Predicting Street Signs using Faster R-CNN and YOLO

by Zachary Newbury

COSC-4235 Thesis

A Thesis submitted in Partial Fulfillment
of the Requirements of the degree of
Bachelor of Computer Science

Department of Mathematics and Computer Science
Faculty of Sciences, Engineering and Architecture
Laurentian University
Sudbury, Ontario, Canada

Contents

abstract . . . . . **iii**

acknowledgements . . . . . **iv**

List of Figures . . . . . **iv**

List of Tables . . . . . **v**

1 introduction **1**
1.1 motivation 1

2 Background **3**
2.1 overview 3
2.2 How do object detectors work? 3
2.3 Machine Learning and its different types 4
2.4 Convolutional Neural Networks 5
2.4.1 convolution operation 5
2.4.2 sliding window 6
2.5 padding 7
2.5.1 pooling layers 7
2.5.2 CNN Network Structure 8
2.6 Back propagation 8
2.7 Regions of Interest and bounding box regression 10
2.8 The different types of object detectors 11
2.9 R-CNN family 11
2.9.1 R-CNN 12
2.9.2 Bounding Box Regression 12
2.9.3 Fast R-CNN 13
2.9.4 Faster R-CNN 14
2.10 YOLO family 15
2.10.1 YOLO version 1 15
2.10.2 YOLO version 2 16
2.10.3 YOLO version 3 16

2.10.4 YOLO version 4 . . . 16
2.11 Related Work . . . 17

3 methodology . . . 18
3.0.1 py-faster-rcnn . . . 18
3.0.2 detectron2 . . . 18
3.0.3 Darknet . . . 18
3.1 Model Setup . . . 19
3.2 Dataset . . . 20
3.3 metric data . . . 21
3.4 Frames Per Second . . . 21
3.5 Object detection precision and recall metrics . . . 22

4 results . . . 24
4.1 Comparison of YOLOv4 and Faster R-CNN . . . 24

5 conclusion . . . 29
5.1 further work . . . 29

A Custom Dataset preparation from LISA Traffic Sign dataset . . . 35
A.1 Detectron2 format . . . 35

B Traffic sign types . . . 38

abstract

Detecting and classifying images is a problem in machine learning., and in the case of use in autonomous vehicles object detection and classification can be crucial in allowing the system to function. Autonomous vehicles need to be able to detect traffic signs and classify what traffic sign they are looking at to adapt the driving behaviour such as stopping at a stop sign. In this thesis two object detection models are compared; You Only Look Once version 4 (YOLOv4) and Faster Region based Convolutional Neural Networks (Faster R-CNN). We show that YOLOv4 is not only more accurate after training on the Laboratory for Intelligent Safety Automobile (LISA) traffic sign dataset in all metrics but also far faster on detection time based on its average Frames Per Second. YOLOv4 recieves a 0.5% to 8% increase in AP over Faster R-CNN and a 30% increase in FPS over Faster R-CNN.

acknowledgements

First I would like to thank my supervisor professor Amr Abdel-Dayem for helping me in being able to achieve writing this thesis. I would also like to thank my academic advisor Professor Oumar Gueye.

List of Figures

1 General representation of a Perceptron, a single neuron neural network. . 4
2 General representation of a Feed Forward Neural Network a model. . . . 5
3 Intersection and Union 10
4 Two Stage Method of object detection 11
5 One Stage Method of object detection 11
6 visual representation of fast r-cnn model. 14
7 Visual explanation of how the Faster R-CNN Region Proposal Network works 14
8 YOLOv1 analysing an image and predicting objects. Figure from [31] . . 15
9 residual learning building block. 19
10 AP50 per batch while training 416x416 Yolov4 model 2 25
11 AP75 per batch while training 416x416 Yolov4 model 2 26
12 Loss per batch while training 416x416 YOLOv4 Model 2 26
13 AP50 and AP75 during training of YOLOv4 416x416 model 1 27
14 AP50 during training of YOLOv4 512x512 27
15 Faster R-CNN Resnet50 28
16 Faster R-CNN Resnet101 28
17 examples of traffic signs used in training from the LISA dataset 38

List of Tables

1 categories used for training models along including how the data set was separated for different tasks 20

2 AP metrics and FPS table for Faster R-CNN and YOLOv4 models evaluated on a Tesla P100 GPU 24

3 categories used for training models along including how the data set was separated for different tasks 36

Chapter 1

introduction

This thesis is a comparison between two Object Detection software being the Faster Region Based Convolutional Neural Network (Faster R-CNN) from the R-CNN family of object detectors and the "You Only Look Once" (YOLO) version 4 (YOLOv4) from the YOLO family of object detectors. I will be comparing the two models ability to detect street signs. Before explaining the purpose it is important to explain what exactly an object detector is. An object detector takes images that may have objects of interest and tries to detect where the objects are spatially in the image. In addition, Object Detectors can use image classifiers to classify what object they are looking at. The object detectors in this thesis will be used to both detect and classify images. There are other technologies besides object detectors that achieve the goal of detecting objects around vehicles such as light detection and ranging (LiDAR), "which utilizes multiple laser beams to locate obstacles in its surroundings and is known for its capability to depict this information in a dense three dimensional (3D) cloud of points[43]". LiDar however only maps the 3D space around it and does not detect the visual information around it which can be important information when driving vehicles. A good example of when to use this technology is in autonomous vehicles or "self driving" cars. In order for vehicles to autonomously drive they need some way to be able to detect street signs to know when to stop for stop signs or to slow down for traffic. There are multiple ways in which autonomous vehicles can determine what street signs to look out for such as satellite data but an object detector can be used to detect the signs with cameras and then have the car react based on what objects it detects.

1.1 motivation

An important aspect in the ability for automobiles to be able to drive autonomously is the ability to perceive objects in the area around it. How quickly the car can detect objects and how accurately is an integral role with cars needing to solve optimization problems at least once a second [20]. Being able to know which object detector would be

best for both speed and accuracy in the application of autonomous cars is one of the main reasons that led to the decision on this topic for this thesis. Modern convolutional neural network based object detectors are the most applied technique for detecting objects for autonomous vehicles[11]. Do to this CNN based object detectors were decided upon as the best choice for comparison. Additionally, object detectors are one of the fields at the forefront of the computer vision field and a good starting point for getting into the Artificial intelligence (AI) area of study. Computer vision is necessary in autonomous vehicles to determine what signs are on the road to determine situations like speed limits, signs warning to stop or yield and signs that warn to be more alert like pedestrian crossing signs.

Chapter 2

Background

2.1 overview

In order to compare the two object detection software, we must first understand what object detection is and how both of these software work. To do this, we will go over the background concepts required to understand object detection starting with the types of machine learning algorithms that modern object detectors are based on. We will then transition to talking about background knowledge required to understand some of the functionality in the models including intersection over union, bounding box regression and some previous object detection models to better understand how the more modern ones work. Finally we will then talk about in detail how Faster RCNN and YOLOv4 work.

2.2 How do object detectors work?

In order to compare the two object detection models it is important to lay out how object detection works in software. First, what is object detection? Object detection is a method of detecting objects in images. As an example, if a object detection software is given an image with several dogs in it, it should be able to determine where in each image the dogs are. By extension a similar concept called image classification is used to then classify what objects are detected by the object detector. In practice, object detectors are often combined with image classification models and are often implemented using some machine learning algorithms. Examples of Object Detectors include the RCNN or YOLO families of object detectors but also older models like the Viola Jones Facial detection. Object detection in more modern times uses more advanced machine learning models, mainly deep learning algorithms using Convolutional Neural Networks.

2.3 Machine Learning and its different types

Machine learning (ML) is a subfield of Artificial Intelligence that is designed to be able to "learn". In practice, this means giving a ML model known data or already analysed data and unanalysed data to produce rules to predict future data[4]. Deep Learning is a subfield of ML with the main difference that it uses successive representations of input data[4]. In object detection models, ML and Deep Learning are often represented using Artificial Neural Networks as are much of ML today.

Artificial Neural Networks or Neural Networks are a network of Artificial Neurons. Neurons can be generalized as a single node or function where the result is derived from multiple inputs multiplied by determined weights and a possible bias value. Figure 1 shows a visual representation of a type of Neuron called a Perceptron[23].

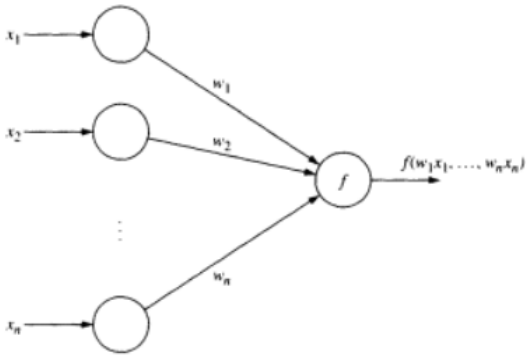


Fig. 1: General representation of a Perceptron, a single neuron neural network. Where each x_i represent an input value, each w_i represent a weight value and the function f calculates the output value from the weights and inputs. Figure from [23].

