CHALMERS
UNIVERSITY OF TECHNOLOGY

Modelling temporal context for traffic light recognition using RNNs

Master's thesis in Mathematics

DAVÍÐ FREYR BJÖRNSSON
MATTIAS WESTERBERG

DEPARTMENT OF MATHEMATICAL SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Modelling temporal context for traffic light recognition using RNNs

DAVÍÐ FREYR BJÖRNSSON
MATTIAS WESTERBERG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2021

Modelling temporal context for traffic light recognition using RNNs
DAVÍÐ FREYR BJÖRNSSON
MATTIAS WESTERBERG

Supervisors: Johan Jonasson, Department of Mathematical Sciences
Mohammad Nazari, Scania CV AB
Ludvig af Klinteberg, Scania CV AB
Examiner: Klas Modin, Department of Mathematical Sciences

Master's Thesis 2021
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Modelling temporal context for traffic light recognition using RNNs
DAVÍÐ FREYR BJÖRNSSON
MATTIAS WESTERBERG
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

The purpose of this thesis is to investigate whether or not including temporal context using recurrent neural networks in real-time object detection systems can improve detection performance in traffic light recognition. This was investigated using the DriveU traffic light dataset. Two variations of the YOLOv4 object detection system were created. The first variation is a LSTM which takes as input the bounding boxes predicted by YOLOv4 and outputs updated predictions. The second variation is a modification of the YOLOv4 network in which convolutional layers are replaced with convolutional LSTMs. With a limited number of experiments, it was found that the baseline model outperforms the more complicated sequential models. However, there is evidence that this is due to the sequential training strategy since the YOLOv4 baseline was outperformed by some sequential models when it adopted the sequential training strategy. The baseline YOLOv4 model achieved best performance on a held-out test set. The best sequential model achieved lower detection performance. When the baseline YOLOv4 was trained with the sequential training strategy, it achieved worse performance than the sequential models. Modelling temporal context using recurrent neural networks may improve detection performance, but answering the question requires an exhaustive search for a training strategy and model architecture. The analysis conducted in this thesis provides no evidence that modelling temporal context with YOLOv4 improves traffic light recognition performance on the DriveU dataset.

Keywords: object detection; traffic light recognition; recurrent neural networks; temporal context; YOLO

Acknowledgements

This thesis was conducted with generous support from many people. First, we would like to thank our industry supervisors Shahab, Ludvig and Lisanu. We have also received vital support from our academic supervisor Johan and examiner Klas. Scania has provided us with a computer cluster of high-performance machines which was utilised during the project. We want to thank Addi for the support with using this cluster. Additionally, we would like to thank the EARA team at Scania for all the fruitful discussions about the project. Lastly, we would like to thank our families and friends for their support throughout the process of writing this thesis.

Davíð Freyr Björnsson & Mattias Westerberg, Gothenburg, May 2021

Contents

List of Figures

List of Tables

List of Algorithms

Figure 1: A traffic scene from Germany with multiple traffic lights. The bounding boxes enclosing the traffic lights are predictions from an object detection system. [11]

1 Introduction

Various systems for autonomous driving have been developed throughout the years. These systems aim to make the job of the driver easier while increasing safety at the same time. There is evidence of rearview cameras and rear parking sensors decreasing the probability of collisions [2]. Additionally, a study found that lane departure warning and lane-keep assist systems decrease the driver injury risk [3]. This work is concerned with developing a traffic light recognition (TLR) system that can be useful in autonomous vehicles. A typical scene with multiple traffic lights is illustrated in figure 1. Previous work in TLR mainly concerns single frame traffic light recognition [[4], [5], [6], [7], [8]]. While these papers show impressive performance, more information about the traffic situation can be extracted when considering a sequence of frames. If a traffic light is in a particular location in a frame, it is likely to be in a similar location in the next frame. The traffic light state is also highly dependant on the previous frame. This observation has been used for improvements in the general field of object detection in video sequences [9]. There are several approaches in the field of deep learning to improve object detection in video with the use of recurrent neural networks (RNN), a machine learning model for sequential data [10]. Important information from previous frames can be remembered in an RNN and potentially result in a more accurate TLR system. To the best of our knowledge, this has not been investigated for the case of TLR.

1.1 Background

Scania is moving towards more sustainable transportation systems. Self-driving vehicles are one of the many investments into this long-term project which Scania estimates will lead to greater sustainability and automotive safety. Extensive research has to be conducted in the field of traffic light recognition before fully autonomous vehicles can enter the roads.

1.2 Aim

There are many approaches to consider when including the temporal context in a model. It is not feasible to investigate all of them, so the investigation is limited to RNNs. The thesis aims to investigate if RNNs can improve the detection accuracy and speed of TLR.

1.3 Limitations

There are many techniques for optimising the training of object detection systems (ODS). This thesis aims not to create the best possible ODS but rather to measure the effect of adding RNNs in deep learning models in the field of TLR. Thus, an exhaustive search for the optimal network architecture will not be performed. Due to both hardware and time constraints, extensive hyperparameter tuning for models cannot be conducted. Instead, the focus is put on a specific model and how RNNs affect its performance. Even if it was possible to perform an exhaustive search for the optimal model, the results would still not be guaranteed to hold for other data than used in the thesis.

2 Theory

Video Object Detection Systems (VODS) are highly complex architectures, often combining many components found in the field of deep learning. The core components are ODSs and RNNs. Those components comprise many systems and operations, e.g. neural networks, convolutions, recurrent models, activation functions and training schemes. This chapter deals with the fundamental theory that the proposed VODS builds upon.

2.1 Deep learning foundations

Deep learning approaches to object detection utilise convolutional neural networks (CNN) for extracting semantic information in images. A naive way of extracting information in images would be to have one input neuron per pixel. However, a CNN is a special case of the neural network model, made to improve this extraction process with the added benefits of reduced training times and lower risk of overfitting. When concerned with video object detection, it is also beneficial to factor in the relationship between frames in a sequence. An RNN can model this relationship.

2.1.1 Feedforward neural networks

The core unit in a neural network is the neuron, which can be considered a vertex in a computational graph holding one value. It is structured as a graph without cycles, where one end of the graph contains the input neurons and the other end the output neurons. A neural network has one or more layers with a varying number of neurons in each layer. The value of a neuron is computed by an activation function of the weighted sum of the outputs of the neurons in the previous layer. The weights are adjusted so that the neural network models the input data. The error is the difference between the ground truth and the prediction.

For a neural network with n layers, the value of the k'th neuron in layer $l \in \{1, ..., n\}$ is $x_l^k := g(b_l^k + \sum_k w_{l-1}^k x_{l-1}^k)$. This is a recursive formula, since all neurons x_{l-1}^i will be dependent on neurons in layer $l-2$. The base case x_0^k is given by the input data, together with the ground truth. Common choices for a activation function g are the sigmoid function σ and tanh.

The activation function should be differentiable since the weights w_l^i are updated with the gradient of the loss function $\ell(\hat{y}, y)$ with respect to w_l^i, where $\hat{y}$ is the prediction (the network's output layer) and $y \in \mathbb{R}^M$ is the true ground. For classification tasks, a common loss function is cross-entropy (1) while the squared error is a common choice for regression tasks. An example neural network is displayed in figure 2. The hidden layer consists of 2 neurons, each of which takes as input all neurons in the previous layer; therefore it is called "fully connected".

$$\ell_{\text{cross entropy}}(y, \hat{y}) = -\frac{1}{M} \sum_{i=1}^{M} y_i \cdot \log(\hat{y}_i) \tag{1}$$

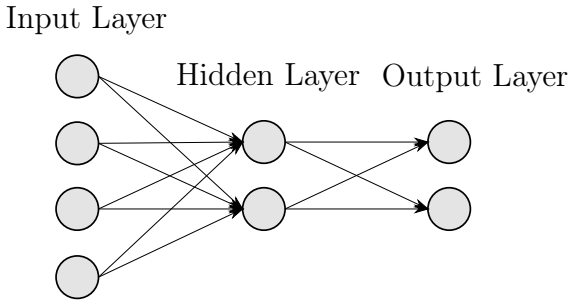


Figure 2: A fully connected neural network with 4 input neurons, 2 hidden neurons and 2 output neurons.

The objective of a neural network is to model an input-output mapping. Finding a mapping means adjusting the weights and biases in the neural network so that they model the true mapping as closely as possible. There is a wide range of training schemes to accomplish this goal. A training scheme consists of forward propagation of the input values, i.e. calculating the output of a neural network, and backpropagation, which is propagation of the gradient of the loss back in the network in order to update the parameters. The parameter update has a magnitude proportional to the gradient with respect to the prediction error, but there are many optimisation strategies for how the gradient can be utilised [12, p. 164-223].

2.1.2 Optimisation strategies

There are many techniques and optimisation algorithms available to get the best out of the training phase. The field of deep learning is not in agreement on what optimisation algorithm to use and what practices are best in general. A relevant subset of techniques and stochastic optimisation algorithms is presented here [12, p. 306-307].

Two common optimisation algorithms are stochastic gradient descent (SGD) and Adam. SGD is an optimisation algorithm that takes a subsample of the datapoints (commonly referred to as a batch) and uses it to calculate the gradient with respect to a loss function ℓ. The gradient is propagated through the network and is used to update the weights and biases. A pseudocode example for learning a parameter θ with SGD is given in algorithm 1. The algorithm requires a learning rate schedule $\epsilon_1, \epsilon_2, ...$, which is used for scaling the gradient. A sufficient convergence condition is that

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \text{ and}$$

$$\sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

In practise, the learning rate is constant after a couple of iterations [12, p. 290-291].

Algorithm 1 SGD update for a parameter θ

Input: Learning rate schedule $\epsilon_1, \epsilon_2, ...$
Output: Parameter θ

procedure SGD($\epsilon_1, \epsilon_2, ...$)
    $k \leftarrow 1$
    while stopping criterion not met **do**
        Sample a batch from the training set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$
        Compute gradient estimate $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} \ell(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$
        Apply update $\theta \leftarrow \theta - \epsilon_k \hat{\mathbf{g}}$
        $k \leftarrow k + 1$

A variation of the SGD optimiser is SGD with momentum, which works just like algorithm 1, but in each step a weighted sum of current and past gradient estimates is calculated, where the weights are decaying exponentially for previous gradients. [12, p. 292-293]

Tuning hyperparameters for optimisation algorithms can be time-consuming, while model performance is largely dependent on them. An attempt to combat this issue is an adaptive learning rate. Adam takes into account the learning rates of past iterates. The algorithm utilises the idea of moments in combination with the average of the past squared gradients. The algorithm can be studied in more detail in algorithm 2. Adam is considered robust to different choices of hyperparameters. [12, p. 306-307]

Algorithm 2 Adam optimiser algorithm [1]

Input: α: step size.
Input: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for moment estimates.
Input: $f(\boldsymbol{\theta})$: objective function with parameters $\boldsymbol{\theta}$.
Input: $\boldsymbol{\theta}_0$: Initial parameter vector
Output: $\boldsymbol{\theta}_t$

procedure ADAM($\alpha, \beta_1, \beta_2, f, \boldsymbol{\theta}_0$)
    $m_0 \leftarrow 0$
    $v_0 \leftarrow 0$
    $t \leftarrow 0$
    while $\boldsymbol{\theta}_t$ not converged **do**
        $t \leftarrow t + 1$
        $g_t \leftarrow \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1})$
        $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
        $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
        $\hat{m}_t \leftarrow m_t / (1 - (\beta_1)^t)$
        $\hat{v}_t \leftarrow v_t / (1 - (\beta_2)^t)$
        $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

Optimisation algorithms aside, other useful techniques can be applied to neural networks to assist in the optimisation process.

The distribution of each layer's input changes during the training process of a neural network. This requires precise initialisation of the network hyperparameters. To combat this, S. Ioffe et al. [13] proposed a method called batch normalisation.

This is an operation which is usually applied right before an activation function. Let X be a batch with m observations and n predictors, X_{ij} denotes the i'th observation in the j'th predictor. Then the normalised batch X' is calculated as:

$$X'_{\cdot j} = \frac{\sum_{i=1}^{m} X_{ij} - \mu_j}{\sigma_j} \forall j \in \{1, ..., n\}, \tag{2}$$

where μ_j and σ_j are the mean and standard deviation of the j'th predictor over m observations. Batch normalisation helps to stabilise the training process and can dramatically decrease the training time [12, p. 313-317].

Dropout is another technique for assisting the optimisation process. It involves setting a random subset of neurons to zero during a forward pass. This technique is a way to emulate the training of an ensemble of sparse networks so that it will not be too reliant on a small subset of neurons. In other words, dropout is a regularisation technique [12, p. 255].

Common choices for activation functions are, as stated earlier, sigmoid and tanh. However, it has been found that those activation functions are likely to exacerbate the vanishing gradient problem. This occurs commonly for deep networks because of the many multiplications in the long chain of gradients. It means that the training may be in a phase where it cannot update some of its parameters and thus cannot learn from the data. To combat the vanishing and exploding gradient problem, one can use the rectified linear unit (ReLU) activation function:

$$g(x) = \begin{cases} x, & \text{for } x > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The ReLU function, however, has another problem: the dead gradient. This occurs when most inputs to the function are negative. To combat this problem, we allow for weakly negative outputs. The new activation function is called leaky ReLU:

$$g(x) = \begin{cases} x, & \text{for } x > 0 \\ kx, & \text{otherwise,} \end{cases} \tag{4}$$

where k is a small scalar. A variation on the leaky ReLU is to let k be a learnable parameter. This activation function is called parametric ReLU [12, p. 189-190].

When the output takes on a finite number of discrete values, a common activation function is the softmax function. For an input vector $x \in \mathbb{R}^K$, the corresponding softmax transformation is

$$\phi(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{K} \exp(x_j)} \quad \forall\, i = 1, ..., K. \tag{5}$$

Therefore all values of the transformed input vector are between 0 and 1 and together sum to 1 [12, p. 180-181].

2.1.3 Convolutional neural networks

A convolutional neural network is a special case of a neural network characterised by two key operations: convolution and pooling operations. The convolution operation involves calculating the Hadamard product between the input matrix and a kernel matrix and sum the resulting matrix elements for each pixel. The output matrix is called a feature map, and a convolutional layer in a neural network typically calculates several feature maps of the input matrix, one per kernel. The network learns the weights of the kernel. Therefore a kernel window with multiple activations gets associated with only one output [12, p. 327-328]. In the case where our input I of size $W \times H$ (e.g. a grayscale image) and kernel K of size $M \times N$ are both two-dimensional, then a feature map F is calculated as

$$F = K * I = \sum_{m=1}^{M} \sum_{n=1}^{N} K_{mn} I_{x-m,y-n}, \qquad \forall\,(x, y) \in (W - M) \times (H - N). \tag{6}$$

This convolution operation is commutative and is denoted by $*$. The convolution operator for the two-dimensional case is visualised in figure 3 [12, p. 327-328]. When the feature map is three-dimensional, as is the case for RGB images, the convolution operation is applied to each colour channel using separate kernels.

There is also a possibility of controlling the step size of the kernel, called the stride, which will affect the size of the resulting feature map. Equation (6) has a stride of $(1, 1)$ which means that it will move the kernel 1 element horizontally and 1 element vertically over I for each feature map pixel. A stride of $(2, 1)$ will halve the feature map size by stepping two elements horizontally instead, i.e. every other element of W would be removed.

A generalisation of the equation (6) is achieved by introducing padding, e.g. zero padding, which is adding zeros around the input matrix I to make it taller and wider. This operation avoids having a smaller feature map than the input matrix, beneficial in applications where the spatial context is of interest [12, p. 343].

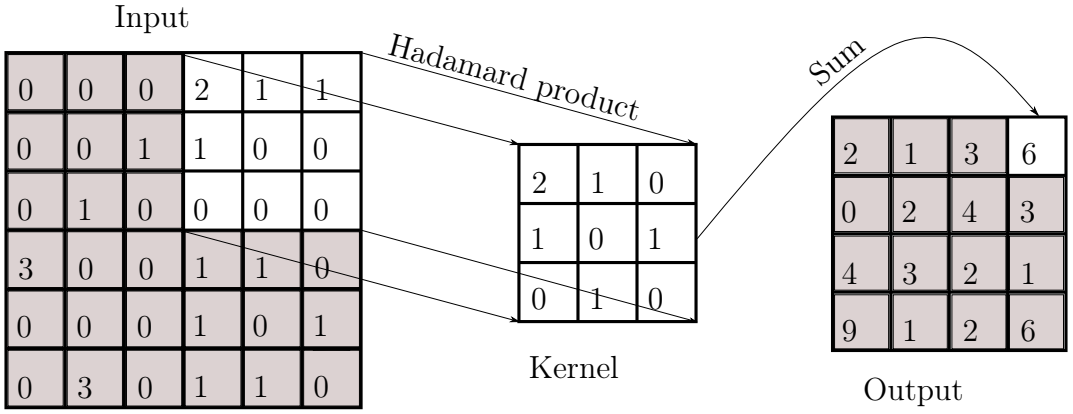


Figure 3: A visualisation of the convolutional operation using a two-dimensional input of size 6×6 and a kernel of size 3×3.

The benefits of using convolutions in comparison to fully connected layers are twofold. They typically reduce the number of network parameters since $M \times N \times$ number of kernels is usually smaller than the number of elements in the input matrix. Secondly, they

effectively increase the size of the training dataset in relation to the number of parameters since the same filters are applied to every kernel window of each image. Convolutions also have a translational invariance property. This means that when a subset of image pixels is shifted, the corresponding feature map elements are shifted accordingly. [12, p. 331-335].

For upsampling a feature map one can use transposed convolutions, where the input elements get associated with multiple outputs. The notion of padding and stride is also used for this operation. Given a set of feature maps F of size $|F| \times W \times H$, a stride s, an input padding p_i, output padding p_o, dilation value d and a kernel size $M \times N$, the transposed convolution operator returns an output of size $|\tilde{F}| \times \tilde{W} \times \tilde{H}$, where

$$\tilde{W} = s(W-1) - 2p_i + d(M-1) + p_o + 1 \tag{7}$$

$$\tilde{H} = s(H-1) - 2p_i + d(N-1) + p_o + 1. \tag{8}$$

The procedure to create an output feature map comprises three steps.

1. Padding the input according to p_i

2. Create an output feature map equal to the feature map but with d padding elements (e.g. zeros) inserted between each row and column. Lastly, pad the output feature map according to p_o

3. Apply a convolution operation (6) to the feature map with the given kernel and stride

This procedure can be conducted on the same feature map as many times as the desired size of the output channel dimension [14].

Another operation that is common in combination with convolutions is pooling. A pooling layer has no parameters, and the motive is to reduce the size of a feature map. There are different types of pooling layers, where a common one is max-pooling. The max-pooling operation extracts the highest pixel value in each kernel of the image. A potential problem with pooling operations is that they cause loss of spatial information of objects in the original matrix, which is a drawback for applications relying on the preservation of spatial context. A visualisation of max-pooling is displayed in figure 4 [15].

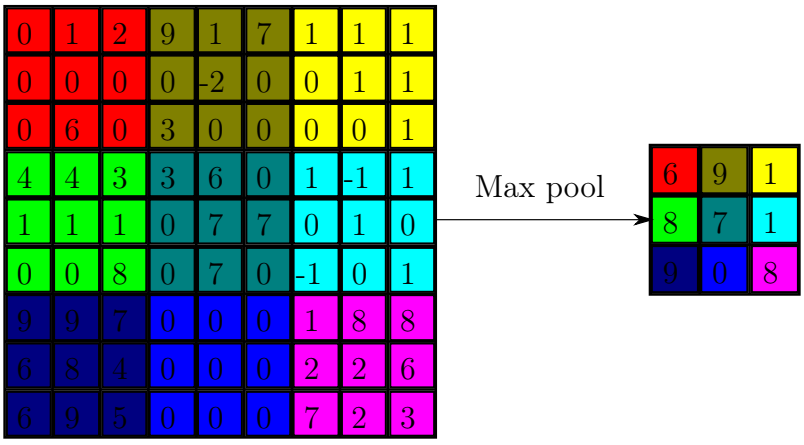


Figure 4: A visualisation of the two-dimensional pooling operation. In this case, max-pooling with a kernel of size 3×3 and a stride of 3×3.

