0.89	0.0050
	1	200	00:00:20	0.2288	100.00%	0.64	0.0050
	1	250	00:00:25	0.2079	100.00%	0.93	0.0050
	1	300	00:00:30	1.3049	100.00%	1.69	0.0050
	1	350	00:00:34	0.3130	100.00%	0.90	0.0050
	1	400	00:00:39	0.2351	100.00%	0.86	0.0050
	1	450	00:00:44	0.5138	100.00%	0.99	0.0050
	1	500	00:00:49	0.1985	100.00%	1.05	0.0050
	1	550	00:00:54	0.2089	100.00%	0.76	0.0050
	1	600	00:00:59	0.2479	100.00%	0.68	0.0050
	1	650	00:01:04	0.1962	100.00%	0.77	0.0050
	1	700	00:01:09	0.1659	100.00%	0.96	0.0050
	1	750	00:01:14	0.1880	100.00%	0.75	0.0050
	1	800	00:01:19	0.3588	100.00%	0.84	0.0050
	1	850	00:01:24	0.0255	100.00%	0.73	0.0050
	1	900	00:01:29	0.1324	100.00%	1.08	0.0050
	1	950	00:01:34	0.0371	100.00%	0.57	0.0050
	1	1000	00:01:39	0.0568	100.00%	0.78	0.0050
	2	1050	00:01:46	0.1751	100.00%	0.68	0.0050
	2	1100	00:01:51	0.2776	100.00%	0.88	0.0050
	2	1150	00:01:56	0.0521	100.00%	1.51	0.0050
	2	1200	00:02:01	0.1173	100.00%	0.66	0.0050
	2	1250	00:02:06	0.1220	100.00%	1.28	0.0050
	2	1300	00:02:11	0.4237	100.00%	0.91	0.0050
	2	1350	00:02:16	0.2722	100.00%	0.92	0.0050
	2	1400	00:02:21	0.2793	100.00%	0.73	0.0050
	2	1450	00:02:26	0.3051	100.00%	1.03	0.0050
	2	1500	00:02:31	0.0788	100.00%	0.69	0.0050
	2	1550	00:02:36	0.2064	100.00%	0.76	0.0050
	2	1600	00:02:41	0.2232	100.00%	0.81	0.0050
	2	1650	00:02:46	0.1025	100.00%	1.55	0.0050
	2	1700	00:02:51	0.2365	100.00%	0.66	0.0050
	2	1750	00:02:56	0.0383	100.00%	0.77	0.0050
	2	1800	00:03:01	0.0436	100.00%	0.64	0.0050
	2	1850	00:03:07	0.1933	100.00%	0.59	0.0050
	2	1900	00:03:11	0.0207	100.00%	0.24	0.0050
	2	1950	00:03:16	0.0202	100.00%	0.35	0.0050
	2	2000	00:03:21	0.0393	100.00%	0.34	0.0050
	2	2026	00:03:24	0.0039	100.00%	0.28	0.0050

```
|=========================================================================
=======================================|

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 1013 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |      Rate      |
|=========================================================================
=======================================|
|     1 |         1 |   00:00:00 |     2.2811 |    1.56% |       0.37 |         0.0050 |
|     1 |        50 |   00:00:12 |     0.3599 |   94.12% |       0.38 |         0.0050 |
|     1 |       100 |   00:00:24 |     0.3928 |   89.84% |       0.30 |         0.0050 |
|     1 |       150 |   00:00:37 |     0.6176 |   85.94% |       0.34 |         0.0050 |
|     1 |       200 |   00:00:49 |     0.5539 |   93.42% |       0.28 |         0.0050 |
|     1 |       250 |   00:01:01 |     0.4922 |   88.28% |       0.39 |         0.0050 |
|     1 |       300 |   00:01:14 |     0.6570 |   82.81% |       0.40 |         0.0050 |
|     1 |       350 |   00:01:26 |     0.3510 |   93.90% |       0.34 |         0.0050 |
|     1 |       400 |   00:01:39 |     0.3814 |   89.84% |       0.39 |         0.0050 |
|     1 |       450 |   00:01:51 |     0.5450 |   87.50% |       0.41 |         0.0050 |
|     1 |       500 |   00:02:04 |     0.1987 |   98.44% |       0.34 |         0.0050 |
|     1 |       550 |   00:02:17 |     0.3008 |   92.17% |       0.33 |         0.0050 |
|     1 |       600 |   00:02:29 |     0.1535 |   96.88% |       0.32 |         0.0050 |
|     1 |       650 |   00:02:41 |     0.1445 |   96.25% |       0.24 |         0.0050 |
|     1 |       700 |   00:02:54 |     0.2625 |   95.00% |       0.38 |         0.0050 |
|     1 |       750 |   00:03:06 |     0.4782 |   87.50% |       0.37 |         0.0050 |
|     1 |       800 |   00:03:19 |     0.1638 |   98.39% |       0.67 |         0.0050 |
|     1 |       850 |   00:03:31 |     0.2185 |   98.44% |       0.34 |         0.0050 |
|     1 |       900 |   00:03:43 |     0.2123 |   95.31% |       0.29 |         0.0050 |
|     1 |       950 |   00:03:56 |     0.1388 |   97.66% |       0.26 |         0.0050 |
|     2 |      1000 |   00:04:11 |     0.2203 |   95.31% |       0.27 |         0.0050 |
|     2 |      1050 |   00:04:23 |     0.0634 |   98.48% |       0.37 |         0.0050 |
|     2 |      1100 |   00:04:36 |     0.1564 |   97.66% |       0.20 |         0.0050 |
|     2 |      1150 |   00:04:49 |     0.2041 |   96.88% |       0.30 |         0.0050 |
|     2 |      1200 |   00:05:01 |     0.0347 |   98.95% |       0.18 |         0.0050 |
|     2 |      1250 |   00:05:14 |     0.0961 |   96.88% |       0.24 |         0.0050 |
|     2 |      1300 |   00:05:26 |     0.1745 |   97.65% |       0.31 |         0.0050 |
|     2 |      1350 |   00:05:39 |     0.1954 |   92.97% |       0.25 |         0.0050 |
|     2 |      1400 |   00:05:51 |     0.2461 |   94.53% |       0.24 |         0.0050 |
|     2 |      1450 |   00:06:04 |     0.1449 |   95.79% |       0.23 |         0.0050 |
```

```
|     2 |       1500 |     00:06:17 |     0.0956 |     98.44% |     0.29 |     0.0050 |
|     2 |       1550 |     00:06:29 |     0.2340 |     92.19% |     0.26 |     0.0050 |
|     2 |       1600 |     00:06:42 |     0.2327 |     93.75% |     0.32 |     0.0050 |
|     2 |       1650 |     00:06:54 |     0.1548 |     96.09% |     0.16 |     0.0050 |
|     2 |       1700 |     00:07:06 |     0.1122 |     97.40% |     0.22 |     0.0050 |
|     2 |       1750 |     00:07:19 |     0.1166 |     96.88% |     0.21 |     0.0050 |
|     2 |       1800 |     00:07:31 |     0.2449 |     96.09% |     0.26 |     0.0050 |
|     2 |       1850 |     00:07:44 |     0.1168 |     98.44% |     0.23 |     0.0050 |
|     2 |       1900 |     00:07:56 |     0.2255 |     95.31% |     0.26 |     0.0050 |
|     2 |       1950 |     00:08:09 |     0.1614 |     97.66% |     0.28 |     0.0050 |
|     2 |       1984 |     00:08:17 |     0.0614 |     99.00% |     0.23 |     0.0050 |
|=======================================================================
=======================================|
```

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.

```
|=======================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |     Rate      |
|=======================================================================
=======================================|
|     1 |         1 |     00:00:00 |     2.8235 |     100.00% |     5.92 |     0.0050 |
|     1 |        50 |     00:00:02 |     0.2037 |     100.00% |     0.49 |     0.0050 |
|     1 |       100 |     00:00:05 |     0.4355 |     100.00% |     1.13 |     0.0050 |
|     1 |       150 |     00:00:08 |     0.1561 |     100.00% |     0.67 |     0.0050 |
|     1 |       200 |     00:00:11 |     0.1234 |     100.00% |     0.72 |     0.0050 |
|     1 |       250 |     00:00:14 |     0.5472 |     100.00% |     1.19 |     0.0050 |
|     1 |       300 |     00:00:17 |     0.2664 |     100.00% |     0.77 |     0.0050 |
|     1 |       350 |     00:00:20 |     0.0437 |     100.00% |     0.37 |     0.0050 |
|     1 |       400 |     00:00:23 |     0.0773 |     100.00% |     0.87 |     0.0050 |
|     1 |       450 |     00:00:25 |     0.0897 |     100.00% |     1.01 |     0.0050 |
|     1 |       500 |     00:00:28 |     0.1214 |     100.00% |     0.53 |     0.0050 |
|     1 |       550 |     00:00:31 |     0.2012 |     100.00% |     0.72 |     0.0050 |
|     1 |       600 |     00:00:34 |     0.2193 |     100.00% |     0.53 |     0.0050 |
|     1 |       650 |     00:00:37 |     0.3145 |     100.00% |     1.08 |     0.0050 |
|     1 |       700 |     00:00:40 |     0.2762 |     100.00% |     1.10 |     0.0050 |
|     1 |       750 |     00:00:43 |     0.3912 |     100.00% |     0.83 |     0.0050 |
|     1 |       800 |     00:00:46 |     0.1742 |     100.00% |     0.64 |     0.0050 |
|     1 |       850 |     00:00:48 |     0.4859 |     100.00% |     0.92 |     0.0050 |
|     1 |       900 |     00:00:51 |     0.1617 |     100.00% |     0.52 |     0.0050 |
|     1 |       950 |     00:00:54 |     0.1874 |     100.00% |     1.24 |     0.0050 |
|     1 |      1000 |     00:00:57 |     0.2255 |     100.00% |     0.60 |     0.0050 |
```

```
|     2 |       1050 |     00:01:02 |     0.1515 |    100.00% |       0.69 |       0.0050 |
|     2 |       1100 |     00:01:05 |     0.1542 |    100.00% |       0.76 |       0.0050 |
|     2 |       1150 |     00:01:08 |     0.3080 |    100.00% |       1.01 |       0.0050 |
|     2 |       1200 |     00:01:11 |     0.3184 |    100.00% |       0.77 |       0.0050 |
|     2 |       1250 |     00:01:14 |     0.5464 |    100.00% |       1.74 |       0.0050 |
|     2 |       1300 |     00:01:17 |     0.2412 |    100.00% |       0.62 |       0.0050 |
|     2 |       1350 |     00:01:20 |     0.0729 |    100.00% |       1.04 |       0.0050 |
|     2 |       1400 |     00:01:23 |     0.2458 |    100.00% |       0.73 |       0.0050 |
|     2 |       1450 |     00:01:25 |     1.0584 |    100.00% |       1.10 |       0.0050 |
|     2 |       1500 |     00:01:28 |     0.2303 |    100.00% |       1.00 |       0.0050 |
|     2 |       1550 |     00:01:31 |     0.3323 |    100.00% |       0.76 |       0.0050 |
|     2 |       1600 |     00:01:34 |     0.0361 |    100.00% |       0.74 |       0.0050 |
|     2 |       1650 |     00:01:37 |     0.2501 |    100.00% |       1.44 |       0.0050 |
|     2 |       1700 |     00:01:40 |     0.0802 |    100.00% |       0.79 |       0.0050 |
|     2 |       1750 |     00:01:43 |     0.0907 |    100.00% |       0.45 |       0.0050 |
|     2 |       1800 |     00:01:46 |     0.2056 |    100.00% |       0.71 |       0.0050 |
|     2 |       1850 |     00:01:49 |     0.0794 |    100.00% |       0.74 |       0.0050 |
|     2 |       1900 |     00:01:52 |     0.4074 |    100.00% |       1.14 |       0.0050 |
|     2 |       1950 |     00:01:54 |     0.1408 |    100.00% |       0.61 |       0.0050 |
|     2 |       2000 |     00:01:57 |     0.1751 |    100.00% |       0.70 |       0.0050 |
|     2 |       2026 |     00:01:59 |     0.0958 |    100.00% |       1.11 |       0.0050 |
|=========================================================================
=======================================|

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 1013 training images...done.

Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.2051 |     93.75% |       0.29 |       0.0050 |
|     1 |        50 |     00:00:09 |     0.1371 |     95.31% |       0.27 |       0.0050 |
|     1 |       100 |     00:00:19 |     0.1721 |     96.09% |       0.19 |       0.0050 |
|     1 |       150 |     00:00:29 |     0.2006 |     94.53% |       0.21 |       0.0050 |
|     1 |       200 |     00:00:39 |     0.2563 |     93.20% |       0.48 |       0.0050 |
|     1 |       250 |     00:00:49 |     0.2309 |     94.53% |       0.34 |       0.0050 |
|     1 |       300 |     00:00:58 |     0.2219 |     94.53% |       0.20 |       0.0050 |
|     1 |       350 |     00:01:08 |     0.1334 |     95.00% |       0.20 |       0.0050 |
|     1 |       400 |     00:01:18 |     0.0856 |     98.44% |       0.27 |       0.0050 |
```

1	450	00:01:27	0.1186	96.88%	0.20	0.0050
1	500	00:01:37	0.1598	96.88%	0.24	0.0050
1	550	00:01:46	0.2355	94.38%	0.28	0.0050
1	600	00:01:56	0.0976	96.88%	0.19	0.0050
1	650	00:02:06	0.1219	96.09%	0.21	0.0050
1	700	00:02:16	0.1710	94.53%	0.19	0.0050
1	750	00:02:26	0.1559	93.88%	0.25	0.0050
1	800	00:02:35	0.4363	88.28%	0.32	0.0050
1	850	00:02:45	0.1032	96.88%	0.20	0.0050
1	900	00:02:55	0.2146	92.97%	0.25	0.0050
1	950	00:03:05	0.1770	96.09%	0.29	0.0050
1	1000	00:03:14	0.2259	92.97%	0.21	0.0050
2	1050	00:03:27	0.0832	99.22%	0.15	0.0050
2	1100	00:03:37	0.1890	94.53%	0.31	0.0050
2	1150	00:03:47	0.1726	95.31%	0.19	0.0050
2	1200	00:03:57	0.0610	98.44%	0.12	0.0050
2	1250	00:04:07	0.1008	96.88%	0.17	0.0050
2	1300	00:04:17	0.2197	96.09%	0.28	0.0050
2	1350	00:04:26	0.1101	98.44%	0.23	0.0050
2	1400	00:04:36	0.2595	91.41%	0.31	0.0050
2	1450	00:04:46	0.1765	93.75%	0.19	0.0050
2	1500	00:04:55	0.1538	96.09%	0.20	0.0050
2	1550	00:05:05	0.0641	99.21%	0.14	0.0050
2	1600	00:05:15	0.0688	97.66%	0.19	0.0050
2	1650	00:05:24	0.0733	97.66%	0.21	0.0050
2	1700	00:05:34	0.1644	96.88%	0.17	0.0050
2	1750	00:05:44	0.0881	97.66%	0.21	0.0050
2	1800	00:05:53	0.1818	96.09%	0.18	0.0050
2	1850	00:06:03	0.1707	96.49%	0.29	0.0050
2	1900	00:06:13	0.0810	97.30%	0.16	0.0050
2	1950	00:06:22	0.1863	94.53%	0.18	0.0050
2	2000	00:06:32	0.1433	96.09%	0.20	0.0050
3	2050	00:06:45	0.0917	98.44%	0.14	0.0050
3	2100	00:06:55	0.1345	96.09%	0.17	0.0050
3	2150	00:07:05	0.2048	96.30%	0.34	0.0050
3	2200	00:07:15	0.0945	99.22%	0.19	0.0050
3	2250	00:07:25	0.1451	96.09%	0.17	0.0050
3	2300	00:07:35	0.2916	94.53%	0.31	0.0050
3	2350	00:07:45	0.1857	94.53%	0.19	0.0050
3	2400	00:07:55	0.1002	97.66%	0.18	0.0050
3	2450	00:08:04	0.0806	96.88%	0.23	0.0050
3	2500	00:08:14	0.1388	96.88%	0.16	0.0050
3	2550	00:08:23	0.1218	97.52%	0.30	0.0050
3	2600	00:08:33	0.1599	96.09%	0.18	0.0050

3	2650	00:08:43	0.1578	96.09%	0.26	0.0050
3	2700	00:08:53	0.0484	100.00%	0.24	0.0050
3	2750	00:09:03	0.0740	97.30%	0.15	0.0050
3	2800	00:09:13	0.0688	97.39%	0.17	0.0050
3	2850	00:09:23	0.1298	96.88%	0.16	0.0050
3	2900	00:09:32	0.1085	96.88%	0.18	0.0050
3	2950	00:09:42	0.2133	96.88%	0.18	0.0050
3	3000	00:09:52	0.1166	94.53%	0.15	0.0050
3	3018	00:09:55	0.0571	99.15%	0.12	0.0050

Detector training complete (with warnings):
Warning: Invalid bounding boxes from 31 out of 1014 training images were removed.
The following rows in trainingData have invalid
bounding box data:

Invalid Rows

_____

17
75
194
199
202
249
253
261
290
309
369
395
399
498
617
652
678
679
703
717
731
752
767
806

808
811
816
864
897
920
937

Bounding boxes must be fully contained within their associated image and must have positive width and height.