A Perceptron is an old type of neuron where the idea is that it has two possible choices. This can be generalized to a linear function, usually in the form of

$$f(x) = \begin{cases} 0 & \text{if } X'W + b \leqslant 0 \\ 1 & \text{if } X'W + b > 0 \end{cases} \tag{2.1}$$

where W is a vector of weights for each input, X is a vector of each input and b is a bias weight [26]. While this neuron type can be helpful with small scale problems like predicting logical outputs like AND gates it isnt as scalable for larger prediction problems that are more based on fuzzy values like probabilities. Due to this other activation functions for outputs have been created that use smooth activation such as the Rectified Linear Unit (ReLU)[17] function $\phi(z) = max\{0, z\}$[10] and others like the sigmoid function $\phi(z) = \frac{1}{1+\exp(-z)}$ where z is the threshold $z = X'W - b$. These functions use continuous values instead of discrete ones allowing for more variations in output. In many

cases, a single neuron is not enough to accurately predict the desired results though so neural networks are needed to network the results from multiple neurons into more neurons for refinement. There are multiple ways of configuring a neural network, but the configuration relevant to object detection is the forward feedback neural network seen in figure 2. A feedforward neural network is a layered network that is fully connected meaning that each neuron result feeds into each of the neurons in the next layer and that there isn't back feeding[23].

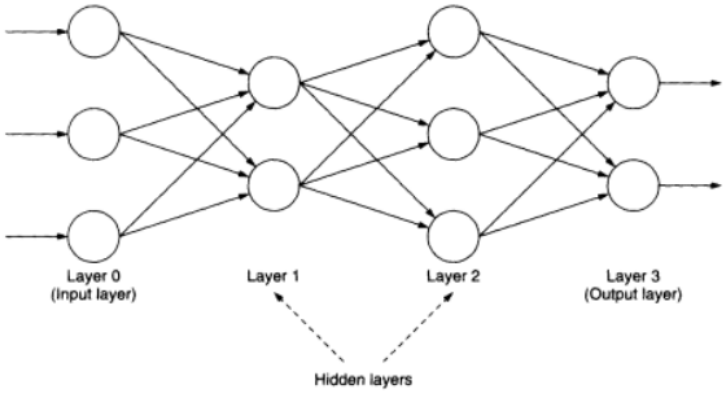


Fig. 2: General representation of a Feed Forward Neural Network a model.
Figure from [23].

2.4 Convolutional Neural Networks

While a feedforward network is useful in training a range of data the above configuration is limited in training and recognizing features in images especially if those features are translated in new images. To a network a dog on the left side of a screen and on the right side of the screen are completely different images even if its the exact same image of the dog. To fix this problem a neural network model that is translation invariant in its detection of image features is needed. A Convolutional Neural Networks[19], also known as a CNN, "are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers"[10]. Before going into the structure of a CNN it is necessary to define what convolution is and how it works in practice as well as how pooling layers in CNN work. Additionally what benefits this change brings to the network.

2.4.1 convolution operation

A convolution operation "gets two tensors as input and produces a tensor as output" [24]. A tensor in this definition being some multidimensional array. An example being a 0D tensor is a scalar as it scales 0 dimensions and a 2D tensor as a 2-dimensional matrix [4]. The following two examples will demonstrate how convolution works in practice based

on the definition given in [24]. First, Consider a 2x2 input matrix **A** and corresponding 2x2 matrix called the Filter or sometimes called the Kernel **K**.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}, \mathbf{K} = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} \tag{2.2}$$

We can apply the convolution operation, denoted with symbol *, on both of these tensors and will get a corresponding scalar value defined in equation 2.3.

$$\mathbf{A} * \mathbf{K} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} = \sum_{i=1}^{4} a_i k_i \tag{2.3}$$

While this operation simple normally input tensors are very large maybe being close to 1080x1920x3 in the case of standard HD images while kernels stay relatively small maybe being 3x3 or 5x5 in dimensions. So a definition of convolution with different sized input tensors are needed.

2.4.2 sliding window

Consider another example where a input tensor **A** has a dimension of 3x3 while the Filter **K** has a dimension of 2x2.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}, \mathbf{K} = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} \tag{2.4}$$

We can get a 2x2 "patch" from the matrix

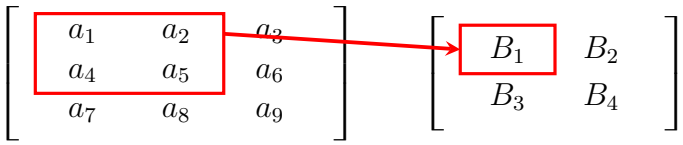
$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \rightarrow \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \tag{2.5}$$

We can then apply the convolution operation on the patch with the filter tensor to get the first scalar value for the output tensor which we'll call $\mathbf{B_1}$

$$B_1 = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix} = \sum_{i=1}^{4} a_i \cdot k_i \tag{2.6}$$

We would then put the resulting sum into a activation function and continue this

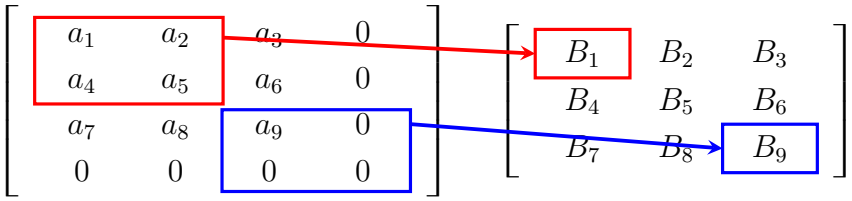
process for all B_i patches we can specify getting a smaller tensor as a result.



(2.7)

2.5 padding

One situation that will come up is that we will want to do a convolution for every pixel in an image but by the definition given above we won't be able to as there won't be a neighborhood around a_3 as an example. In order to solve this issue we use padding to make it so that the resulting feature map from the convolution will be the same dimension as the image that entered. Padding is done by adding dummy rows and columns of 0's around a tensor to conserve the shape of the output. If you use **A** and **K** from equation 2.4 you could pad **A** to conserve the dimension by adding a column of 0's next to the right most column and a row of 0's below the bottom most column like so:



(2.8)

2.5.1 pooling layers

Pooling is similar to convolution but much simpler, in image analysis "pooling helps to make the representation approximately invariant to small translations of the input" [10]. There are several type of pooling functions but we can use max pooling as a simple example. Consider a 3x3 tensor **A** like we used before in equation 2.4.

$$\mathbf{A} = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \tag{2.9}$$

Just as we did in equation 2.10 we will get a patch from the tensor, but instead of applying a convolution operation with a filter, we will instead apply a pooling function, in this

case we will get the max value from the patch shown in equation 2.11.

$$\begin{bmatrix} \boxed{\begin{matrix} a_1 & a_2 \\ a_4 & a_5 \end{matrix}} & a_3 \\ a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{\begin{matrix} a_1 & a_2 \\ a_4 & a_5 \end{matrix}} \end{bmatrix} \tag{2.10}$$

$$\mathbf{B}_j = \max_i a_i \tag{2.11}$$

Which in the example given in equation 2.10 equation 2.11 would give $B_1 = max(a_1, a_2, a_4, a_5)$ and so on for each possible j.

2.5.2 CNN Network Structure

Now that we have a basis for how each of the layer types in a CNN works we can now explain how it is implemented in a network. The first few stages are convolutional layers and pooling layers. Convolutional layers organize units into a feature map where each unit is connected to a patch from the feature map of the previous layer with a set of weights known as the filter bank[18]. The result is then passed through a non-linearity activation function like ReLU[17], defined by in section 2.3 and then the output is passed on to the next layer. [24]

The Pooling layer is used to merge semantically similar features into one. "A typical pooling unit computes the maximum of a local path in one feature map (or in a few feature maps)[18]." Max pooling will decrease the dimension of the representation. Following several layers of convolutional layers and pooling layers, the CNN will then give a final result in its output layer, this could be represented using a softmax function to give a representation as a probability.

2.6 Back propagation

Now that we understand the different types of Neural Networks required for doing object detection, we now need to know how these models are trained. The main method of training neural networks is by using the back propagation method with gradient descent. There are several steps involved in this process so we will summarize Rumelharts[34] original explanation of the concept as well as using some newer ideas brought up by Nielsen[26].

First, the aim of training a neural network is that when an input is given to the network a subsequent output is as close to its expected result as possible. So if in the case of object detection we give an image of an apple, we expect the object detector to result in a prediction that it is an apple. We can use a loss function to get the errors of the results and use that to update the weights. Equation 2.12 shows an example loss

function often used being the Mean Squared Error.

$$MSE = C(x) = \frac{1}{2n}\sum_{x}\left|\left|y(x) - a^{L}\right|\right|^{2} \tag{2.12}$$

Where C is the loss function, x is the input tensor, y(x) is the expected results of x, and a^L is the actual output of the network where L is the number of layers. This function will get us the total loss or error of the network, which can then be used to get the amount to which each weight needs to be updated. Next, the derivative of the total loss with respect to each weight $\frac{\partial C}{\partial w_{i,j}}$ where $w_{i,j}$ is the weight from the ith neuron to the jth neuron. This gives us a "direction" in which to update the weight values, if the derivative is positive it needs to increase and the reverse vice versa.

$$\frac{\partial C}{\partial w_{i,j}} \tag{2.13}$$

The original paper described just adding a Δw for each w to update each weight after a training input defined as

$$\Delta w = -\epsilon\frac{\partial C}{\partial w_{i,j}} \tag{2.14}$$

Where ϵ is the learning rate (LR) effecting how much each meta parameter is updated per iteration. A common LR is between 0.001 and 0.01. To summarize, you can write backpropogation as a simplified algorithm[18] defined below:

1. input a data set of training examples.
2. for each training example x: set the corresponding activation output a and perform the steps:
   - **Feedforward:** For each layer compute each neurons activateion $z^{x,l} = w^l a + b^l$ where l is the layer and a is the input from the previous layers activation's.
   - **Output Error:** calculate the loss vector C for the networks activation
   - **Backpropogate error:** for each layer from output to input compute the derivative relative to the loss for each metaparameter t, $\frac{\partial C}{\partial t}$
3. **Gradient Descent:** for each metaparameter t in the layer update them based on the rule $t- > t - \epsilon\frac{\partial C}{\partial t}$, do this for each layer using the cost function for the current layer until the input layer is reached.