A common design of a CNN [16] is successive use of the pattern

1. Convolution
2. Batch normalization
3. Activation
4. Pooling

2.1.4 Recurrent Neural Networks

An RNN is used for modelling sequential data by using connections to later realisations of itself. The core components of a simple RNN are an activation function g, a loss function ℓ and a hidden state vector h_t. The output vector at time step t is denoted as $\hat{y}_t$.

To perform forward propagation we begin with an initialised hidden state h_0, then for each time-step $t, t = 1, ..., T$ we use the following update equations:

$$h_t = f(W_x^{(t)} x_t + W_h^{(t)} h_{t-1} + b_h) \tag{9}$$
$$o_t = W_o^{(t)} h_t + b_o \tag{10}$$
$$\hat{y}_t = g(o_t), \tag{11}$$

where f is a transition function that is applied at each time step, $W_h^{(t)}$, $W_x^{(t)}$, $W_o^{(t)}$ are weight matrices and b_h and b_o are biases [12, p. 372-374].

The output of the network, denoted o_t, is used in a loss function ℓ to measure the distance between the predicted labels $\hat{y}$ and true labels y. The total loss is the sum of the losses at all timesteps: $\ell = \sum_{t=1}^{T} \ell_t(\hat{y}_t, y_t)$. A visualisation of the RNN is displayed in figure 5.

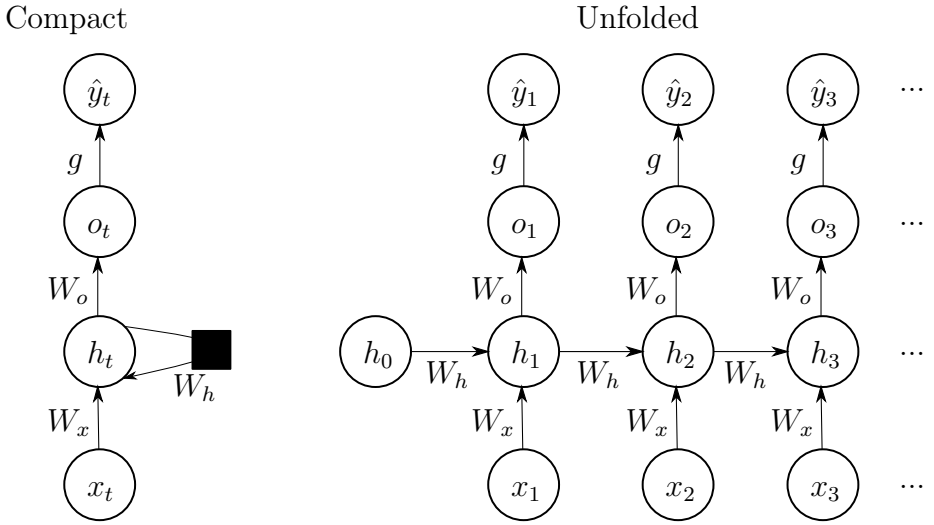


Figure 5: An illustration of the RNN model in compact and unfolded form.

With long sequences, the network becomes deep. As with all neural networks with many hidden layers, they can suffer from the vanishing or exploding gradient problem. To

illustrate this problem, consider a network with only one neuron per layer but many layers. The nested activation functions give the output of the network

$$\hat{y}_T = g(W_h^T g(W_h^{T-1}...g(W_h^2 g(W_h^1 x - b_h^1) - b_h^2)... - b^{T-1}) - b^T). \tag{12}$$

A single SGD update of the weight vector W_h

$$W_h^{(t+1)} := W_h^{(t)} + \alpha \frac{\delta \ell}{\delta W_h^{(t)}}, \tag{13}$$

where α is the learning rate. The computation of the gradient $\frac{\delta \ell}{\delta W_h}$ involves multiplication with the term

$$\frac{\delta W_h^T}{\delta W_h^t} = \Pi_{k=T}^{t+1} g'(b^k) W_h^k. \tag{14}$$

Weights are often initialised with small values. However, if the weights are small, then the product $g'(b_h^{t-1})W_h^t$ is likely to be smaller than one. If T is large, the gradient $\frac{\delta \ell}{\delta W_h}$ becomes close to zero, and the training, therefore, slows down considerably. On the other hand, if the weights are large in magnitude then the gradient factors are likely to be larger than one. In that case, the gradients increase exponentially with network depth. This is more common in later stages of the training since the weights tend to grow over time. These scenarios are descriptions of the vanishing and exploding gradient problem. These problems cause difficulties for recurrent neural networks to learn long-term dependencies [17, p.131-132].

One way of tackling this problem is to have the hidden units with self-loops where another hidden unit controls each loop, called a forget gate. This gate allows the network to retain long-term knowledge and learn from the data what it should forget and remember. This is the idea behind gated RNNs and specifically the long short-term memory (LSTM) model. There are three types of gates in an LSTM: forget gate, input gate and output gate. Those are visualised in figure 6 and the corresponding update formulas are listed in the equations (15)-(20) [14]. The key idea is to learn what information should be remembered and what should be forgotten [12, p. 404-406].

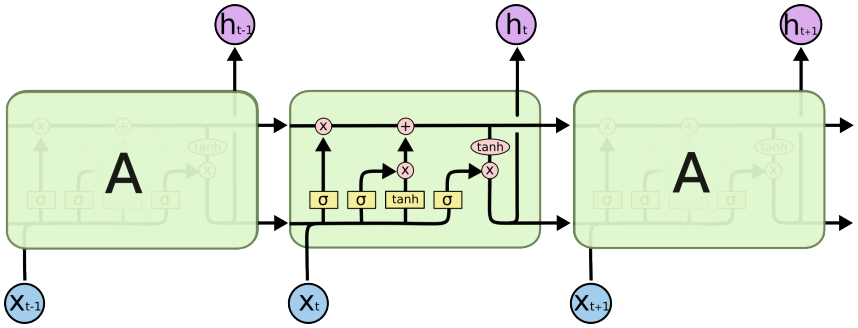


Figure 6: An illustration of the LSTM model with three time steps. [18]

The output of an LSTM at time step t is denoted o_t and the input as x_t.

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{15}$$
$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{16}$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{17}$$
$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{18}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ g_t \tag{19}$$
$$h_t = o_t \circ \tanh(c_t) \tag{20}$$

Although LSTMs reduce the risk of exploding gradients, they do not eliminate it. It is therefore vital to be aware of methods to tackle that problem when using LSTMs. The authors of [19] propose a method that involves scaling gradients when their norm reaches a certain threshold. This introduces another hyperparameter, but the paper's authors find that looking at the average norm over many updates provides a good foundation for deciding on a threshold.

There is a recurrent model which combines convolutional neural networks with an LSTM. It is called a convolutional LSTM (ConvLSTM) and the corresponding update equations are given by (21)-(26). The drawback the authors saw in using a fully connected LSTM is that no spatial information can be encoded because the image data would have to be flattened into a vector. Using a ConvLSTM in a neural network provides the network with visual memory of previous images in an image sequence [20].

$$\mathcal{I}_t = \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \tag{21}$$
$$\mathcal{F}_t = \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \tag{22}$$
$$\mathcal{G}_t = \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \tag{23}$$
$$\mathcal{C}_t = \mathcal{F}_t \circ \mathcal{C}_{t-1} + \mathcal{I}_t \circ \mathcal{G}_t \tag{24}$$
$$\mathcal{O}_t = \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \tag{25}$$
$$\mathcal{H}_t = \mathcal{O}_t \circ \tanh(\mathcal{C}_t) \tag{26}$$

An alternative to LSTMs when it comes to gated RNNs is gated recurrent units (GRU), which is a model with fewer parameters. It is a similar architecture but without an output gate and a cell state. The update equations are listed in (27)-(30), where r is the reset gate, z is the update gate, $\tilde{h}$ is the hidden state and h is the output [21].

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{27}$$
$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{28}$$
$$\tilde{h}_t = \phi(W_h x_t + U_h(r \circ h_{t-1})) \tag{29}$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t \tag{30}$$

Similarly, a convolutional GRU model is achieved by exchanging the matrix-vector multiplications for convolution operations, considering the weight matrices as kernels. [22]

2.2 Object detection systems

There are four tasks that deep learning-based object detection systems need to perform efficiently and accurately: extract features, generate candidate regions, classify and refine

bounding box locations. The process of extracting features from an image using deep learning can be done with a typical classification network for images, such as a residual neural network (ResNet). A common architectural pattern in object detection networks is where the following operations are applied repeatedly in order [[16], [23]]

1. Convolution
2. Batch normalisation
3. Activation

This part of an object detection system is called the backbone network. Deep learning object detectors either tackle task two and three in one or two stages that comprise the network's head. Two-stage detectors divide the task into a region proposal step and a classification-regression step, where region proposals are initial guesses of where an object might be. For the classification step, a background class is included in addition to the classes of interest. This class represents boxes not containing any objects. The background class is used during inference for filtering out boxes that supposedly does not contain an object. On the other hand, one-stage detectors take as input an image and a set of predefined boxes and perform the classification and regression step directly, and a confidence score determines if a box contains an object or not. Sometimes there is an intermediate step between the backbone and the head for processing the features. This stage of a detection system is called the neck [24, p. 84-85].

This section provides explanations of some fundamental concepts that are often used in the object detection literature.

2.2.1 Anchor boxes and confidence scores

Anchor boxes are used in several object detection systems. They are predefined bounding boxes that the network uses as initial guesses of the true bounding boxes. Those are predefined boxes that can, for example, be selected by running k-means on the width and height-space of the boxes in the training data. Suitable aspect ratios and sizes are determined from the training data. Object detectors then refine these initial guesses by predicting the deviation from them. This prediction often includes a confidence score that ranges between 0 and 1. This score represents how confident the algorithm is that the bounding box contains an object. Predicting the confidence score is a regression problem. Therefore, the regression output from such object detectors consists of tuples of five values - four values determining the location and size and one confidence score [25].

In object detection, an optimal system finds bounding boxes that perfectly localise the ground truth boxes and predicts the correct classes. The localisation is often quantified by Intersection over Union (IOU), the fraction of overlapping area between the ground truth and the prediction. The label we use for training the confidence regressor of an ODS is the IOU. The confidence of a bounding box is thus the predicted overlap with a ground truth object. This feature of the system is used during inference to filter out predictions with low confidence.

2.2.2 Non-maximum suppression

Non-maximum suppression (NMS) is a post-processing method in object detection. Most likely, some bounding box predictions will refer to the same object, which results in many redundant predictions. NMS is a greedy algorithm for removing duplicate predictions by selecting the most confident boxes in each region. It is specified in algorithm 3 [26].

Algorithm 3 Non-maximum suppression

Input: List of bounding box predictions $\mathcal{B}$, confidence threshold t, NMS threshold τ
Output: A reduced list of bounding box prediction $\tilde{\mathcal{B}}$

procedure NMS($\mathcal{B}, t, \tau$)
    $\tilde{\mathcal{B}} \leftarrow \{\}$
    Remove boxes from $\mathcal{B}$ where confidence is less than t.
    Sort $\mathcal{B}$ on confidence in a descending order
    for B in $\mathcal{B}$ **do**
        for $\tilde{B}$ in $\tilde{\mathcal{B}}$ **do**
            $IOU_{\tilde{B}} \leftarrow$ IOU between B and $\tilde{B}$
            if $IOU_{\tilde{B}} > \tau$ **then**
                Break loop
        if $IOU_{\tilde{B}} < \tau$ for all $\tilde{B}$ **then**
            Append B to $\tilde{\mathcal{B}}$

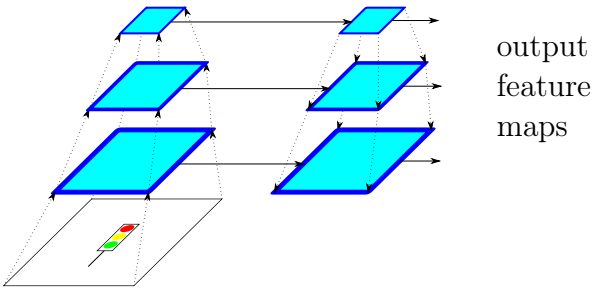
2.2.3 Selective search

Selective search is an algorithm for segmenting an image by finding coherent matter and bounding boxes. It is a greedy similarity-based algorithm that combines several similarity measures (e.g. color and size) to distinguish objects from each other. In order to not miss any objects, the algorithm emphasises high recall over high precision [27].

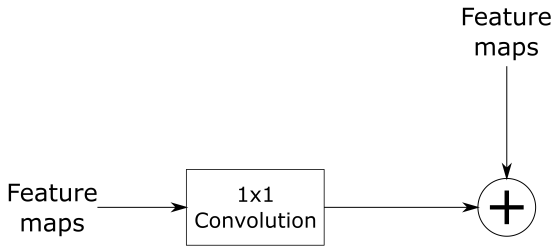
This technique can propose regions of interest that may contain an object to a two-stage ODS. This algorithm is not used in recently developed object detectors since it is slow compared to modern region proposal methods.

2.2.4 Feature pyramid networks

A feature pyramid network's (FPN) structure is a CNN for generating feature maps of different scales and semantic complexity. It is structured as a bottom-up and top-down pathway with lateral connections between them, as illustrated in figure 7a. The output feature maps can be reorganised into box predictions. Intuitively, feature maps generated with fewer convolutions are more coarse than others, and predictions are made independently on the different top-down pathway scales. This means that we make predictions on both coarse and fine features and at different scales, which intuitively encompass a greater variety of objects. A detection system utilising FPNs may also be better at recognising objects at different scales. It has been shown experimentally that FPNs increase detection accuracy on common ODSs with little impact on the inference time [28].



(a) An illustration of a feature pyramid network consisting of bottom-up and top-down pathways. The bottom-up pathway consists of convolutional layers for downsampling the maps, while the top-down pathway consists of upsampling layers. The lateral connections are copies of the feature maps that are convoluted and concatenated with features at later stages. The channel dimension is omitted.



(b) An illustration of the fusion of bottom-up and top-down features in the FPN network.

Figure 7: A graphical explaination of the FPN network.

2.2.5 Focal loss and RetinaNet

Lin et al. [29] found that the main reason one-stage object detection systems were inferior to two-stage detectors in terms of detection accuracy is the general imbalance between the object and background class. They proposed a solution to this problem by inventing a new loss function called focal loss. Focal loss down-weights correctly detected objects during training to put focus on the falsely detected ones. Let p be the model's estimated probability that class y is correct. We define p_t as p if the prediction is correct, and $1 - p$ otherwise. Focal loss is defined in equation (31). However, the focal loss is often used with a scaling parameter α_t in practice.

$$\mathrm{FL}(p_t) = -(1 - p_t)^{\gamma} log(p_t). \tag{31}$$

The authors created a one stage object detection system called RetinaNet using an FPN backbone network and focal loss, which surpassed all the existing two-stage detectors at the time in terms of detection accuracy.

2.2.6 ROI pooling and warping

Two-stage object detection systems generate candidate bounding boxes and feed them to a CNN. A CNN requires the bounding boxes to have a particular input size, but a region proposal can be of any size. Region of interest (ROI) pooling was introduced to solve this problem. It divides a candidate region into an $K \times K$ grid of cells and then performs a pooling operation on each cell. But one downside of this operation is apparent when candidate regions are non-integer, e.g. when relative coordinates are used. In this case, some object information may be lost in the translation from relative to absolute coordinates due to integer approximations [30]. A variation of ROI pooling is ROI warping, which considers the lost information by interpolating all the pixels in a cell instead [31].

2.3 Evaluation of object detection systems

To demonstrate the quality of an ODS, it is necessary to define suitable metrics. Object detectors output a confidence score representing the assessed probability that the predicted bounding box contains an object and how accurately placed and shaped the predicted bounding box is. The metric mean average precision (mAP) attempts to assess this in one value between 0 and 1. For a precise definition of this metric, several concepts need to be defined.

The confidence score that accompanies each bounding box prediction can be used to classify a prediction as positive (the bounding box contains an object of a specific class) or negative (the bounding box does not contain an object, i.e. the background class is predicted). When calculating mAP, a detection is labelled positive if its confidence score is equal or larger than a threshold t, else it is considered negative.

Whether the prediction is considered to be true or false depends on a chosen IOU threshold s and whether or not the class prediction is correct. The IOU of a ground truth bounding box and the predicted bounding box is the area of overlap between the bounding boxes as a percentage of the total area of the two boxes.

The following metrics can then be defined:

- **True positive (TP):** A predicted box where the IOU with the ground-truth bounding box is greater than a threshold s and the class is correct
- **False positive (FP):** A predicted box that is not a true positive, i.e. either the class is correct, but the IOU is less than s, or the class is incorrect
- **False negative (FN):** A ground-truth bounding box that was not correctly detected

For a fixed IOU threshold s, the precision and recall values are calculated as a function of t

$$precision(t) = \frac{TP(t)}{TP(t) + FP(t)} \tag{32}$$

$$recall(t) = \frac{TP(t)}{TP(t) + FN(t)}. \tag{33}$$

A key property of a good object detector is one where when recall increases, precision decreases by only a small amount. This results in a large area under the precision-recall curve (AUC). To compute the mAP metric and the average precision (AP), the following steps are carried out:

1. Order the K different confidence values outputted by the object detector in an increasing order

$$t(k), k = 1, 2, ..., K \text{ where } t(y) > t(x) \text{ for } y > x \tag{34}$$

Each confidence value is used as a detection threshold t as in (32) and (33). There is a one-to-one, monotonic correspondence between the recall and the confidence t which has the identical correspondence with the index k. The precision-recall curve is therefore discontinuous

2. Define an ordered set of reference recall values, R, sorted in a decreasing order: $R(i) < R(j)$ for $i > j$.

3. Interpolate the precision-recall curve so that it is continuous and monotonic over a finite set
$$PREC_{int}(i) = \max_{k|recall(t(k))\geq i} precision(t(k)), \tag{35}$$
where i is a given recall value

4. The AP metric is then defined as an approximation (Riemann integral) of the area under the precision-recall curve
$$AP = \sum_{k=0}^{K} (R(k) - R(k+1)) \cdot PREC_{int}(R(k)). \tag{36}$$

The resulting Riemann integral is evaluated using interpolation, where two approaches can be taken:

- N-point interpolation, where the reference recall values are used for the calculations
$$R(n) = \frac{N-n}{N-1}, n = 1, .., N. \tag{37}$$
The AP becomes
$$AP = \frac{1}{N} \sum_{n=1}^{N} PREC_{int}(R(n)). \tag{38}$$
Common values for N are 101 and 11.

- All-point interpolation, where the obtained K recall values are used and the following function values are set:
$$t(0) = 0$$
$$t(K+1) = 1,$$
so $recall(0) = 1$ and $recall(K+1) = 0$ by definition. Then, $recall(k)$ is the recall at the k'th confidence value $t(k)$. The precision values are calculated using the interpolated precision-recall curve $PREC_{int}$.