****

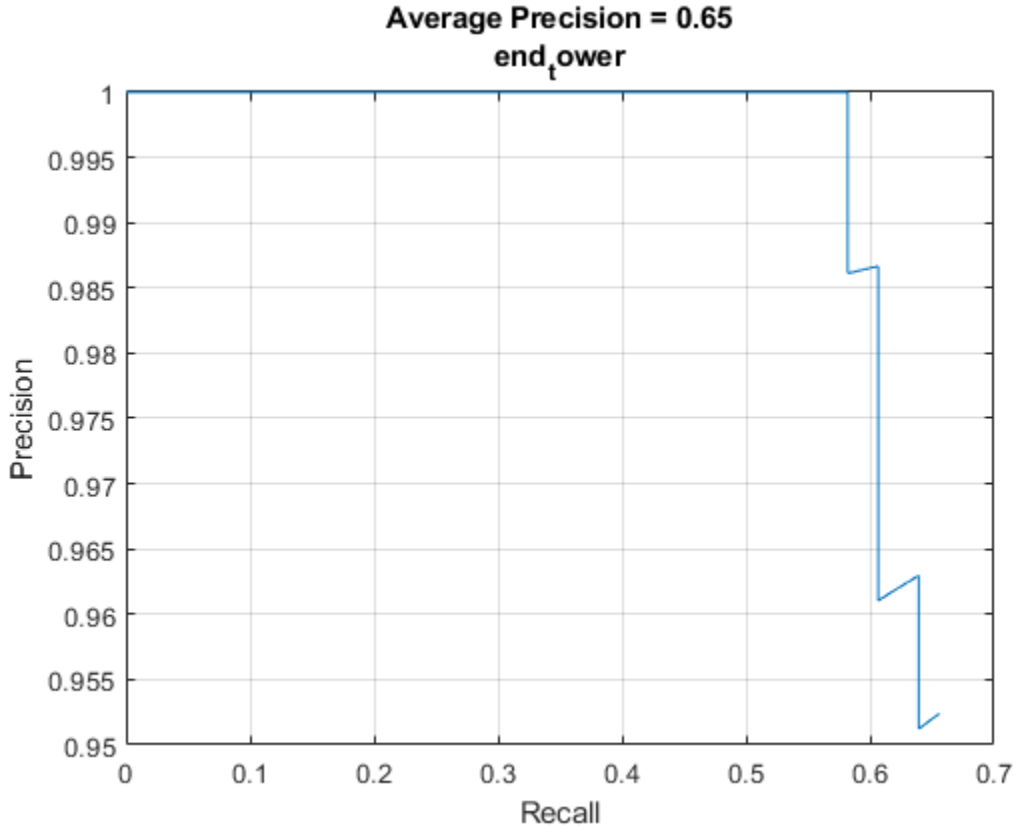
trainedDetector =

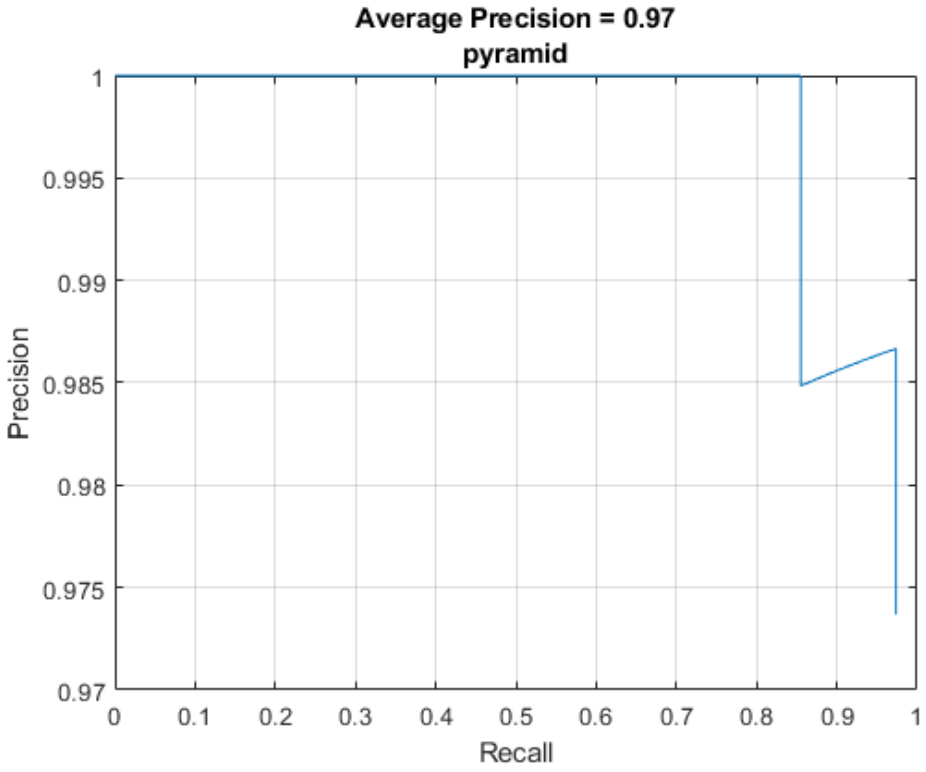
  fasterRCNNObjectDetector with properties:

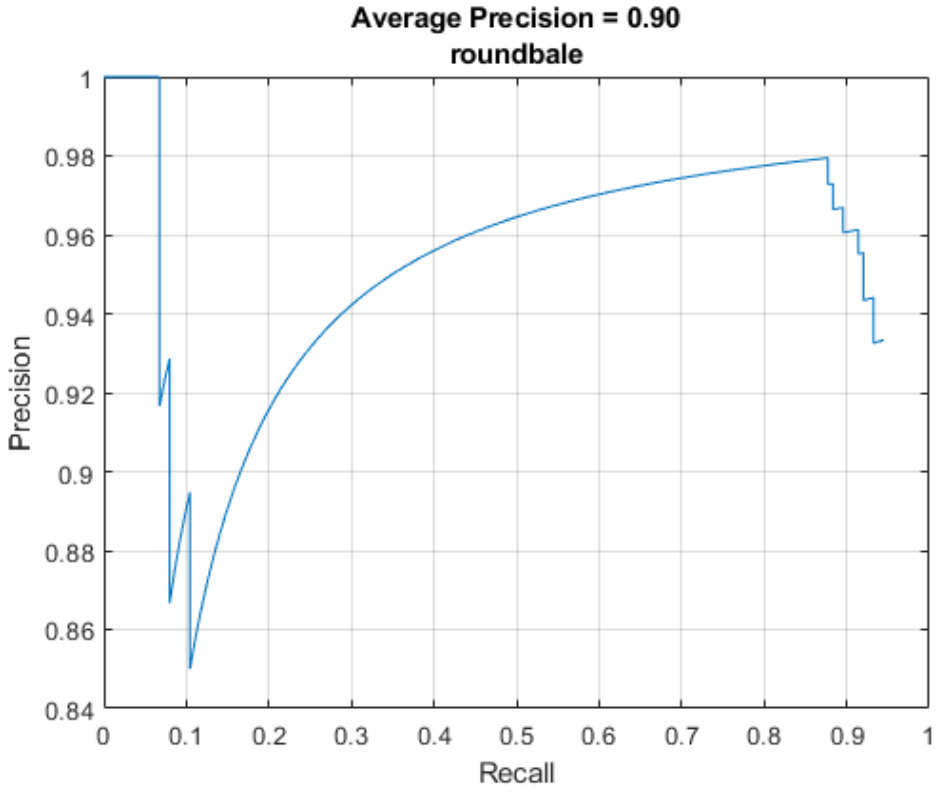
       ModelName: 'tractors'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'tractors' 'roundbale' 'end_tower' 'pyramid' 'span' 'tower' 'Background'}
    MinObjectSize: [16 16]

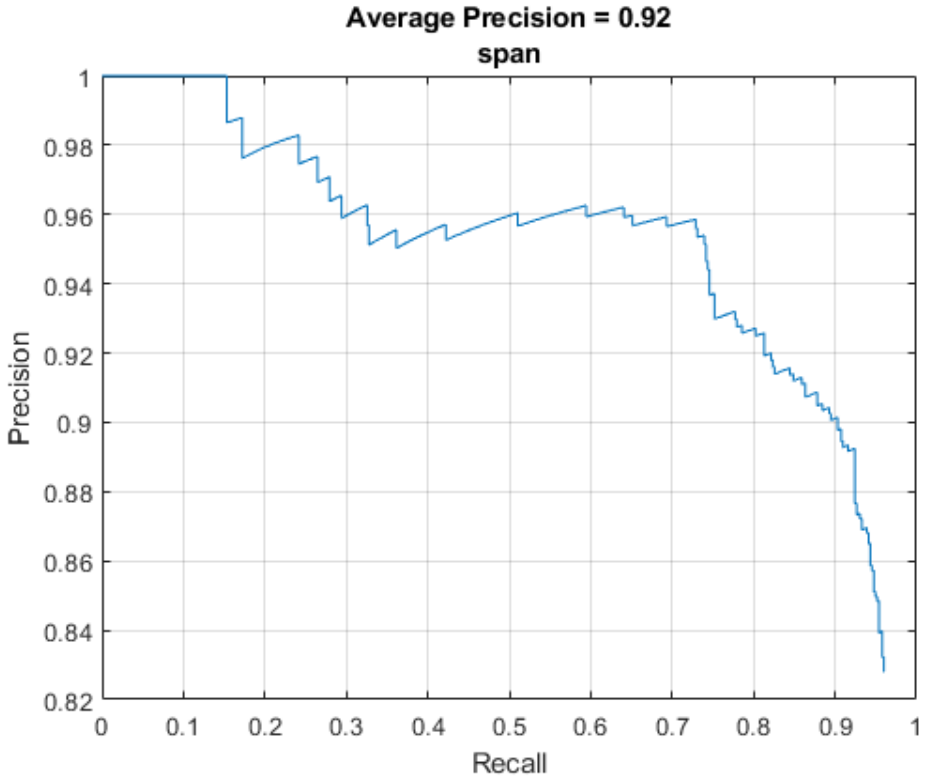
Elapsed time is 1605.589488 seconds.

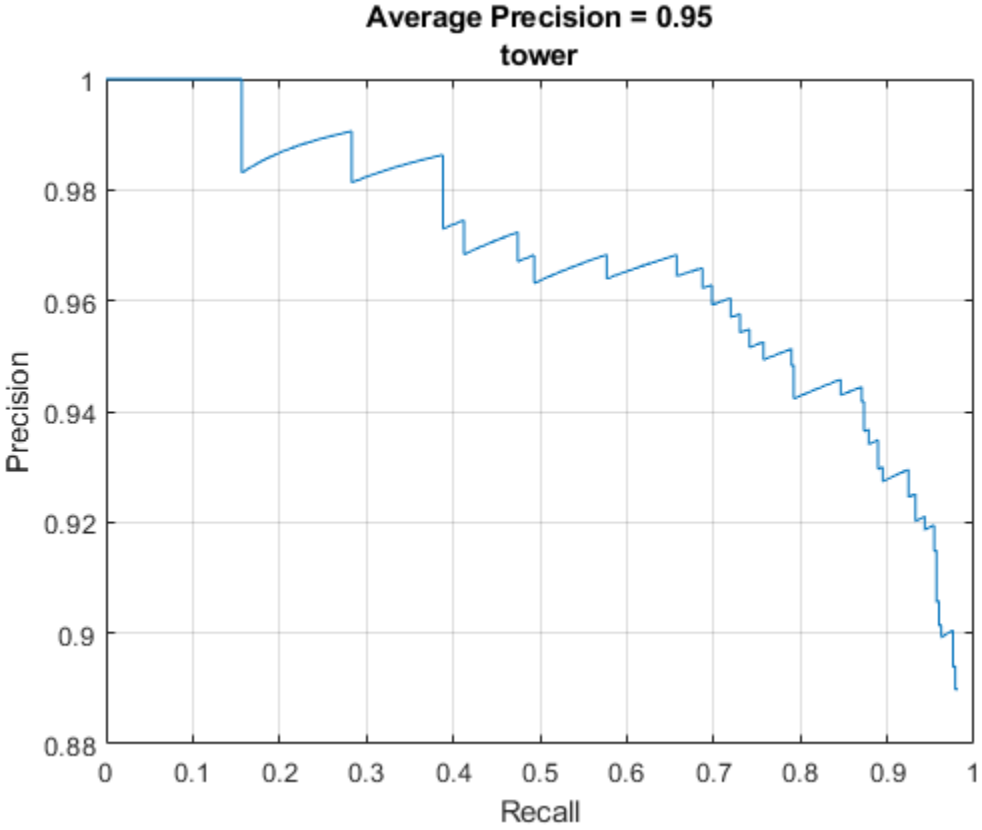
APPENDIX F.11.1.1 Result

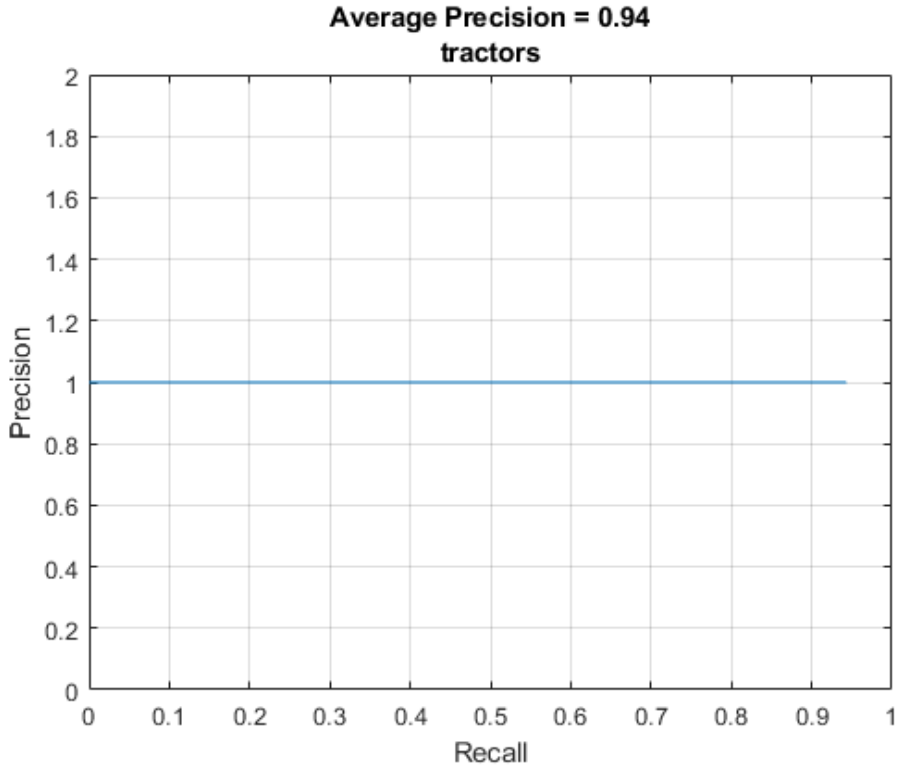





Average Precision = 0.90
roundbale
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.92
span
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0.82
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.95
tower
Precision
Recall
1
0.98
0.96
0.94
0.92
0.9
0.88
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.94
tractors
Precision
Recall
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

APPENDIX F.11.1.2 Results on test pictures

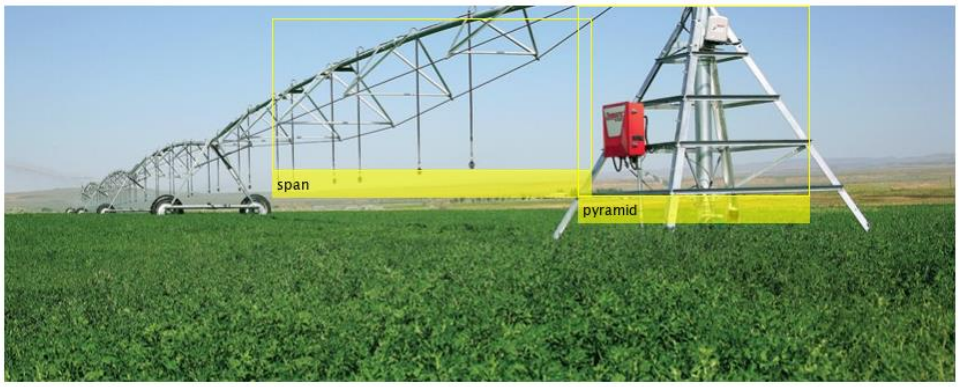

tractors
tractors
CASE IH
MAGNUM
ASI


tractors
tractors
tractors
tractors


tractors
tractors


span


VODAR
span
vodarirrigation.en.alibaba.com


span
pyramid

APPENDIX F.11.2 GoogLeNet Run 2

>> combinedMergedDetectorv2

optionsStage1 =

```
  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0050
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 2
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
                  CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
             ExecutionEnvironment: 'auto'
                      WorkerLoad: []
                       OutputFcn: []
                           Plots: 'none'
                  SequenceLength: 'longest'
               SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                        Momentum: 0.9000
                InitialLearnRate: 0.0050
           LearnRateScheduleSettings: [1×1 struct]
                L2Regularization: 1.0000e-04
            GradientThresholdMethod: 'l2norm'
                GradientThreshold: Inf
                       MaxEpochs: 2
                   MiniBatchSize: 1
                         Verbose: 1
                VerboseFrequency: 50
                  ValidationData: []
             ValidationFrequency: 50
              ValidationPatience: Inf
                         Shuffle: 'once'
```

```
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                     Verbose: 1
            VerboseFrequency: 50
              ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                     Shuffle: 'once'
           CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
            SequenceLength: 'longest'
        SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
```

```
    GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
         ValidationFrequency: 50
          ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
        ExecutionEnvironment: 'auto'
                  WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
       SequencePaddingValue: 0
```

****
*
Training a Faster R-CNN Object Detector for the following object classes:

* tractors
* roundbale
* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 31 out of 1014 training images were removed.
The following rows in trainingData have invalid
bounding box data:

Invalid Rows
_____

17
75
194
199
202
249
253

261
290
309
369
395
399
498
617
652
678
679
703
717
731
752
767
806
808
811
816
864
897
920
937

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.