While simple gradient descent worked some what well for older neural network models, newer models have decided to use a slightly different approach for back propagation. While the theory is still the same they don't use the whole training set to update the

weights at once, rather they create "batches" to train the model iteratively. Both Faster R-CNN and YOLOv4 train their networks in random batches from their training data sets.

2.7 Regions of Interest and bounding box regression

Before proceeding to explain how each of the different object detectors work it is necessary to get an understanding of how object detectors detect, where in the image an object is, and how to handle inaccurate predictions. While CNNs can be used to create a feature map to classify what objects are in an image it can not be used to determine where in the image the objects are on its own. Object detectors tend to separate the object detection phase and the classification so that the model can first try and predict where in the image objects are. When an object detector makes a prediction as to where in an image an object is this is known as a region of interest (RoI). There are multiple different approaches in finding objects including algorithms like Selective Search[37] used in R-CNN or anchor based methods like the Region Proposal Network used in Faster R-CNN[32] and YOLOv2[29]. In both of these methods the RoI is denoted as a bounding box which is a rectangular region where an object is. Regardless of method, the ability to detect if a RoI is accurate needs to be trained, but to do that there needs to be some metric to determine this.The Intersection over Union (IOU) is used to determine the prediction accuracy. The IoU is the intersection of the preset bounding box and the predicted bounding box area divided by its union[33]. Figure 3 gives a visual representation of the area, the equation for IoU is seen in the following equation.

$$IoU = \frac{Intersection(box1, box2)}{Union(box1, box2)}$$

If a bounding box prediction has a IoU below a certain threshold, it is factored in as a negative prediction. IoU usually has a threshold of at least 0.5[6].

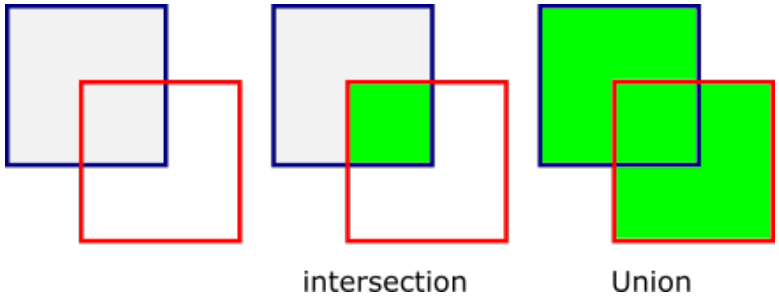


Fig. 3: Intersection and Union

2.8 The different types of object detectors

Now that we have an understanding of what a neural network is and how it works we can now describe how modern object detectors use deep learning to predict objects in images. The first thing to understand before going into the different object detectors is that there are two distinct approaches to CNN based object detection, the one-stage method and two-stage method [28]. The main difference between the two methods is the speed where two stage methods tend to be slower then the one-stage approach[28]. The two models that are being tested in this thesis each use a different approach, where Faster R-CNN uses the two-stage method while YOLO uses the one stage method. We can use the original R-CNN model to describe how the two-stage approach works. The original paper on RCNN describes the model as a 4 step process where it "(1)takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs"[8]. We can generalize this as seen in figure 4.

Fig. 4: Two Stage Method of object detection

One stage detectors simplify this process by predicting the RoI and generating the feature set at the same time[31]. You can see a visual representation in Figure 5.

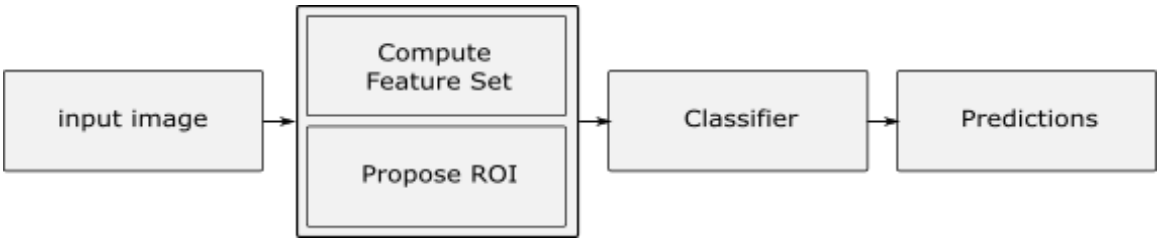


Fig. 5: One Stage Method of object detection

2.9 R-CNN family

As mentioned in section 2.8 Faster R-CNN is a two stage object detector, the previous versions in the R-CNN family also share this feature. From the original R-CNN to Faster R-CNN there were many changes, a big one being that in the original only the feature detector was a neural network, but by Faster R-CNN it was a full neural network from end to end. R-CNN was an important milestone in the development of CNN object detection.

2.9.1 R-CNN

The process of image detection originally defined by Girshick[8] using R-CNN involves the following steps, first the model takes an input image and "extracts around 2000 bottom-up region proposals" using an algorithm like Selective Search. Second, it "computes features for each proposal using a large convolutional neural network", the feature detector CNN has a fixed size input for images of 227x227 RGB inputs so the proposals are warped to those dimensions before input. Finally it "classifies each region using class-specific linear SVMs". An SVM being a Support Vector Machine, a machine learning algorithm with the goal of "finding good decision boundaries between two sets of points belonging to two different categories[4]." R-CNN "optimize one linear SVM per class"[8] meaning each linear SVM determines how likely it is for it to be the given class or not. R-CNN will also try and refine the predicted RoI, or bounding box since the RoI is defined in a rectangular region in this model, by using a method called bounding box regression on the predicted bounding boxes.

2.9.2 Bounding Box Regression

The way in which RoI are predicted can vary by model, but even then after a prediction is made it will probably have to be refined to better represent the object it's predicting, the way in which object detectors refine their RoI is called Bounding Box Regression. R-CNN[8] described bounding box regression with the goal being to find a function $d_i(P)$ for each value that describes a bounding box defined as a 4-tuple (x,y,w,h) that can be used to transform a predicted bounding box P to the ground truth box G that fits the object. Not all object detectors use the same bounding box regression model but to give an example the following is how bounding box regression worked in R-CNN. P is described as a 4-tuple $P = (P_x, P_y, P_w, P_h)$ where x and y are the center coordinates of the bounding box and w,h are defined as the width and height in pixels, similarly the ground truth box G is defined as a 4-tuple $G = (G_x, G_y, G_w, G_h)$. The transformation from the predicted bounding box to the ground truth box is defined in equation 2.15 to 2.18 for each 4-tuple parameter.

$$\hat{G}_x = P_w d_x(P) + P_x \tag{2.15}$$

$$\hat{G}_y = P_h d_y(P) + P_y \tag{2.16}$$

$$\hat{G}_w = P_w \exp\left(d_w(P)\right) \tag{2.17}$$

$$\hat{G}_h = P_h \exp\left(d_h(P)\right) \tag{2.18}$$

Where each d_i can be defined from the feature map of the image as well as a trained set of weight meta-parameters $d_i = w_i^T \phi_5(P)$[8] where ϕ_5 is the output of the feature map

from the 5th pooling layer. The weights are updated with a ridge regression:

$$w_i = arg\min_{\hat{w}_i} \sum_{k}^{N} \left(t_i^k - \hat{w}_i^T \phi_5\left(P^k\right)\right)^2 + \lambda ||\hat{w}_i||^2 \tag{2.19}$$

Where each P^k is a different bounding box prediction for some class and $\hat{w}$ is the current value of the w meta-parameter that will be updated. The λ variable gives the strength that it is updated at

The t_i metaparameter is based on the prediction and is just used for predicting the target bounding box, it is defined in equations 2.20. The duplicate equations t_i^* in equations 2.21 are used in faster r-cnn for getting the bounding box regression loss for its region proposal network.

$$t_x = \frac{x - x_a}{w_a}, t_y = \frac{y - y_a}{h_a}, t_w = \log\left(\frac{w}{w_a}\right), t_h = \log\left(\frac{h}{h_a}\right), \tag{2.20}$$

$$t_x^* = \frac{x^* - x_a}{w_a}, t_y^* = \frac{y^* - y_a}{h_a}, t_w^* = \log\left(\frac{w^*}{w_a}\right), t_h^* = \log\left(\frac{h^*}{h_a}\right) \tag{2.21}$$

Where x,y,w,h are the center of the bounding box and its width and height, x, x^* and x_a are values from the predicted bounding box, anchor box (this is the prediction gotten from selective search in the original R-CNN) and the ground truth box.