5. The mean average precision is the average AP over all the classes:
$$mAP = \frac{1}{C} \sum_{i=1}^{C} AP_i \tag{39}$$
where AP_i is the AP for class i and C is the total number of classes

Common mAP metrics are mAP50, i.e. mAP calculated at an IOU threshold of 0.5 and mAP@[0.5:0.05:0.95], mAP averaged over IOU thresholds from 0.5 to 0.95 with a step size of 0.05. Another version of this metric takes the area of the ground-truth objects into consideration:

- AP_S only takes into consideration small ground-truth objects (area $< 32^2$ pixels)
- AP_M only takes into account medium-sized ground-truth objects ($32^2 <$ area $< 96^2$ pixels)
- AP_L only takes into considerations large ground-truth objects (area $> 96^2$ pixels)

Another version of IOU that takes adjacent frames in video into account is the spatio-temporal tube intersection over union (STT-IOU). Considering a sequence of predictions over several frames, the ground-truths and the predictions form tubes in time. STT-IOU measures the volume of overlap as a fraction of the volume of union, thus quantifying the tracking performance. This is formulated mathematically in equation (40), where $T_p^{(i)}$ denotes a bounding box in frame i in a tube of predicted boxes T_p and, similarly, $T_{gt}^{(i)}$ for a ground truth tube [32].

$$\text{STT-IOU} = \frac{volume(T_p \cap T_{gt})}{volume(T_p \cup T_{gt})} = \frac{\sum_{i=1}^{\text{num. frames}} area(T_p^{(i)} \cap T_{gt}^{(i)})}{\sum_{i=1}^{\text{num. frames}} area(T_p^{(i)} \cup T_{gt}^{(i)})}. \tag{40}$$

2.4 Regions with CNN features (R-CNN)

Two-stage object detection systems can be divided into a region proposal phase and a combined classification and regression phase. The region proposal phase consists of two steps. In the first step, ROIs are generated. In the second step, the coordinates of the proposed bounding boxes are refined. Additionally, class predictions are updated, and boxes that do not correspond to objects are removed. What follows is a brief description of some of the most well-known two-stage algorithms in chronological order: R-CNN, Fast R-CNN and Faster R-CNN.

2.4.1 R-CNN

Regions with CNN features (R-CNN) starts by generating region proposals using the algorithm selective search. The backbone is used to extract features from images and outputs a 4096-dimensional feature vector. In order to use the CNN, all images must be of the same size. This issue was solved by resizing the images to a fixed size of 227×227 pixels, potentially changing the aspect ratio. When the feature vector has been calculated for a region proposal, a support vector machine (SVM) is run for each class. Each SVM predicts whether a region proposal contains an object of a certain class or not. This means that the backbone network calculates a feature vector for each region proposal, and the feature vector for each proposal is fed to an SVM C times. Each classified proposal is fed to a class-specific linear regressor that adjusts the bounding box coordinates. When it comes to classifying examples as positive or negative in the loss function, the IOU threshold was treated as a hyperparameter that was tuned on a validation set. A value of 0.3 was found to be optimal in terms of mAP on a benchmark dataset. During inference, NMS is used

to reject regions with high IOU between different predicted boxes in combination with low SVM scores. Training is conducted in four stages: pre-training the backbone network on an image classification task, domain-specific fine-tuning, training separate SVMs for each class, and finally using a class-specific regressor to correct the bounding box locations [33].

2.4.2 Fast R-CNN

Fast R-CNN was introduced and builds upon R-CNN with the objective of reducing inference time. The main inference bottleneck of the R-CNN implementation is that it is necessary to perform one CNN forward pass for each region proposal. Another advantage of Fast R-CNN is that it is trained end-to-end. The loss function consists of a sum of partial loss functions, one for each ROI, and it combines classification and regression loss as a weighted sum

$$\ell(p, u, t^u, v) = \ell_{\text{cls}}(p, u) + \lambda[u \geq 1]\ell_{\text{loc}}(t^u, v),$$

where u is the ground truth class, p is the softmax output, t^u is the regression output for class u and v is the bounding box ground truth. A bounding box regression output is represented by a tuple containing coordinates. Hyperparameter λ is used for weighting the localisation loss. The indicator function $[u \geq 1]$ is used to ignore localisation error for the background class (which by convention is class 0). The number of region proposals at test time is approximately 2000, all of which are passed through a CNN for prediction. For the region proposals to be fed to the same CNN, they have to be of equal size. This is achieved by using ROI pooling on each region proposal. Inference time in Fast R-CNN is 10 to 100 times lower than R-CNN. To make Fast R-CNN more robust to different image scales, it is trained using each image at several predefined scales [30].

2.4.3 Faster R-CNN

Faster R-CNN was created to further reduce the inference time of Fast R-CNN by speeding up the proposal generation process. The bottleneck of Fast R-CNN was generating region proposals using selective search. To this end, a region proposal network (RPN) was introduced. The RPN is a CNN that shares feature maps with the backbone, thus adding little computational effort. The RPN is a small network taking the feature maps as input and predicting ROIs with corresponding k confidence scores. The ROIs are determined using the feature map output of the CNN component in the RPN. The RPN slides a 3×3 kernel over each pixel in the feature map and produces k anchor boxes for each pixel. NMS is used on the output of the RPN to reduce the number of region proposals which decreases the inference time. Finally, the head of Fast R-CNN is used as the head of the ODS, where it predicts the box classes and adjusts the coordinates. A positive label is assigned to two kinds of anchors: those with the highest IOU with a ground-truth box or an anchor with an IOU higher than 0.7 with any ground-truth box. A negative label is assigned to non-positive anchor boxes if the IOU is lower than 0.3 for all ground truth boxes. Boxes that are neither positive nor negative do not affect the model training. The Faster R-CNN objective function is given by

$$\ell(p_i, t_i) = \frac{1}{N_{cls}} \sum_i \ell_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* \ell_{reg}(t_i, t_i^*),$$

where i is the index of an anchor in a mini-batch, and p_i is the probability that the anchor box contains an object. Furthermore, p_i^* is one if the ground-truth label is positive but 0; otherwise, t_i is a four-dimensional vector containing the four coordinates of the predicted bounding box while t_i^* contains the same values but for a positive ground truth anchor box. Finally, the classification loss ℓ_{cls} is defined for a binary class with the two labels being object and no object, while the regression loss ℓ_{reg} is only calculated as deviations from positive ground truth anchors. The fact that the regression loss is calculated in terms of deviations from the ground truth is what mainly differentiates the objective function of Faster R-CNN from Fast R-CNN. The architecture of faster R-CNN is visualised in figure 8 [34].

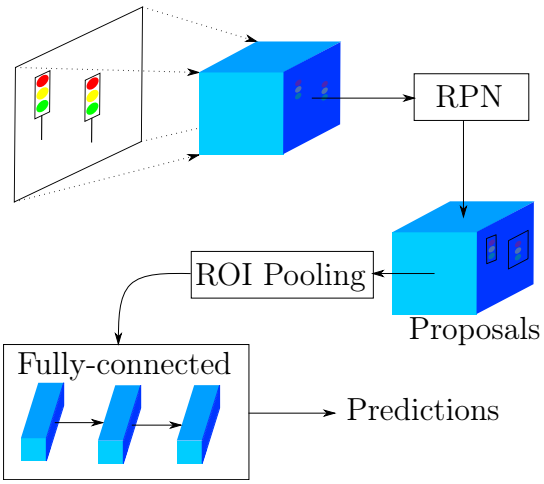


Figure 8: Visualisation of the architecture of Faster R-CNN.

2.5 You only look once (YOLO)

What follows are brief descriptions of the four versions of a commonly used one-stage ODS called YOLO. Another common one-stage ODS is single shot detector (SSD) which has a similar architecture to YOLOv2-YOLOv4, outputting coordinates, confidence score and class probabilities based on anchor boxes [35]. SSD achieves lower detection accuracy than YOLOv4 on the benchmark dataset MSCOCO 2017 but to a lower inference time [23].

There exist multiple versions of YOLO, but all of them have the following steps in common:

1. Resize images to a square size
2. Run a convolutional network
3. Non-max suppression

2.5.1 YOLOv1

In the first version, all images are resized to a resolution of 448×448. Then the image is divided into an $S \times S$ grid. For each grid cell, B bounding boxes and corresponding confidence scores are predicted. Each bounding box prediction is composed of a 5 dimensional vector: $[x, y, w, h, \text{confidence score}]$ where x, y represent the top left corner of the

bounding box and w, h the width and height. This means that the regression output is of size $S \times S \times B \times 5$. YOLOv1 also predicts class probabilities for each grid cell which means the classification output is of size $S \times S \times C$, where C is the number of classes. This creates a score map where the C values per grid cell can be interpreted as probabilities since they are nonnegative and sum to 1.

When training YOLOv1, a combined regression and classification loss function is used, and the two different losses are weighted by two hyperparameters λ_{coord} and λ_{noobj}. In YOLOv1, a confidence score is defined as the IOU between a predicted and ground truth box. One predicted box per grid cell is responsible for a ground truth object which is the one with highest confidence score. If the center of no ground truth object falls into the particular grid cell, no box is responsible for the prediction. The loss function is expressed as

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \Big] \tag{41}$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big[(\sqrt{w}_i - \sqrt{\hat{w}_i})^2 + (\sqrt{h}_i - \sqrt{\hat{h}_i})^2 \Big] \tag{42}$$

$$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big(C_{ij} - \hat{C}_{ij} \Big)^2 \tag{43}$$

$$+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \Big(C_{ij} - \hat{C}_{ij} \Big)^2 \tag{44}$$

$$+\sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{\text{c} \in \text{ classes}} (p_i(c) - \hat{p}_i(c))^2, \tag{45}$$

where $\mathbb{1}_{ij}^{\text{obj}}$ evaluates to 1 if an object's center is located in cell i and bounding box j is responsible for detecting that object, otherwise 0. $\mathbb{1}_{ij}^{\text{noobj}}$ is the opposite. x_i, y_i and $\hat{x}_i, \hat{y}_i$ are the ground truth and predicted x and y-coordinate values for grid cell i. Similarly, w and h denote the width and height of a bounding box. C_{ij} is the confidence score for box j in grid cell i while $\hat{C}_{ij}$ is the previously mentioned predicted confidence score. $\mathbb{1}_{i}^{\text{obj}}$ evaluates to 1 when cell i contains an object, otherwise 0. $\hat{p}_i(c)$ is the predicted probability that the i'th cell contains class c and $(p_i(c) - \hat{p}_i(c))^2$ is thus the classification loss for class c in grid i. Notice that, due to the indicator function $\mathbb{1}_{i}^{\text{obj}}$, the confidence loss is the only active loss when there is not an object in grid i. The hyperparameter λ_{noobj} is usually small, in order to handle the natural class imbalance occurring since most grid cells won't contain an object. A typical selection of values for λ_{noobj} and λ_{coord} is 0.5 and 5. Small localisation errors in large bounding boxes should matter less than the same error in small bounding boxes. Taking the square root of the bounding box width and height in the loss function takes this into account.

When using YOLOv1 for inference, the predicted confidence score is used to discard bounding boxes with no object using a confidence threshold, along with NMS for removing overlapping boxes referring to the same object [36].

2.5.2 YOLOv2

YOLOv1 has several shortcomings, like poor localisation accuracy in comparison to two-staged alternatives. Version one of YOLO also has relatively low recall, so the main focus in version two of YOLO is to improve those things. One improvement was adding batch normalisation, which increased the network's regularisation capabilities. Images of varying resolutions are used, which improves overall mAP. The main difference, however, is the introduction of anchor boxes in the YOLO head architecture. The paper authors discovered that picking the right dimensions for the anchor boxes is crucial for maximising the mAP. This was done using k-means clustering on a benchmark dataset and hand-picking a value for k that lead to a good trade-off between recall and model complexity. In YOLOv2 the bounding box centre has to fall inside the anchor boxes corresponding grid cell, leading to less accurate localisation but decreasing training time. This is in stark contrast to Faster R-CNN, which does not constrain the location of bounding boxes. During inference, a bounding box prediction $\mathbf{b}$ is calculated according to the equations (46)-(50), where $\hat{x}, \hat{y}, \hat{w}, \hat{h}, \hat{C}$ is the regression output from YOLOv2 per anchor box. The offsets c_x, c_y are for locating the correct grid cell for the anchor box, and p_w, p_h corresponds to the anchor box width and height [37].

$$b_x = \sigma(\hat{x}) + c_x \tag{46}$$
$$b_y = \sigma(\hat{y}) + c_y \tag{47}$$
$$b_w = p_w e^{\hat{w}} \tag{48}$$
$$b_h = p_h e^{\hat{h}} \tag{49}$$
$$b_{\hat{C}} = \sigma(\hat{C}) \tag{50}$$

2.5.3 YOLOv3

Like with YOLOv2, YOLOv3 takes as input B predefined anchor boxes. The regression loss is identical to YOLOv1. Logistic regression is used to predict the confidence score for each bounding box. Ideally, the logistic regressor should output one if a particular anchor box has the greatest IOU with a ground truth box. If an anchor box has an IOU of more than 0.5 with a ground truth box, but it is not the highest IOU, the prediction is ignored. This way, we can train the logistic regressor to predict a confidence score for a given bounding box prediction. For classification, separate logistic classifiers are used instead of a softmax since each bounding box may contain more than one class, e.g. a traffic light may be red and yellow. Boxes are predicted at three scales using an FPN to reflect both the coarse and fine-grained features of the image in the predictions. Bounding box anchors are chosen using k-means clustering on training data, as in YOLOv2. The loss function is altered in YOLOv3, in which the confidence and classification loss is calculated using cross-entropy [38].

FPN is not the only addition in the YOLOv3 backbone but also residual blocks. A residual block in YOLOv3 consists of a 1x1 convolution and a following 3x3 convolution, both with batch normalisation and activation. The number of output channels is the same as the number of input channels. When several blocks are attached, there is a residual connection from every block to every succeeding block [16] [38].

2.5.4 YOLOv4

The YOLOv4 ODS is structured in the following way:

- Backbone: CSPDarknet53
- Neck: spatial pyramid pooling (SPP), path aggregation network (PAN),
- Head: YOLOv3 head network

The backbone network in YOLOv4 is CSPDarknet53. It is a cross-stage partial network (CSPNet) and a deep fully convolutional neural network consisting of 53 network layers. CSPNet was developed in order to combat inefficient gradient computations. In modern backbone networks, it is common to apply successive convolutions in a ResBlock fashion. This is called a Dense Block and means that instead of residual connections, we concatenate the input feature map with the output feature map and perform a new convolution recurrently. The authors of CSPNet noticed that by splitting feature maps into parts, performing operations on them individually and fusing them afterwards, one saves computations with little degradation in accuracy [39].

YOLOv4's neck network consists of layers that are used to process feature maps from the backbone network. This includes the PAN and SPP modules.

The processed feature maps are then fed to the YOLOv3 head network, which is part of YOLOv3. It consists of successive convolutional layers, batch normalisations and leaky ReLU activations. Like stated earlier, the YOLOv3 head predicts feature maps at different scales.

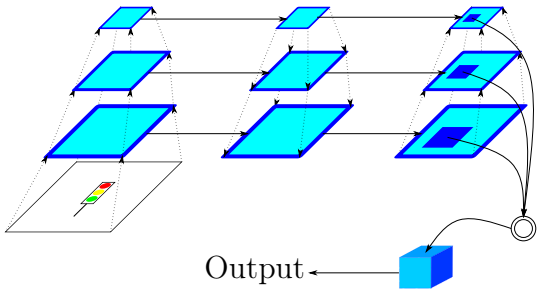
The goal of the creators of YOLOv4 was to create a network with low inference times and high accuracy. Therefore YOLOv4 utilises several techniques from the so-called bag of freebies (BOF) and bag of specials (BOS) sets of modifications to the network architecture and training strategies for improving detection accuracy. BOF contains techniques that increase detection accuracy without compromising inference time, e.g. data augmentation methods called Mosaic and CutMix. BOS instead offers higher accuracy by paying a small price in inference time, e.g. using the Mish activation function. BOF and BOS techniques are applied at all parts of the YOLOv4 network. The following sections introduce some of the essential elements of YOLOv4.

2.5.4.1 SPP SPP is a technique used in the neck of YOLOv4 for further processing the backbone features. It takes as input a set of feature maps and then performs pooling with three different kernel sizes: 5, 9 and 13, where the stride is 1 and the padding is adjusted so that the output is of the same shape as the input. SPP proceeds by concatenating the pooled feature maps with each other and the input set of feature maps. This means that the output is four different variations on the input, with varying semantic emphasis.

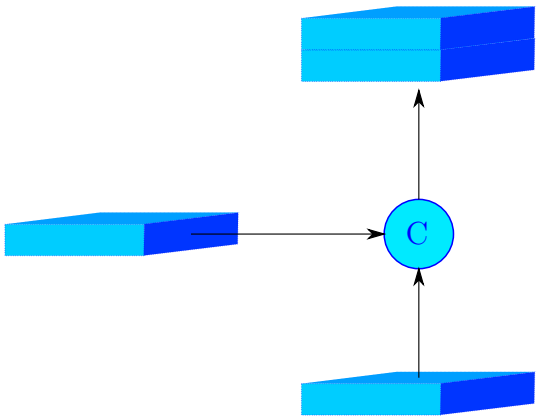
The original SPP was created for outputting feature maps of a fixed size, independently of the input image resolution. This is beneficial for two-stage systems, which often have fully connected layers. However, it is not the case in YOLOv4, where the output feature maps are larger if the input image is larger. The output of YOLOv4 for the typical input image size 608×608 is three grids of shapes $76 \times 76 \times A \times (4 + 1 + C)$, $38 \times 38 \times A \times (4 + 1 + C)$

and $19 \times 19 \times A \times (4 + 1 + C)$, where A is the number of anchor boxes. With a higher resolution, the grids will increase in size [40] [23].

2.5.4.2 PAN YOLOv4 uses a PAN in the neck. A central feature of a PAN is the efficient propagation of features through the network. A PAN builds upon the FPN architecture by adding an additional bottom-up pathway, a different strategy for combining features of different semantic levels and a pooling strategy for aggregating the output feature maps. The lower semantic layers contain more accurate localisation information. The accurate localisation information can be combined with the high-level semantics in the last convolutional layer by allowing shortcuts in the network. The combination of features propagated through lateral connections with the mainline of convolutions is conducted by copying feature maps, downsampling them through convolutions and performing element-wise addition with the mainline feature maps of matching size. This allows for the feature propagation shortcuts in the network. In YOLOv4, a modification of the combination of features is used where feature maps are concatenated instead of added together. Furthermore, the output feature maps are not combined using a pooling strategy. Instead, they are concatenated together and processed by additional convolutional layers. A general PAN is visualised in figure 9a, where the sideways arrows represent lateral connections. The combination of features from lateral connections used in YOLOv4 is visualised in figure 9b [41] [23].



(a) Simplified schematic of the PAN architecture. Vertical arrows denote downsampling or upsampling operations. The copy operation copies a feature map and combines it with later-stage feature maps. In YOLOv4, there is no adaptive feature pooling operation. The output is instead essentially the last layer of feature maps. The channel dimension is omitted.



(b) An illustration of the concatenation of lateral connection-features and later stage-features in the PAN network of YOLOv4.

Figure 9: A graphical explaination of the PAN network.

2.5.4.3 Mish activation D. Misra et al. [42] observed that the ReLU activation function, apart from suffering from the dying ReLU problem, also generates sharp and rough loss landscapes. This makes it hard to optimise over since the first-order gradient is noisy. This problem is solved using Mish, which is a smooth, continuously differentiable activation function. It is formulated in equation (51) and plotted in figure 10.

$$g(x) = x \tanh\left(\text{softplus}(x)\right) = x \tanh\left(\ln\left(1 + e^{x}\right)\right) \tag{51}$$

The activation function yields a smoother loss landscape than many other activation functions. This was demonstrated experimentally by randomly generating neural networks with two output neurons and plotting them for many different inputs. The resulting loss landscapes from using ReLU and Mish as activation functions were plotted, and the Mish loss landscape was confirmed to be smoother. The hypothesis that this would yield better performance in neural networks was confirmed through benchmarks on state-of-the-art classification networks such as ResNet and DarkNet. The benchmark showed that Mish was the ideal choice for an activation function in most scenarios, which is why it is used in the YOLOv4 backbone.