Epoch	Iteration	Time Elapsed	Mini-batch	Mini-batch	Mini-batch	Base Learning
		(hh:mm:ss)	Loss	Accuracy	RMSE	Rate
1	1	00:00:00	0.6249	88.28%	1.45	0.0050
1	50	00:00:05	0.1310	100.00%	0.60	0.0050
1	100	00:00:10	0.4711	100.00%	0.92	0.0050
1	150	00:00:15	0.1454	100.00%	0.78	0.0050
1	200	00:00:19	0.1230	100.00%	0.79	0.0050
1	250	00:00:24	0.3991	100.00%	1.82	0.0050
1	300	00:00:29	0.4517	100.00%	1.43	0.0050
1	350	00:00:34	0.2469	100.00%	0.64	0.0050
1	400	00:00:39	1.1272	100.00%	1.69	0.0050

1	450	00:00:44	0.2508	100.00%	0.98	0.0050
1	500	00:00:49	0.1379	100.00%	0.92	0.0050
1	550	00:00:54	0.0881	100.00%	0.74	0.0050
1	600	00:00:59	0.5188	100.00%	0.95	0.0050
1	650	00:01:04	0.2294	100.00%	0.83	0.0050
1	700	00:01:09	0.2506	100.00%	0.66	0.0050
1	750	00:01:14	0.3604	100.00%	0.99	0.0050
1	800	00:01:19	0.1320	100.00%	0.74	0.0050
1	850	00:01:24	0.1032	100.00%	1.13	0.0050
1	900	00:01:29	0.3944	100.00%	0.76	0.0050
1	950	00:01:33	0.2899	100.00%	1.06	0.0050
1	1000	00:01:38	0.2820	100.00%	0.90	0.0050
2	1050	00:01:46	0.1721	100.00%	1.84	0.0050
2	1100	00:01:52	0.3402	100.00%	1.07	0.0050
2	1150	00:01:57	0.1391	100.00%	1.11	0.0050
2	1200	00:02:01	0.2053	100.00%	0.60	0.0050
2	1250	00:02:06	0.3389	100.00%	0.83	0.0050
2	1300	00:02:12	0.4458	100.00%	0.88	0.0050
2	1350	00:02:17	0.2158	100.00%	0.76	0.0050
2	1400	00:02:21	0.2399	100.00%	0.96	0.0050
2	1450	00:02:26	0.6801	100.00%	1.84	0.0050
2	1500	00:02:31	0.0795	100.00%	0.53	0.0050
2	1550	00:02:36	0.3180	100.00%	0.76	0.0050
2	1600	00:02:41	0.2726	100.00%	1.09	0.0050
2	1650	00:02:46	1.5224	100.00%	2.05	0.0050
2	1700	00:02:51	0.1518	100.00%	0.79	0.0050
2	1750	00:02:56	0.3028	100.00%	0.96	0.0050
2	1800	00:03:01	0.2267	100.00%	0.72	0.0050
2	1850	00:03:06	0.2464	100.00%	0.74	0.0050
2	1900	00:03:11	0.1739	100.00%	0.76	0.0050
2	1950	00:03:16	0.0853	100.00%	0.83	0.0050
2	2000	00:03:21	0.2062	100.00%	0.61	0.0050
2	2026	00:03:23	0.4706	100.00%	0.88	0.0050
===

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 1013 training images...done.

Training on single GPU.
|===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |

		(hh:mm:ss)	Loss	Accuracy	RMSE	Rate
1	1	00:00:00	2.2772	8.59%	0.37	0.0050
1	50	00:00:12	0.0789	99.22%	0.42	0.0050
1	100	00:00:25	0.3636	92.19%	0.40	0.0050
1	150	00:00:37	0.3624	93.75%	0.37	0.0050
1	200	00:00:50	0.3553	94.53%	0.45	0.0050
1	250	00:01:03	0.1770	96.77%	0.27	0.0050
1	300	00:01:15	0.3383	94.53%	0.51	0.0050
1	350	00:01:28	0.3494	95.31%	0.41	0.0050
1	400	00:01:41	0.1126	96.91%	0.19	0.0050
1	450	00:01:54	0.4382	89.84%	0.30	0.0050
1	500	00:02:06	0.3788	92.19%	0.36	0.0050
1	550	00:02:19	0.2647	93.75%	0.35	0.0050
1	600	00:02:31	0.1062	96.09%	0.23	0.0050
1	650	00:02:44	0.2024	93.75%	0.22	0.0050
1	700	00:02:56	0.0739	98.44%	0.46	0.0050
1	750	00:03:09	0.1603	92.97%	0.22	0.0050
1	800	00:03:22	0.1049	97.53%	0.27	0.0050
1	850	00:03:34	0.3344	89.84%	0.25	0.0050
1	900	00:03:46	0.1894	92.97%	0.26	0.0050
1	950	00:03:59	0.0682	100.00%	0.30	0.0050
2	1000	00:04:15	0.2079	95.15%	0.24	0.0050
2	1050	00:04:27	0.4733	94.69%	0.45	0.0050
2	1100	00:04:40	0.3033	92.97%	0.33	0.0050
2	1150	00:04:52	0.1191	96.09%	0.26	0.0050
2	1200	00:05:05	0.0523	99.22%	0.32	0.0050
2	1250	00:05:18	0.4926	92.19%	0.35	0.0050
2	1300	00:05:31	0.3772	89.84%	0.39	0.0050
2	1350	00:05:44	0.1663	96.88%	0.28	0.0050
2	1400	00:05:56	0.2909	91.41%	0.20	0.0050
2	1450	00:06:09	0.1409	96.09%	0.19	0.0050
2	1500	00:06:21	0.1754	94.53%	0.29	0.0050
2	1550	00:06:34	0.2258	96.09%	0.31	0.0050
2	1600	00:06:46	0.0838	96.09%	0.24	0.0050
2	1650	00:06:59	0.0497	98.90%	0.27	0.0050
2	1700	00:07:11	0.1205	96.88%	0.16	0.0050
2	1750	00:07:24	0.1551	96.88%	0.27	0.0050
2	1800	00:07:36	0.0929	96.88%	0.19	0.0050
2	1850	00:07:49	0.1133	96.88%	0.22	0.0050
2	1900	00:08:02	0.0980	97.40%	0.36	0.0050
2	1912	00:08:05	0.1720	96.09%	0.20	0.0050

```
|=========================================================================
=======================================|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|=========================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=========================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.0676 |    100.00% |       1.11 |        0.0050 |
|     1 |        50 |     00:00:02 |     0.7910 |    100.00% |       1.44 |        0.0050 |
|     1 |       100 |     00:00:05 |     0.1167 |    100.00% |       0.57 |        0.0050 |
|     1 |       150 |     00:00:08 |     0.1519 |    100.00% |       0.73 |        0.0050 |
|     1 |       200 |     00:00:11 |     0.3640 |    100.00% |       1.05 |        0.0050 |
|     1 |       250 |     00:00:14 |     0.3126 |    100.00% |       0.77 |        0.0050 |
|     1 |       300 |     00:00:17 |     0.1675 |    100.00% |       0.56 |        0.0050 |
|     1 |       350 |     00:00:20 |     0.3435 |    100.00% |       0.74 |        0.0050 |
|     1 |       400 |     00:00:23 |     0.2712 |    100.00% |       0.99 |        0.0050 |
|     1 |       450 |     00:00:26 |     0.1314 |    100.00% |       0.60 |        0.0050 |
|     1 |       500 |     00:00:28 |     0.1576 |    100.00% |       0.70 |        0.0050 |
|     1 |       550 |     00:00:31 |     0.4529 |    100.00% |       0.91 |        0.0050 |
|     1 |       600 |     00:00:34 |     0.3069 |    100.00% |       0.99 |        0.0050 |
|     1 |       650 |     00:00:37 |     0.5441 |    100.00% |       1.35 |        0.0050 |
|     1 |       700 |     00:00:40 |     0.1994 |    100.00% |       0.52 |        0.0050 |
|     1 |       750 |     00:00:43 |     0.4377 |    100.00% |       0.95 |        0.0050 |
|     1 |       800 |     00:00:45 |     0.2146 |    100.00% |       0.71 |        0.0050 |
|     1 |       850 |     00:00:48 |     0.0893 |    100.00% |       0.54 |        0.0050 |
|     1 |       900 |     00:00:51 |     0.2969 |    100.00% |       0.90 |        0.0050 |
|     1 |       950 |     00:00:54 |     0.2539 |    100.00% |       0.97 |        0.0050 |
|     1 |      1000 |     00:00:57 |     0.0979 |    100.00% |       0.51 |        0.0050 |
|     2 |      1050 |     00:01:02 |     0.0985 |    100.00% |       1.71 |        0.0050 |
|     2 |      1100 |     00:01:05 |     0.0849 |    100.00% |       0.60 |        0.0050 |
|     2 |      1150 |     00:01:08 |     0.1217 |    100.00% |       0.68 |        0.0050 |
|     2 |      1200 |     00:01:11 |     0.1436 |    100.00% |       0.75 |        0.0050 |
|     2 |      1250 |     00:01:14 |     0.5591 |    100.00% |       1.09 |        0.0050 |
|     2 |      1300 |     00:01:16 |     0.1609 |    100.00% |       0.57 |        0.0050 |
|     2 |      1350 |     00:01:19 |     0.1188 |    100.00% |       0.57 |        0.0050 |
|     2 |      1400 |     00:01:22 |     0.2047 |    100.00% |       0.71 |        0.0050 |
|     2 |      1450 |     00:01:25 |     0.2465 |    100.00% |       1.00 |        0.0050 |
|     2 |      1500 |     00:01:28 |     0.1634 |    100.00% |       0.62 |        0.0050 |
|     2 |      1550 |     00:01:31 |     0.1807 |    100.00% |       0.76 |        0.0050 |
```

2	1600	00:01:33	0.0471	100.00%	0.48	0.0050
2	1650	00:01:36	0.2842	100.00%	0.69	0.0050
2	1700	00:01:39	0.0994	100.00%	1.09	0.0050
2	1750	00:01:42	2.5899	100.00%	1.99	0.0050
2	1800	00:01:45	0.0407	100.00%	0.76	0.0050
2	1850	00:01:47	0.1287	100.00%	0.61	0.0050
2	1900	00:01:50	0.3404	100.00%	0.72	0.0050
2	1950	00:01:53	0.7236	100.00%	1.46	0.0050
2	2000	00:01:56	0.2315	100.00%	0.58	0.0050
2	2026	00:01:57	0.2857	100.00%	1.27	0.0050
===

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 1013 training images...done.

Training on single GPU.
|===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===|
1	1	00:00:00	0.1005	97.39%	0.23	0.0050
1	50	00:00:09	0.4831	89.84%	0.29	0.0050
1	100	00:00:18	0.2010	95.31%	0.24	0.0050
1	150	00:00:28	0.2692	93.48%	0.27	0.0050
1	200	00:00:38	0.0941	99.03%	0.25	0.0050
1	250	00:00:47	0.1704	96.09%	0.22	0.0050
1	300	00:00:57	0.0185	100.00%	0.15	0.0050
1	350	00:01:06	0.1110	98.44%	0.24	0.0050
1	400	00:01:16	0.2149	96.09%	0.21	0.0050
1	450	00:01:25	0.1369	97.66%	0.28	0.0050
1	500	00:01:35	0.0626	98.15%	0.22	0.0050
1	550	00:01:45	0.2462	95.31%	0.27	0.0050
1	600	00:01:54	0.2644	94.53%	0.33	0.0050
1	650	00:02:04	0.0704	99.22%	0.15	0.0050
1	700	00:02:13	0.0947	97.22%	0.25	0.0050
1	750	00:02:23	0.0540	98.44%	0.25	0.0050
1	800	00:02:33	0.3255	92.19%	0.35	0.0050
1	850	00:02:42	0.4601	86.72%	0.23	0.0050
1	900	00:02:52	0.1347	97.66%	0.17	0.0050
1	950	00:03:01	0.1491	96.09%	0.21	0.0050

2	1000	00:03:14	0.3532	91.41%	0.25	0.0050
2	1050	00:03:23	0.1486	97.66%	0.24	0.0050
2	1100	00:03:33	0.1374	96.88%	0.22	0.0050
2	1150	00:03:43	0.0293	100.00%	0.17	0.0050
2	1200	00:03:52	0.1027	96.88%	0.17	0.0050
2	1250	00:04:02	0.2351	93.75%	0.29	0.0050
2	1300	00:04:11	0.0390	98.86%	0.17	0.0050
2	1350	00:04:21	0.3175	90.63%	0.20	0.0050
2	1400	00:04:31	0.1470	95.31%	0.23	0.0050
2	1450	00:04:40	0.0741	97.66%	0.18	0.0050
2	1500	00:04:50	0.1185	97.66%	0.20	0.0050
2	1550	00:05:00	0.0037	100.00%	0.11	0.0050
2	1600	00:05:09	0.2169	93.75%	0.25	0.0050
2	1650	00:05:19	0.2676	94.53%	0.19	0.0050
2	1700	00:05:29	0.1196	95.31%	0.18	0.0050
2	1750	00:05:39	0.0215	99.22%	0.16	0.0050
2	1800	00:05:48	0.3551	91.41%	0.21	0.0050
2	1850	00:05:57	0.1678	98.44%	0.22	0.0050
2	1900	00:06:07	0.1888	92.19%	0.17	0.0050
2	1950	00:06:16	0.2389	93.75%	0.26	0.0050
3	2000	00:06:29	0.0860	97.66%	0.17	0.0050
3	2050	00:06:39	0.1149	98.44%	0.19	0.0050
3	2100	00:06:48	0.2092	94.53%	0.28	0.0050
3	2150	00:06:58	0.0137	100.00%	0.12	0.0050
3	2200	00:07:07	0.1156	96.88%	0.18	0.0050
3	2250	00:07:17	0.0583	98.32%	0.22	0.0050
3	2300	00:07:27	0.1172	95.31%	0.23	0.0050
3	2350	00:07:36	0.0689	96.88%	0.16	0.0050
3	2400	00:07:45	0.0226	98.94%	0.15	0.0050
3	2450	00:07:55	0.2406	94.53%	0.24	0.0050
3	2500	00:08:05	0.0751	98.44%	0.22	0.0050
3	2550	00:08:14	0.1048	96.88%	0.18	0.0050
3	2600	00:08:24	0.1909	94.53%	0.32	0.0050
3	2650	00:08:34	0.1899	96.09%	0.25	0.0050
3	2700	00:08:44	0.2912	92.19%	0.19	0.0050
3	2750	00:08:53	0.0770	99.22%	0.16	0.0050
3	2800	00:09:02	0.0789	97.27%	0.25	0.0050
3	2850	00:09:12	0.2126	96.88%	0.24	0.0050
3	2900	00:09:21	0.0332	98.90%	0.14	0.0050
3	2937	00:09:28	0.0553	98.44%	0.14	0.0050
==========

Detector training complete (with warnings):

Warning: Invalid bounding boxes from 31 out of 1014 training images were removed.
The following rows in trainingData have invalid
bounding box data:

Invalid Rows

_____

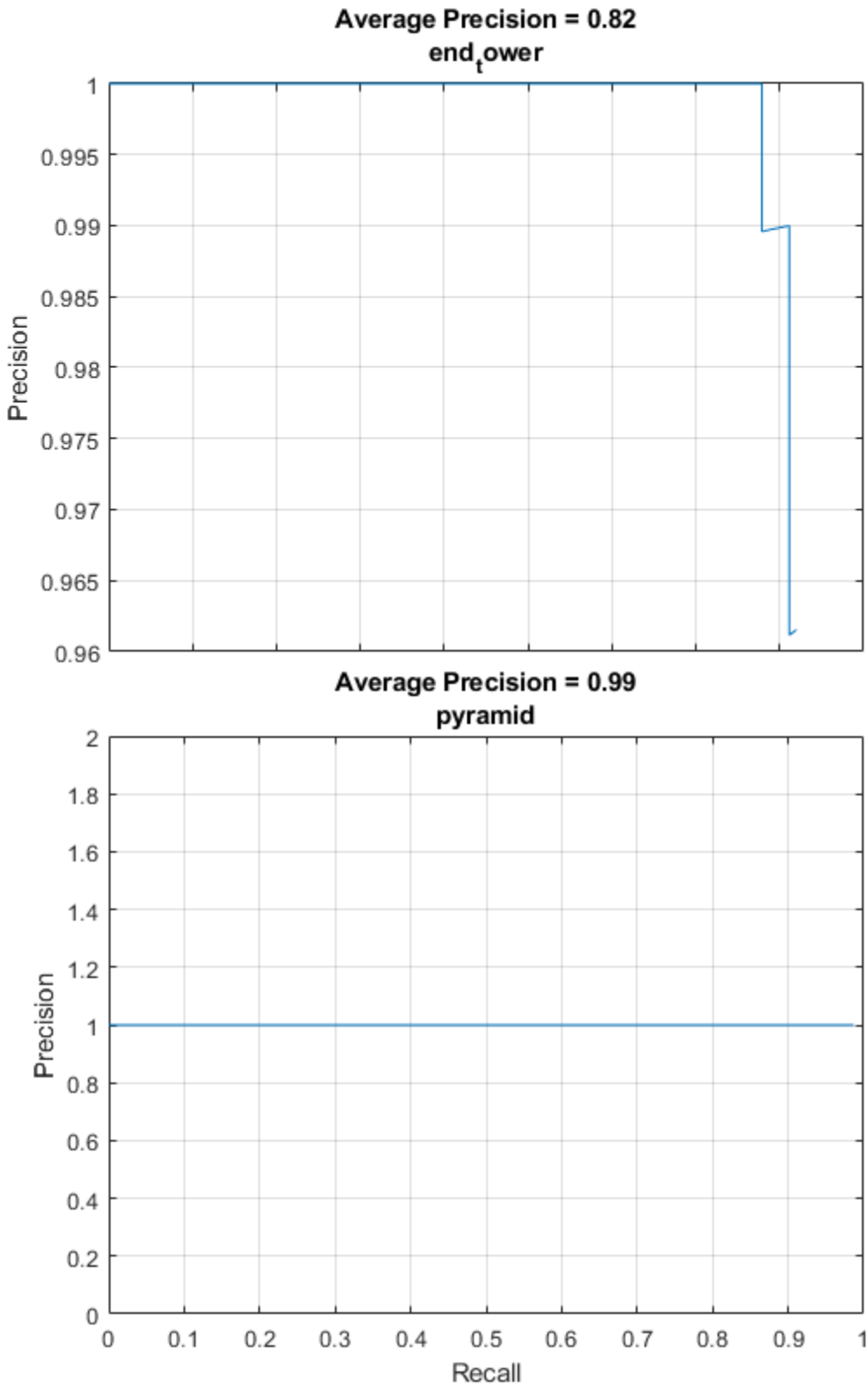
17
75
194
199
202
249
253
261
290
309
369
395
399
498
617
652
678
679
703
717
731
752
767
806
808
811
816
864
897
920
937