2.9.3 Fast R-CNN

Fast R-CNN improved on R-CNN by implementing structural changes in the network that increases speed and accuracy over the original. One of the biggest changes is that Fast R-CNN no longer uses a SVM for the class outputs, now instead using a softmax function as a part of the same network as the feature detector[7]. Another change is the implementation of the RoI pooling layer which changes how RoI are analysed. Instead of resizing each region and then inputting each one into the feature map CNN, the CNN now takes in both the original image warped to fit the CNN and the predicted RoI at once and then uses max pooling "to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of H x W.[7]" Figure 6 shows a visual representation of the model and how it detects objects.

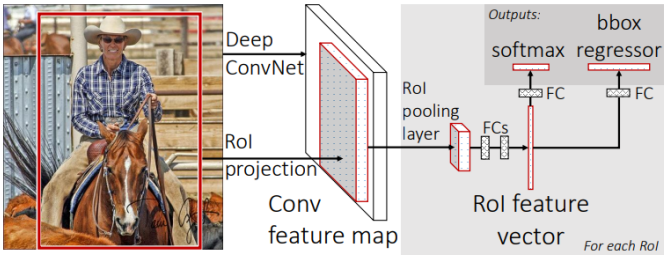
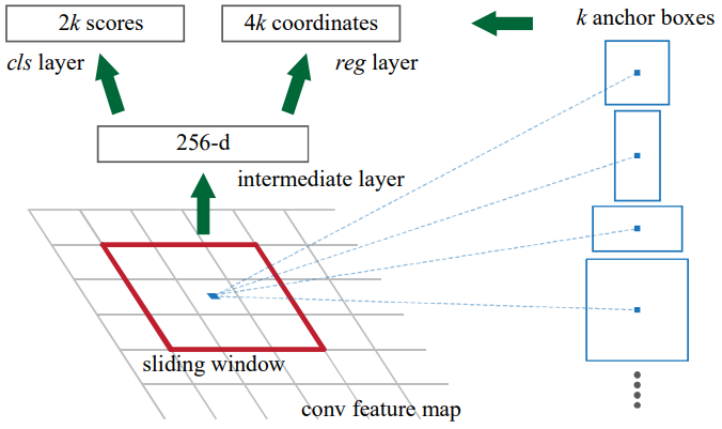


Fig. 6: visual representation of fast r-cnn model.
Figure from [7]

2.9.4 Faster R-CNN

Finally, Faster R-CNN[32] iterated on Fast R-CNN to create a model which is a neural network from end to end, removing the need to use an algorithm to predict the RoI. Faster R-CNN instead uses a Region Proposal Network (RPN) to predict it's RoI. Summarizing Ren[32], the RPNs takes the feature map taken from a CNN after processing an image and then will output region proposals and objectness scores using an anchor based approach and the feature detector. In figure 7 it demonstrates the anchor based approach visually. The anchor based approach predicts k bounding boxes simultaneously in each sliding window location, in the case of Faster R-CNN they use k=9 as there is 3 bounding boxes for 3 possible different scales to be predicted, and then passes to a lower dimension feature detector and then passes the outputs to a bounding box regressor layer and classification layer to get the predicted proposal box coordinates and classification scores. Each anchor box is trained separately to get its aspect ratio.



(a) (b)

Fig. 7: Visual explanation of how the Faster R-CNN Region Proposal Network works (a) shows the sliding window, in each grid place that the sliding window checks it proposes k anchor boxes shown in (b) which predicts predicts an object detection score for each box and the anchor box proposed coordinates. Figure from [32].

2.10 YOLO family

In the following section we will look over how each version of the You Only Look Once (YOLO) family works so we have an understanding of how YOLOv4 which we will be comparing also works.

2.10.1 YOLO version 1

In this thesis YOLO version 4 (YOLOv4) will be investigated so having an understanding of how the model works is essential in studying its capabilities. The YOLO family of object detectors were iteratively improved so understanding how YOLO version 1 (YOLOv1) analyses images will help in understanding its later version. As stated in the previous section, YOLOv1 has 3 distinct steps in analysing images starting with resizing the input image. It will resize the image to 448x448 px, this is used as a setup for the next step which uses a grid approach to predict its RoI [31]. Next is to input the resized image into a CNN that simultaneously detects RoI predictions and the feature set in parallel. For the RoI predictions the network first separates the image into a SxS grid where S is specified before training, so if S=7 as is the case in the original YOLO paper[31] the image will have a 7x7 grid of equally sized square grid cells. The network then predicts B bounding boxes for each grid region in the image, if B=2 as an example then each grid region predicts 2 bounding boxes. Alongside predicting bounding boxes, it also predicts the probabilities of each class the object detector is setup to detect for each grid. Once both of these are done, the network then gives the outputs to the classifier to combine the data and make the final predictions based on the best results.

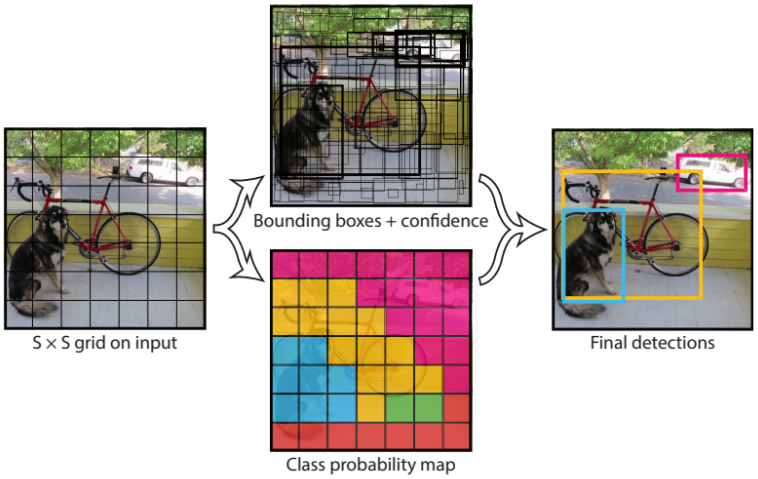


Fig. 8: YOLOv1 analysing an image and predicting objects. Figure from [31]

2.10.2 YOLO version 2

YOLO version 2 (YOLOv2) changed the RoI prediction step by instead using an altered version of Faster R-CNN's Region Proposal Network (RPN) to predict bounding boxes in an anchor approach speeding up detection. This approach did decrease the accuracy of YOLO but increased not only speed but also the amount of objects that could be detected as it now predicts over a thousand bounding boxes[29]. The bounding box regression was altered as well now using direct location prediction, this limits the change in location for the bounding box to within the grid the object was predicted. One of the biggest changes was also the introduction of the Darknet19 feature detector which is a 19 layer deep ConvNet.

2.10.3 YOLO version 3

YOLOv3 was a more iterative update to YOLOv2, it's biggest change to its feature extractor now a 53 layer deep Residual CNN called Darknet-53. Darknet-53 boasts to be more accurate but with a hit to speed, citing a accuracy of 77.2 to YOLOv2 74.1 and a FPS of 78 vs YOLOv2 171 [30]. The other big change YOLOv3 brought was to predict bounding boxes across 3 different scales.

2.10.4 YOLO version 4

Finally YOLOv4 introduced what it calls the bag of freebies and bag of specials to improve upon yolov3 model. The bag of freebies are "methods that only change the training strategy or only increase the training cost"[3] while the bag of specials are plugin modules and post-processing methods that "plugin modules and post-processing methods that only increase the inference cost by a small amount but can significantly improve the accuracy of object detection". Bag of freebies in YOLOv4 mainly consists of data augmentation steps which can include augmenting the geometry of the image like random scaling and cropping but also photometric distortion like image brightness and saturation.

YOLOv4 network is broken up into 4 parts, the input, backbone (or feature detector), neck and head. The input is the input layer to the network for input images, the backbone is a feature detector where they use darknet53 from YOLOv3, the neck are layers that connect the head and backbone of the model which can include layers like a feature pyramid network[21] but YOLOv4 uses the SPP[16] and PAN[22] layers and finally the head is the layers that are used for object detection where YOLOv4 uses the same layers from YOLOv3.

2.11 Related Work

Interest in detecting traffic signs using software has become popular over the last decade. Initial research on detecting traffic signs didn't start in the field of autonomous vehicles but rather as a need for being able to take inventory of traffic signs [39]. Later on they would compare image classification model like histogram of gradients (HOG) to CNN models finding good results for the CNN models[35]. Eventually CNN object detection models became quite popular within the object detection space and in [36] they would then compare CNN object detection models precision using modern models like R-CNN, Faster R-CNN and Mask R-CNN compared using the DFG traffic-sign dataset. There was also [1] which compared one stage detector YOLO and its variants using the same DFG dataset for training.

Chapter 3

methodology

Now that we have an understanding of how the two object detectors works we can now explain how the experiment was set up. It was never considered to manually program the models using languages like python with the Tensorflow library. Several software were considered before the final decision.

3.0.1 py-faster-rcnn

py-faster-rcnn is the original release of faster R-CNN implemented in python2 code. Some time was spent in trying to implement this version of the model but due to it being difficult to implement and having been deprecated an superceded by detectron and later detectron2 it was decided to move on to the newer release.