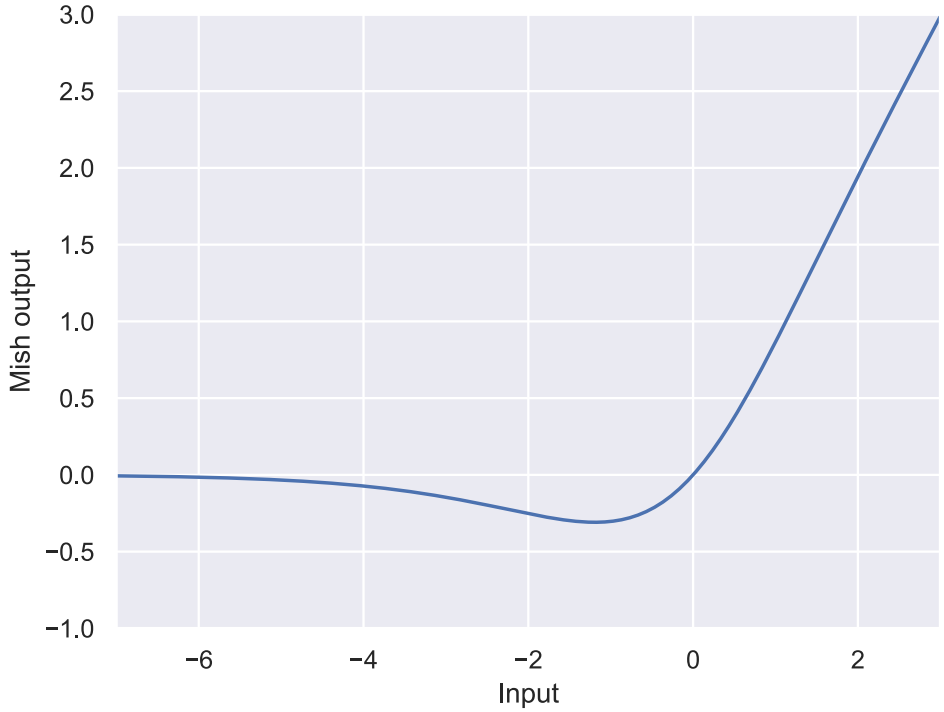


Figure 10: Mish activation function.

2.5.4.4 SAM SAM is an attention system that extracts the most useful parts in feature maps. The basic idea is to make a copy of a feature map, use average or max-pooling in combination with performing convolutions. The outputted, modified feature map is then fed to a sigmoid function before doing elementwise multiplication with the original input feature map. This is done both channel-wise and along the spatial dimension. In this way, the network can learn to become robust to perturbations in data [43].

The implementation of SAM in the YOLOv4 system is slightly modified. The pooling operations are removed, and there is only a convolution and sigmoid operation left in the attention module. This modified version of SAM is visualised in figure 11 [23].

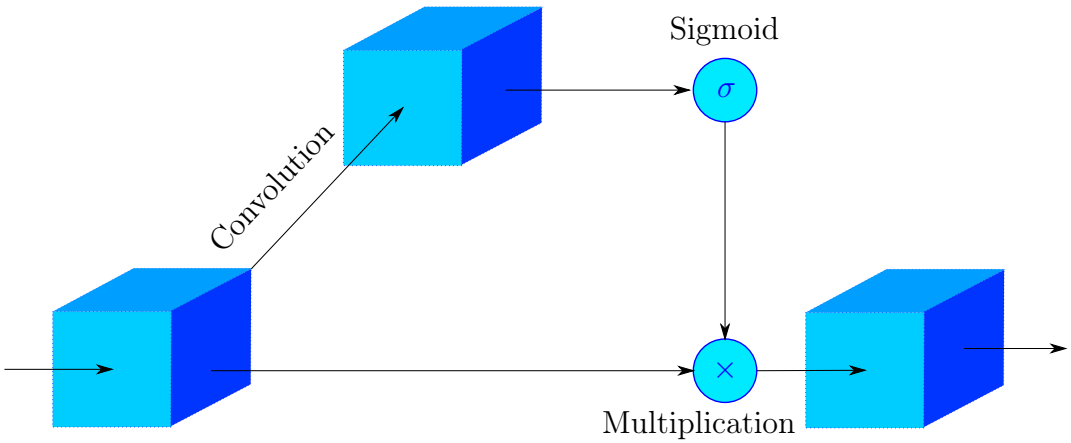


Figure 11: Modified version of the Spatial Attention Module, used in YOLOv4. It is a component of the neck network for processing feature maps from the backbone net.

2.5.4.5 Cross mini-batch normalisation When a batch has low or zero variance, that leads to numerical instability, e.g. due to floating point inaccuracies. This problem can be solved by adding a small term in the denominator of the batch normalisation formula (2). In equation (52), X denotes a batch of mini-batches, and $X_{ti}(\theta_t)$ denotes the i'th activation in the t'th minibatch using the network weights θ_t.

$$\hat{X}_{ti}(\theta_t) = \frac{X_{ti}(\theta_t) - \mu_t(\theta_t)}{\sqrt{\sigma_t(\theta_t)^2 + \epsilon}} \tag{52}$$

An attempt to increase the expressive power of batch normalization, one could scale and shift by learnable parameters γ and β. This scaled and shifted normalized batch, y_{ti}, is calculated as in equation (53) [44].

$$y_{ti}(\theta_t) = \gamma \hat{X}_{ti}(\theta_t) + \beta \tag{53}$$

The problem that occurs in deep learning with images is that y_{ti} becomes noisy due to small batch size. Nevertheless, given that weights change by a small amount between iterations, it is possible to estimate the mean and variance needed using Taylor expansions of both statistics. One can further stabilise the estimations by averaging over the current and $k-1$ previous means. This alternative to batch normalisation is called Cross-iteration batch normalisation [45].

YOLOv4 uses a modified version of CBN that does not incorporate statistics from previous batches but only between the different mini-batches within the present batch. This means that the estimate is a bit more accurate, as we do not have to rely on mean and standard deviation estimates from previous training steps [23].

2.5.4.6 DropBlock regularisation The dropout technique does not work well for regularising convolutional layers since neighbouring elements in a feature map are spatially correlated. This means that removing an element will have little effect since its neighbours have similar semantics. DropBlock is a strategy to mitigate the problem of dropout in

convolution operations by dropping regions instead of elements at random. This has a higher chance of achieving a regularisation effect than dropout in CNNs. An algorithm for the DropBlock technique is given in 4 [46].

Algorithm 4 DropBlock Regularisation

Input: Activations $g(L)$ from the previous layer L, $size$, α, $mode$.
Output: DropBlock-modified activations.

procedure DropBlock($g(L), size, \alpha, mode$)
  if $mode$ is inference **then**
    Return $g(L)$
  $A \leftarrow g(L)$
  Randomly sample mask $M \sim Bernoulli(\alpha)$
  for each element m at position (i, j) in M **do**
    Let m be the center of a square of size $size$, zero all corresponding elements in A.
  Normalize A
  Return A

2.5.4.7 Loss function YOLOv4 replaces the localisation loss in the YOLOv3 loss function, which is the same as in (41) and (42), with a loss that considers the IOU instead of the box coordinates.

In the following description of various IOU losses, the assumption is that only one predicted bounding box $\mathcal{B}$ and ground truth box $\mathcal{B}^{gt}$ exists. The coordinates-based bounding box loss metric (usually l_1 or l_2 norm) is different from the IOU evaluation metric. The inventors of Distance-IOU (DIOU) [47] show that the methods that attempt to take the IOU metric into account offer an inadequate solution to this problem:

- IOU loss (originally proposed by [48]):

$$\mathcal{L}_{IOU} = 1 - IOU \tag{54}$$

  This metric does not distinguish between bounding boxes that are close or far away from the ground truth bounding box if they are non-overlapping.

- Generalised IOU (GIOU) loss (originally proposed by [48]):

$$\mathcal{L}_{GIOU} = \mathcal{L}_{IOU} + \frac{|\mathcal{C} \setminus \mathcal{B} \cup \mathcal{B}^{gt}|}{|\mathcal{C}|}, \tag{55}$$

  where $\mathcal{C}$ is the smallest convex shape that contains both the predicted bounding box $\mathcal{B}$ and ground truth box $\mathcal{B}^{gt}$ and $|\cdot|$ denotes the area of a shape.

In simulation experiments, the inventors of DIOU show that the IOU loss converges to sub-optimal bounding box locations when the predicted and ground truth box are non-overlapping. While the GIOU loss addresses the main problem of the IOU loss function, it is still not ideal as it takes many iterations for it to converge.

Due to these problems they propose the DIOU loss:

$$\mathcal{L}_{DIOU} = \mathcal{L}_{IOU} + \frac{\rho^2(\mathcal{B}, \mathcal{B}^{gt})}{c^2}, \tag{56}$$

where c is the diagonal length of the smallest box that contains both the predicted bounding box $\mathcal{B}$ and ground truth box $\mathcal{B}^{gt}$ and ρ is the Euclidean distance between the central points of the bounding boxes. So instead of only trying to minimize the enclosing area of the two boxes the goal is now to minimize the normalised distance between the boxes directly. This fact leads to a faster convergence than for the GIOU loss. Incorporating aspect ratio prediction consistency, the authors define the Complete-IOU (CIOU) loss:

$$\mathcal{L}_{CIOU} = \mathcal{L}_{IOU} + \frac{\rho^2(\mathcal{B}, \mathcal{B}^{gt})}{c^2} + \alpha v. \tag{57}$$

The value v measures the difference in aspect ratios between the ground truth and prediction:

$$v = \frac{4}{\pi^2}(\arctan(\frac{w^{gt}}{h^{gt}}) - \arctan(\frac{w}{h}))^2, \tag{58}$$

where w, h and w^{gt}, h^{gt} are the width and height of the predicted and ground truth bounding box respectively and α is a parameter which represents the tradeoff between emphasizing the overlap and aspect ratio similarity, as specified in equation (59) [47].

$$\alpha = \frac{v}{(1 - IOU) + v}. \tag{59}$$

The loss function used in YOLOv4 is similar to what is presented in equations (41)-(45). However, the localisation loss is replaced by the CIOU loss. The loss function can be written as

$$\mathcal{L}_{CIOU} \tag{60}$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \Big(\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i) \Big) \tag{61}$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \Big(\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i)) \Big) \tag{62}$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{\text{c} \in \text{ classes}} \Big(\hat{p}_i(c) \log(p_i(c)) + (1 - \hat{p}_i(c)) \log(1 - p_i(c)) \Big), \tag{63}$$

where $\mathcal{L}_{CIOU}$ is defined in equation (57). [49]

2.5.4.8 DIOU-NMS In DIOU-NMS, the IOU metric used in NMS is replaced by DIOU to avoid falsely removing bounding boxes of occluded objects. Given a predicted box, $\mathcal{K}$, with the highest confidence score we update the confidence score of a predicted box predicted box $\mathcal{B}_i$ in the following way:

$$r_i = \begin{cases} r_i, & \text{if } IOU - \frac{\rho^2(\mathcal{K}, \mathcal{B}_i)}{c^2} < \epsilon \\ 0, & \text{if } IOU - \frac{\rho^2(\mathcal{K}, \mathcal{B}_i)}{c^2} \geq \epsilon, \end{cases} \tag{64}$$

where ϵ is the NMS threshold and r_i is the confidence score of box i. Then the bounding boxes with zero confidences are removed. This results in the non-removal of bounding boxes that have a large central point distance from the bounding box with the highest confidence score [47].

2.5.4.9 Data augmentation YOLOv4 uses basic data augmentation techniques like mirroring images, altering hue, value, saturation and blur. The user has to decide what data augmentation techniques to use. Mirroring an image, for example, is not necessarily a good idea if the orientation of the ground truth object is the only way to characterise it.

Another technique that improves mAP on an object detection benchmark dataset is Mosaic [23]. The Mosaic methodology is to merge different parts of images into one frame. At most, four images are merged. First, the algorithm randomly selects whether to use Mosaic or not. If it is used, the algorithm proceeds by selecting if a split on width, height or both width and height should be split. If it decides to split on both, a width and a height split is decided, w_s and h_s. They are pixel coordinates and are randomly sampled among the 60 % central width and height pixels. All the images in the frame are randomly sampled from the dataset, and the different cutouts from the frames are randomly selected. Additionally, all images that are part of the Mosaic frame are processed with random settings for the basic data augmentation techniques. An example Mosaic frame is shown in figure 12.

Another data augmentation technique in YOLOv4 is CutMix, which places randomly sampled bounding boxes from the ground truth data on other frames.

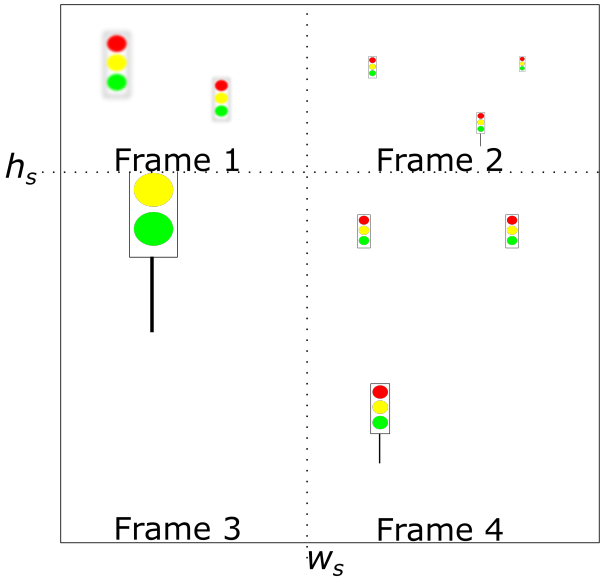


Figure 12: An example Mosaic frame.

2.6 Video object detections systems

A simple system for performing video object detection would be to run an ODS on every frame. However, such a system does not consider temporal context, and an ideal system would use this context to improve detection accuracy, inference time or both. In this work, only systems that use RNNs for modelling the temporal context are considered. Despite having access to more context in video, it is a more challenging problem than single frame object detection. This is due to the large amount of data to process; motion blur more frequently appears in video and objects may temporarily be partially or entirely occluded.

[10] On top of this, RNNs are known to be difficult to train due to vanishing or exploding gradients.

There are two categories of RNN video object detectors, feature-based and box-based.

2.6.1 Feature-based video object detection systems

The main idea behind feature-based VODS is to implement a visual memory of previous frames. An attempt to do this was done in multi-frame prediction ([50]), where the idea was to extract features from several frames before making a prediction. The implementation is inspired by a variant of the SSD architecture called SqueezeDet+. The difference lies in that the head network is only used in the last frame, and the addition of convolutional GRUs at the end of the backbone network. Figure 13 shows a minimal illustration of this architecture. The experiments that were conducted using this implementation indicated that temporal context improves detection accuracy. In addition, that long-term temporal context (frames more than one timestep ago) was more important than short-term temporal context [9]. Similar architectures for predictions on every frame and models using convolutional LSTMs have also been developed.

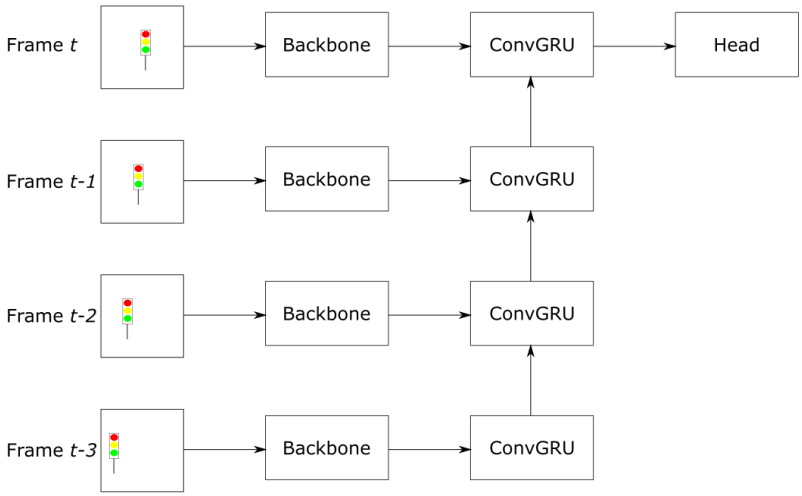


Figure 13: Multi-frame VODS inspired by SSD but with the addition of a convolutional GRU.

Another interesting attempt at creating a VODS using RNNs is the system "*Looking Fast and Slow*" [51]. The architecture utilises two feature extractors: a large and slow one and a small and fast one. The idea behind the architecture is that it would be sufficient to use a large feature extractor for every k'th frame and mainly rely on temporal context for the other frames, where it would suffice to use the small feature extractor to retrieve the coarse features of the image. The architecture is visualised in figure 14.

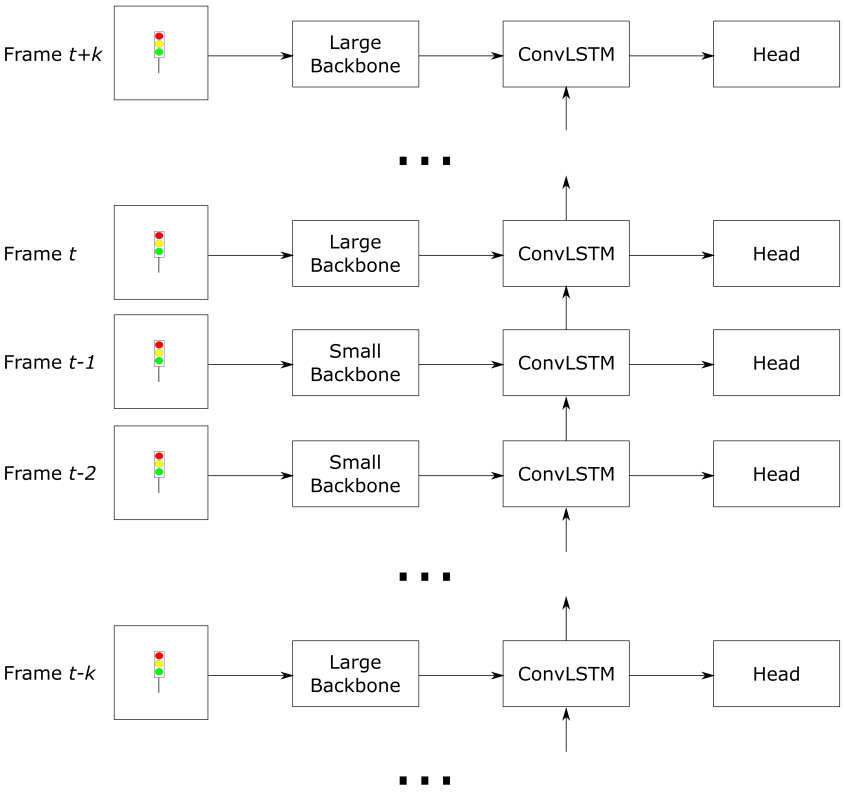


Figure 14: The architecture of "Looking Fast and Slow", using one large and one small feature extractor. The large feature extractor is used every k'th frame.

2.6.2 Box-level video object detection systems

Some VODS use the predicted bounding boxes and class probabilities as input to an RNN to refine them using temporal context. An example of this can be found in the paper *"Context Matters: Refining Object Detection in Video with Recurrent Neural Networks"* [52]. S. Tripathi et al. implemented a recurrent model using YOLOv1 as a base ODS. It takes $S \times S$ bounding boxes outputted from YOLOv1 (the ones having the highest confidence score in each grid cell) as input to a two-layer GRU that outputs predicted bounding boxes. An illustration of this architecture can be found in figure 15. The system was trained in two stages: a standalone pre-trained YOLOv1 was first trained on the dataset of interest, and secondly, the GRU unit was trained.