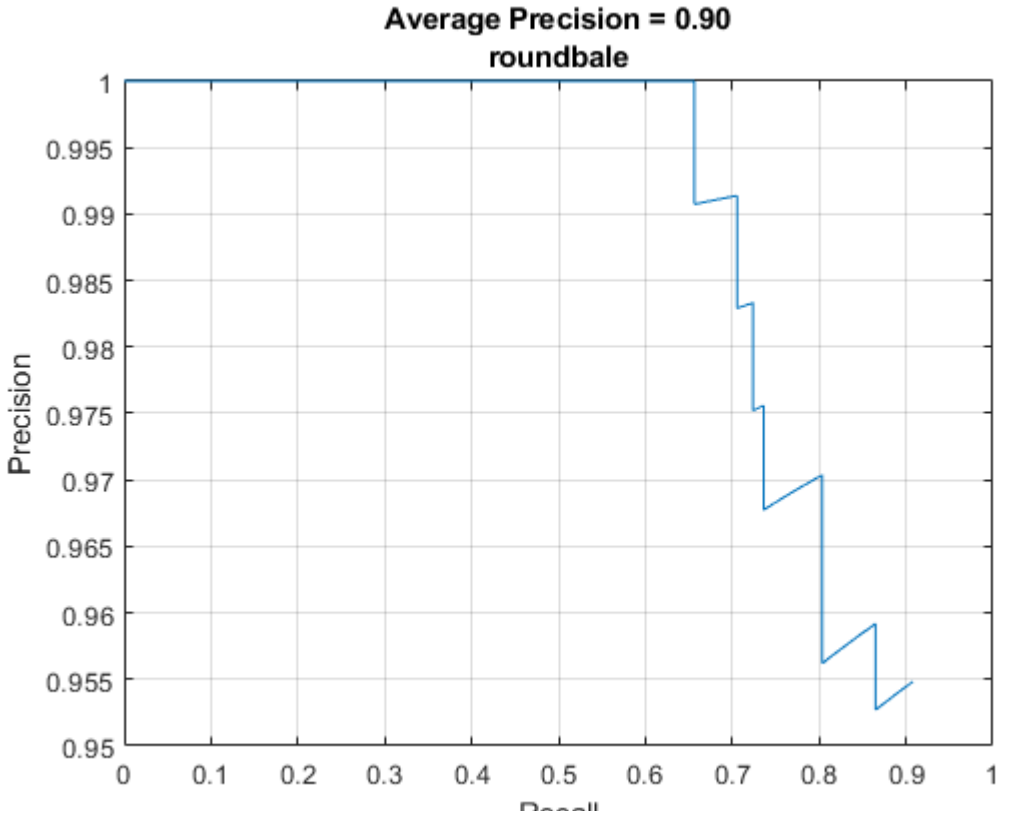
Bounding boxes must be fully contained within their associated image and must have positive width and height.

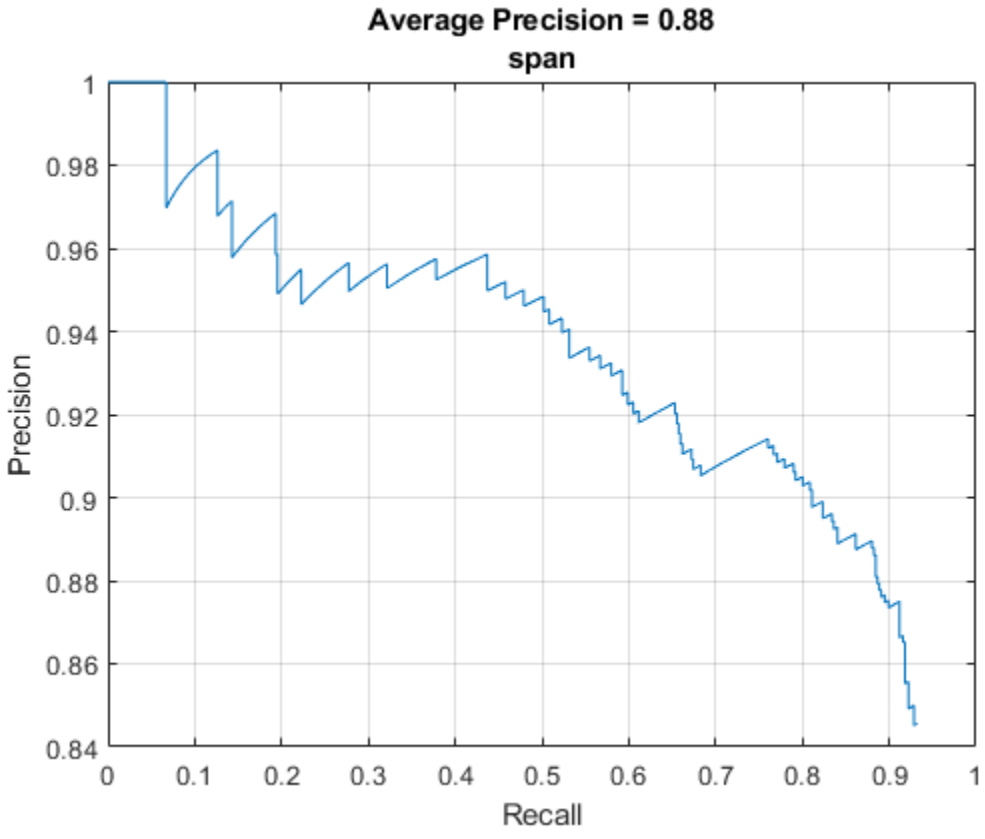
****

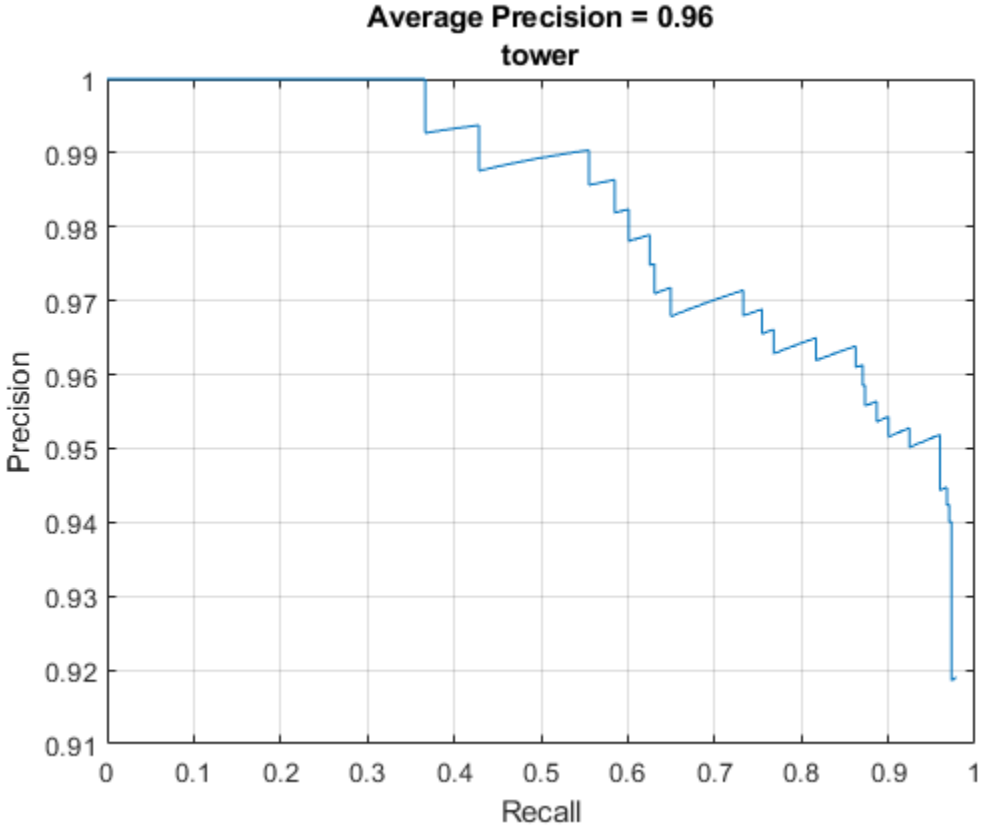
```
trainedDetector =

  fasterRCNNObjectDetector with properties:

       ModelName: 'tractors'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'tractors'  'roundbale'  'end_tower'  'pyramid'  'span'  'tower'
'Background'}
    MinObjectSize: [16 16]

Elapsed time is 1583.434285 seconds.
```

APPENDIX F.11.2.1 Result

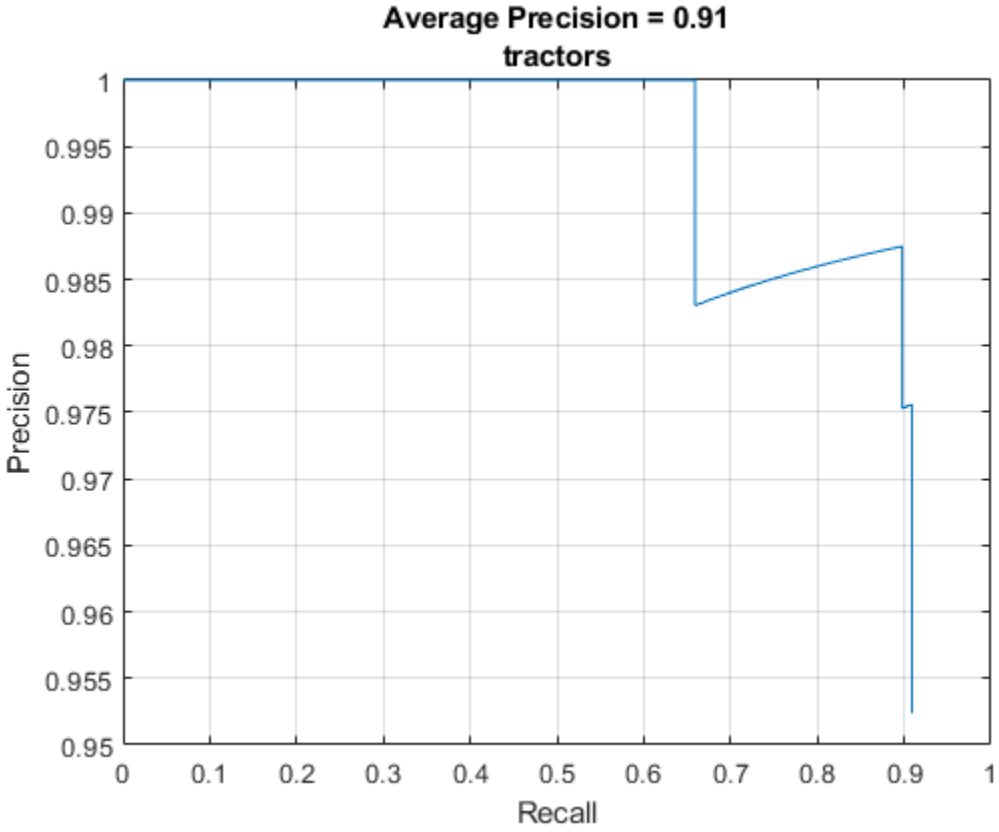


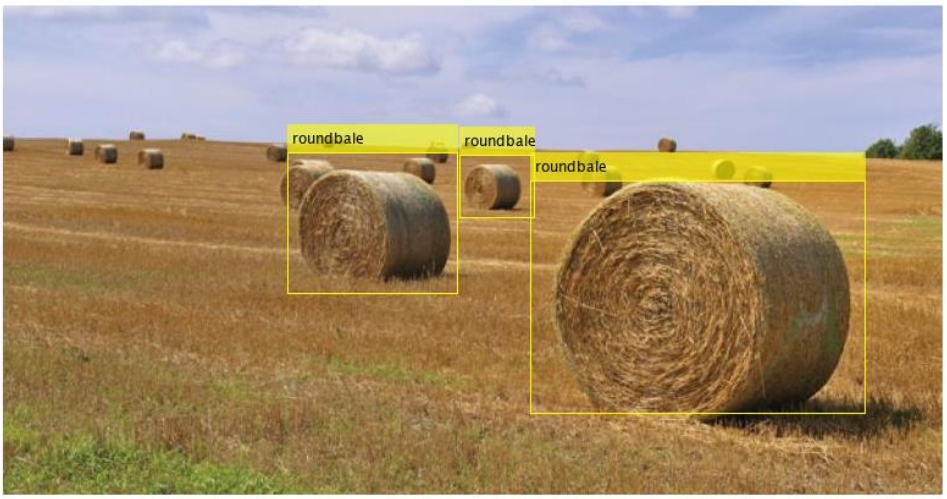

Average Precision = 0.90
roundbale
Precision
1
0.995
0.99
0.985
0.98
0.975
0.97
0.965
0.96
0.955
0.95
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.88
span
Precision
1
0.98
0.96
0.94
0.92
0.9
0.88
0.86
0.84
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Recall



APPENDIX F.11.2.2 Results on test pictures



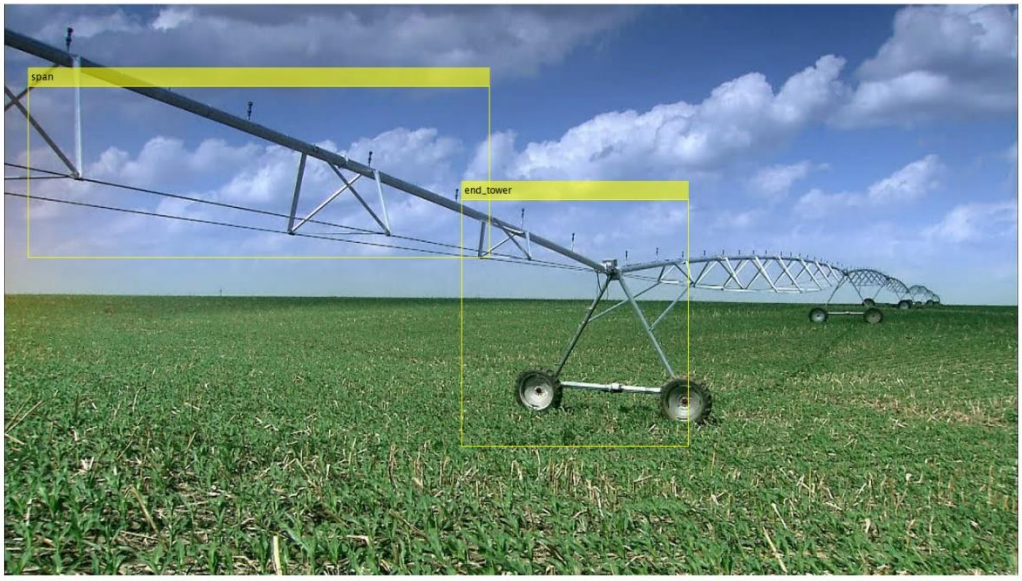

roundbale
roundbale
roundbale


roundbale


tractors
CASE IH
MAGNUM
ASI


tractors


tractors


span
end_tower


span
span


roundbale
tractors
roundbale
roundbale

APPENDIX F.12 Pivotv2

APPENDIX F.12.1 GoogLeNet

```
>> pivotDetectorGoogLeNetv3v2

optionsStage1 =

  TrainingOptionsSGDM with properties:

                     Momentum: 0.9000
            InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
           L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
           GradientThreshold: Inf
                   MaxEpochs: 2
                MiniBatchSize: 1
                     Verbose: 1
             VerboseFrequency: 50
              ValidationData: []
```

```
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage2 =

  TrainingOptionsSGDM with properties:

                   Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
     GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
               MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                 WorkerLoad: []
                  OutputFcn: []
                      Plots: 'none'
             SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage3 =

  TrainingOptionsSGDM with properties:

                   Momentum: 0.9000
```

```
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 2
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
                     Plots: 'none'
          SequenceLength: 'longest'
       SequencePaddingValue: 0


optionsStage4 =

  TrainingOptionsSGDM with properties:

                  Momentum: 0.9000
          InitialLearnRate: 0.0050
    LearnRateScheduleSettings: [1×1 struct]
          L2Regularization: 1.0000e-04
      GradientThresholdMethod: 'l2norm'
          GradientThreshold: Inf
                  MaxEpochs: 3
              MiniBatchSize: 1
                    Verbose: 1
           VerboseFrequency: 50
             ValidationData: []
        ValidationFrequency: 50
         ValidationPatience: Inf
                    Shuffle: 'once'
             CheckpointPath: 'C:\Users\clindhorst2\AppData\Local\Temp\'
       ExecutionEnvironment: 'auto'
                WorkerLoad: []
                 OutputFcn: []
```

```
                Plots: 'none'
        SequenceLength: 'longest'
    SequencePaddingValue: 0
```

****
*
Training a Faster R-CNN Object Detector for the following object classes:

* end_tower
* pyramid
* span
* tower

Step 1 of 4: Training a Region Proposal Network (RPN).
Warning: Invalid bounding boxes from 143 out of 3688 training images were removed.
The following rows in trainingData have invalid
bounding box data:

Invalid Rows
_____

39
55
61
62
82
99
101
108
113
127
145
156
199
222
260
311
316
323
328
399
431
452
472

474
525
538
546
551
575
604
607
646
696
698
739
744
763
766
775
804
815
827
844
893
903
1018
1029
1030
1035
1044
1066
1131
1139
1142
1146
1156
1218
1315
1351
1361
1430
1479
1488
1518
1527
1530
1567

1592
1698
1709
1716
1773
1792
1830
1864
1875
1917
1960
1965
1972
1979
1986
2103
2116
2148
2171
2177
2211
2232
2258
2297
2342
2386
2388
2473
2477
2482
2521
2556
2561
2563
2579
2628
2638
2674
2707
2752
2759
2772
2813
2847

2851
2881
2882
2896
2916
2921
2960
2963
2986
3025
3039
3109
3157
3210
3272
3320
3328
3340
3354
3406
3433
3442
3444
3563
3568
3594
3597
3624
3627
3636
3642
3648

Bounding boxes must be fully contained within their associated image and must have positive width and height.
Training on single GPU.

```
|=========================================================================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
|       |           | (hh:mm:ss)   |    Loss    |  Accuracy  |    RMSE    |      Rate     |
|=========================================================================================|
|     1 |         1 |     00:00:02 |     1.2349 |     51.56% |       1.34 |        0.0050 |
```

	1	50	00:00:08	0.4384	100.00%	1.09	0.0050
	1	100	00:00:14	0.0189	100.00%	0.54	0.0050
	1	150	00:00:20	0.9844	100.00%	2.14	0.0050
	1	200	00:00:25	0.1836	100.00%	0.61	0.0050
	1	250	00:00:31	0.1525	100.00%	0.68	0.0050
	1	300	00:00:37	0.1284	100.00%	0.51	0.0050
	1	350	00:00:43	0.0992	100.00%	1.05	0.0050
	1	400	00:00:49	0.0189	100.00%	0.55	0.0050
	1	450	00:00:54	0.2021	100.00%	0.67	0.0050
	1	500	00:01:00	0.1805	100.00%	1.18	0.0050
	1	550	00:01:06	0.0941	100.00%	0.89	0.0050
	1	600	00:01:12	0.1239	100.00%	0.64	0.0050
	1	650	00:01:18	0.0505	100.00%	0.43	0.0050
	1	700	00:01:24	0.0728	100.00%	0.64	0.0050
	1	750	00:01:30	0.1458	100.00%	0.71	0.0050
	1	800	00:01:35	0.1788	100.00%	0.56	0.0050
	1	850	00:01:41	0.1477	100.00%	0.56	0.0050
	1	900	00:01:47	0.0787	100.00%	0.41	0.0050
	1	950	00:01:53	0.0404	100.00%	0.48	0.0050
	1	1000	00:01:59	0.0558	100.00%	0.37	0.0050
	1	1050	00:02:05	0.0673	100.00%	0.55	0.0050
	1	1100	00:02:11	0.0103	100.00%	0.45	0.0050
	1	1150	00:02:17	0.0045	100.00%	0.27	0.0050
	1	1200	00:02:23	0.0412	100.00%	0.42	0.0050
	1	1250	00:02:29	0.0417	100.00%	0.63	0.0050
	1	1300	00:02:35	0.0122	100.00%	0.20	0.0050
	1	1350	00:02:40	0.0459	100.00%	0.48	0.0050
	1	1400	00:02:46	0.0934	100.00%	0.61	0.0050
	1	1450	00:02:52	0.0058	100.00%	0.43	0.0050
	1	1500	00:02:58	0.0128	100.00%	0.35	0.0050
	1	1550	00:03:04	0.0711	100.00%	0.41	0.0050
	1	1600	00:03:10	0.0507	100.00%	0.36	0.0050
	1	1650	00:03:16	0.0422	100.00%	0.35	0.0050
	1	1700	00:03:22	0.0244	100.00%	0.36	0.0050
	1	1750	00:03:28	0.0156	100.00%	0.27	0.0050
	1	1800	00:03:34	0.0976	100.00%	0.78	0.0050
	1	1850	00:03:39	0.0026	100.00%	0.29	0.0050
	1	1900	00:03:45	0.0437	100.00%	0.31	0.0050
	1	1950	00:03:51	0.0578	100.00%	0.62	0.0050
	1	2000	00:03:57	0.0112	100.00%	0.21	0.0050
	1	2050	00:04:03	0.0450	100.00%	0.35	0.0050
	1	2100	00:04:09	0.0819	100.00%	0.68	0.0050
	1	2150	00:04:15	0.0377	100.00%	0.59	0.0050
	1	2200	00:04:21	0.0650	100.00%	0.53	0.0050

1	2250	00:04:27	0.0582	100.00%	0.66	0.0050
1	2300	00:04:32	0.0517	100.00%	0.36	0.0050
1	2350	00:04:38	0.0491	100.00%	0.35	0.0050
1	2400	00:04:44	0.0115	100.00%	0.24	0.0050
1	2450	00:04:50	0.1720	100.00%	0.75	0.0050
1	2500	00:04:56	0.0566	100.00%	0.39	0.0050
1	2550	00:05:01	0.0078	100.00%	0.40	0.0050
1	2600	00:05:07	0.0423	100.00%	0.26	0.0050
1	2650	00:05:13	0.0042	100.00%	0.16	0.0050
1	2700	00:05:19	0.0234	100.00%	0.30	0.0050
1	2750	00:05:25	0.0229	100.00%	0.23	0.0050
1	2800	00:05:30	0.0435	100.00%	0.37	0.0050
1	2850	00:05:36	0.1025	100.00%	0.58	0.0050
1	2900	00:05:42	0.0480	100.00%	0.34	0.0050
1	2950	00:05:48	0.0110	100.00%	0.23	0.0050
1	3000	00:05:54	0.0185	100.00%	0.23	0.0050
1	3050	00:05:59	0.0463	100.00%	0.46	0.0050
1	3100	00:06:05	0.5121	100.00%	1.72	0.0050
1	3150	00:06:11	0.0571	100.00%	0.54	0.0050
1	3200	00:06:17	0.0388	100.00%	0.45	0.0050
1	3250	00:06:22	0.0533	100.00%	0.44	0.0050
1	3300	00:06:28	0.0421	100.00%	0.33	0.0050
1	3350	00:06:34	0.0231	100.00%	0.32	0.0050
1	3400	00:06:40	0.0829	100.00%	1.37	0.0050
1	3450	00:06:46	0.0679	100.00%	0.61	0.0050
1	3500	00:06:51	0.0488	100.00%	0.28	0.0050
1	3550	00:06:57	0.0118	100.00%	0.22	0.0050
1	3600	00:07:03	0.1608	100.00%	0.56	0.0050
1	3650	00:07:09	0.0245	100.00%	0.20	0.0050
2	3700	00:07:18	0.0253	100.00%	0.37	0.0050
2	3750	00:07:23	0.0314	100.00%	0.30	0.0050
2	3800	00:07:29	0.0122	100.00%	0.24	0.0050
2	3850	00:07:35	0.0716	100.00%	0.50	0.0050
2	3900	00:07:41	0.0137	100.00%	0.23	0.0050
2	3950	00:07:47	0.0023	100.00%	0.22	0.0050
2	4000	00:07:53	0.0275	100.00%	0.36	0.0050
2	4050	00:07:58	0.0220	100.00%	0.23	0.0050
2	4100	00:08:04	0.2807	100.00%	0.56	0.0050
2	4150	00:08:10	0.0030	100.00%	0.14	0.0050
2	4200	00:08:16	0.0835	100.00%	0.50	0.0050
2	4250	00:08:22	0.0226	100.00%	0.21	0.0050
2	4300	00:08:27	0.0025	100.00%	0.23	0.0050
2	4350	00:08:33	0.1059	100.00%	0.55	0.0050
2	4400	00:08:39	0.0133	100.00%	0.18	0.0050

	2	4450	00:08:45	0.0404	100.00%	0.41	0.0050
	2	4500	00:08:51	0.0089	100.00%	0.19	0.0050
	2	4550	00:08:56	0.0406	100.00%	0.26	0.0050
	2	4600	00:09:02	0.0027	100.00%	0.41	0.0050
	2	4650	00:09:08	0.0404	100.00%	0.28	0.0050
	2	4700	00:09:14	0.0456	100.00%	0.28	0.0050
	2	4750	00:09:20	0.0105	100.00%	0.23	0.0050
	2	4800	00:09:25	0.0029	100.00%	0.09	0.0050
	2	4850	00:09:31	0.0078	100.00%	0.26	0.0050
	2	4900	00:09:37	0.0082	100.00%	0.15	0.0050
	2	4950	00:09:43	0.0102	100.00%	0.17	0.0050
	2	5000	00:09:49	0.0129	100.00%	0.45	0.0050
	2	5050	00:09:54	0.0332	100.00%	0.32	0.0050
	2	5100	00:10:00	0.0372	100.00%	0.39	0.0050
	2	5150	00:10:06	0.0007	100.00%	0.14	0.0050
	2	5200	00:10:12	0.0961	100.00%	0.56	0.0050
	2	5250	00:10:17	0.0502	100.00%	0.51	0.0050
	2	5300	00:10:23	0.0203	100.00%	0.22	0.0050
	2	5350	00:10:29	0.0285	100.00%	0.41	0.0050
	2	5400	00:10:35	0.0337	100.00%	0.29	0.0050
	2	5450	00:10:41	0.0246	100.00%	0.29	0.0050
	2	5500	00:10:47	0.0461	100.00%	0.50	0.0050
	2	5550	00:10:52	0.0209	100.00%	0.23	0.0050
	2	5600	00:10:58	0.0319	100.00%	0.48	0.0050
	2	5650	00:11:04	0.0231	100.00%	0.24	0.0050
	2	5700	00:11:10	0.0297	100.00%	0.31	0.0050
	2	5750	00:11:16	0.0069	100.00%	0.33	0.0050
	2	5800	00:11:21	0.0931	100.00%	0.58	0.0050
	2	5850	00:11:27	0.0158	100.00%	0.41	0.0050
	2	5900	00:11:33	0.0135	100.00%	0.19	0.0050
	2	5950	00:11:39	0.0334	100.00%	0.29	0.0050
	2	6000	00:11:44	0.0232	100.00%	0.31	0.0050
	2	6050	00:11:50	0.0205	100.00%	0.21	0.0050
	2	6100	00:11:56	0.0254	100.00%	0.21	0.0050
	2	6150	00:12:02	0.0240	100.00%	0.35	0.0050
	2	6200	00:12:08	0.0132	100.00%	0.23	0.0050
	2	6250	00:12:13	0.0595	100.00%	0.35	0.0050
	2	6300	00:12:19	0.0252	100.00%	0.30	0.0050
	2	6350	00:12:25	0.0126	100.00%	0.23	0.0050
	2	6400	00:12:31	0.0551	100.00%	0.40	0.0050
	2	6450	00:12:37	0.0077	100.00%	0.14	0.0050
	2	6500	00:12:42	0.0210	100.00%	0.28	0.0050
	2	6550	00:12:48	0.0209	100.00%	0.26	0.0050
	2	6600	00:12:54	0.0147	100.00%	0.29	0.0050

```
|     2 |       6650 |     00:13:00 |     0.0082 |     100.00% |       0.18 |       0.0050 |
|     2 |       6700 |     00:13:06 |     0.0267 |     100.00% |       0.22 |       0.0050 |
|     2 |       6750 |     00:13:11 |     0.0086 |     100.00% |       0.16 |       0.0050 |
|     2 |       6800 |     00:13:17 |     0.0058 |     100.00% |       0.17 |       0.0050 |
|     2 |       6850 |     00:13:23 |     0.0120 |     100.00% |       0.19 |       0.0050 |
|     2 |       6900 |     00:13:29 |     0.0586 |     100.00% |       0.47 |       0.0050 |
|     2 |       6950 |     00:13:35 |     0.0055 |     100.00% |       0.12 |       0.0050 |
|     2 |       7000 |     00:13:40 |     0.0091 |     100.00% |       0.22 |       0.0050 |
|     2 |       7050 |     00:13:46 |     0.0273 |     100.00% |       0.46 |       0.0050 |
|     2 |       7100 |     00:13:52 |     0.0121 |     100.00% |       0.18 |       0.0050 |
|     2 |       7150 |     00:13:58 |     0.0058 |     100.00% |       0.16 |       0.0050 |
|     2 |       7200 |     00:14:04 |     0.0368 |     100.00% |       0.23 |       0.0050 |
|     2 |       7250 |     00:14:09 |     0.0297 |     100.00% |       0.28 |       0.0050 |
|     2 |       7300 |     00:14:15 |     0.0279 |     100.00% |       0.37 |       0.0050 |
|     2 |       7348 |     00:14:21 |     0.0187 |     100.00% |       0.25 |       0.0050 |
|=======================================================================================|
```

Step 2 of 4: Training a Fast R-CNN Network using the RPN from step 1.
--> Extracting region proposals from 3674 training images...done.

Training on single GPU.

```
|=======================================================================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
|       |           |  (hh:mm:ss)  |    Loss    |  Accuracy  |    RMSE    |     Rate      |
|=======================================================================================|
|     1 |         1 |     00:00:00 |     2.1942 |      0.00% |       0.37 |       0.0050 |
|     1 |        50 |     00:00:13 |     0.5455 |     89.06% |       0.52 |       0.0050 |
|     1 |       100 |     00:00:26 |     0.7003 |     83.59% |       0.50 |       0.0050 |
|     1 |       150 |     00:00:39 |     0.6194 |     86.72% |       0.48 |       0.0050 |
|     1 |       200 |     00:00:52 |     0.5615 |     85.94% |       0.46 |       0.0050 |
|     1 |       250 |     00:01:04 |     0.6985 |     85.94% |       0.50 |       0.0050 |
|     1 |       300 |     00:01:17 |     0.1527 |     96.09% |       0.58 |       0.0050 |
|     1 |       350 |     00:01:30 |     0.5843 |     89.84% |       0.52 |       0.0050 |
|     1 |       400 |     00:01:43 |     0.1478 |     97.66% |       0.40 |       0.0050 |
|     1 |       450 |     00:01:55 |     0.4044 |     90.38% |       0.53 |       0.0050 |
|     1 |       500 |     00:02:08 |     0.3228 |     92.97% |       0.41 |       0.0050 |
|     1 |       550 |     00:02:21 |     0.2865 |     97.66% |       0.32 |       0.0050 |
|     1 |       600 |     00:02:34 |     0.2557 |     93.16% |       0.48 |       0.0050 |
|     1 |       650 |     00:02:47 |     0.2103 |     95.31% |       0.33 |       0.0050 |
|     1 |       700 |     00:03:00 |     0.3787 |     89.84% |       0.43 |       0.0050 |
```

	1	750	00:03:13	0.3603	92.19%	0.45	0.0050
	1	800	00:03:26	0.2874	91.41%	0.45	0.0050
	1	850	00:03:39	0.2166	94.53%	0.32	0.0050
	1	900	00:03:51	0.2064	94.53%	0.32	0.0050
	1	950	00:04:04	0.2284	96.09%	0.29	0.0050
	1	1000	00:04:18	0.1408	97.66%	0.25	0.0050
	1	1050	00:04:30	0.2315	93.75%	0.27	0.0050
	1	1100	00:04:43	0.2026	92.97%	0.30	0.0050
	1	1150	00:04:56	0.1959	96.09%	0.36	0.0050
	1	1200	00:05:09	0.4171	89.06%	0.40	0.0050
	1	1250	00:05:22	0.2345	95.31%	0.28	0.0050
	1	1300	00:05:35	0.1849	94.53%	0.27	0.0050
	1	1350	00:05:48	0.1774	96.09%	0.32	0.0050
	1	1400	00:06:01	0.1883	94.53%	0.34	0.0050
	1	1450	00:06:14	0.2603	92.97%	0.27	0.0050
	1	1500	00:06:27	0.2524	93.75%	0.30	0.0050
	1	1550	00:06:40	0.1213	96.88%	0.25	0.0050
	1	1600	00:06:53	0.1842	95.31%	0.30	0.0050
	1	1650	00:07:06	0.1759	94.83%	0.36	0.0050
	1	1700	00:07:18	0.3942	92.19%	0.41	0.0050
	1	1750	00:07:31	0.1626	97.66%	0.29	0.0050
	1	1800	00:07:44	0.0945	98.06%	0.39	0.0050
	1	1850	00:07:57	0.1850	97.66%	0.35	0.0050
	1	1900	00:08:10	0.1374	95.31%	0.20	0.0050
	1	1950	00:08:23	0.1204	98.44%	0.26	0.0050
	1	2000	00:08:35	0.2484	91.58%	0.26	0.0050
	1	2050	00:08:49	0.1048	99.22%	0.23	0.0050
	1	2100	00:09:02	0.5840	89.06%	0.37	0.0050
	1	2150	00:09:15	0.0661	97.66%	0.16	0.0050
	1	2200	00:09:27	0.0668	100.00%	0.50	0.0050
	1	2250	00:09:40	0.2263	94.53%	0.22	0.0050
	1	2300	00:09:53	0.2199	94.74%	0.42	0.0050
	1	2350	00:10:06	0.1563	94.53%	0.24	0.0050
	1	2400	00:10:19	0.0979	98.44%	0.24	0.0050
	1	2450	00:10:32	0.1708	96.09%	0.25	0.0050
	1	2500	00:10:45	0.1309	96.09%	0.20	0.0050
	1	2550	00:10:58	0.0875	97.66%	0.18	0.0050
	1	2600	00:11:11	0.0774	97.66%	0.20	0.0050
	1	2650	00:11:24	0.1037	96.88%	0.20	0.0050
	1	2700	00:11:37	0.2142	94.21%	0.28	0.0050
	1	2750	00:11:50	0.0719	98.44%	0.19	0.0050
	1	2800	00:12:03	0.3317	92.19%	0.38	0.0050
	1	2850	00:12:16	0.1402	96.81%	0.42	0.0050
	1	2900	00:12:29	0.1100	96.88%	0.23	0.0050