3.0.2 detectron2

For training the Faster R-CNN model we used facebooks Detectron2 software[40]. The model was taken from the detectron2 model zoo being pretrained on the Common Objects in Context (COCO) dataset and using Resnet 50 as its backbone.

3.0.3 Darknet

YOLOv4 has many different forks based on the original release from AlexeyAB, we decided to use the Roboflow fork of Yolov4 based on AlexeyAB release. This release was chosen due to its built in metric reporting such as a map function for getting the mean average precisions, which also allows for specifying of the IOU threshold so it can be compared to the COCO style metrics produced from detectron2's evaluator. The software also saves the weights file every 1000 iterations so if an error occurs and training has to stop it could be continued.

3.1 Model Setup

Both Faster R-CNN and YOLOv4 while different in approach have some features that can be shared for a fair comparison. For this experiment it was decided that each model would be trained on a learning rate of 0.01 based on recommendation from Bengio[2]. Each model would be trained on at least 26000 batch iterations based on the recommendations from darknet saying that there should be at least 2000 times the number of classes batches trained on. This isn't a rule as if the model doesn't converge on a good AP by the 26000 iteration it would be trained on more iterations as will be seen later. Next the is the backbones that each model will be trained on. As stated in Chapter 1 image classification and detection are different so the choice of what classifier is used can have some effect on both precision and speed. For YOLOv4 the CSPDarknet53 head used in the original YOLOv4 paper was used for the experiment, it was tested on an input of 416x416 and 512x512 image input. For Faster R-CNN we trained on the Resnet 50 and Resnet 101 networks, the numbers representing the depth (number of layers) in these networks.

Resnet[12] is a residual network, the intention of this architecture to remedy what is known as the vanishing gradient problem. The vanishing gradient comes from an issue from the back propagation algorithm for neural networks where it has been found that "error signals become progressively more diffused as the signal goes back through each hidden layer"[9]. In other words "with the network depth increasing, accuracy gets saturated"[12], so ResNets found a way of remedying this problem by introducing a deep residual learning framework using identity mappings.

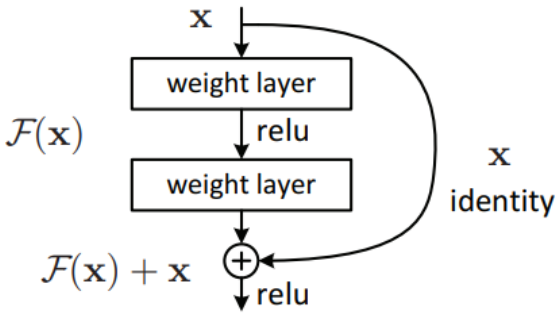


Fig. 9: residual learning building block.
Figure from [12]

Figure 9 gives a visual explanation of how a residual mapping works. Instead of just having the outputs from the previous layer go to the next through the whole network ResNets also map CNN blocks of same output and input size and pass the input of the beginning of the block to the output to the end of the block adding it to the blocks output before going into the linear function. In the Resnet paper they claimed that this was able

to get a 6% increase in average precision using Faster R-CNN over the original backbone used VGG-16[32].

Darknet53 was originally described in the YOLOv3 paper by Redmon[30] and is a 53 layered Resnet CNN. In YOLOv4 they altered the darknet53 architecture based on CSPNets Cross-Stage-Partial connections[38] in some of its "skip connections" which are similar to resnets identity connections.

3.2 Dataset

The Laboratory for intelligent and safe automobiles (LISA) traffic sign dataset[25] is used for training both models for analysis. It was chosen because it has traffic signs already individually annotated as their street sign type unlike other datasets like googles openimages that groups all traffic signs except stop signs into a single class. The LISA dataset doesn't use the same standard notation of YOLO and Detectron2 so code had to be created to convert the annotation files for each model. The annotations were first converted to Detectron2's format for custom datasets. They were then converted from that format to the YOLOv4 custom format. An explanation on how they were converted to those formats are in Appendix A.

categories	train	valid	test
pedestrianCrossing	740	237	108
signalAhead	618	203	104
speedLimit35	369	117	52
speedLimit25	229	86	34
keepRight	223	71	37
addedLane	207	58	29
merge	185	59	22
yield	151	69	16
laneEnds	133	56	21
stopAhead	113	33	22
speedLimit45	102	32	7
speedLimit30	94	29	17
school	88	33	12

Table 1: categories used for training models along including how the data set was separated for different tasks

It was decided to use the 13 classes shown in figure 1 for training the data set as each

class has over 100 images at least. The stop sign class was ignored as both datasets have already been pre-trained on the class. The classes where split randomly with 10% being used for the test images, and 60% of the remaining images being training images, and the remainder being used for validation. A total of 3252 images where used for training and 4816 images in the total dataset.

3.3 metric data

When training the models both detectron2 and darknet keep track of the models performance from start to finish. Darknet gives the ability to track the mAP defined as the AP where the IOU threshold is 0.5 and prints a graph showing the mAP and total loss per batch iteration. With Detectron2 you can specify a Evaluator that calculates the metrics every evaluation period specified in the configuration file. For our testing we used an evaluation period of 500 batches, so once every every 500 batch iterations it would calculate its metric data. Yet once we have our final model we need to define by what metrics we can compare our two models on. It was decided based on previous comparisons between object detection models like in [5] that the COCO AP metrics and the FPS would be the best metrics to compare the models on. All comparisons are done in the cloud using Google Colab using a Tesla P100 16GB GPU. Training was done using a Tesla V100 GPU instead due to its 3x quicker speeds which should have sped up training time.

3.4 Frames Per Second

When an object detector analyses an image there is a gap in time between the image being inputted into the model and the final predictions being outputted. We can use the inference time it takes to analyse an image to get the inference time per image. Getting the average inference time measured in seconds per image can then be turned to images or frames per seconds by getting the inverse since it is a measure of $\frac{second}{image}$ and not $\frac{image}{second}$.

$$\text{Frames Per Second} = \frac{1}{\text{Inference Time per Image}} \tag{3.1}$$

In Detectron2 they give the average inference time per image after analysing an image so we just had to apply equation 3.1 to get the FPS of the Faster R-CNN model. Darknet does provide the detection time it takes to analyse all of the validation images but not the inference time per image. Hoewever, Darknet provides a benchmarking tool for getting the FPS of the model which gives similar results to getting the FPS by instead dividing the total images analysed by the detection time. As an example, if we got a detection time of 25 seconds, we analysed 1048 images then $\frac{1048 \text{ images}}{25 \text{ seconds}}$ gets around 41 seconds which would be similar to what we get in the benchmarking tool. Inference time is also effected

by the hardware in use so the same GPU will be used between tests, in the case of this thesis the Tesla P100 GPU will be used in all our evaluations.

3.5 Object detection precision and recall metrics

In section 2.7 we talked about the IOU threshold and how it can be used to determine if the predicted results are a positive or negative prediction. In total, there are 4 prediction metrics neccessary in calculating the relevant metrics for analysing the results being true positive (TP), false positive (FP), true negative (TN) and false negative (FN) counts per class. In order for a prediction to be positive, it needs to pass 2 requirements, it needs to succeed the confidence threshold that an object is being detected and its IOU needs to exceed a specified IOU threshold, else it would be a negative prediction[27]. If a prediction confidence threshold and makes a correct prediction based on the ground truth label it is a true prediction, if it predicts the wrong label it is a false prediction. We can count the TP, FP and TN for each class to then calculate the precision and recall metrics for each. These metrics will then be used to calculate the Average Precision which will be needed to compare the object detection models. Precision and Recall are defined in equation 3.2 and 3.3.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

Precision defines the accuracy of results when accepted by the model while the recall gives the models ability to predict accurate results. Using these two metrics we can get the average precision value for comparing results. In the Pascal Visual Objects Challenge challenge they defined average precision as "the mean precision at a set of eleven equally spaced recall levels [0,0.1,...,1]"[6]. In simplified terms they get the average precisions based on different recall values. This is a quick way of getting the integral of whats known as the precision-recall curve based on ranked inputs. In equation 3.4 it gets the average of 11 different precision values gotten from the recall values from 0 to 1, in equation 3.5 it gets the biggest precision value for any recall bigger than the one given.

$$\text{AP} = \frac{1}{11} \sum_{r \epsilon \{0, 0.1, \ldots, 1\}} p_{\text{interp}}(r) \tag{3.4}$$

$$p_{\text{interp}}(r) = \max_{\tilde{r}:\tilde{r} \geqslant r} p(\tilde{r}) \tag{3.5}$$

For comparison we use the COCO evaluation metrics for determining our AP values which differ slightly from the VOC definition of AP. First is that COCO defines looks at AP at different IoU thresholds, IoU threshold of 0.5, 0.75 and [0.5:0.95] are used in

comparisons. AP at IoU threshold 0.5 and 0.75 are denoted as AP50 and AP75 in this paper. AP[0.5:0.95] defines the mean average precision of every AP from IoU threshold 0.5 to IoU threshold 0.95 stepping up by 0.05 defined in equation 3.6. It will be denoted as AP in this paper.

$$mAP = \frac{AP50 + AP55 + ... + AP95}{10} \tag{3.6}$$

Detectron2 implements an evaluator method called COCOEvaluator that calculates the COCO AP metrics as its training and can be used afterwards to get the final results. Darknet does not implement a builtin method of getting coco methods but does include a mAP feature on its detector. We can calculate the mAP or AP50 at different IoU thresholds to get not only the AP50 and AP75 but also calculate all of the AP from IoU 0.5 to 0.95 to get the AP[0.5:0.95] main metric. A main issue with darknet is that it will only log the map in the terminal log not as a file, so the only way to keep track of the metrics is to save the terminal logs and to parse the output strings for each of the values on each batch iteration. This was implemented by using a regular expression to find each of the AP and loss values in the log which could then be used to chart out the training metrics.