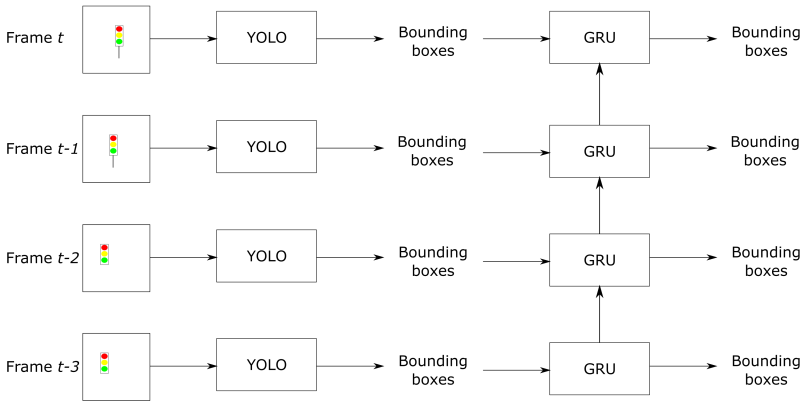


Figure 15: An illustration of a box-level-based VODS using the most confident boxes per grid cell in YOLOv1 as input to a two-layer GRU cell.

There are also papers in the field describing systems that are not online methods. K. Chen. et al. [53] describe a system that uses detected bounding boxes at two different timesteps to predict the bounding boxes on the frames in between. They use two operators, one responsible for modelling the temporal context and another for refining bounding boxes based on this context. The system predicts bounding boxes for a downscaled version of a past frame and the present frame, which are then upscaled to the correct size. They use a spatial refinement operator to derive the location of bounding boxes on the frame in between. Notice that in the next timestep, the previously present frame will be among those that are refined. This means that all but the first and last frame in a sequence will be refined using temporal context.

2.6.3 Other video object detection systems

One could also combine feature-based and box-level approaches in one system. This is the case for recurrent YOLO (ROLO), where both feature maps (flattened as a one-dimensional tensor) and the most confident bounding box is concatenated into one vector. This vector is then fed to an LSTM, which refines the prediction. Since only the most confident bounding box is used, the system can only detect one object in an image. The architecture is illustrated in figure 16 [54].

ROLO was trained in three phases. The first phase involves the pre-training of the backbone network of YOLO. The second involves training YOLO end-to-end. Then in the last phase, only the LSTM of ROLO is trained.

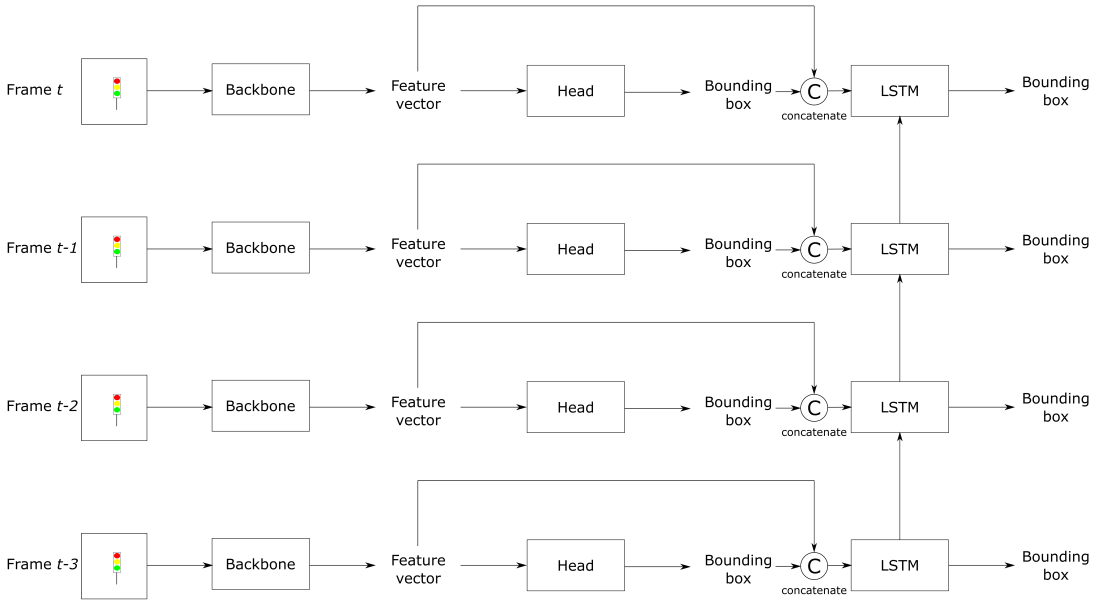


Figure 16: An illustration of the ROLO architecture.

Video object detectors based on attention techniques and optical flow also exist. Attention systems in the literature often achieve higher detection accuracy at the cost of high inference times. Flow systems use so-called flow networks for modelling the optical flow in a sequence of frames. This includes propagating information retrieved in subsequent frames to previous frames to improve bounding box location estimates [9].

3 Related work

Various approaches have been taken in an attempt to solve the problem of traffic light recognition. Traffic light recognition systems that have been built vary from completely rule-based systems that utilise image processing methods, to using completely deep learning based methods. The deep learning methods commonly use different versions of widely known one and two-stage object detectors as starting points for their systems, e.g. YOLO, SSD and R-CNN.

3.1 Non-deep learning approaches to TLR

A. Gómez et al. [55] propose a traffic light recognition system using image processing for detection in combination with hidden Markov models (HMM) for state classification. The base system is completely rule-based, where an image processing method and conversion from the RGB to the HSV colourspace. The system is therefore not affected by class imbalance problems. Traffic light state transitions can be viewed as Markov chains. Therefore, in an attempt to improve upon the baseline, an HMM was added to the system to make it more robust without decreasing inference speed. The authors used their own dataset for evaluating their method, consisting of 649 manually annotated images. The chosen performance metric was accuracy. The baseline system achieved an accuracy of 78.54% while the addition of the HMM resulted in an accuracy of 90.55%.

3.2 Non-temporal deep learning approaches to TLR

L. Possatti et al. [4] realised many studies in TLR were framed as single frame object detection problems. For autonomous vehicles, it is also necessary to detect the relevant traffic lights for the planned route. The researchers use the DriveU and LISA dataset for training and a dataset created specifically for the project. Their own dataset was created using a car equipped with LiDAR and a camera which was driven on five different routes. The LiDAR data was annotated with 3D coordinates of the relevant traffic lights and stored in the prior map. When the system was tested, the prior knowledge of relevant traffic light positions was used to filter out the irrelevant traffic lights. The choice of ODS was YOLOv3. The traffic light detection system performance was measured using mAP as well as precision and recall. The images were shrunk to a resolution of 608x608 RGB pixels. It was possible to achieve an inference time of 21 Hz on an Nvidia Titan Xp GPU, though capped by the car camera's framerate of 16 Hz. The ODS was trained on both the DriveU and LISA training sets, and the resulting models were evaluated on the test sets. In addition, the system was evaluated on the LiDAR dataset. It was reported that precision and recall of 88.6% and 86.5% was achieved on the DriveU test set along with an mAP50 of 85.6%. Corresponding values for the LISA test set were 66.5%, 54.8% and 50.6%. Finally, similar resulting values of 69.5%, 57.2% and 55.2% were obtained on the researchers' own dataset. Those results were obtained using a confidence threshold of 0.5, meaning that all bounding boxes with confidence below the threshold were discarded.

Ouyang et al. [5] propose a lightweight system for the detection of traffic lights. They are aware of the trade-off between accuracy and inference time in a real-world application of TLR and build a system in light of this. The system reaches an inference frequency of 10 Hz on an Nvidia Jetson TX1/TX2. The proposed system consists of two phases: traffic light detection and state classification. The detection phase is a heuristic method using

several image processing techniques. It extracts all illuminants in a frame (traffic lights, braking lights, etc.) and centres a fixed-size bounding box around it. This means that it does not detect the whole traffic light but a specific bulb on it. The classification part is a lightweight CNN that classifies the illuminants as green, red or other objects. It is reported that the proposed system yields at least the same detection and classification performance as the state of the art systems with computationally heavy backbone networks such as ResNet, DarkNet, GoogleNet but with up to 50 times lower inferences times.

Bach et al. [8] propose a traffic light recognition system based on the Faster R-CNN two-stage object detector. The DriveU dataset was used where only lights with three bulbs were included. Tram, bus, pedestrian or cyclist lights were discarded as well as those that did not belong to the nearest intersection of the vehicle. The dataset was then split on images, where 70% was allocated to the training set and 30% to the test set. The recognition system then consists of a feature extractor and a box classifier. The feature extractor is a slightly modified version of the residual network used in Faster-RCNN. Like in Faster-RCNN, anchor priors were utilised but defined based on the scales and aspect ratios present in the training set (only one aspect ratio was used but nine different scales). Finally, a regressor head outputs the relative bounding box coordinates, and a classifier head classifies the states of traffic lights and their type, e.g. arrow-right. The mAP metric is used to measure the recognition performance of the system. An mAP of only 1% was achieved for the off class. No inference time system evaluation was reported.

Müller and Dietmayer [6] built a system based on SSD for TLR on the DriveU dataset. They state that the main drawback of separating the region proposal and state classification phases is increased inference times. On the other hand, they claim that pooling operations in some CNN-based ODS:s result in an inability to detect small objects. They claim that despite the detection of traffic lights may seem like a simple task at first glance, they are easily confused with other objects in traffic scenes when little context is provided, e.g. braking lights. To mitigate this problem, the researchers defined anchor box sizes that were based on the bounding box sizes in the training set. The strides and kernel sizes used were adjusted accordingly. Inspiration was taken from VGG-16, but with the modification that feature maps from different stages of the network were concatenated into one set of maps. The maps finally act as input to three different small networks that output bounding boxes, confidence scores and class probabilities. The authors use Log Average Miss Rate (LAMR) for evaluation of the network and its variations. The best model in terms of LAMR achieved a score of 0.015 with an inference time of 117 ms. The fastest network had an inference time of 101 ms but with a LAMR of 0.064. The experiments were conducted using an Nvidia Titan Xp GPU.

K. Yoneda et al. [7] propose another method using prior maps to increase accuracy and reduce the inference times of a TLR system. In order to avoid discomfort to passengers when stopping a vehicle that is travelling at 50 km/h, typically, a minimum halting distance of close to 100 meters is needed. Therefore, the authors state that it is important that a TLR system recognises traffic light states at a distance of considerably more than 100 meters. A challenge in recognising traffic lights at a distance greater than 100 meters is that traffic lights appear very small. A strength of their system is that it can detect traffic lights that are smaller than 10 pixels. The data used for the prior maps comes from Japan and contains information on the precise location, orientation and type of traffic lights. Although the inclusion of prior maps seem to result in only a minor performance

increase for normal traffic lights, the increase is substantial for arrow traffic lights where substantially less data was available. The proposed method outperforms YOLOv3 on the test set for circle traffic lights, both in terms of accuracy (mean F-score of 91.8% for the proposed system, 91.5% without prior maps information but a value of 72.2% for YOLO). The corresponding F-values for arrow lights were 56.7%, 42.9%, 41.7%. Inference times were lower than YOLOv3 (YOLOv3 had an average computational time of around 94 ms while the proposed system a time of around 64 ms), and their system was more robust to different lighting conditions. They allege that deep learning detectors could outperform image processing-based detectors in situations where the traffic lights have low brightness, e.g. due to light pollution.

3.3 Temporal deep learning approaches to TLR

K. Behrendt et al. [56] proposes a system that modifies and builds upon the YOLO ODS to detect, classify and track traffic lights over multiple frames. The data used is the Bosch Small Traffic Light dataset, which contains 13 334 frames in total. The dataset is then formed by taking random crops of the upper part of the image, where most traffic lights are found. The authors run a modified version of YOLOv3 without classification. Since traffic lights in their dataset are only 10 pixels in width on average, they use a greater number of grid cells for more precise bounding box localisation ability. The detected bounding boxes are then fed to a classification network that classifies the states of the bounding boxes. Additionally, false positives are removed by thresholding the bounding box confidence estimates of the network. In order to track objects, an odometry based model in combination with a neural network is used to estimate the movement of traffic lights between frames. They use disparity images from a stereo camera where the median disparity value of a traffic light represents the whole traffic light. The traffic light coordinates and disparity values are then triangulated onto the 2D reference frame of the dataset. Then linear triangulation is used to reconstruct the 4 bounding box corners of a frame. We can therefore write:

$$X_t^c = T_{t-1}^t X_{t-1}^c$$
$$x_t^c = P X_t^c$$

where c represents a corner id, and $X_{t-1}^c = [x^c, y^c, z^c]^T$ and x_t^c represent the projected and re-projected corner coordinates. The estimate is then created from the re-projected corners. Since traffic lights are of small size in the dataset, precise localisation becomes difficult. Therefore a neural network is used to refine the bounding box estimates. There is no other performance evaluation on the full system other than precision-recall plots. From an isolated evaluation of the traffic light state classifier, 95.1% accuracy was observed on the test set. The full systems is reported to run at 15 FPS on an NVIDIA Titan Black GPU. Gains are reported in an online evaluation where the system is run in real-time. The YOLOv3 detector was compared with their system in an online and offline manner. The reported gains in accuracy were observed in the online evaluation. Since the YOLOv3 detector needs 100 ms to process each frame, some frames are skipped in an online setting. In that scenario, the tracker combined with the classifier could replace the detector in many instances, so a significant gain in accuracy was observed compared to the baseline YOLOv3 system.

Wang et al. [57] propose combining object tracking and a deep learning based detection method for traffic light recognition. Detection is done by using a CNN model that consists

of a backbone and a backend network. The backend network takes the multi-scale feature maps outputted by the backbone as input and fuses them in a way to make the detection network more robust to different object scales and image resolutions with a minimal increase in computational complexity. The output of the detection network is the bounding box location. An integrated channel feature tracking module is applied to track the traffic light location and state when a detection is obtained. The system achieves better detection accuracy, as they report 15% fewer false positives than YOLOv3. Running the system on a computer with an Nvidia Titan XP they report an inference speed of 17.92 FPS compared to YOLOv3's 14.47 FPS.

4 Methodology

To investigate whether or not RNNs can improve detection accuracy, a modern and high-performing ODS was modified to incorporate temporal dependencies using RNNs. The original and modified detection system are compared on the traffic light dataset DriveU. An alternative approach would have been to create the simplest possible ODS and modify it to include temporal context. However, this does not guarantee that including the temporal context in the same way in a modern and advanced ODS would increase the performance. The aim of the thesis is to explore whether it is beneficial or not to model temporal context in TLR systems using RNNs. This could not be evaluated using a minimal ODS since the results are not guaranteed to hold for a state-of-the-art ODS. For this reason, YOLOv4 was chosen as a baseline, as it achieves high performance in terms of mAP on benchmark datasets while still running in real-time. The YOLO detection systems are also common in TLR literature and achieve high detection accuracy on various traffic light datasets. Similar experiments were run for YOLOv4 and different modifications of it, so that the effect of adding temporal context could be measured.

4.1 Dataset

The data used in this project comes from the DriveU dataset, consisting of 232 039 traffic light annotations in 11 cities in Germany. The data is in the form of 40 925 video frames of resolution 2048x1024 which form 2 110 sequences in total. A sequence begins around 150 m before the vehicle arrives at a crossing controlled by traffic lights and ends after immediately passing the crossing (the recording is paused while the vehicle has halted). The videos were filmed at a frame rate of 15 FPS. Two different cameras were used to record videos, one with a wide horizontal field of view (130 degrees) for capturing traffic lights placed near the vehicle and the other with a narrow field of view (60 degrees) for capturing traffic lights further away [11].

A label is represented by a six digit number representing the following properties:

- Viewpoint orientation (front-, back-, right- and left facing)
- Relevancy and occlusion (not relevant, relevant, not relevant and occluded, relevant and occluded)
- Installation orientation (vertical and horizontal)
- Number of light units (1-4)
- State (off, red, yellow, red-yellow and green)
- Pictogram (circle, arrow, pedestrian, cyclist and bus)

The pre-processing of the dataset consists of two main steps: (1) cropping and resizing the images and (2) filtering the labels.

In step 1, the goal is to reduce the size of the images to something that is feasible to run using current state-of-the-art ODS (the common benchmark dataset MSCOCO 2017 has images of size 608x608). The size was chosen arbitrarily to be 640x640 with this in

mind. Resizing from 2048x1024 to 640x640 using bilinear interpolation removes many details in an image and the performance of an ODS would be low. It would also alter the aspect ratio from 2:1 to 1:1. In order to combat this, two ideas were applied. The first one comes from the observation that it is rare for traffic lights to appear at the bottom of an image or the left or right edges. Therefore, a window of 1920x960 was cropped out, vertically at the top but horizontally centred. This cropped region also maintains the original aspect ratio. This reduces the image size without affecting the quality of the image. The second idea was that instead of resizing the images to 640x640 directly, two squares of size 640x640 were cropped from the image. Due to this, it was only necessary to resize down to 1280x640 and then extract a left crop and right crop from each image. This part of the pre-processing is visualised in figure 17. Traffic lights whose width is more than 70 % outside the cropped regions are removed.

Unlike most other works in TLR, this work considers all of the traffic light state classes: off, red, yellow, red-yellow and green. However, different pictograms are merged. This means that e.g. a green circle traffic light and a green arrow traffic light are considered to belong to the same class. Around 85 % of the traffic lights are circle lights. The only classes that are filtered out are horizontal, pedestrian and cyclist traffic lights. The ones kept are all relevant for a self-driving vehicle, except for those not facing the vehicle. The single reason for detecting off traffic lights in the developed systems is for the case of flashing yellow traffic lights. Off traffic lights were found to be challenging to detect, one of the reasons being that they look similar to non-front-facing traffic lights from a large distance. Therefore, the non- front-facing off traffic lights were not removed. This way, the relevant off traffic lights may more easily be found by the ODS. A positive side effect of this, is a less imbalanced dataset since most off traffic lights are not front-facing.

The resulting pre-processed dataset consists of 4 220 sequences and 81 850 frames of resolution 640x640 with five classes: off, red, yellow, red-yellow and green. The dataset was split by selecting random sequences in each city, resulting in a training, validation and test set. They consist of 3 030, 548 and 642 sequences, respectively. Label statistics for each set is provided in table 1. Small traffic lights are defined to have a bounding box area of less than 32^2 pixels, medium between 32^2 and 96^2 and large having area greater than 96^2.

Statistic	**Training**	**Validation**	**Test**
No. of sequences	3 030	548	642
No. of frames	58 924	10 984	11 942
No. of annotations	134 014	24 025	25 946
% Off	20.8	22.1	17.8
% Red	30.6	28.9	32.6
% Yellow	3.5	4.0	4.5
% Red-Yellow	1.5	1.0	1.4
% Green	43.7	44.0	43.7
% Front-facing	82.7	80.9	84.4
% Small traffic lights	93.8	93.5	93.3
% Medium traffic lights	6.2	6.5	6.7
% Large traffic lights	0	0	0

Table 1: Statistics for the pre-processed training, validation and test set.