1	2950	00:12:42	0.0988	96.88%	0.22	0.0050
1	3000	00:12:55	0.1094	98.44%	0.26	0.0050
1	3050	00:13:08	0.0526	99.22%	0.32	0.0050
1	3100	00:13:21	0.1174	97.66%	0.40	0.0050
1	3150	00:13:34	0.1505	96.88%	0.28	0.0050
1	3200	00:13:47	0.0201	100.00%	0.24	0.0050
1	3250	00:14:00	0.2310	94.53%	0.26	0.0050
1	3300	00:14:14	0.1280	97.66%	0.27	0.0050
1	3350	00:14:26	0.1744	94.53%	0.21	0.0050
1	3400	00:14:39	0.1453	96.88%	0.23	0.0050
1	3450	00:14:52	0.2402	93.75%	0.26	0.0050
1	3500	00:15:05	0.0459	98.91%	0.17	0.0050
1	3550	00:15:18	0.0559	100.00%	0.21	0.0050
1	3600	00:15:31	0.0960	98.44%	0.26	0.0050
1	3650	00:15:43	0.0423	100.00%	0.28	0.0050
2	3700	00:16:00	0.1062	97.66%	0.19	0.0050
2	3750	00:16:12	0.1444	96.09%	0.21	0.0050
2	3800	00:16:25	0.2578	96.88%	0.44	0.0050
2	3850	00:16:38	0.0905	98.44%	0.22	0.0050
2	3900	00:16:51	0.1811	93.75%	0.19	0.0050
2	3950	00:17:04	0.1853	94.53%	0.23	0.0050
2	4000	00:17:16	0.0431	99.22%	0.28	0.0050
2	4050	00:17:29	0.0399	98.44%	0.15	0.0050
2	4100	00:17:42	0.0620	97.66%	0.16	0.0050
2	4150	00:17:55	0.1807	93.75%	0.30	0.0050
2	4200	00:18:08	0.3190	92.97%	0.24	0.0050
2	4250	00:18:20	0.1161	96.88%	0.28	0.0050
2	4300	00:18:33	0.1033	96.70%	0.27	0.0050
2	4350	00:18:46	0.1572	96.09%	0.28	0.0050
2	4400	00:18:59	0.2067	96.88%	0.36	0.0050
2	4450	00:19:12	0.1092	97.66%	0.27	0.0050
2	4500	00:19:25	0.1781	94.53%	0.27	0.0050
2	4550	00:19:38	0.1294	96.88%	0.22	0.0050
2	4600	00:19:51	0.1313	95.31%	0.30	0.0050
2	4650	00:20:04	0.0831	97.66%	0.29	0.0050
2	4700	00:20:17	0.0624	97.30%	0.26	0.0050
2	4750	00:20:30	0.0397	99.15%	0.18	0.0050
2	4800	00:20:43	0.0689	98.28%	0.30	0.0050
2	4850	00:20:56	0.1399	97.66%	0.26	0.0050
2	4900	00:21:09	0.0483	98.82%	0.31	0.0050
2	4950	00:21:22	0.1046	96.88%	0.21	0.0050
2	5000	00:21:35	0.2406	94.53%	0.30	0.0050
2	5050	00:21:48	0.0581	100.00%	0.30	0.0050
2	5100	00:22:01	0.2954	93.75%	0.34	0.0050

2	5150	00:22:14	0.0943	98.57%	0.24	0.0050
2	5200	00:22:27	0.1456	96.67%	0.20	0.0050
2	5250	00:22:40	0.0963	98.44%	0.20	0.0050
2	5300	00:22:52	0.0962	99.22%	0.20	0.0050
2	5350	00:23:05	0.0327	98.44%	0.21	0.0050
2	5400	00:23:18	0.1401	96.09%	0.20	0.0050
2	5450	00:23:31	0.1569	96.88%	0.25	0.0050
2	5500	00:23:44	0.0935	98.44%	0.18	0.0050
2	5550	00:23:57	0.0858	98.44%	0.22	0.0050
2	5600	00:24:09	0.0578	98.44%	0.16	0.0050
2	5650	00:24:22	0.1674	95.31%	0.26	0.0050
2	5700	00:24:35	0.0609	98.44%	0.26	0.0050
2	5750	00:24:48	0.0489	97.66%	0.24	0.0050
2	5800	00:25:01	0.1556	95.56%	0.22	0.0050
2	5850	00:25:14	0.0894	98.44%	0.33	0.0050
2	5900	00:25:27	0.0399	100.00%	0.24	0.0050
2	5950	00:25:40	0.1169	97.66%	0.21	0.0050
2	6000	00:25:53	0.1732	93.75%	0.20	0.0050
2	6050	00:26:06	0.0997	96.88%	0.16	0.0050
2	6100	00:26:19	0.0563	100.00%	0.17	0.0050
2	6150	00:26:32	0.1805	95.31%	0.22	0.0050
2	6200	00:26:45	0.1075	96.51%	0.34	0.0050
2	6250	00:26:58	0.1217	98.44%	0.19	0.0050
2	6300	00:27:12	0.1718	96.09%	0.26	0.0050
2	6350	00:27:25	0.1219	97.66%	0.23	0.0050
2	6400	00:27:38	0.0787	99.22%	0.18	0.0050
2	6450	00:27:51	0.1024	98.44%	0.27	0.0050
2	6500	00:28:04	0.0294	100.00%	0.34	0.0050
2	6550	00:28:17	0.1316	95.31%	0.17	0.0050
2	6600	00:28:30	0.1200	96.09%	0.17	0.0050
2	6650	00:28:43	0.1656	95.31%	0.25	0.0050
2	6700	00:28:56	0.0623	98.44%	0.18	0.0050
2	6750	00:29:09	0.1640	95.31%	0.26	0.0050
2	6800	00:29:22	0.0789	98.44%	0.17	0.0050
2	6850	00:29:36	0.1039	97.66%	0.18	0.0050
2	6900	00:29:49	0.0954	97.66%	0.19	0.0050
2	6950	00:30:02	0.0396	98.96%	0.20	0.0050
2	7000	00:30:16	0.1187	96.09%	0.14	0.0050
2	7050	00:30:29	0.3400	92.97%	0.33	0.0050
2	7100	00:30:42	0.0948	96.09%	0.19	0.0050
2	7150	00:30:55	0.0812	97.66%	0.20	0.0050
2	7200	00:31:08	0.1291	98.44%	0.26	0.0050
2	7250	00:31:21	0.0705	99.22%	0.16	0.0050
2	7300	00:31:34	0.1372	96.88%	0.21	0.0050

| 2 | 7330 | 00:31:42 | 0.1820 | 94.53% | 0.22 | 0.0050 |
|===|

Step 3 of 4: Re-training RPN using weight sharing with Fast R-CNN.
Training on single GPU.
|===|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base Learning |
| | | (hh:mm:ss) | Loss | Accuracy | RMSE | Rate |
|===|
1	1	00:00:00	2.1770	100.00%	3.86	0.0050
1	50	00:00:03	0.1195	100.00%	0.75	0.0050
1	100	00:00:07	0.0270	100.00%	0.41	0.0050
1	150	00:00:10	0.1698	100.00%	0.65	0.0050
1	200	00:00:14	0.1117	100.00%	0.50	0.0050
1	250	00:00:17	0.0578	100.00%	0.52	0.0050
1	300	00:00:21	0.0572	100.00%	0.43	0.0050
1	350	00:00:25	0.0564	100.00%	0.40	0.0050
1	400	00:00:28	0.1285	100.00%	0.41	0.0050
1	450	00:00:32	0.1136	100.00%	0.53	0.0050
1	500	00:00:35	0.0470	100.00%	0.43	0.0050
1	550	00:00:39	0.0097	100.00%	0.37	0.0050
1	600	00:00:42	0.0224	100.00%	0.25	0.0050
1	650	00:00:46	0.0381	100.00%	0.39	0.0050
1	700	00:00:49	0.0160	100.00%	0.24	0.0050
1	750	00:00:53	0.0093	100.00%	0.31	0.0050
1	800	00:00:57	0.0781	100.00%	0.61	0.0050
1	850	00:01:00	0.0554	100.00%	0.38	0.0050
1	900	00:01:04	0.0842	100.00%	0.50	0.0050
1	950	00:01:07	0.1288	100.00%	0.56	0.0050
1	1000	00:01:11	0.0256	100.00%	0.75	0.0050
1	1050	00:01:14	0.0170	100.00%	0.28	0.0050
1	1100	00:01:18	0.0294	100.00%	0.50	0.0050
1	1150	00:01:21	0.0220	100.00%	0.30	0.0050
1	1200	00:01:25	0.0561	100.00%	0.56	0.0050
1	1250	00:01:29	0.1359	100.00%	0.66	0.0050
1	1300	00:01:32	0.0222	100.00%	0.32	0.0050
1	1350	00:01:36	0.0987	100.00%	0.45	0.0050
1	1400	00:01:39	0.0514	100.00%	0.40	0.0050
1	1450	00:01:43	0.0645	100.00%	0.48	0.0050
1	1500	00:01:46	0.0109	100.00%	0.21	0.0050

1	1550	00:01:50	0.0286	100.00%	0.43	0.0050
1	1600	00:01:54	0.0228	100.00%	0.71	0.0050
1	1650	00:01:57	0.0489	100.00%	1.11	0.0050
1	1700	00:02:01	0.0577	100.00%	0.46	0.0050
1	1750	00:02:04	0.0222	100.00%	0.30	0.0050
1	1800	00:02:08	0.0136	100.00%	0.24	0.0050
1	1850	00:02:11	0.1187	100.00%	0.48	0.0050
1	1900	00:02:15	0.0570	100.00%	0.37	0.0050
1	1950	00:02:18	0.0024	100.00%	0.35	0.0050
1	2000	00:02:22	0.0049	100.00%	0.22	0.0050
1	2050	00:02:26	0.0204	100.00%	0.25	0.0050
1	2100	00:02:29	0.0831	100.00%	0.58	0.0050
1	2150	00:02:33	0.0744	100.00%	1.48	0.0050
1	2200	00:02:36	0.1439	100.00%	0.60	0.0050
1	2250	00:02:40	0.0156	100.00%	0.27	0.0050
1	2300	00:02:43	0.0342	100.00%	0.37	0.0050
1	2350	00:02:47	0.0870	100.00%	0.67	0.0050
1	2400	00:02:51	0.0494	100.00%	0.40	0.0050
1	2450	00:02:54	0.0714	100.00%	0.44	0.0050
1	2500	00:02:58	0.0841	100.00%	1.01	0.0050
1	2550	00:03:01	0.0509	100.00%	0.44	0.0050
1	2600	00:03:05	0.0636	100.00%	0.40	0.0050
1	2650	00:03:08	0.0704	100.00%	0.65	0.0050
1	2700	00:03:12	0.0104	100.00%	0.23	0.0050
1	2750	00:03:15	0.0839	100.00%	0.42	0.0050
1	2800	00:03:19	0.1825	100.00%	0.50	0.0050
1	2850	00:03:23	0.0466	100.00%	0.45	0.0050
1	2900	00:03:26	0.0668	100.00%	0.35	0.0050
1	2950	00:03:30	0.0540	100.00%	0.40	0.0050
1	3000	00:03:33	0.0307	100.00%	0.37	0.0050
1	3050	00:03:37	0.6474	100.00%	1.66	0.0050
1	3100	00:03:40	0.1222	100.00%	0.66	0.0050
1	3150	00:03:44	0.0341	100.00%	0.37	0.0050
1	3200	00:03:48	0.0328	100.00%	0.80	0.0050
1	3250	00:03:51	0.0247	100.00%	0.33	0.0050
1	3300	00:03:55	0.0478	100.00%	0.52	0.0050
1	3350	00:03:58	0.0108	100.00%	0.36	0.0050
1	3400	00:04:02	0.0520	100.00%	0.38	0.0050
1	3450	00:04:05	0.0681	100.00%	0.42	0.0050
1	3500	00:04:09	0.0277	100.00%	0.32	0.0050
1	3550	00:04:12	0.0561	100.00%	0.52	0.0050
1	3600	00:04:16	0.0096	100.00%	0.22	0.0050
1	3650	00:04:20	0.0103	100.00%	0.38	0.0050
2	3700	00:04:26	0.0192	100.00%	0.26	0.0050

	2	3750	00:04:29	0.0212	100.00%	0.32	0.0050
	2	3800	00:04:33	0.0531	100.00%	0.36	0.0050
	2	3850	00:04:36	0.0688	100.00%	0.61	0.0050
	2	3900	00:04:40	0.0932	100.00%	0.49	0.0050
	2	3950	00:04:43	0.0627	100.00%	0.37	0.0050
	2	4000	00:04:47	0.0628	100.00%	0.37	0.0050
	2	4050	00:04:50	0.0461	100.00%	0.37	0.0050
	2	4100	00:04:54	0.0525	100.00%	0.47	0.0050
	2	4150	00:04:58	0.0209	100.00%	0.47	0.0050
	2	4200	00:05:01	0.0399	100.00%	0.43	0.0050
	2	4250	00:05:05	0.0385	100.00%	0.28	0.0050
	2	4300	00:05:08	0.1238	100.00%	0.55	0.0050
	2	4350	00:05:12	0.0420	100.00%	0.30	0.0050
	2	4400	00:05:16	0.0313	100.00%	0.57	0.0050
	2	4450	00:05:19	0.1315	100.00%	0.57	0.0050
	2	4500	00:05:23	0.0023	100.00%	0.24	0.0050
	2	4550	00:05:26	0.0537	100.00%	0.37	0.0050
	2	4600	00:05:30	0.0881	100.00%	0.49	0.0050
	2	4650	00:05:33	0.1195	100.00%	1.05	0.0050
	2	4700	00:05:37	0.0435	100.00%	0.33	0.0050
	2	4750	00:05:41	0.0756	100.00%	0.65	0.0050
	2	4800	00:05:44	0.0458	100.00%	0.34	0.0050
	2	4850	00:05:48	0.0512	100.00%	0.73	0.0050
	2	4900	00:05:51	0.1453	100.00%	0.99	0.0050
	2	4950	00:05:55	0.0482	100.00%	0.57	0.0050
	2	5000	00:05:58	0.2906	100.00%	0.89	0.0050
	2	5050	00:06:02	0.0277	100.00%	0.34	0.0050
	2	5100	00:06:06	0.0089	100.00%	0.16	0.0050
	2	5150	00:06:09	0.0164	100.00%	0.22	0.0050
	2	5200	00:06:13	0.0128	100.00%	0.25	0.0050
	2	5250	00:06:16	0.0389	100.00%	0.45	0.0050
	2	5300	00:06:20	0.0515	100.00%	0.52	0.0050
	2	5350	00:06:23	0.0109	100.00%	0.27	0.0050
	2	5400	00:06:27	0.0801	100.00%	0.46	0.0050
	2	5450	00:06:31	0.4282	100.00%	1.41	0.0050
	2	5500	00:06:34	0.0099	100.00%	0.23	0.0050
	2	5550	00:06:38	0.0335	100.00%	0.26	0.0050
	2	5600	00:06:41	0.0899	100.00%	0.47	0.0050
	2	5650	00:06:45	0.0137	100.00%	0.32	0.0050
	2	5700	00:06:48	0.0360	100.00%	0.46	0.0050
	2	5750	00:06:52	0.0600	100.00%	0.50	0.0050
	2	5800	00:06:55	0.0035	100.00%	0.27	0.0050
	2	5850	00:06:59	0.0142	100.00%	0.25	0.0050
	2	5900	00:07:03	0.0144	100.00%	0.28	0.0050

```
|     2 |       5950 |     00:07:06 |     0.0116 |     100.00% |       0.29 |       0.0050 |
|     2 |       6000 |     00:07:10 |     0.0868 |     100.00% |       0.77 |       0.0050 |
|     2 |       6050 |     00:07:13 |     0.0479 |     100.00% |       0.44 |       0.0050 |
|     2 |       6100 |     00:07:17 |     0.0523 |     100.00% |       0.58 |       0.0050 |
|     2 |       6150 |     00:07:20 |     0.7442 |     100.00% |       0.94 |       0.0050 |
|     2 |       6200 |     00:07:24 |     0.0337 |     100.00% |       0.36 |       0.0050 |
|     2 |       6250 |     00:07:28 |     0.0131 |     100.00% |       0.30 |       0.0050 |
|     2 |       6300 |     00:07:31 |     0.0322 |     100.00% |       0.34 |       0.0050 |
|     2 |       6350 |     00:07:35 |     0.0119 |     100.00% |       0.27 |       0.0050 |
|     2 |       6400 |     00:07:38 |     0.0267 |     100.00% |       0.24 |       0.0050 |
|     2 |       6450 |     00:07:42 |     0.0375 |     100.00% |       0.56 |       0.0050 |
|     2 |       6500 |     00:07:45 |     0.0385 |     100.00% |       0.39 |       0.0050 |
|     2 |       6550 |     00:07:49 |     0.0817 |     100.00% |       0.46 |       0.0050 |
|     2 |       6600 |     00:07:52 |     0.0243 |     100.00% |       0.41 |       0.0050 |
|     2 |       6650 |     00:07:56 |     0.0382 |     100.00% |       0.39 |       0.0050 |
|     2 |       6700 |     00:08:00 |     0.0331 |     100.00% |       0.34 |       0.0050 |
|     2 |       6750 |     00:08:03 |     0.0223 |     100.00% |       0.57 |       0.0050 |
|     2 |       6800 |     00:08:07 |     0.0565 |     100.00% |       0.36 |       0.0050 |
|     2 |       6850 |     00:08:10 |     0.0691 |     100.00% |       0.58 |       0.0050 |
|     2 |       6900 |     00:08:14 |     0.0332 |     100.00% |       0.33 |       0.0050 |
|     2 |       6950 |     00:08:17 |     0.1619 |     100.00% |       0.62 |       0.0050 |
|     2 |       7000 |     00:08:21 |     0.0906 |     100.00% |       0.41 |       0.0050 |
|     2 |       7050 |     00:08:24 |     0.0235 |     100.00% |       0.39 |       0.0050 |
|     2 |       7100 |     00:08:28 |     0.0263 |     100.00% |       0.30 |       0.0050 |
|     2 |       7150 |     00:08:32 |     0.0160 |     100.00% |       0.27 |       0.0050 |
|     2 |       7200 |     00:08:35 |     0.0130 |     100.00% |       0.53 |       0.0050 |
|     2 |       7250 |     00:08:39 |     0.1554 |     100.00% |       0.65 |       0.0050 |
|     2 |       7300 |     00:08:42 |     0.0331 |     100.00% |       0.27 |       0.0050 |
|     2 |       7348 |     00:08:46 |     0.0217 |     100.00% |       0.35 |       0.0050 |
|=======================================================================
=======================================|
```

Step 4 of 4: Re-training Fast R-CNN using updated RPN.
--> Extracting region proposals from 3674 training images...done.