It is also of interest to show the AP during the training and how it changed as well as the total loss or error after each batch sample. This is of interest as we would like to know how quickly each model converged to a given AP on their validation set. We don't need to know the FPS during training as it shouldn't change. The COCOEvaluator will also save these metric during the training process so we just had to add the evaluator to our trainer to check after a certain number iterations. For speed sake we decided to set our testing evaluator to test every 500 batches.

Chapter 4

results

For this thesis there are 3 main metrics to compare the 2 object detection methods being mean average precision. The average frames per second (FPS), the mean average precision (mAP) as well as other AP metrics like AP50 and AP75 along with the Loss values for each trained model.

4.1 Comparison of YOLOv4 and Faster R-CNN

As explained in section 3.4 and 3.5 we used COCO AP metrics and the FPS when detecting images with the validation set as our reference. Table 2 shows the results for each model tested. Note that the YOLOv4 with input size 416x416 for its backbone had two different models made for it based as the first model failed to log its training results, the one in the figure shows the results from the second model that was able to log its results.

Model	Backbone	size	batches	AP	AP50	AP75	FPS
Yolov4	CSPDarknet53	416	26000	65.96%	79.53%	77.32%	49.5
Yolov4	CSPDarknet53	512	26000	62.57%	81.58%	76.90%	42.5
Faster R-CNN	resnet50	224	26000	62.91%	80.66%	75.96%	4.41
Faster R-CNN	resnet101	224	52000	57.32%	76.22%	70.89%	3.93

Table 2: AP metrics and FPS table for Faster R-CNN and YOLOv4 models evaluated on a Tesla P100 GPU

The conclusions from these results show good signs for YOLOv4 over Faster R-CNN though with some problems with the results from YOLOv4. First is the AP values, regardless of input size the AP values for YOLOv4 showed significantly better results than that of Faster R-CNN by at least a small margin. The only exception is the AP50 value for the 416 input model that was slightly lower than the resnet50 model of Faster

R-CNN meaning that when the IOU is lower it is more likely for the resnet50 model to correctly make a prediction than for the YOLOv4 model. The AP of YOLOv4 416x416 is about 3 points higher than the resnet50 Faster R-CNN model and about 8 points higher than the resnet101 model. The YOLOv4 512x512 model gets lower scores but not by much with it being outperformed in AP by resnet50 Faster R-CNN model by about 0.4 points and still beating resnet101 by around 5 points.

The FPS of the two models has a incredibly large difference compared to their AP score where YOLOv4 regardless of input size had a FPS at least 38 images more every second. This result was expected, in the YOLOv4 paper they show that YOLOv4 had a similar difference in FPS. They compared the model to Faster R-CNN using resnet50 and found a FPS of 9.4 for Faster R-CNN while getting 54fps for YOLOv4 416x416[3] when trained on the COCO dataset. While this is not trained on the same dataset or using the same hardware the difference in the FPS shows a comparable gap to what was observed in our results. Even excluding the difference between the models that give such a big speed increase the difference in the backbones speed also give YOLOv4 an advantage since as shown in the YOLOv3 paper were the darknet53 framework was introduced it showed it getting a higher FPS on average than that of Resnet101[30]. The reason YOLO is much faster than Faster R-CNN is down to its architecture. The one stage detector does its region proposals and classification in parallel significantly speeding it up while Faster R-CNN has to do these steps seperately even with the new RPN; the RPN also has to analyse images in sequence slowing the network down more.

One issue with the trained YOLO models however is that it appears to be overfitting as it is trained. Figure 10 shows the AP50 as the size 416 model was trained, while initially the AP50 was very high peaking at around 87% by the time it reached the final batch it had dropped by 10% to an AP50 of 76%.

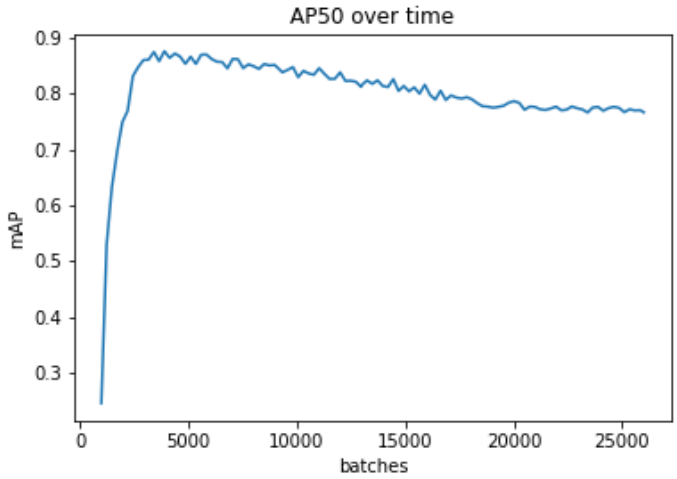


Fig. 10: AP50 per batch while training 416x416 Yolov4 model 2

Interestingly this trend wasn't shown as much with AP75 as seen in figure 11, meaning

the model has an easier time with predictions that have a higher overlap with the ground truth. This is an issue that was probably caused by the learning rate being too high which caused the over fitting.

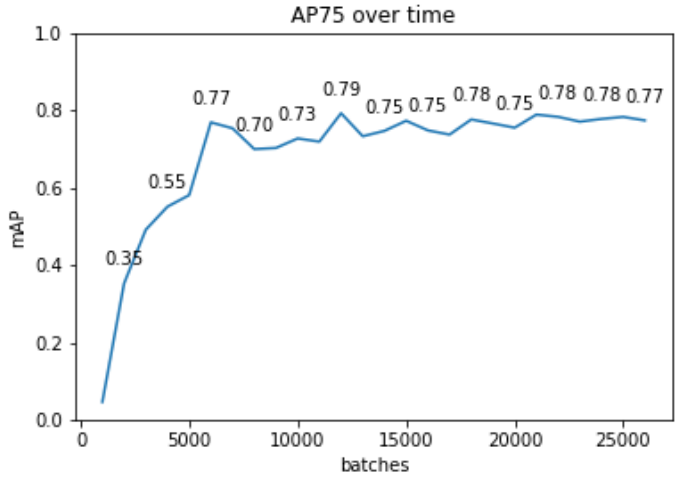


Fig. 11: AP75 per batch while training 416x416 Yolov4 model 2

The Loss during training also didn't show any reasons for this which is why it is assumed it may be an issue with the learning rate. The loss was converging towards 0 as seen in figure 12.

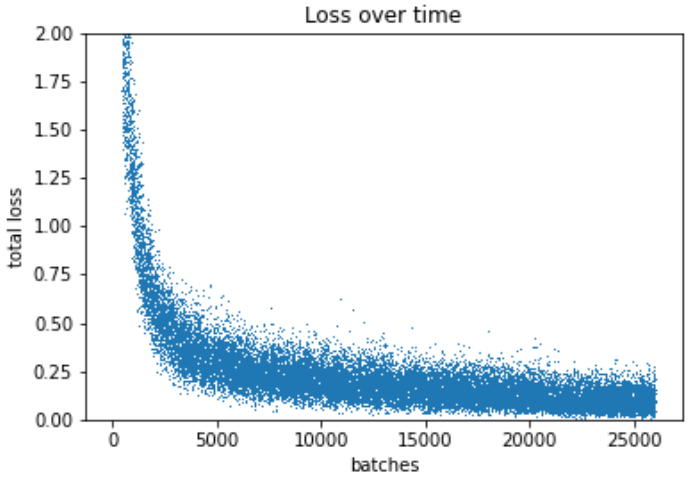
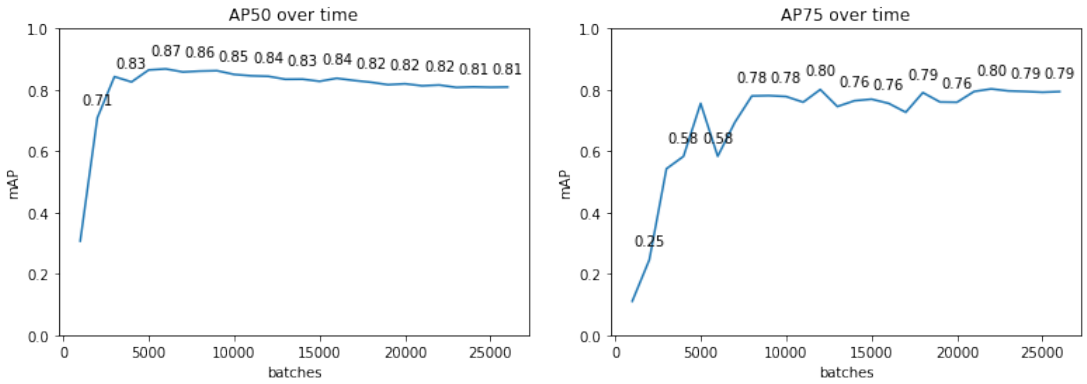