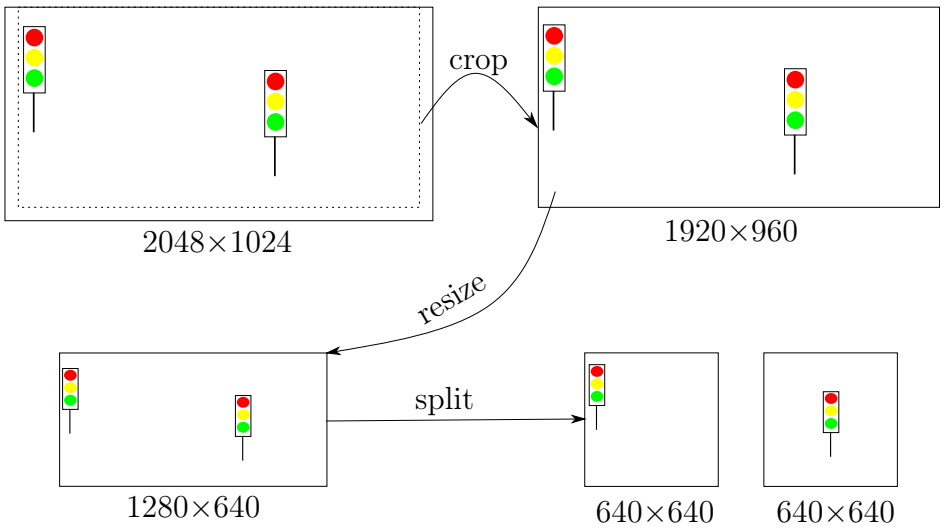


Figure 17: A visualization of the pre-processing of the DriveU dataset, consisting of a crop, resize and a split step.

An important note on the DriveU dataset that affects the performance of a TLR system is an inconsistency in the annotations and labelling errors. One of them is that in some cities, only the front-facing off traffic lights were annotated. This means that many of the off traffic lights do not have annotations. Additionally, there are errors in the annotation labels [11]. An attempt to fix this has recently been conducted, and a new version of the DriveU dataset has been released. Unfortunately, this occurred too late in the thesis creation process for the updated data to be utilised.

Because of the errors in the dataset and the way labels were filtered, the result is an underestimate of what performance can be achieved with the models, especially for the off traffic lights.

4.2 Evaluation

Apart from measuring how accurate a system is at making predictions, inference times are relevant for a real-time system. The accuracy of an ODS is often measured with mAP. Inference times are commonly measured in Frames Per Second (FPS) or milliseconds (ms) in the field of object detection. Therefore the aim is to assess the impact of incorporating temporal context in terms of two key aspects:

1. Accuracy

2. Inference time

For TLR, the exact localisation of traffic lights is not important compared with, e.g. cars where there is a risk of crashing when localisation accuracy is low. Therefore the IOU threshold at which we consider a traffic light to be localised correctly can be low. The lowest threshold commonly used in literature was therefore chosen, namely 50 %.

Inference times that are measured for different models vary significantly due to the hardware used when running them. To enable comparison, all the measured inference times have been recorded on the same machine. Inference times may differ due to how much

the machine is being utilised when measuring the inference time, so an average inference time over many runs is reported along with the variance. The main interest is not to minimise the inference time since the running conditions are not similar to an actual deployed version of the system. The central point is to show how the inference time changes with different modifications of the baseline.

4.3 Baseline

A PyTorch [14] implementation of YOLOv4 was used for all baseline experiments. Some of the BOF and BOS used in the original YOLOv4 system were not implemented due to time limitations (e.g. DropBlock regularisation, CutMix and DIOU-NMS). A schematic view of the specific implementation of YOLOv4 used in this project is illustrated in figure 18. The five downsampling stages of the YOLOv4 backbone network are fully convolutional. Most of the convolutional layers use a stride of one to lose as little localisation information as possible. Batch normalisation and a Mish activation function are applied after each convolutional layer. Stages two to five also includes one residual block each. Stages three to five together with the neck and head network work as a PAN. The neck comprises an SPP module and upsampling modules, where each of those modules also contains further convolutional layers. The features then flow through the fully convolutional YOLOv3 head network, where predictions are made at several points in the network. Both the neck and head network use leaky ReLU activations after batch normalisation instead of Mish.

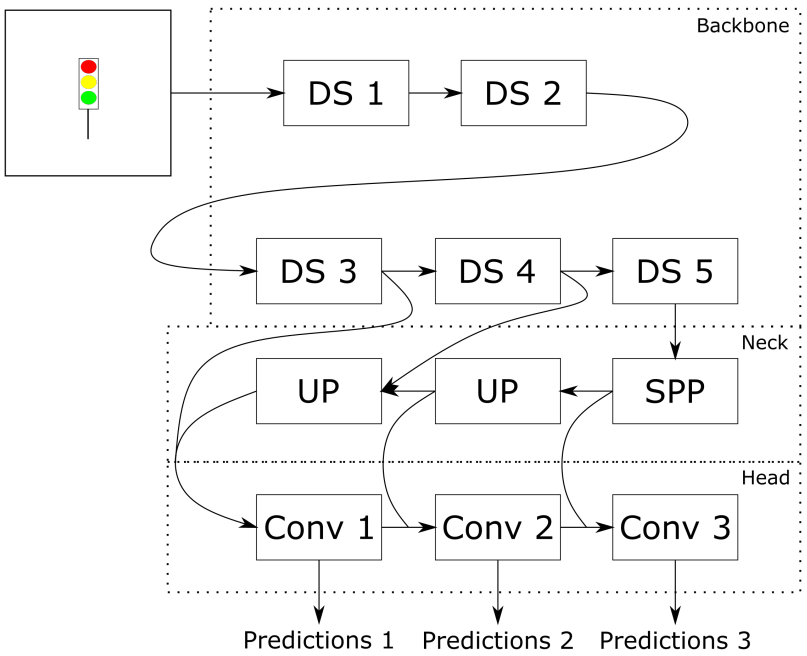


Figure 18: A schematic view of the baseline YOLOv4 architecture where downsampling is abbreviated as DS and upsampling as UP. The DS layers consists of fully convolutional networks with batch normalisation and Mish activation after each convolution operator. The neck is comprised of a PAN network and the head consists of a fully convolutional network. The neck and head networks uses leaky ReLU activations.

YOLOv4 was trained with a batch size of four and Adam as optimiser with hyperpa-

rameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. The base step size α was 10^{-4} but was varied according to a scheduler function $S(t) : \mathbb{R} \to \mathbb{R}$, yielding a step size factor to multiply the base step size with for a given time step. This means that the step size in each time step is $10^{-4} \cdot S(t)$. The time step t is given according to algorithm 2, but the weights are only updated after batch size-many forward propagations. The scheduler takes three additional hyperparameters: b, the number of burn-in steps and s_1, s_2, the number of steps before stepping down the step size factor. $S(t)$ can be specified as

$$S(t) = \begin{cases} (\frac{t}{b})^4 & \text{if } t < b, \\ 1 & \text{if } b \leq t < s_1, \\ 0.1 & \text{if } s_1 \leq t < s_2, \\ 0.01 & \text{otherwise.} \end{cases}$$

The YOLO loss function was used with the addition of CIOU loss, given in (57). An epoch consists of sampling all frames in the dataset uniformly at random without replacement.

When training YOLOv4, there is a possibility of starting with pre-trained weights to speed up training and potentially reach higher detection accuracy. However, this is not available for the other models developed for this project. To be able to fairly compare the models, pre-trained weights were therefore not used for the YOLOv4 baseline. There is one experiment with Mosaic data augmentation and one without, to be able see how the effect of Mosaic translates to sequential data. Those two experiments are denoted as `YOLO` and `YOLO_MOS`.

It was noted in the class-specific performance analysis that the off state is most difficult to detect. An alternative training session without the off state was conducted. This experiment is denoted as `YOLO_NOFF` and does not use Mosaic.

4.4 Main systems

Two systems were developed for this thesis, a box-level LSTM version of YOLOv4 and a feature-based architecture that replaces convolutional layers in the backbone, neck and head networks with convolutional LSTMs.

The training strategy for the sequential models requires the introduction of additional hyperparameters. While the batch size could be four as in `YOLO` and `YOLO_MOS`, it is set to one in the experiments with the sequential models due to computer memory constraints. Instead of maximising the batch size, the sequence subdivision length was maximised. This is the number of images fed to the sequential model until backpropagation is conducted and computer memory is freed. A higher sequence subdivision length than one is necessary to model temporal context. In the experiments with sequential models, the sequence subdivision length is six. This, however, does not mean that the weight update occurs after six forward propagations. The cycle of weight updates is regulated using the sequential batch size. If not stated otherwise, this hyperparameter is given a value of two, meaning that one weight update is performed after completing two sequences. The average gradient of the loss is used for the weight update. The hidden and cell states are kept unchanged after a backpropagation within a sequence and only reset after completing a whole sequence. The weight update cycle consists of running several backpropagations

over two sequences and accumulating gradients, and updating the weights. A schematic view of one training step is illustrated in figure 19. As in the YOLOv4 experiments, Adam optimiser was used with the same settings and the same step size scheduler $S(t)$. The loss function used is the same as in `YOLO`, `YOLO_MOS` and `YOLO_NOFF`.

The data augmentation techniques for the sequential models are similar to the ones in `YOLO` and `YOLO_MOS`. However, the data augmentation parameters such as blur and rotation are not randomly sampled within a sequence but only at the start of a new sequence. This is performed to ensure that the transformations to images in a sequence is consistent, intuitively keeping more temporal dependencies. The Mosaic data augmentation method is altered with the same idea in mind. With Mosaic zero, one, two or three extra sequences are running simultaneously in the same frames, and the extra sequences are selected at the start of a new sequence. Since the sequences are potentially of different lengths, the sequence of frames stops after the shortest sequence is finished. The same logic applies if the batch size is greater than one, the processing of the next sequence(s) is started once the shortest sequence in the batch is over.

One additional data augmentation method was experimented with, namely reversed sequences. With 50% probability, this method will feed a sequence of frames in reverse order during training. When this is used in an experiment, the experiment's name gets the name extension `REV`. The complete data augmentation pipeline and how it differs between the single-frame object and video object detector experiments is illustrated in figure 20.

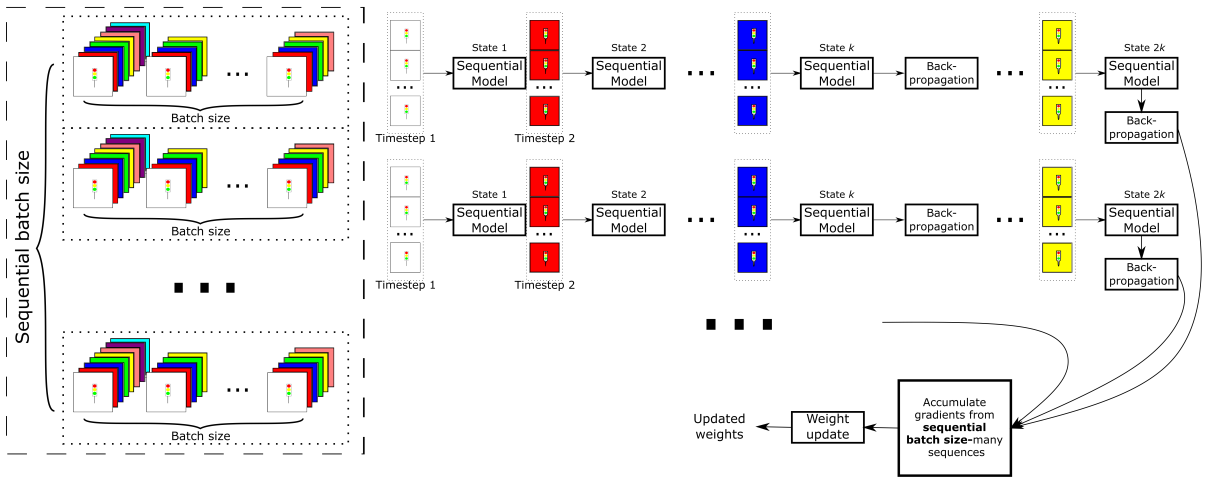


Figure 19: Illustration of one training step and how batch size, sequential batch size and the sequence subdivision length parameter modifies it. The sequence subdivision parameter is given the name k in the illustration.

4.4.1 Box-level VODS

The box-level VODS uses the YOLOv4 architecture and hyperparameters as in the `YOLO` experiment but further processes the predicted boxes with a simple LSTM module. This architecture is inspired by the system described in [52], presented in section 2.6.2. The LSTM module requires the hyperparameter N_B, the number of boxes to feed to the LSTM. In practice, it takes the N_B most confident boxes among the three prediction tensors, stores them in a tensor of size $N_B \times (4 + 1 + C)$, where C is the number of classes and $4 + 1$ are the coordinates and confidence score. This is a naive approach to box-level LSTMs since

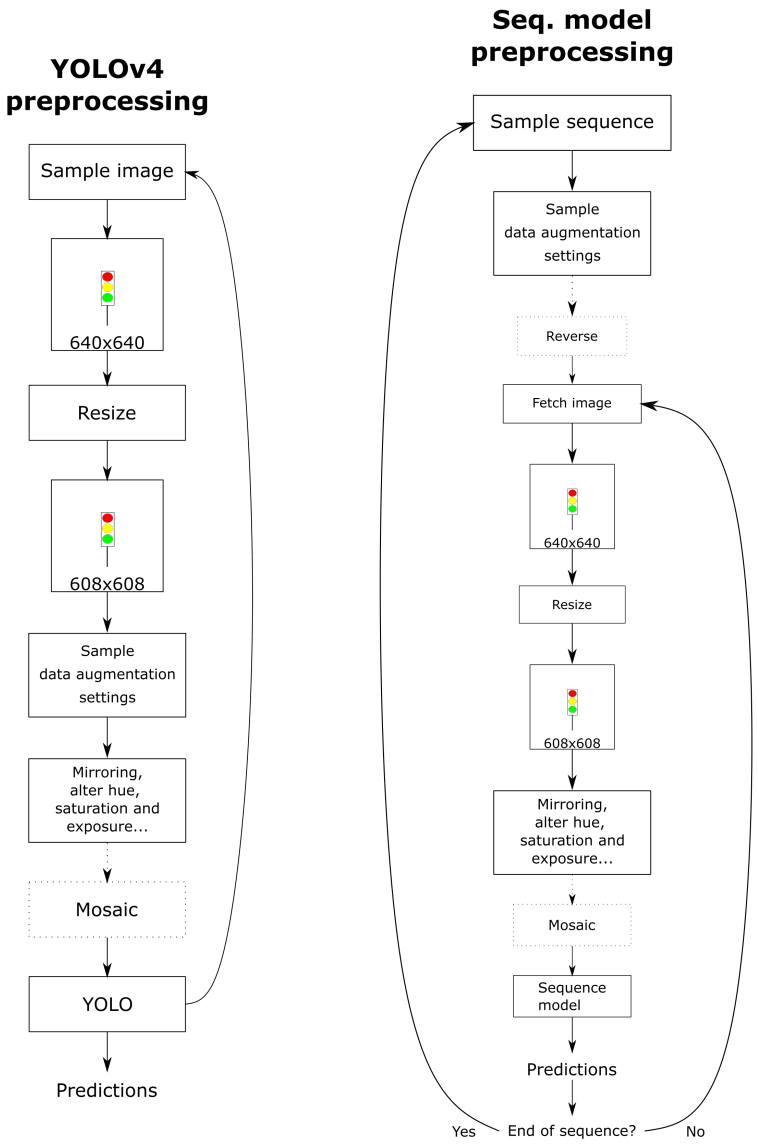


Figure 20: An illustration of the data augmentation pipeline that is applied during the training of the architectures. The Mosaic and reverse operations are optional.

there is no association of boxes between the frames. On the other hand, considerably more boxes are predicted in YOLOv4 compared to YOLOv1, which is why the approach from [52] is computationally intractable for YOLOv4. Naturally, the boxes that the LSTM does not process could contain objects not covered by the processed ones, which is why the tensor of processed boxes is put back among the non-processed ones. In short, the three prediction tensors are concatenated, sorted on confidence, the N_B most confident boxes are processed, and finally, the sorting operation is reversed. The box-level VODS experiments are denoted as `BLSTM_NB`, where `NB` is the hyperparameter N_B.

4.4.2 Feature-level VODS

The feature-level VODS replaces or adds convolutional layers in the different stages (downsampling stages one to five, neck network and the head network) with convolutional LSTMs. The most common unit in those stages is convolution, batch normalisation and activation function. In some places, the convolution operation is replaced. If there is a convolutional layer with a stride of one but a kernel size greater than one, it is replaced by a ConvLSTM with a matching kernel size. If such a layer does not exist, a ConvLSTM with a kernel size of three is added. A downsampling stage is denoted as `DSX`, where `X` is replaced by the number 1-5. The neck and head stages are denoted as `N` and `H` respectively. For the head experiment, the ConvLSTM is put at the start of the head, to affect all the prediction stages. One experiment was conducted when a ConvLSTM was placed as the first layer, i.e. before the YOLOv4 model. This experiment is denoted as `FLSTM_P`.

There is also a possibility of stacking ConvLSTMs. This was, however not explored due to time limitations. Notice that when ConvLSTMs are placed in the neck, the architecture is similar to [51] and [50], described in section 2.6.1. If the box-level LSTM is added on top of a model used in `FLSTM` experiments, the architecture is similar to ROLO.

Some additional experiments were conducted with feature-level LSTMs, with other hyperparameters related to the sequential models. One of them is `FLSTM_DS1_SHORT`, which uses a sequence subdivision length of three, but with a batch size of two. This experiment is run to investigate how important the sequence length is sequential model's performance compared to the batch size. Some other experiments are `FLSTM_DS4_KER5` and `FLSTM_DS4_KER7`, where kernel sizes of 5 and 7 of the ConvLSTM in the DS4 phase are tested. Like with the baseline experiment, there is one experiment where the off states are removed from the dataset. This experiment is denoted as `FLSTM_DS1_NOFF`. Since the hyperparameters from the `YOLO` experiment were used for the sequential models, the training is biased towards the YOLOv4 baseline model. Therefore, an experiment with the YOLOv4 baseline using the sequential training strategy (shown in figure 19) was also run. In this experiment, the batch size is one, the sequential batch size is two, and the subdivision length is six, like for the sequential models. We denote this experiment as `YOLO_SEQ`.

As a sanity check, the learned weights for the ConvLSTM layers were looked into. If weights such as W_{hi}, W_{ci} are all zeros while W_{xi}, W_{xc}, W_{xo} are not, it would mean that temporal context is ignored and the ConvLSTM is reduced to a near to ordinary convolutional layer with activation. It was, however verified that the sequential models weigh in previous frames to different extents.

4.5 Experimental setup

Computers with an x86 and PowerPC architecture with Nvidia Tesla P100 GPU were utilised for the experiments, running either PyTorch 1.5 or 1.7. OpenCV 4 was used for processing images for training, e.g. data augmentation. The library Pycocotools was utilised and modified to enable the computation of mAP at any IOU threshold.

Many hyperparameters need to be defined for the different experiments. YOLOv4 was trained with the hyperparameter values $b = 1000, s_1 = 120000, s_2 = 180000$ for the step size scheduler. For the sequential models, the scheduler hyperparameters need to be adjusted due to a lot fewer training steps. Therefore, those hyperparameter values were reduced so that, in most experiments, $b = 100, s_1 = 13000, s_2 = 17000$. Those settings were chosen empirically to shorten model convergence and training time. The sequential batch size was set to two, the batch size was set to one, and the sequence subdivision length was set to six, if not stated otherwise. Similarly, the kernel sizes for feature-level VODS was set to three, if not stated otherwise.

5 Results

In this section, a basis for evaluation and discussion of the systems is constructed. Inference times on an Nvidia Tesla P100 GPU as well as mAP50 of all models are presented. Loss curves, as well as validation mAP50 curves, are also displayed. However, they are all rather similar, which is why only a subset of them are included.

5.1 Baseline

The resulting baseline mAP50 scores for the validation and test set is provided in table 2. The average inference time and its standard deviation for 1000 frames is given in table 3.