Training on single GPU.

```
|=======================================================================
=======================================|
| Epoch | Iteration | Time Elapsed | Mini-batch | Mini-batch | Mini-batch | Base
Learning |
|       |           | (hh:mm:ss) |    Loss    | Accuracy |    RMSE    |      Rate      |
|=======================================================================
=======================================|
|     1 |         1 |     00:00:00 |     0.1084 |     96.30% |       0.31 |       0.0050 |
```

1	50	00:00:10	0.3983	79.17%	0.97	0.0050
1	100	00:00:20	0.1143	96.09%	0.22	0.0050
1	150	00:00:30	0.1340	96.88%	0.23	0.0050
1	200	00:00:40	0.1632	97.70%	0.37	0.0050
1	250	00:00:51	0.0579	98.28%	0.20	0.0050
1	300	00:01:01	0.0863	98.44%	0.21	0.0050
1	350	00:01:11	0.1928	96.88%	0.32	0.0050
1	400	00:01:21	0.2507	94.53%	0.30	0.0050
1	450	00:01:32	0.3113	91.41%	0.30	0.0050
1	500	00:01:42	0.1999	97.32%	0.41	0.0050
1	550	00:01:53	0.0861	97.66%	0.18	0.0050
1	600	00:02:03	0.1939	94.53%	0.31	0.0050
1	650	00:02:13	0.0990	98.77%	0.36	0.0050
1	700	00:02:23	0.0727	98.44%	0.25	0.0050
1	750	00:02:33	0.0523	97.66%	0.16	0.0050
1	800	00:02:43	0.3887	89.84%	0.35	0.0050
1	850	00:02:54	0.0562	99.22%	0.16	0.0050
1	900	00:03:04	0.1458	100.00%	0.24	0.0050
1	950	00:03:15	0.1734	96.09%	0.21	0.0050
1	1000	00:03:25	0.0537	99.22%	0.18	0.0050
1	1050	00:03:35	0.0797	96.88%	0.17	0.0050
1	1100	00:03:45	0.1300	96.09%	0.19	0.0050
1	1150	00:03:55	0.1516	96.88%	0.24	0.0050
1	1200	00:04:06	0.0372	100.00%	0.20	0.0050
1	1250	00:04:16	0.0851	98.44%	0.19	0.0050
1	1300	00:04:26	0.2603	94.53%	0.29	0.0050
1	1350	00:04:37	0.0950	97.94%	0.28	0.0050
1	1400	00:04:47	0.0619	99.05%	0.22	0.0050
1	1450	00:04:57	0.1629	96.88%	0.23	0.0050
1	1500	00:05:08	0.0337	97.66%	0.20	0.0050
1	1550	00:05:18	0.0910	97.14%	0.23	0.0050
1	1600	00:05:28	0.0626	100.00%	0.19	0.0050
1	1650	00:05:38	0.1079	94.53%	0.18	0.0050
1	1700	00:05:49	0.1106	96.88%	0.18	0.0050
1	1750	00:06:00	0.0824	97.66%	0.19	0.0050
1	1800	00:06:10	0.0910	95.88%	0.18	0.0050
1	1850	00:06:20	0.0639	97.66%	0.15	0.0050
1	1900	00:06:30	0.1214	97.66%	0.25	0.0050
1	1950	00:06:41	0.1118	95.59%	0.23	0.0050
1	2000	00:06:51	0.1406	96.88%	0.18	0.0050
1	2050	00:07:01	0.2906	92.97%	0.29	0.0050
1	2100	00:07:11	0.1259	98.37%	0.30	0.0050
1	2150	00:07:22	0.0652	98.44%	0.36	0.0050
1	2200	00:07:32	0.1152	97.66%	0.23	0.0050

1	2250	00:07:42	0.3570	91.41%	0.29	0.0050
1	2300	00:07:52	0.0807	98.44%	0.19	0.0050
1	2350	00:08:02	0.1162	95.31%	0.18	0.0050
1	2400	00:08:12	0.1225	96.88%	0.21	0.0050
1	2450	00:08:23	0.0755	97.66%	0.18	0.0050
1	2500	00:08:33	0.1407	96.09%	0.24	0.0050
1	2550	00:08:43	0.2317	96.88%	0.34	0.0050
1	2600	00:08:53	0.1937	92.97%	0.26	0.0050
1	2650	00:09:03	0.0845	97.56%	0.32	0.0050
1	2700	00:09:13	0.1349	97.66%	0.21	0.0050
1	2750	00:09:23	0.0897	96.09%	0.15	0.0050
1	2800	00:09:34	0.2610	95.31%	0.36	0.0050
1	2850	00:09:44	0.0500	98.44%	0.16	0.0050
1	2900	00:09:54	0.1528	96.09%	0.26	0.0050
1	2950	00:10:04	0.0920	97.66%	0.17	0.0050
1	3000	00:10:15	0.1996	95.31%	0.27	0.0050
1	3050	00:10:25	0.1602	95.31%	0.21	0.0050
1	3100	00:10:35	0.1267	96.09%	0.19	0.0050
1	3150	00:10:46	0.0858	96.88%	0.23	0.0050
1	3200	00:10:56	0.1734	95.31%	0.24	0.0050
1	3250	00:11:06	0.0921	99.22%	0.21	0.0050
1	3300	00:11:16	0.0859	96.88%	0.22	0.0050
1	3350	00:11:26	0.1762	94.53%	0.27	0.0050
1	3400	00:11:36	0.0716	98.80%	0.19	0.0050
1	3450	00:11:46	0.2367	95.31%	0.21	0.0050
1	3500	00:11:56	0.0746	97.52%	0.19	0.0050
1	3550	00:12:07	0.1649	94.53%	0.18	0.0050
1	3600	00:12:17	0.0706	98.44%	0.17	0.0050
1	3650	00:12:27	0.1098	96.88%	0.18	0.0050
2	3700	00:12:41	0.0640	99.22%	0.13	0.0050
2	3750	00:12:51	0.0413	100.00%	0.14	0.0050
2	3800	00:13:01	0.0748	96.88%	0.15	0.0050
2	3850	00:13:11	0.1156	96.09%	0.20	0.0050
2	3900	00:13:22	0.3913	89.84%	0.32	0.0050
2	3950	00:13:32	0.0645	99.22%	0.17	0.0050
2	4000	00:13:42	0.0822	98.44%	0.17	0.0050
2	4050	00:13:53	0.0597	98.44%	0.17	0.0050
2	4100	00:14:03	0.0269	100.00%	0.13	0.0050
2	4150	00:14:14	0.0491	99.22%	0.15	0.0050
2	4200	00:14:24	0.0952	96.88%	0.19	0.0050
2	4250	00:14:34	0.0805	98.44%	0.17	0.0050
2	4300	00:14:44	0.1091	96.88%	0.18	0.0050
2	4350	00:14:55	0.0739	98.44%	0.20	0.0050
2	4400	00:15:05	0.0337	100.00%	0.20	0.0050

2	4450	00:15:15	0.2852	92.97%	0.28	0.0050
2	4500	00:15:25	0.1971	94.53%	0.28	0.0050
2	4550	00:15:36	0.0739	98.44%	0.20	0.0050
2	4600	00:15:46	0.0753	98.44%	0.16	0.0050
2	4650	00:15:56	0.1167	95.31%	0.19	0.0050
2	4700	00:16:07	0.1833	94.53%	0.23	0.0050
2	4750	00:16:17	0.1365	96.88%	0.31	0.0050
2	4800	00:16:27	0.2083	94.53%	0.23	0.0050
2	4850	00:16:37	0.1105	97.66%	0.17	0.0050
2	4900	00:16:48	0.0864	96.88%	0.13	0.0050
2	4950	00:16:58	0.3063	92.19%	0.30	0.0050
2	5000	00:17:08	0.1612	96.88%	0.21	0.0050
2	5050	00:17:19	0.2151	94.53%	0.21	0.0050
2	5100	00:17:29	0.0839	96.88%	0.36	0.0050
2	5150	00:17:39	0.0543	98.44%	0.15	0.0050
2	5200	00:17:50	0.0196	100.00%	0.13	0.0050
2	5250	00:18:00	0.0755	98.44%	0.16	0.0050
2	5300	00:18:10	0.0379	100.00%	0.14	0.0050
2	5350	00:18:21	0.0538	100.00%	0.29	0.0050
2	5400	00:18:31	0.1053	96.88%	0.19	0.0050
2	5450	00:18:41	0.0801	99.07%	0.30	0.0050
2	5500	00:18:52	0.0489	100.00%	0.24	0.0050
2	5550	00:19:02	0.0706	96.88%	0.15	0.0050
2	5600	00:19:12	0.1564	97.66%	0.31	0.0050
2	5650	00:19:22	0.1613	97.66%	0.19	0.0050
2	5700	00:19:33	0.1070	97.66%	0.17	0.0050
2	5750	00:19:43	0.1335	97.87%	0.37	0.0050
2	5800	00:19:53	0.1165	96.88%	0.23	0.0050
2	5850	00:20:04	0.0945	96.88%	0.20	0.0050
2	5900	00:20:14	0.0936	97.66%	0.18	0.0050
2	5950	00:20:24	0.0669	97.66%	0.18	0.0050
2	6000	00:20:34	0.0663	98.44%	0.13	0.0050
2	6050	00:20:44	0.2671	92.97%	0.30	0.0050
2	6100	00:20:54	0.0979	97.66%	0.18	0.0050
2	6150	00:21:04	0.0987	97.66%	0.28	0.0050
2	6200	00:21:14	0.1157	96.88%	0.18	0.0050
2	6250	00:21:25	0.1950	97.66%	0.32	0.0050
2	6300	00:21:35	0.0768	97.66%	0.20	0.0050
2	6350	00:21:45	0.0804	97.66%	0.24	0.0050
2	6400	00:21:55	0.1543	94.53%	0.22	0.0050
2	6450	00:22:05	0.1118	95.31%	0.19	0.0050
2	6500	00:22:15	0.0967	97.66%	0.19	0.0050
2	6550	00:22:26	0.1044	96.09%	0.20	0.0050
2	6600	00:22:36	0.0423	99.22%	0.15	0.0050

2	6650	00:22:46	0.1042	96.88%	0.14	0.0050
2	6700	00:22:56	0.1070	98.97%	0.24	0.0050
2	6750	00:23:07	0.0543	99.22%	0.16	0.0050
2	6800	00:23:17	0.0571	98.13%	0.18	0.0050
2	6850	00:23:27	0.1729	95.31%	0.23	0.0050
2	6900	00:23:37	0.0331	99.22%	0.16	0.0050
2	6950	00:23:48	0.0889	98.44%	0.17	0.0050
2	7000	00:23:58	0.1037	95.24%	0.16	0.0050
2	7050	00:24:08	0.1070	97.66%	0.22	0.0050
2	7100	00:24:18	0.1114	94.53%	0.18	0.0050
2	7150	00:24:28	0.0332	100.00%	0.25	0.0050
2	7200	00:24:38	0.1217	96.09%	0.18	0.0050
2	7250	00:24:48	0.1275	97.66%	0.23	0.0050
2	7300	00:24:59	0.0750	96.88%	0.13	0.0050
3	7350	00:25:12	0.0417	97.96%	0.12	0.0050
3	7400	00:25:22	0.1276	96.09%	0.23	0.0050
3	7450	00:25:33	0.1187	97.66%	0.23	0.0050
3	7500	00:25:43	0.0595	98.44%	0.20	0.0050
3	7550	00:25:53	0.0745	98.44%	0.17	0.0050
3	7600	00:26:04	0.1040	95.31%	0.13	0.0050
3	7650	00:26:14	0.1008	97.70%	0.28	0.0050
3	7700	00:26:24	0.1734	93.75%	0.15	0.0050
3	7750	00:26:34	0.0522	100.00%	0.15	0.0050
3	7800	00:26:45	0.2330	92.97%	0.29	0.0050
3	7850	00:26:55	0.0496	99.22%	0.14	0.0050
3	7900	00:27:05	0.0956	97.66%	0.16	0.0050
3	7950	00:27:15	0.0820	98.46%	0.27	0.0050
3	8000	00:27:26	0.0660	98.10%	0.16	0.0050
3	8050	00:27:36	0.1123	98.44%	0.19	0.0050
3	8100	00:27:46	0.0541	100.00%	0.21	0.0050
3	8150	00:27:57	0.1026	96.09%	0.17	0.0050
3	8200	00:28:07	0.0778	98.44%	0.21	0.0050
3	8250	00:28:17	0.1023	96.25%	0.22	0.0050
3	8300	00:28:27	0.1394	96.09%	0.20	0.0050
3	8350	00:28:38	0.1324	95.31%	0.16	0.0050
3	8400	00:28:48	0.0827	95.31%	0.15	0.0050
3	8450	00:28:58	0.1425	96.09%	0.25	0.0050
3	8500	00:29:08	0.3084	92.19%	0.31	0.0050
3	8550	00:29:19	0.0689	99.03%	0.27	0.0050
3	8600	00:29:29	0.0783	96.09%	0.15	0.0050
3	8650	00:29:39	0.1003	97.66%	0.18	0.0050
3	8700	00:29:50	0.0863	96.88%	0.17	0.0050
3	8750	00:30:00	0.1329	96.09%	0.16	0.0050
3	8800	00:30:10	0.0614	99.22%	0.16	0.0050

3	8850	00:30:21	0.0981	98.77%	0.27	0.0050
3	8900	00:30:31	0.1037	96.09%	0.20	0.0050
3	8950	00:30:41	0.1056	96.88%	0.17	0.0050
3	9000	00:30:52	0.1330	96.09%	0.17	0.0050
3	9050	00:31:02	0.1641	94.53%	0.20	0.0050
3	9100	00:31:12	0.0530	97.75%	0.21	0.0050
3	9150	00:31:23	0.0967	97.66%	0.16	0.0050
3	9200	00:31:33	0.1435	95.31%	0.21	0.0050
3	9250	00:31:43	0.0510	98.44%	0.17	0.0050
3	9300	00:31:53	0.1272	96.09%	0.17	0.0050
3	9350	00:32:04	0.0924	98.44%	0.20	0.0050
3	9400	00:32:14	0.0632	97.66%	0.14	0.0050
3	9450	00:32:24	0.0685	99.22%	0.16	0.0050
3	9500	00:32:35	0.0734	96.88%	0.17	0.0050
3	9550	00:32:45	0.0255	100.00%	0.21	0.0050
3	9600	00:32:55	0.0562	97.96%	0.14	0.0050
3	9650	00:33:05	0.1928	91.15%	0.20	0.0050
3	9700	00:33:15	0.0629	99.22%	0.17	0.0050
3	9750	00:33:25	0.1940	93.75%	0.20	0.0050
3	9800	00:33:35	0.0795	99.22%	0.20	0.0050
3	9850	00:33:46	0.1094	94.53%	0.15	0.0050
3	9900	00:33:56	0.1037	96.88%	0.18	0.0050
3	9950	00:34:06	0.0497	99.22%	0.12	0.0050
3	10000	00:34:16	0.0721	97.66%	0.17	0.0050
3	10050	00:34:26	0.0346	98.89%	0.24	0.0050
3	10100	00:34:36	0.1197	94.53%	0.19	0.0050
3	10150	00:34:47	0.0988	96.88%	0.15	0.0050
3	10200	00:34:57	0.0949	96.88%	0.12	0.0050
3	10250	00:35:07	0.0881	97.66%	0.16	0.0050
3	10300	00:35:18	0.0792	98.70%	0.26	0.0050
3	10350	00:35:28	0.1643	95.31%	0.17	0.0050
3	10400	00:35:38	0.1457	96.88%	0.24	0.0050
3	10450	00:35:48	0.2173	92.97%	0.28	0.0050
3	10500	00:35:59	0.0726	98.44%	0.13	0.0050
3	10550	00:36:09	0.0953	98.44%	0.25	0.0050
3	10600	00:36:19	0.0441	98.44%	0.12	0.0050
3	10650	00:36:29	0.2321	95.31%	0.33	0.0050
3	10700	00:36:39	0.1261	98.44%	0.22	0.0050
3	10750	00:36:50	0.1120	97.66%	0.19	0.0050
3	10800	00:36:59	0.0891	96.88%	0.13	0.0050
3	10850	00:37:10	0.2532	92.97%	0.25	0.0050
3	10900	00:37:20	0.0905	97.66%	0.18	0.0050
3	10950	00:37:30	0.2776	92.97%	0.27	0.0050
3	10989	00:37:38	0.0657	97.66%	0.15	0.0050

```
|========================================================================
=======================================|
```

Detector training complete (with warnings):
Warning: Invalid bounding boxes from 143 out of 3688 training images were removed.
The following rows in trainingData have invalid
bounding box data:

Invalid Rows
_____

39
55
61
62
82
99
101
108
113
127
145
156
199
222
260
311
316
323
328
399
431
452
472
474
525
538
546
551
575
604
607
646
696

698
739
744
763
766
775
804
815
827
844
893
903
1018
1029
1030
1035
1044
1066
1131
1139
1142
1146
1156
1218
1315
1351
1361
1430
1479
1488
1518
1527
1530
1567
1592
1698
1709
1716
1773
1792
1830
1864
1875
1917

1960
1965
1972
1979
1986
2103
2116
2148
2171
2177
2211
2232
2258
2297
2342
2386
2388
2473
2477
2482
2521
2556
2561
2563
2579
2628
2638
2674
2707
2752
2759
2772
2813
2847
2851
2881
2882
2896
2916
2921
2960
2963
2986
3025

3039
3109
3157
3210
3272
3320
3328
3340
3354
3406
3433
3442
3444
3563
3568
3594
3597
3624
3627
3636
3642
3648

Bounding boxes must be fully contained within their associated image and must have positive width and height.

****

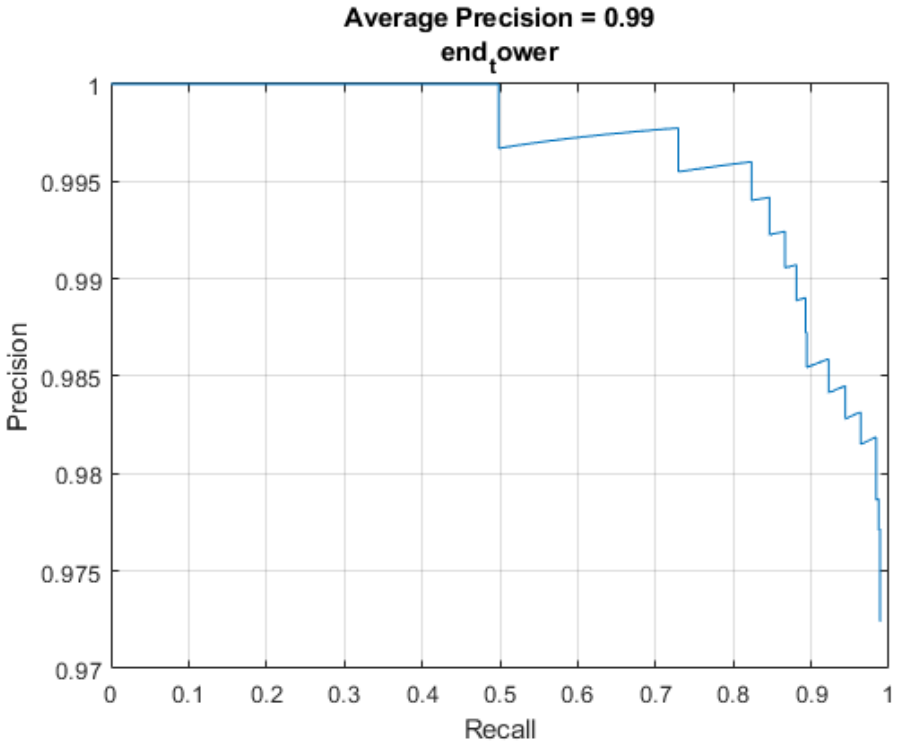
trainedDetector =

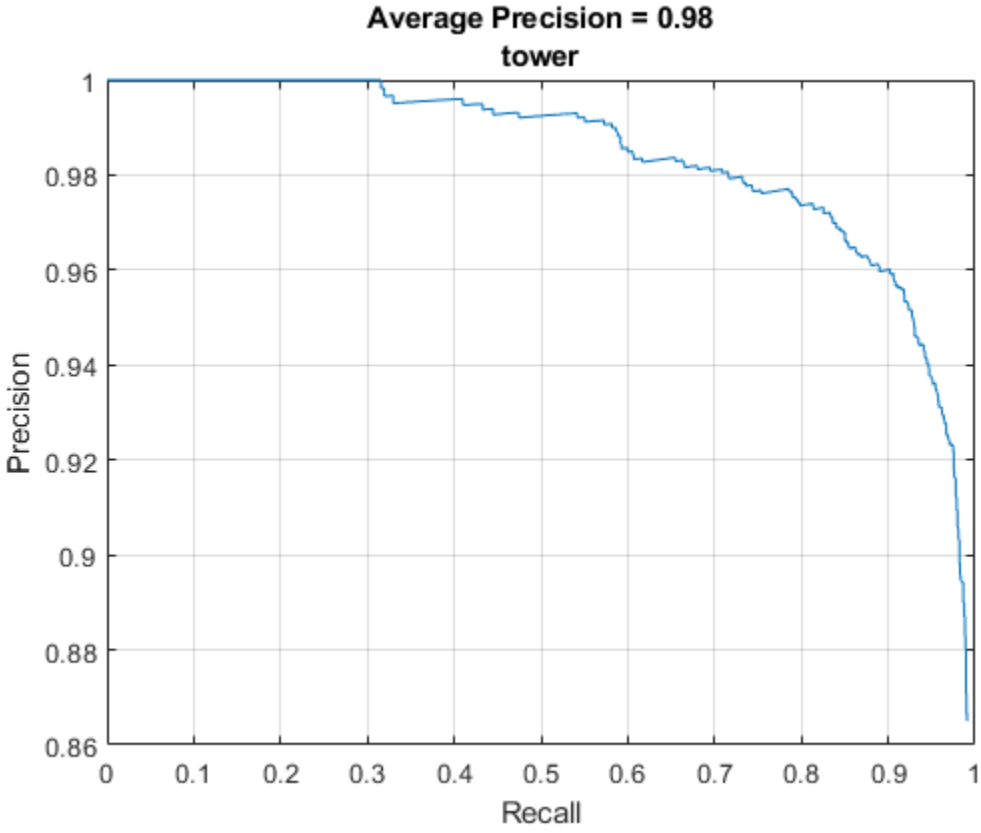
  fasterRCNNObjectDetector with properties:

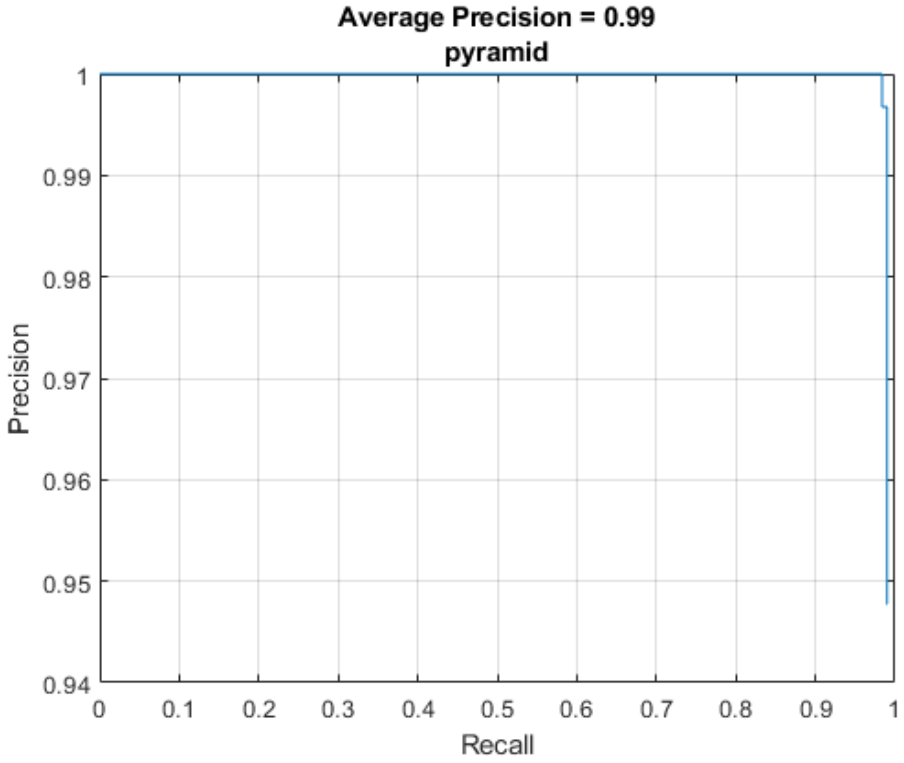
       ModelName: 'end_tower'
         Network: [1×1 DAGNetwork]
     AnchorBoxes: [6×2 double]
      ClassNames: {'end_tower' 'pyramid' 'span' 'tower' 'Background'}
    MinObjectSize: [16 16]

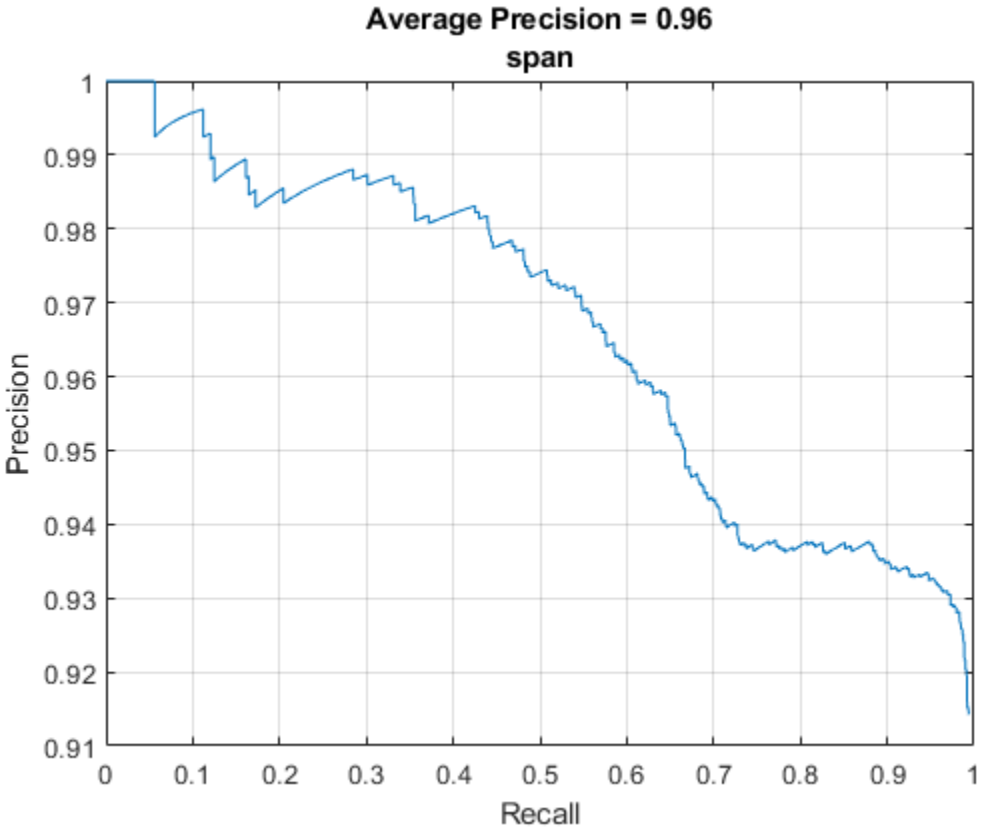
Elapsed time is 6092.868297 seconds.

APPENDIX F.12.1.1 Results

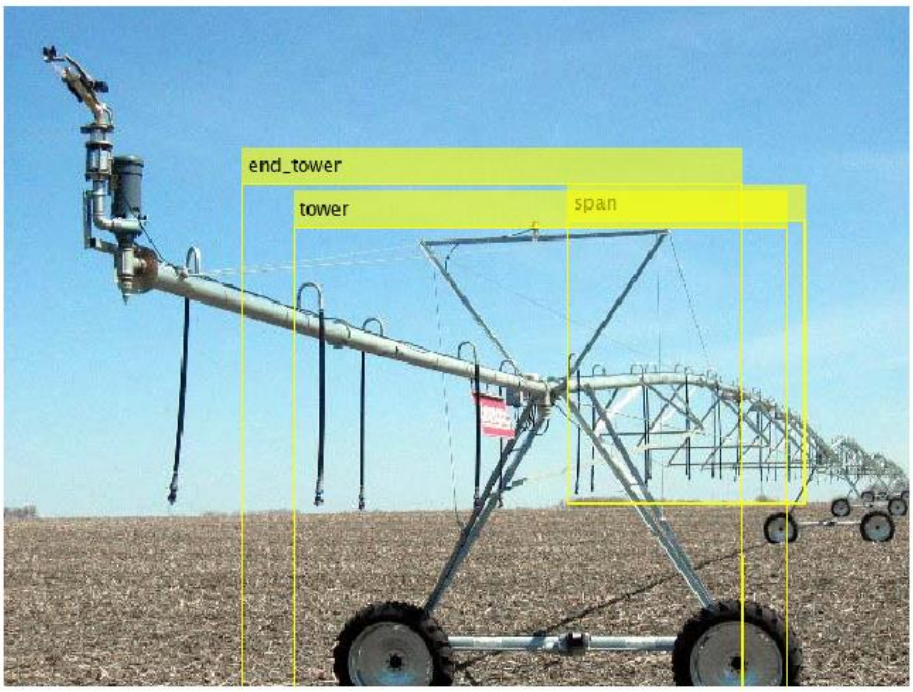


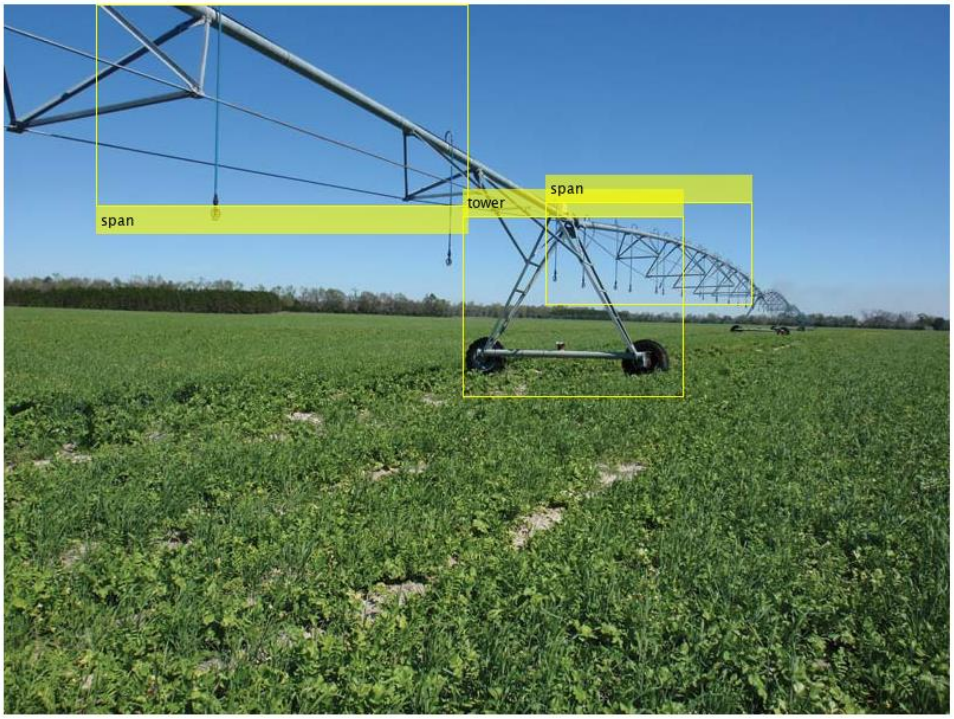



Average Precision = 0.99
pyramid
Precision
Recall
1
0.99
0.98
0.97
0.96
0.95
0.94
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Average Precision = 0.96
span
Precision
Recall
1
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

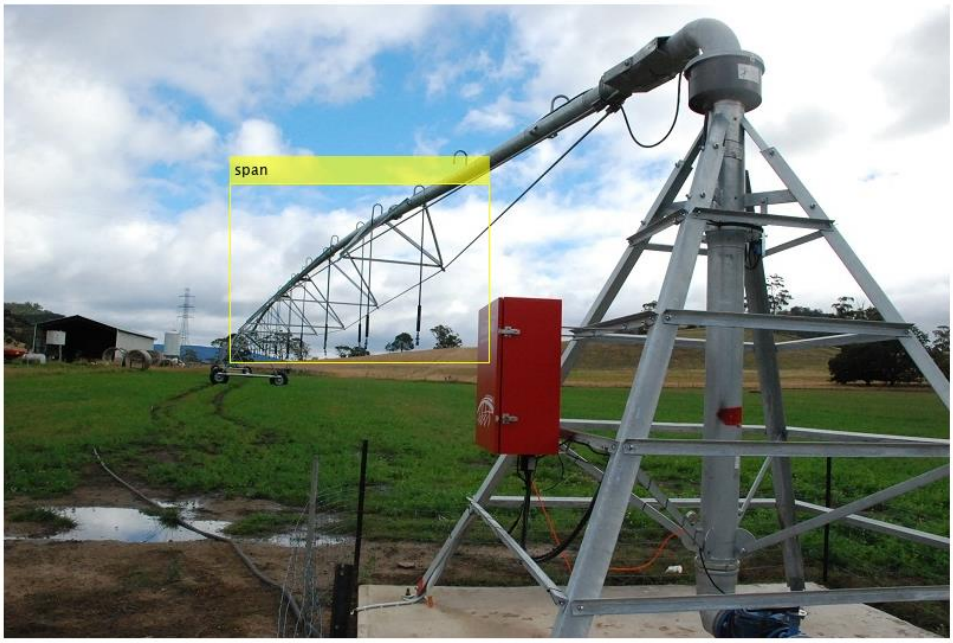
APPENDIX F.12.1.2 Results on test pictures






end_tower
tower


tower


span