Fig. 12: Loss per batch while training 416x416 YOLOv4 Model 2

Before the newer 416x416 model another one was trained but the logging for the training data failed so only the AP50 during training could be gotten based on the using the logged weight files. Due to this the newer YOLOv4 416x416 model was trained instead. The AP50 of that model did begin to shrink but it did it at a much lesser degree only shrinking around 6% instead of 11% shown in the current model. Figure 13 shows the AP50 in (a) and AP75 in (b) during the training of the original model. The only

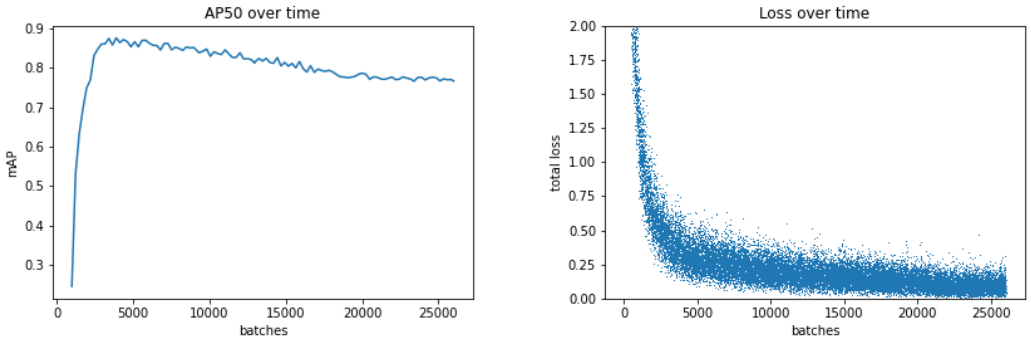
difference in settings between the two models was in the configuration file were in model 1 the steps section was based on an iteration of 13000 iterations instead of 26000 by mistake for the first 13000 iterations. The steps counter tells the trainer during which batch iteration to augment the data to different input sizes.



(a) AP50 (b) AP75

Fig. 13: AP50 and AP75 during training of YOLOv4 416x416 model 1

The same issues can be seen with the 512x512 trained model showing that it isn't an issue of the input size but rather the learning rate or some other factor.



(a) AP50 (b) loss

Fig. 14: AP50 during training of YOLOv4 512x512

Both the Resnet50 and Resnet101 models of faster R-CNN didn't experience this issue during their training as shown in figure 17 and figure 16

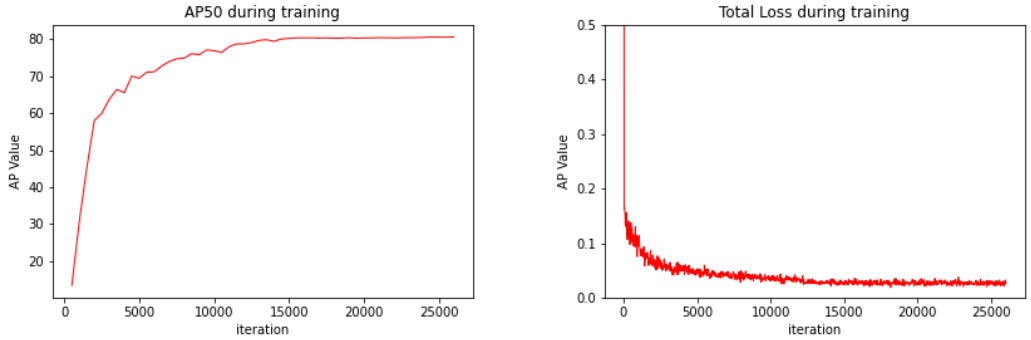


Fig. 15: Faster R-CNN Resnet50

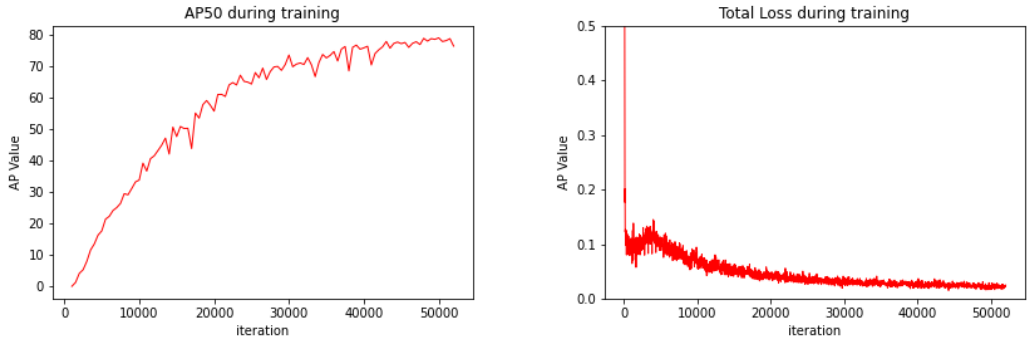


Fig. 16: Faster R-CNN Resnet101

Chapter 5

conclusion

Based on the results shown in this thesis it can be concluded that You Only Look Once version 4 is better than Faster R-CNN in not only speed but also in accuracy. It was found that YOLOv4 even with the overfitting issues still had FPS values at least 30% higher than Faster R-CNN and AP values at least 1% greater than the Faster R-CNN models.

5.1 further work

In this thesis the predictive capability and speed of Faster R-CNN and YOLOv4 were compared using the LISA dataset. It was shown in the results that YOLOv4 was both faster and better at predicting street signs. However there are limitations to what has been researched and further studies will need to explore other parts of the autonomous vehicle requirements and how it relates to the use of YOLOv4 and Faster R-CNN when making a decision.

detection functionality

Once a detection has been made the autonomous vehicle will then need to use the prediction to determine where around the car it is in a 3D space and then act accordingly to the data. One way in which object detection and methods and autonomous vehicles have been able to work in conjunction is with using subtasks like the Simultaneous Localization And Mapping (SLAM) which has been investigated previously[20, 44]. SLAM isn't a single algorithm but an entire field with multiple approaches including the Visual SLAM [14] field that focuses on how to use visual data to create a map for driving. There should be some study on how each of these object detection software work in conjunction with SLAM technologies like LiDAR which has been explored previously[42].

Followup comparison to YOLOv5

Yolo version 5 was released by Ultralytics in August 2020 by Glenn Jocher[15]. While no academic paper has been presented by Glenn or anyone from Ultralytics there has been some results with YOLOv5 with generally good results. Jiawei Xing and Wei Qi Yan[41] created a traffic sign recognition system based on an edited version of YOLOv5 and Faster R-CNN. Their augmented version of Faster R-CNN was shown to be getting higher mAP scores than basic Faster R-CNN with the base version getting a mAP of 80.3% while theirs got an additional 15 points at 95.3% mAP. It did this with comparable speeds as well.

Larger Dataset

In the experiments for this thesis we used the LISA data set but ended up splitting the dataset into 3 distinct groups to separate for testing and training portions of the model, however we didn't use the entire data set due to the low count of images within their classes. At the same time we ignored the largest data class the stop sign because both networks should have already been pretrained on the image before based on assumption. For future review of the two networks it would give more accurate numbers if the networks were trained on larger datasets with more classes to get a better picture of how the object detectors are able to handle multiple classes.

Bibliography

[1] Avramović, A., et al. "Neural-Network-Based Traffic Sign Detection and Recognition in High-Definition Images Using Region Focusing and Parallelization." IEEE Access, vol. 8, pp. 189855–189868., 2020, IEEE Xplore.

[2] Bengio, Y. "Practical Recommendations for Gradient-Based Training of Deep Architectures." Neural Networks: Tricks of the Trade, 2nd Edition, Springer Berlin Heidelberg, 2012, pp. 437–78.

[3] Bochkovskiy A., Wang C., and Liao G., "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv:2004.10934 [cs, eess](Apr. 2020). `arxiv.org/abs/2004.10934`

[4] Chollet F., "Deep Learning with Python." Vol.1. Manning, 2017.

[5] Du J., "Understanding of Object Detection Based on CNN Family and YOLO." Journal of Physics: Conference Series, Vol.1004, IOP Publishing, pp.1-8, 2018.

[6] Everingham M. et al., "The Pascal Visual Object Classes (VOC) Challenge." International Journal of Computer Vision Vol.88, pp.303–338, 2010.

[7] Girshick R., "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), pp.1440-1448, 2015.

[8] Girshick R. et al., "Rich feature hierarchies for accurate object detection and semantic segmentation." 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580-587, 2014.

[9] Goh, Garrett B., et al. "Deep Learning for Computational Chemistry." Journal of Computational Chemistry, Vol. 38, pp.1291-1307, 2017.

[10] Goodfellow I., Bengio Y., and Courville A., "Deep Learning." MIT Press, pp. 336, 2016.

[11] Gupta A. et al., "Deep learning for object detection and scene perceptionin self-driving cars: Survey, challenges, and open issues." Array Vol.10, p.1-20. 2021.

[12] He K. et al., “Deep Residual Learning for Image Recognition.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770-778. 2015.

[13] Informatik F. et al. “Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies.” A Field Guide to Dynamical Recurrent Neural Networks, 2003.

[14] Jin G., et al. “Keyframe-Based Dynamic Elimination SLAM System Using YOLO Detection.” Intelligent Robotics and Applications, Springer International Publishing, pp. 697–705, 2019.

[15] Jocher G., et al., “ultralytics/yolov5: v3.0.” Github Repository, 2020.

[16] He K., et al., ”Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.37, no.9, pp. 1904-1916, 2015.

[17] Nair V. and Hinton G., “Rectified linear units improve restricted boltzmann machines,” International Conference on Machine Learning (ICML), 2010.

[18] LeCun Y., Bengio Y., and Hinton G., “Deep learning.” Nature Vol.521. pp.436-444, 2015.

[19] Le Cun Y., ”Generalization and Network Design Strategies.” 1989.

[20] Lewis G., “Object Detection for Autonomous Vehicles.” 2014.

[21] Lin T., et al., “Feature Pyramid Networks for Object Detection.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.

[22] Liu S., et al., “Path Aggregation Network for Instance Segmentation.” 2018 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8759-8768, 2018.

[23] Mehrotra K., et al., ”Elements of Artificial Neural Networks.” A Bradford Book, pp. 10-21, 1997.

[24] Michelucci U. “Fundamentals of Convolutional Neural Networks.” Advanced Applied Deep Learning: Convolutional Neural Networks and Object Detection, Apress, pp. 79–123, 2019.

[25] Møgelmose A., Trivedi M., and Moeslund T., ”Vision based Traffic Sign Detection and Analysis for Intelligent Driver Assistance Systems: Perspectives and Survey,” IEEE Transactions on Intelligent Transportation Systems, 2012.

[26] Nielsen M., "How the backpropagation algorithm works." Neural Networks and Deep Learning. Vol.1, Determination Press, 2015.

[27] Olson D. and Delen D., "Advanced data mining techniques." Springer, pp. 138, 2008.

[28] Pang Y. and Cao J., "Deep Learning in Object Detection and Recognition." pp. 21, 2019.

[29] Redmon J. and Farhadi A., "YOLO9000: Better, Faster, Stronger." arXiv:1612.08242, 2016. arxiv.org/abs/1612.08242

[30] Redmon J., and Farhadi A., "YOLOv3: An Incremental Improvement." arXiv:1804.02767, 2018. arxiv.org/abs/1804.02767

[31] Redmon J., et al., "You Only Look Once: Unified, Real-Time Object Detection." 2016 IEEE Conference on Computer Vision and Pattern Recognition(CVPR), IEEE, pp. 779–788, 2016.

[32] Ren S., et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 39, pp. 1137-1149, 2017.

[33] Rezatofighi H., et al., "Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 658–666, 2019.

[34] Rumelhart D., Hintont G., and Williams R., "Learning representations by back-propagating errors," Nature, Vol.323, p.4., 1986.

[35] Stallkamp, J. et al., "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition." Neural Networks, vol. 32, Pergamon, pp. 323–32, 2012.

[36] Tabernik D., and Danijel S. "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition." IEEE Transactions on Intelligent Transportation Systems, IEEE Xplore, Vol. 21, no. 4, pp. 1427–40, 2020.

[37] Uijlings J., et al., "Selective Search for Object Recognition," International Journal of Computer Vision, Vol. 104, pp. 154–171, 2013.

[38] Wang C., et al., "CSPNet: A New Backbone That Can Enhance Learning Capability of CNN." ArXiv:1911.11929 [Cs], Nov. 2019. arXiv.org, arxiv.org/abs/1911.11929.

[39] Wang K., et al. “Automated Road Sign Inventory System Based on Stereo Vision and Tracking.” Computer-Aided Civil and Infrastructure Engineering, Wiley Online Library, vol. 25, no. 6, pp. 468–477, 2010.

[40] Wu Y., et al., ”Detectron2” Github Repository, `github.com/facebookresearch/detectron2`, 2019.

[41] Xing J. and Yan W., “Traffic Sign Recognition Using Guided Image Filtering,” Geometry and Vision, Communications in Computer and Information Science, Springer International Publishing, pp. 85–99, 2021.

[42] Yu Z., et al. “LiDAR Target Fusion and Algorithm Detection Based on Improved YOLO.” Journal of Physics: Conference Series, vol. 1682, no. 1, IOP Publishing, p. 012010, 2020.

[43] Zermas D., et al. “Fast Segmentation of 3D Point Clouds: A Paradigm on LiDAR Data for Autonomous Vehicle Applications.” 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE Xplore, pp. 5067, 2017.

[44] Zhang L. et al. “Semantic SLAM Based on Object Detection and Improved Octomap,” IEEE, Access, Vol. 6, pp.75545 – 75559, 2018.

Appendix A

Custom Dataset preparation from LISA Traffic Sign dataset

In order to train our models we had to convert the LISA dataset into the applicable format for each program we used. Both Detectron2 and Darknet use different formats for defining annotations like bounding boxes and the file types used.

A.1 Detectron2 format

First is the Detectron2 dataset format for custom datasets. Detectron2 uses python dictionaries for its annotations, the items in each dictionary entry is as follows:

```
"file_name": name of the file,
"image_id": unique id representing the image,
"height" and "width": dimensions of the image,
"annotations": a dictionary of information to train the image
```

Where annotations specify a list of each bounding box and its associated category using the following parameters.

```
"bbox": list representing the target bounding box,
        in format [x1, y1, x2, y2] for top left and
        bottom left box point
"bbox_mode": type of bounding box being represented,
"category_id": unique integer representing the category class
```

The LISA dataset includes a comma separated file call allAnotations.csv that has the paths and annotations of every image in the dataset which was used to convert the file over to the detectron2 format. The image width and height wasn't included in the annotations so each image file was read using opencv to get the image shape. "image_id" was set to the name of the image as it just has to be unique and "bbox_mode" was set to the detectron2 constant detectron2.structures.BoxMode.XYXY_ABS which tells

detectron2 that the bounding box is using the top left and bottom right vertices location to describe itself. Finally the category id isn't used as an integer in the LISA annotations file but as the category name string. In order to convert it to we made a switch function in python that returns an index if the inputed name is valid. With this we were able to convert the dataset to the detectron2 format. Next we had to split the dataset into 3 groups, the Training, Testing and Validation dataset. The training dataset is the largest and is used to teach the network the object classes, next is the validation set which is used for getting data metrics during training and the test set was used for testing after training.

It was decided to use the followining 13 classes for training the data set as each class has over 100 images at least. The stop sign class was ignored as both datasets have already been pre-trained on the class. The classes where split randomly with 10% being used for the test images, and 60% of the remaining images being training images, and the remainder being used for validation.

categories	train	valid	test
pedestrianCrossing	740	237	108
signalAhead	618	203	104
speedLimit35	369	117	52
speedLimit25	229	86	34
keepRight	223	71	37
addedLane	207	58	29
merge	185	59	22
yield	151	69	16
laneEnds	133	56	21
stopAhead	113	33	22
speedLimit45	102	32	7
speedLimit30	94	29	17
school	88	33	12

Table 3: categories used for training models along including how the data set was separated for different tasks

Finally, the data set needs the names of the categories saved as metadata as an array called things_classes. Once all of these are specified, the model can then proceed to training. Detectron2 specifies a

For YOLO the input fields are similar but instead of being stored in a python dic-

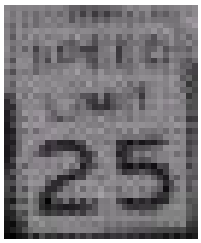
tionary they are instead stored in a text file for each individual image. Each text file contains the image annotations in the following format:

```
category_id xc yc height width
```

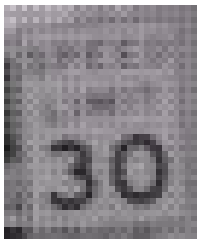
where xc and yc are the center of the bbox as a percent of the image and height and width are the dimensions of the bbox as a percent of the image. So $xc = (x2 + x1)/(2 * image\_width)$ and similarily $width = (x2 - x1)/img\_width$. For each bbox in a image a new row will be written to, the category names will be saved as a text file in the same directory for each subset called _darknet.labels.

Appendix B

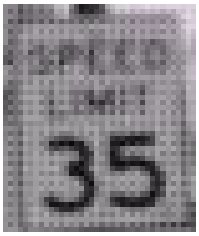
Traffic sign types



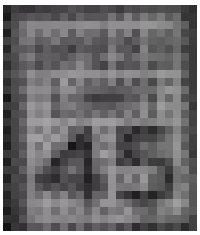
(a) speedLimit25



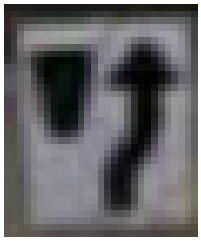
(b) speedLimit30



(c) speedLimit35

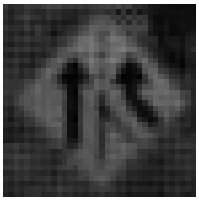


(d) speedLimit45



(e) keepRight

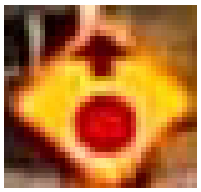
(f) merge



(g) laneEnds

(h) signalAhead

(i) addedLane



(j) stopAhead

(k) yield

(l) school

(m) pedestrian

Fig. 17: examples of traffic signs used in training from the LISA dataset