Experiment	Validation mAP50	Test mAP50
YOLO	45.0	47.3
YOLO_MOS	45.4	47.9
YOLO_SEQ	39.8	39.4

Table 2: Baseline results in terms of mAP50 on the validation and test sets.

Experiment	Inference time (ms)	Variance
YOLO	40.2	0.7

Table 3: Baseline result in terms of average inference time and variance over 1 000 forward propagations.

5.2 Main systems

The resulting mAP50 scores for most sequential models on the validation and test set is provided in table 4. The average inference time and its standard deviation for 1000 frames is given in table 5.

Experiment	Validation mAP50	Test mAP50
BLSTM_1	23.5	23.4
BLSTM_100	<1	<1
FLSTM_P	40.8	40.4
FLSTM_DS1	40.2	39.8
FLSTM_DS2	39.5	39.3
FLSTM_DS3	40.0	40.1
FLSTM_DS4	37.9	37.6
FLSTM_DS5	38.8	38.1
FLSTM_N	38.4	38.4
FLSTM_H	38.8	38.3

Table 4: Main systems results in terms of mAP50 on the validation and test sets.

The best-performing model among the main system variations was FLSTM_DS1. Further experimentation with this model was conducted, using data augmentation, different batch size, sequential batch size and sequence subdivision length. The result from those experiments is provided in table 6.

Experiment	Inference time (ms)	Variance
BLSTM_1	45.2	0.5
FLSTM_P	56.9	0.4
FLSTM_DS1	55.7	0.6
FLSTM_DS2	46.3	0.2
FLSTM_DS3	53.0	0.7
FLSTM_DS4	54.4	0.6
FLSTM_DS5	49.0	0.2
FLSTM_N	62.2	0.7
FLSTM_H	68.7	0.9

Table 5: Main systems results in terms of mean inference time and variance over 1 000 forward propagations.

Experiment	Validation mAP50	Test mAP50
FLSTM_DS1_MOS	40.7	40.8
FLSTM_DS1_REV	39.3	39.4
FLSTM_DS1_SHORT	35.9	35.0

Table 6: Result for variations on the experiment FLSTM_DS1.

Experiments with a larger kernel size than three were also conducted. The results of those experiments are provided in table 7.

Experiment	Validation mAP50	Test mAP50	Kernel size
FLSTM_DS4_KER5	39.4	38.9	5
FLSTM_DS4_KER7	39.2	38.3	7

Table 7: Results when using different kernel sizes.

Experiments were conducted when the off traffic lights were removed from the dataset. The results from those experiments are available in table 8.

Experiment	Validation mAP50	Test mAP50
YOLO_NOFF	58.0	57.9
FLSTM_DS1_NOFF	49.1	49.7

Table 8: Results when all off traffic lights were removed from the dataset.

Apart from the results in tables 2-8, two training variations of the FLSTM_P experiment were conducted. In this variation, the sequential batch size was set to 1 and 10, respectively. Setting the sequential batch size to 10 resulted in a model stuck at close to zero validation mAP50 during training. This experiment was interrupted prematurely because of this reason. When the sequential batch size was set to one, the model achieved a validation mAP50 of 37.1 %.

5.3 Evaluation of results

In table 9, class-specific mAP50 for selected models is displayed.

Experiment	Off		Red		Red-Yellow		Yellow		Green	
YOLO	16.9	20.2	57.7	56.9	43.4	49.9	51.3	49.9	68.0	67.6
YOLO_NOFF	-	-	58.7	56.9	53.6	52.6	60.0	60.5	67.4	67.6
YOLO_SEQ	3.9	3.3	53.7	52.9	38.4	38.6	40.5	40.0	62.8	62.4
FLSTM_DS1	4.0	3.5	54.5	54.5	35.7	36.4	41.8	40.4	65.0	64.4
FLSTM_DS1_NOFF	-	-	53.0	51.6	40.0	39.1	45.7	49.6	57.7	58.6
FLSTM_P	5.9	5.6	54.2	54.1	36.4	36.1	42.7	41.1	64.8	64.9

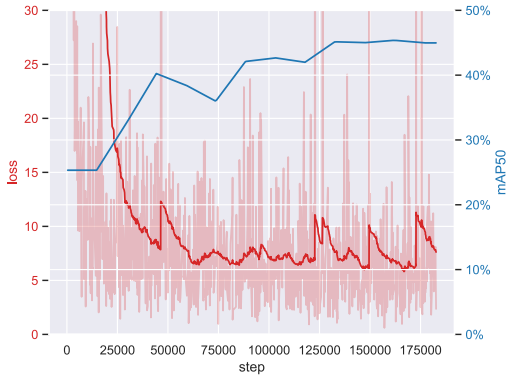
Table 9: Per-class mAP50 for some selected experiments. It is provided under each heading as validation mAP50 and test mAP50.

It was also looked into how the performance differs for different sizes of traffic lights. Since there are no large traffic lights in the dataset, only medium and small traffic lights could be considered. Results for traffic lights of sizes small and medium is given for some selected models in table 10. Notice that this time, the metric is mAP, i.e. the mean over each class for mAP@[0.5:0.05:0.95].

Experiment	Small		Medium	
YOLO	22.0	23.1	45.7	42.7
YOLO_NOFF	29.8	30.2	53.1	49.0
FLSTM_DS1	19.3	18.9	39.8	36.5
FLSTM_DS1_NOFF	23.1	23.2	45.7	42.7

Table 10: Per-traffic light size mAP for some selected experiments. It is provided under each heading as validation mAP and test mAP.

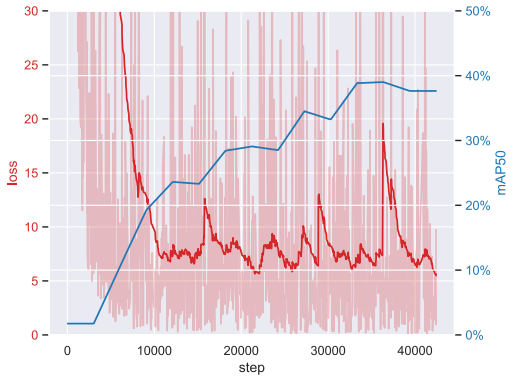
The training loss and the validation mAP50 plotted against training steps for the experiment YOLO_MOS is given in figure 21a and for FLSTM_DS1_MOS in figure 21b. The validation mAP50 is only calculated after each epoch, and is therefore interpolated. Analogous plots are given for experiments FLSTM_DS4_KER5, FLSTM_DS4_KER7, FLSTM_P and FSLTM_H in figures 21c, 21d, 21e and 21f.



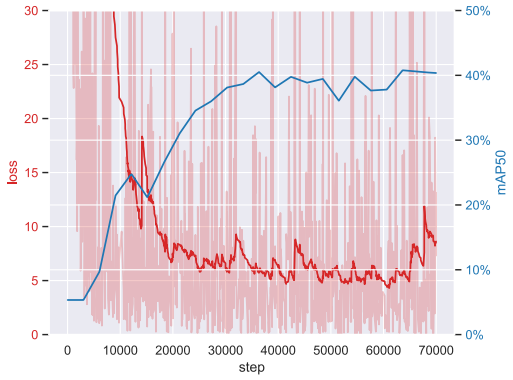
(a) In red, the training loss in `YOLO_MOS` is plotted against training steps.

(b) In red, the training loss in `FLSTM_DS1_MOS` is plotted against training steps.

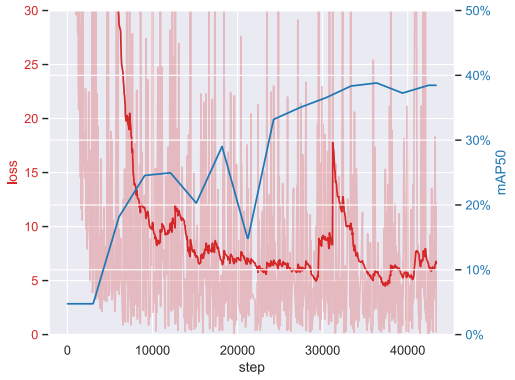
(c) In red, the training loss in `FLSTM_DS4_KER5` is plotted against training steps.



(d) In red, the training loss in `FLSTM_DS4_KER7` is plotted against training steps.



(e) In red, the training loss in `FLSTM_P` is plotted against training steps.



(f) In red, the training loss in `FLSTM_H` is plotted against training steps.

Figure 21: The light red colour marks the actual loss, while the bright red curve has been smoothed with exponential weighted moving average with weight $\alpha = 0.03$. In blue, the validation mAP50 is plotted.

The learned parameters of the ConvLSTM layer of `FLSTM_DS1` and `FLSTM_DS3` were looked into. It was observed that all parameter values of the convolutional layer were in the

range -1 to 1, similar to other convolutional layers in the first downsampling phases of the backbone. It was also investigated how the accuracy of the models `YOLO` and `FLSTM_P` differs for different IOU thresholds. In figure 22, the result from IOU thresholds ranging from 1 % to 99 % is displayed.

Some side by side comparisons of predictions on test images from DriveU from different models is available in appendix A.

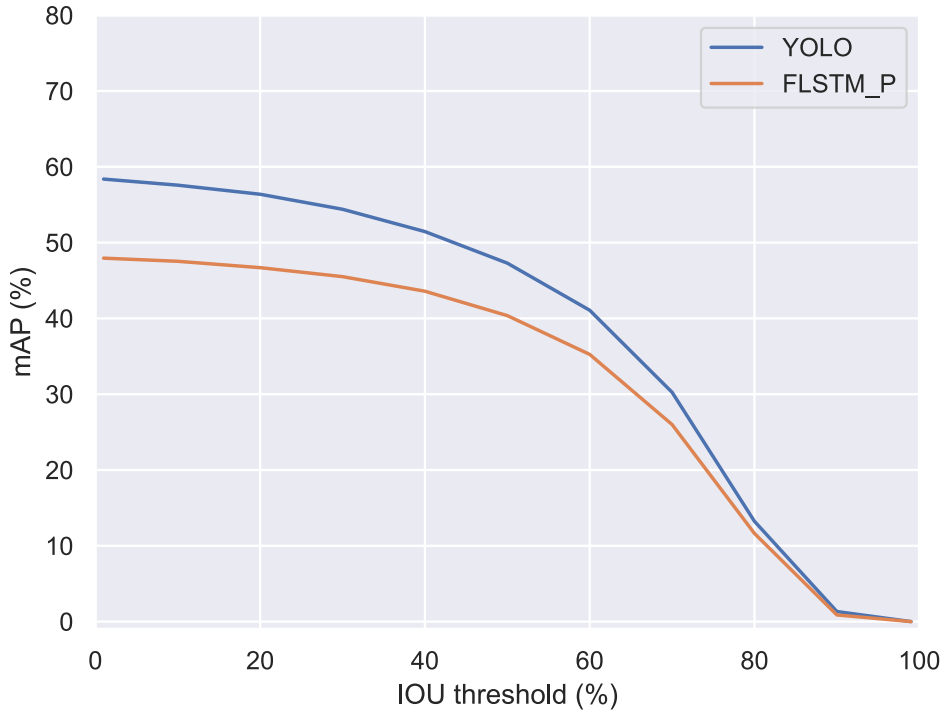


Figure 22: Plot of mAP at different IOU thresholds for the models `YOLO` and `FLSTM_P`.

6 Discussion

The results were in favour of the baseline model, YOLOv4. In other words, modelling temporal context using either ConvLSTMs or box-level LSTMs did not improve the results. However, this does not prove that it would never do so, especially since it has been successful in other applications. There are several possible reasons why it did not happen in this case. The first possible reason is the selection of hyperparameters and architecture design. Due to the time constraints, it was not possible to do proper hyperparameter tuning or conduct an exhaustive search for the optimal model architecture. The hyperparameters for YOLOv4 were used as starting point for all alternative models, which means that the training was biased towards the baseline. Additionally, most of the hyperparameters for YOLOv4 were adopted from recommendations from the original developers, where an exhaustive search for optimal hyperparameters was possible. This may have had a large impact on the outcome of the experiments. An indication of this can be seen in the results of experiments `FLSTM_DS4` and `FLSTM_DS4_KER5`, which differ a lot in mAP50 for a slight change of hyperparameters. Therefore, researchers attempting to model temporal context in video using RNNs should be aware that careful tuning is vital.

The results are also highly affected by the data. There are two properties of the DriveU dataset which may have had a large impact on the result. The first one is the low camera FPS of 15. Lower camera FPS means that there is a larger difference between two consecutive frames, thus the temporal dependencies are weaker. When the vehicle is turning or when traffic lights are close, this is especially true. Of course, higher camera FPS requires a faster model as well. The temporal dependencies are further weakened by the fact that the images are split in half, meaning that a particular traffic light may stay on camera for less time. The results of both the baseline and the sequential models are also negatively impacted by the aggressive resizing of the images, despite several operations being performed to avoid it. The second property about DriveU, which may have harmed the sequential models is the annotation inconsistencies and inaccuracies. Several traffic lights were found to have the wrong label, which causes confusion about what the correct class of a traffic light is. However, the most important error to point out is inconsistent labelling of off traffic lights for different cities. This was fixed but too late for use in this thesis. Further work may therefore provide more accurate results for off traffic lights. The `YOLO` model achieved significantly higher mAP50 for the off traffic lights, but we also experienced that it was less hesitant to make predictions overall. The behaviour of some selected models, including `YOLO`, can be studied in appendix A for a small number of scenes.

The reason for why off traffic lights were difficult to detect cannot only be explained with annotation errors. Red, yellow and green traffic lights are strong illuminants, so those objects differ greatly from most other objects in traffic scenes. Off traffic lights appear dark or blend in with the background in many images, which requires a model that can rely on context instead of only colour. The difficulty of detecting off traffic lights was also concluded by [8], where an mAP of 1 % was achieved. A possible solution to this problem arises from the scenario in which they matter. In Sweden, the only time off traffic lights are interesting to drivers is when a traffic light flashes yellow. Thus, an off traffic light can be indirectly detected using the yellow state, which is present in only some of the frames. Not directly detecting off traffic lights would allow a model to focus on only the simpler

states, which yields a more accurate system. An additional benefit would be with regards to class imbalance since the front-facing off traffic light is infrequent.

Two experiments with data augmentation were conducted. The first experiment tries to answer whether or not Mosaic improves mAP in sequential models. Experiments `FLSTM_DS1` and `FLSTM_DS1_MOS`, indicate that using Mosaic increases mAP50.

While most experiments used the highest possible sequence length (six) before performing backpropagation, a model with lower sequence subdivision lengths but larger batch size was also tested. The experiment `FLSTM_DS1_SHORT` used a sequence subdivision length of three but a batch size of two. However, this resulted in lower detection accuracy. A natural follow-up experiment would be also to test a sequential subdivision length higher than six, which unfortunately was not possible due to memory constraints. Two related experiments were conducted where the sequential batch size was set to one and ten. As stated in the result section, setting the sequential batch size to ten led to extremely slow convergence, and the mAP50 was close to zero. This is likely due to a noisy gradient, so that the average gradient over many sequences is close to zero. This could potentially have been fixed by increasing the learning rate or switching to another learning rate scheduler. Setting the sequential batch size to one did not improve mAP50 either.

Another interesting experiment was to increase the kernel size. This was done as an alternative to `FLSTM_DS4`, with a kernel size of three, which reached an mAP50 of 37.6 on the test set. The alternative experiments `FLSTM_DS4_KER5` and `FLSTM_DS4_KER7` improved on this, getting test mAP50's of 38.9% and 38.3%. This indicates that a more complicated sequential module is beneficial when dealing with a semantically complex feature.

The box-level VODS performed poorly. A box-level LSTM has been implemented with success, but then the task was only to find one object per frame [54]. However, the implementation does not hold for more than one object since when more than one bounding box is fed to the LSTM at a time, there is no mechanism for the association of boxes between different frames. For the LSTM to meaningfully adjust the box predictions, the objects must be fed to the LSTM to be in the same order each frame. The resulting representation, therefore, becomes too weak. For example, let us consider the case when the most confident box in a particular frame does not contain the same object as the most confident box in the next frame. The system wrongly tries to associate the two bounding boxes although they are in no way related, i.e. erroneously adjusting the bounding box locations and labels.

Although the results of the experiments indicate some differences between models, they need to be taken with a grain of salt for two reasons. First, due to time constraints, it was only possible to run each experiment once. It is therefore possible that with repeated runs that the difference in the mAP50 metric would not be statistically significant (although the large size of the dataset reduces the effect of the randomness in parts of the model). Second, the models were trained and tested on a German dataset, so it is not guaranteed that results would hold up for data from other parts of the world.

The sequential models did have consistently higher inference time than the baseline but to a varying degree. The largest factor was how big the feature maps of the added convolutional LSTM's were. This is natural since replacing a convolutional layer with a

ConvLSTM is essentially replacing a number of convolutional layers with more convolutional layers. This generally low impact on the inference time indicates that if adding ConvLSTMs in a network can improve the accuracy of a model and inference time simultaneously, since the network probably can be shrunk but maintaining higher accuracy.

It has been shown by [58] that a convolutional layer can be replaced by a so-called self-attention layer, which has a smaller computational cost but does not compromise on expressive power. A recent invention in the vision field is the Swin transformer [59], which is a completely convolution-free backbone network that can be used for various vision tasks like image classification and detection. It is a variation on the attention-based Transformer model originating from the natural language processing field. The Swin transformer backbone with a variation of the R-CNN head achieves state-of-the-art performance on a benchmark dataset, outperforming YOLOv4. This indicates that future work should explore how attention can be used in an ODS in general, but most importantly how it may incorporate temporal context. According to [60], the attention modules should be applied in the later stages of the ODS, in which we have seen some evidence that ConvLSTM's are the least useful. Our experiments indicate that the further down the CNN a sequential model is attached, the more advanced it has to be. The finding further confirms that advanced attention modules are useful in the later stages of a CNN. [61] constructed variants of ConvLSTMs where attention was included. The result was a nine-fold reduction in the number of parameters for all variants compared to the standard ConvLSTM. The system was used to predict hand gestures in video and achieved similar or better performance than the ConvLSTM-based system. Therefore, an attention-based ConvLSTM version of YOLO may potentially have outperformed the baseline.

A problem with high-resolution images and CNN's is that aggressive resizing is necessary, where much quality is removed. A simple solution would be to divide each image into n grid cells and then run an ODS on each cell. This would most likely increase the detection accuracy but to a high cost of inference time. Therefore, a significantly smaller model would be necessary to compensate for dividing the image into cells. According to [58], if the input and output channels are 128, a convolutional layer with kernel size being 3 has similar computational cost as an attention layer with "kernel size" being 19 (in vision attention models, it is called spatial extent). A potential approach to combat the resizing problem would be to use an attention system on the full-resolution image that predicts locations where traffic lights may be. This would be more likely to be computationally feasible since attention models can consider much larger receptive fields to the same computational cost.

For the classification task of a detection system, conversion from RGB to HSV colour space can be beneficial. The reason is that for distinguishing between the colours yellow, red and green it is sufficient to look only at the hue component. It is, therefore, simpler and more efficient to perform colour segmentation in the HSV space [55]. Additionally [62] show that colour segmentation algorithm achieves better detection accuracy on road signs when using the HSV colour space as opposed to the RGB colour space. They attribute the better performance of the algorithm mainly to the robustness of HSV colour space to different types of lighting conditions.

The primary metric used in this thesis to evaluate the models was mAP50. There is a possibility that the developed sequential model is better in another sense, e.g. consistency

of predictions, an mAP metric with a lower IOU threshold or with respect to another metric, e.g. STT-IOU. Because of the rather big difference in mAP50, there is a high probability for that the baseline system is better in most ways.

A potential way of encouraging a model to learn temporal dependencies would be to incorporate them in the loss function. For example, incorporating STT-IOU in the loss function of a sequential detection system is possibly a more efficient way of learning the temporal dependencies, just as incorporating the IOU loss using $\mathcal{L}_{CIOU}$ improved single-frame detection systems. This would also allow for tracking how well the model learns to model temporal context during training.

It should be noted that even if a VODS would confidently show higher mAP with the proper architecture and hyperparameters, it comes with some additional problems compared to a single-framed ODS. One example is the bias towards the behaviour of the vehicle the annotated data was recorded on. For example, if inference is run on a new vehicle that is more springy, it may lead to worse performance. An extreme version of this problem can be seen in figure 26 in appendix A, using footage of traffic lights recorded on a phone while walking. In this example, even though there is not a significant difference in the performance on the DriveU dataset between the sequential model and the baseline, the sequential model makes severe mistakes. This means that using a model utilising ConvLSTMs requires annotated data from not only different environments but also different vehicles. This makes the process of creating annotated data more expensive and complicated. Notice that all models perform significantly worse on those images, suggesting that the models also learned bias towards the cameras and scenes used in DriveU. A potential way to mitigate this problem is to use weights from earlier epochs, as the validation mAP50 observed during training does not display any hints of bias towards the DriveU dataset. A benefit of the ConvLSTM models over the baseline is reduced flickering in their predictions over several frames. From manual inspection of predictions on the test set (some of which are available in appendix A), it was observed that the sequential model makes more stable predictions but also fewer predictions overall. The baseline thus makes more predictions but also produces more false positives. This is likely the reason for the difference in performance on off traffic lights between the sequential models and the baseline. By running inference on the same sequences with `YOLO_SEQ`, however, yields similar predictions as with the sequential models. This indicates that this behaviour is associated with the training strategy and not necessarily the model.

Finally, `YOLO` and `FLSTM_P` were evaluated for different IOU thresholds. It was found that `YOLO` outperforms the sequential model for all thresholds, but the gap shrinks dramatically as the IOU threshold is increased. This is yet another indication that `YOLO` is less hesitant to make predictions than the sequential alternative. As the IOU threshold is increased, the models only detect the highly confident cases.

7 Conclusion

Modelling temporal context for traffic light recognition using RNNs is non-trivial. A proper investigation of whether temporal context can be efficiently modelled for improving detection accuracy involves exhaustive hyperparameter and model architecture search. This is a computationally expensive and time-consuming process. Our experiments indicate that the further down the network a sequential module is attached, the more complex it has to be to improve recognition accuracy.

The problem of recognising traffic lights is made considerably easier because traffic lights are strong illuminants with similar aspect ratios. Despite this fact, there are still difficult cases that a TLR system has to be able to handle. One of them is differentiating a traffic light from a vehicle brake light. Here the system has to rely heavily on context, which can be done by considering large enough receptive fields. However, the most challenging problem is the detection of off traffic lights. This problem has received very little attention in the literature since researchers in the field routinely exclude this class when developing TLR systems. Off traffic lights can be indistinguishable from other objects when seen from far away. But off traffic lights are only relevant in the case of blinking yellow traffic lights. We, therefore, suggest that a future TLR system should instead focus on inferring the relevant off states using the yellow traffic light, which is much easier to find.

Our experiments provide some evidence that considering longer sequences ($t > 3$) is necessary to maximise accuracy in a sequential vision model utilising ConvLSTMs. Since this requires a large amount of computation time and computer memory, using attention models to reduce the time complexity is an attractive alternative that needs to be explored further.

Lastly, a box-level LSTM without the association of boxes between frames yielded a lower detection accuracy than the baseline and the feature-level sequential models.

References

[1] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[2] J. B. Cicchino, "Effects of rearview cameras and rear parking sensors on police-reported backing crashes," *Traffic injury prevention*, vol. 18, no. 8, pp. 859–865, 2017.

[3] S. Sternlund, J. Strandroth, M. Rizzi, A. Lie, and C. Tingvall, "The effectiveness of lane departure warning systems—a reduction in real-world passenger car injury crashes," *Traffic Injury Prevention*, vol. 18, no. 2, pp. 225–229, 2017, PMID: 27624313. DOI: 10.1080/15389588.2016.1230672. eprint: https://doi.org/10.1080/15389588.2016.1230672. [Online]. Available: https://doi.org/10.1080/15389588.2016.1230672.

[4] L. C. Possatti, R. Guidolini, V. B. Cardoso, R. F. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Traffic light recognition using deep learning and prior maps for autonomous cars," in *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2019, pp. 1–8.

[5] Z. Ouyang, J. Niu, Y. Liu, and M. Guizani, "Deep cnn-based real-time traffic light detector for self-driving vehicles," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 300–313, 2020. DOI: 10.1109/TMC.2019.2892451.

[6] J. Müller and K. Dietmayer, *Detecting traffic lights by single shot detection*, 2018. arXiv: 1805.02523 [cs.CV].

[7] K. Yoneda, A. Kuramoto, N. Suganuma, T. Asaka, M. Aldibaja, and R. Yanase, "Robust traffic light and arrow detection using digital map with spatial prior information for automated driving," *Sensors*, vol. 20, no. 4, 2020, ISSN: 1424-8220. DOI: 10.3390/s20041181. [Online]. Available: https://www.mdpi.com/1424-8220/20/4/1181.

[8] M. Bach, D. Stumper, and K. Dietmayer, "Deep convolutional traffic light recognition for automated driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2018, pp. 851–858.

[9] H. Zhu, H. Wei, B. Li, X. Yuan, and N. Kehtarnavaz, "A review of video object detection: Datasets, metrics and methods," *Applied Sciences*, vol. 10, no. 21, 2020, ISSN: 2076-3417. DOI: 10.3390/app10217834. [Online]. Available: https://www.mdpi.com/2076-3417/10/21/7834.

[10] A. B. Qasim and A. Pettirsch, *Recurrent neural networks for video object detection*, 2020. arXiv: 2010.15740 [cs.CV].

[11] A. Fregin, J. Muller, U. Krebel, and K. Dietmayer, "The driveu traffic light dataset: Introduction and comparison with existing datasets," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3376–3383. DOI: 10.1109/ICRA.2018.8460737.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[15] K. Yamaguchi, K. Sakamoto, T. Akabane, and Y. Fujimoto, "A neural network for speaker-independent isolated word recognition," in *First International Conference on Spoken Language Processing*, 1990.

[16] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: `1512.03385 [cs.CV]`.

[17] B. Mehlig, *Machine learning with neural networks*, 2021. arXiv: `1901 . 05639 [cs.LG]`.

[18] Christopher Olah, *Lstm chain figure*, [Online; accessed February 17, 2021], 2015. [Online]. Available: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png`.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*, PMLR, 2013, pp. 1310–1318.

[20] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, *Convolutional lstm network: A machine learning approach for precipitation nowcasting*, 2015. arXiv: `1506.04214 [cs.CV]`.

[21] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: `1406.1078 [cs.CL]`.

[22] M. Siam, S. Valipour, M. Jagersand, and N. Ray, *Convolutional gated recurrent networks for video segmentation*, 2016. arXiv: `1611.05435 [cs.CV]`.

[23] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, *Yolov4: Optimal speed and accuracy of object detection*, 2020. arXiv: `2004.10934 [cs.CV]`.

[24] H. Wang, Y. Yu, Y. Cai, X. Chen, L. Chen, and Q. Liu, "A comparative study of state-of-the-art deep learning algorithms for vehicle detection," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 2, pp. 82–95, 2019.

[25] *MathWorks anchor boxes for object detection*, `https://se.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html`, Accessed: 2021-03-04.

[26] N. O. Salscheider, "Featurenms: Non-maximum suppression by learning feature embeddings," *arXiv preprint arXiv:2002.07662*, 2020.

[27] J. Uijlings, K. Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, pp. 154–171, Sep. 2013. DOI: `10.1007/s11263-013-0620-5`.

[28] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, *Feature pyramid networks for object detection*, 2017. arXiv: `1612.03144 [cs.CV]`.

[29] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, *Focal loss for dense object detection*, 2018. arXiv: 1708.02002 [cs.CV].

[30] R. Girshick, *Fast r-cnn*, 2015. arXiv: 1504.08083 [cs.CV].

[31] J. Dai, K. He, and J. Sun, "Instance-aware semantic segmentation via multi-task network cascades," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3150–3158.

[32] R. Padilla, W. L. Passos, T. L. B. Dias, S. L. Netto, and E. A. B. da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, 2021, ISSN: 2079-9292. DOI: 10.3390/electronics10030279. [Online]. Available: https://www.mdpi.com/2079-9292/10/3/279.

[33] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[34] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, 2016. arXiv: 1506.01497 [cs.CV].

[35] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[36] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[37] J. Redmon and A. Farhadi, *Yolo9000: Better, faster, stronger*, 2016. arXiv: 1612.08242 [cs.CV].

[38] ——, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. [Online]. Available: http://arxiv.org/abs/1804.02767.

[39] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, *Cspnet: A new backbone that can enhance learning capability of cnn*, 2019. arXiv: 1911.11929 [cs.CV].

[40] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Lecture Notes in Computer Science*, pp. 346–361, 2014, ISSN: 1611-3349. DOI: 10.1007/978-3-319-10578-9_23. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[41] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, *Path aggregation network for instance segmentation*, 2018. arXiv: 1803.01534 [cs.CV].

[42] D. Misra, *Mish: A self regularized non-monotonic activation function*, 2020. arXiv: 1908.08681 [cs.LG].

[43] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, *Cbam: Convolutional block attention module*, 2018. arXiv: 1807.06521 [cs.CV].

[44] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG].

[45] Z. Yao, Y. Cao, S. Zheng, G. Huang, and S. Lin, *Cross-iteration batch normalization*, 2021. arXiv: 2002.05712 [cs.LG].

[46] G. Ghiasi, T.-Y. Lin, and Q. V. Le, *Dropblock: A regularization method for convolutional networks*, 2018. arXiv: 1810.12890 [cs.CV].

[47] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, *Distance-iou loss: Faster and better learning for bounding box regression*, 2019. arXiv: 1911.08287 [cs.CV].

[48] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, *Generalized intersection over union: A metric and a loss for bounding box regression*, 2019. arXiv: 1902.09630 [cs.CV].

[49] L. Wu, J. Ma, Y. Zhao, and H. Liu, "Apple detection in complex scene using the improved yolov4 model," *Agronomy*, vol. 11, no. 3, p. 476, 2021.

[50] A. Broad, M. Jones, and T.-Y. Lee, "Recurrent Multi-frame Single Shot Detector for Video Object Detection," Tech. Rep.

[51] M. Liu, M. Zhu, M. White, Y. Li, and D. Kalenichenko, *Looking fast and slow: Memory-guided mobile video object detection*, 2019. arXiv: 1903.10172 [cs.CV].

[52] S. Tripathi, Z. C. Lipton, S. Belongie, and T. Nguyen, *Context matters: Refining object detection in video with recurrent neural networks*, 2016. arXiv: 1607.04648 [cs.CV].

[53] K. Chen, J. Wang, S. Yang, X. Zhang, Y. Xiong, C. C. Loy, and D. Lin, *Optimizing video object detection via a scale-time lattice*, 2018. arXiv: 1804.05472 [cs.CV].

[54] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, and H. Wang, *Spatially supervised recurrent convolutional neural networks for visual object tracking*, 2016. arXiv: 1607.05781 [cs.CV].

[55] A. Gómez Hernandez, F. Ribeiro de Alencar, P. Prado, F. Osorio, and D. Wolf, "Traffic lights detection and state estimation using hidden markov models," Jun. 2014, pp. 750–755, ISBN: 978-1-4799-3638-0. DOI: 10.1109/IVS.2014.6856486.

[56] K. Behrendt, L. Novak, and R. Botros, "A deep learning approach to traffic lights: Detection, tracking, and classification," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 1370–1377.

[57] K. Wang, X. Tang, S. Zhao, and Y. Zhou, "Simultaneous detection and tracking using deep learning and integrated channel feature for ambint traffic light recognition," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–11, 2021.

[58] J.-B. Cordonnier, A. Loukas, and M. Jaggi, *On the relationship between self-attention and convolutional layers*, 2020. arXiv: 1911.03584 [cs.LG].

[59] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, *Swin transformer: Hierarchical vision transformer using shifted windows*, 2021. arXiv: 2103.14030 [cs.CV].

[60] P. Ramachandran, N. Parmar, A. Vaswani, I. Bello, A. Levskaya, and J. Shlens, "Stand-alone self-attention in vision models," *arXiv preprint arXiv:1906.05909*, 2019.

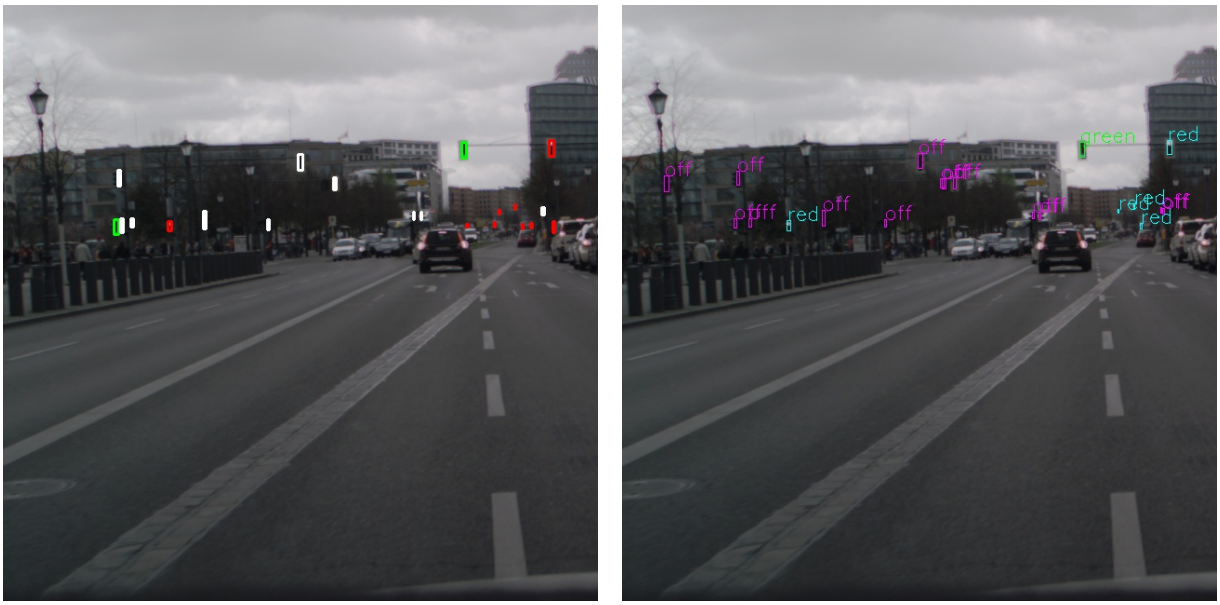
[61] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. Shah, and M. Bennamoun, "Attention in convolutional lstm for gesture recognition," *NIPS*, 2018.

[62] N. M. Ali, N. K. A. M. Rashid, and Y. M. Mustafah, "Performance comparison between rgb and hsv color segmentations for road signs detection," *Applied Mechanics and Materials*, vol. 393, no. 1, pp. 550–555, 2013.

Appendices

A Inference examples

In this appendix, there are side by side comparisons of a few different models when conducting inference on the DriveU test set. In ground truth images, white boxes represent the off state, red for the red state, orange for the red-yellow state, yellow for the yellow state and finally green for the green state.



(a) Ground truth.

(b) `YOLO`

(c) `YOLO_SEQ`

(d) `FLSTM_P`

Figure 23: Model comparison for a scene with many traffic lights. The YOLOv4 baseline model detects considerably more off traffic lights.

(a) Ground truth.

(b) YOLO

(c) YOLO_SEQ

(d) FLSTM_P

Figure 24: Model comparison for a scene with few traffic lights. YOLO is the only model that detects the off traffic light. Notice also the annotation error in the dataset, where a red traffic light was labeled as a yellow.

(a) Ground truth.

(b) YOLO

(c) YOLO_SEQ

(d) FLSTM_P

Figure 25: Model comparison for a scene with some traffic lights that are both far away and somewhat occluded by trees. YOLO has the best recognition performance but FLSTM_P recognises the most relevant traffic lights.

(a) `YOLO` (b) `YOLO_SEQ`

(c) `FLSTM_P`

Figure 26: Model comparison for a scene with some traffic lights from Uppsala, Sweden. `YOLO_SEQ` and `FLSTM_P` both wrongly detect non-traffic lights as traffic lights.

B Difficulties in DriveU

The DriveU dataset is the largest traffic light dataset in terms of number of frames and annotations at the time of writing the thesis. There are some errors in the dataset and some other difficulties for machine learning models, especially for sequential models. In this appendix, some of those difficulties are visualised. During the writing of this thesis, an update of DriveU was released with annotation errors and off state inconsistencies fixed. This was however published too late to incorporate in the thesis, which is why all image labels are from version one.

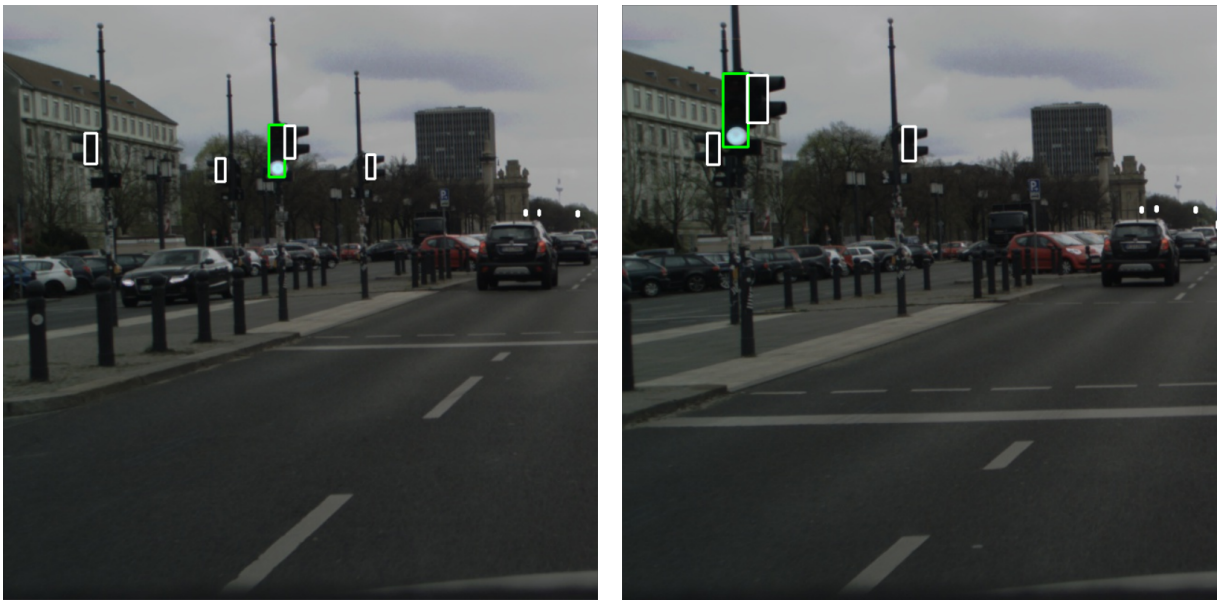
(a) A red traffic light which is annotated as yellow.

(b) Another red traffic light labeled as yellow.

Figure 27: Examples of annotation errors in version one of DriveU.

(a) First frame. (b) Second frame.

Figure 28: Example of how two consecutive frames in DriveU can be significantly different from each other, due to a camera FPS of 15. This makes it difficult to model temporal dependencies between frames.



(a) First frame. (b) Second frame.

Figure 29: Another example of how different two consecutive frames are in DriveU.

DEPARTMENT OF MATHEMATICAL SCIENCES
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY