BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

DEEP LEARNING FOR OBJECT DETECTION

DETEKCE OBJEKTŮ POMOCÍ HLUBOKÝCH NEURONOVÝCH SÍTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **RADOSLAV PITOŇÁK**
AUTOR PRÁCE

SUPERVISOR **Ing. LUKÁŠ TEUER**
VEDOUCÍ PRÁCE

BRNO 2019

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

17159

Student:	**Pitoňák Radoslav**
Program:	Informační technologie
Název:	**Deep Learning for Object Detection**
	Deep Learning for Object Detection
Kategorie:	Zpracování obrazu

Zadání:

1. Study the basics of deep neural networks and backpropagation.
2. Learn about the current methods for creating deep neural networks, especially convolutional neural networks.
3. Choose a method suitable for object detection.
4. Obtain a dataset suitable for experiments.
5. Implement the chosen method and perform experiments on the chosen dataset.
6. Compare achieved results and discuss the potential continuation of your work.
7. Create a poster representing your work and results.

Literatura:

- Krizhevsky, A., Sutskever, I. and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
- Grishick et. al.: Rich feature hierarchies for accurate object detection and semantic segmentation. CVPR 2014.

Pro udělení zápočtu za první semestr je požadováno:

- Points 1 to 3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce:	**Teuer Lukáš, Ing.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2018
Datum odevzdání:	15. května 2019
Datum schválení:	14. května 2019

Abstract

This thesis analyzes different object detection methods which are based on deep neural networks. In the beginning, the convolutional neural networks are described and commonly used object detection methods are compared. In the following parts, the proposal and implementation of the object detection model trained on the specific dataset are described. In conclusion, the achieved results of this model are discussed and compared with the results of other methods.

Abstrakt

Táto práca sa zaoberá metódami používanými na detekciu objektov ktoré používajú hlboké neurónové siete. Na začiatku sú popísané konvolučné neurónové siete a porovnané bežne používané metódy na detekciu objektov. V dalšej časti sa venuje návrhu a implementácii vybranej metódy natrénovanej na špecifickom datasete. Na konci tejto práce sú výsledky, ktoré tento model dosiahol diskutované a porovnané s výsledkami iných metód.

Keywords

Object detection, deep neural networks, convolutional neural networks, computer vision, BDD, YOLO

Kľúčové slová

Detekcia objektov, hlboké neurónové siete, konvolučné neurónové siete, počítačové videnie, BDD, YOLO

Reference

PITOŇÁK, Radoslav. *Deep Learning for Object Detection.* Brno, 2019. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Lukáš Teuer

Rozšírený abstrakt

V posledných rokoch zažívame rýchly vývoj v oblasti hlbokých neurónových sietí. Tieto siete prekonali tradičné algoritmy používané v mnohých oblastiach strojového učenia. Jednou z týchto oblastí je aj počítačové videnie. Táto bakalárska práca sa zaoberá problémom detekcie objektov. Detekcia objektov je problém, ktorý okrem klasifikovania objektov v obraze identifikuje aj ich presné umiestnenie. Toto je jednoduchá úloha pre ľudí, pretože náš zrakový systém je presný a rýchly, ale neurónové siete potrebujú veľa dát a výpočetného výkonu na dosiahnutie dobrých výsledkov. V dnešnej dobe však týchto dát vieme nazbierať pomerne veľké množstvo a tak tieto systémy dosahujú kvalitné výsledky, ktoré sú už využívané v mnohých oblastiach priemyslu, ako napríklad autonómne autá alebo diagnostika chorôb v medicíne.

V prvej časti tejto práce sú stručne vysvetlené neurónové siete a proces učenia neurónových sietí, ktoré sú inšpirované spôsobom, akým funguje ľudský mozog. V ďalšej časti sú detailnejšie opísané konvolučné nurónové siete a vrstvy, používané v týchto sieťach. Konvolučné siete sú najviac využívane pri práci so signálmi ako obraz a zvuk. Okrem základných vrstiev ako konvolučná a plne prepojená vrstva sú popísané vrstvy ktoré zlepšujú generalizovanie siete na testovacie dáta a zabraňujú problému pretrénovania siete. Ďalej sú opísané súčasné metódy využívané na detekciu objektov a to hlavne tzv. *region proposal* metódy a *one-stage* metódy.

Súčasťou práce je aj návrh a implementácia jedného modelu. V návrhu je opísaná vybraná dátová sada ktorá obsahuje obrázky z mestského prostredia, zachytené počas rôznych častí dňa a poveternostných podmienok. Zároveň sú v návrhu detailnejšie popísané architektúry sietí použitých na experimenty. Implementovaný model využíva metódu YOLO, ktorá je vhodná na detekciu v reálnom čase. Na implementáciu bola využitá knižnica PyTorch. PyTorch je knižnica na prácu s tenzormi, ktorá okrem toho poskytuje množstvo ďalšej funkcionality na prácu s neurónovými sieťami. Výsledný model je dostupný ako Python balík a konfigurovateľná aplikácia ovládaná z príkazového riadku.

V poslednej časti boli vykonané experimenty nad dátovou sadou. Prvý experiment bol zameraný na metódu YOLO, ktorá bola najprv natrénovaná bez použitia predtrénovanej siete na extrakciu príznakových vektorov. V druhom experimente bola použitá predtrénovaná sieť na extrakciu vektorov čo urýchlilo proces trénovania. Ďalej bol vykonaný experiment s modelom Tiny YOLOv3, natrénovaným v prostredí Darknet, na rovnakej dátovej sade. Výsledkom je model s nižšou presnosťou ale veľmi dobrou rýchlosťou detekcie. Posledný experiment bol vykonaný s modelom YOLOv3, taktiež natrénovaným v prostredí Darknet. Tento model dosiahol najlepšie výsledky na testovacej dátovej sade. Na záver sú všetky výsledky experimentov porovnané podľa dosiahnutej presnosti.

Deep Learning for Object Detection

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Ing. Lukáš Teuer. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Radoslav Pitoňák
May 15, 2019

Acknowledgements

I would like to thank to my supervisior Lukáš Teuer for provided help and guidance. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Contents

Chapter 1

Introduction

In recent years there has been a rapid development in deep neural networks. Deep neural networks outperform traditional algorithms used in many fields of machine learning. One of these fields is computer vision. Object detection is an interesting problem in computer vision. The definition of the problem is to determine where are the objects located in the image and also classify them to appropriate categories. It is an easy task for a human, as our visual system is accurate and fast, but computers need a big amount of data and computational power to be able to perform object detection tasks with good accuracy. Object detection models have been already proven to work for various use-cases such as autonomous vehicles or diagnosis of diseases in the medicine.

This thesis analyzes the different techniques used for object detection. The main goal is to create an object detection model which is able to predict objects on the street. Models like this are nowadays already used in autonomous vehicle systems. In these systems, besides the accuracy, we aim also on the real-time speed of detection. Because of this reason, YOLO object detection method is used for our experiments. Results are compared using mean average precision object detection metric.

In the next chapter, the basic concepts used within neural networks are described, how they work and how the training of a neural network is performed. In the same chapter also convolutional neural networks, which are mainly used for computer vision problems nowadays are described. Then in Chapter 3 an overview of existing methods used for object detection and history of this area is presented. Specifically, region proposal methods and one-stage detectors are described and compared in detail. In Chapter 4, the dataset and architectures of implemented neural networks are described. In Chapter 5, the implementation details and results of the experiments are presented. In conclusion, the achieved results are summarized and future development is discussed.

Chapter 2

Neural networks

From its earliest days, work on artificial neural networks was motivated by the observation that the human brain computes in a completely different way from the standard digital computer [11]. This is especially needed for tasks like image recognition where the standard way of programming is not effective, because of complexity to define all scenarios. A standard neural network is composed of many simple, connected processors called neurons, each producing a sequence of non-linear activations.

Biological neuron

A biological neuron is composed of cell *body* a tubular *axon* and a lot of hair-like *dendrites* [29]. A scheme of a biological neuron is shown in Figure 2.1. Information is propagated through *synapsies* which are the small gaps between the end of a bulb and a dendrite. Axon of a neuron is connected with many other neurons through synapsis. A neuron will send an output if impulse signals from other neurons which fall upon its dendrites exceed a certain threshold.

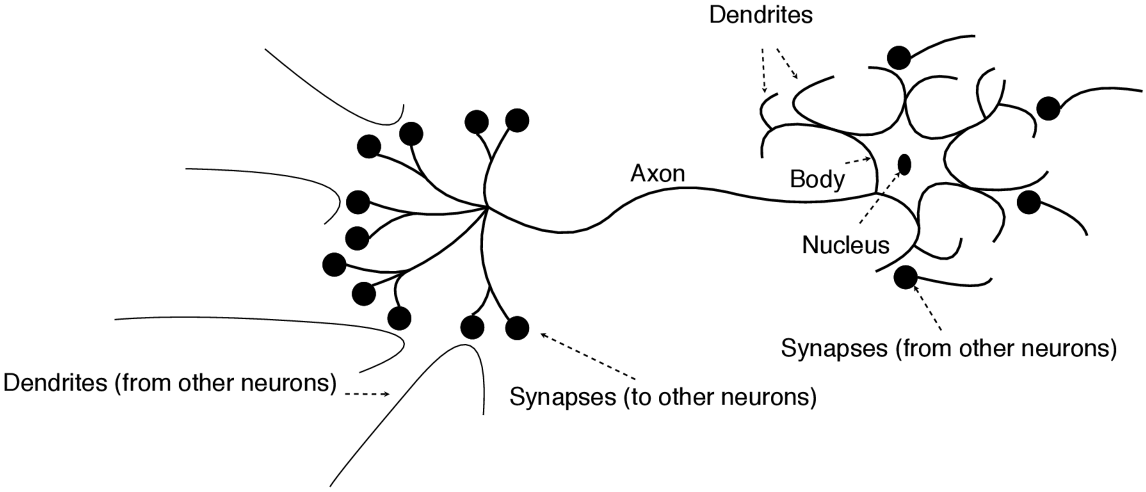

Dendrites
Axon
Body
Nucleus
Synapses (from other neurons)
Dendrites (from other neurons)
Synapses (to other neurons)

Figure 2.1: Human brain neuron. Adapted from: [29].

Artificial neuron

An artificial neuron is an essential unit for the operation of neural networks [12]. It is inspired by human brain neuron shown in Figure 2.1. The neural model consists of three elements: synapses, adder, and activation function. Synapses are characterized by their weight. The signal a_i that comes to the input of synapse i, which is connected to neuron j is multiplied by input weight w_{ij} of that synapse. There is one extra input a_0 which is called *bias* and it equals to a constant. An adder produces a sum of input signals multiplied by the weights of the corresponding synapses of the neuron. And finally, an activation function g is usually a non-linear function controlling the output of the neuron. A simple mathematical model of neuron introduced by McCulloch and Pitts (1943) [28] is shown in Figure 2.2.

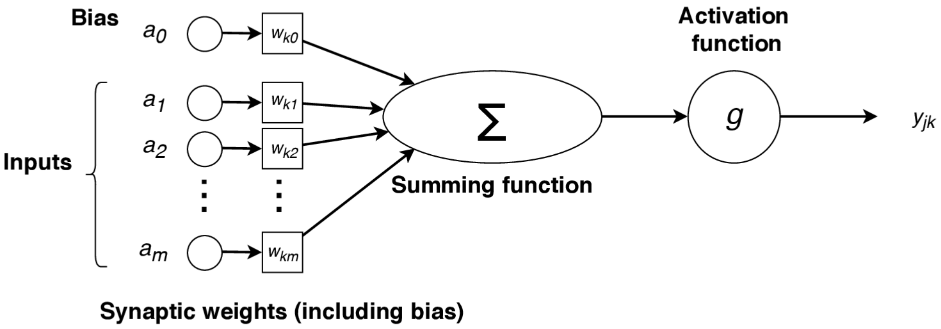


Figure 2.2: Mathematical model of the neuron. Adapted from: [12].

Neural network structure

Once we understand how artificial neuron works, the next step is to show how can we form them to build a network. There are two ways how to do this *A feed-forward network* and *Recurrent network* [35]. In this chapter just feed-forward network is explained, if you want to know more about recurrent neural networks please refer to book *Deep learning* [10] (Chapter 10).

A feed-forward neural network forms a directed acyclic graph which means it has connections just in one direction [35]. Every node receives input from previous nodes and sends output to the next nodes. A feed-forward network has no internal state other than the weights themselves. They are built from layers such that each unit receives input only from units in the immediately preceding layer. There are cases when this is not true, i.e Residual neural networks [15]. In practice, these networks are usually a multilayer which means they have one or more layers of *hidden units*, that are not connected to the outputs of the network. On Figure 2.3 a multilayer feed-forward network is shown.

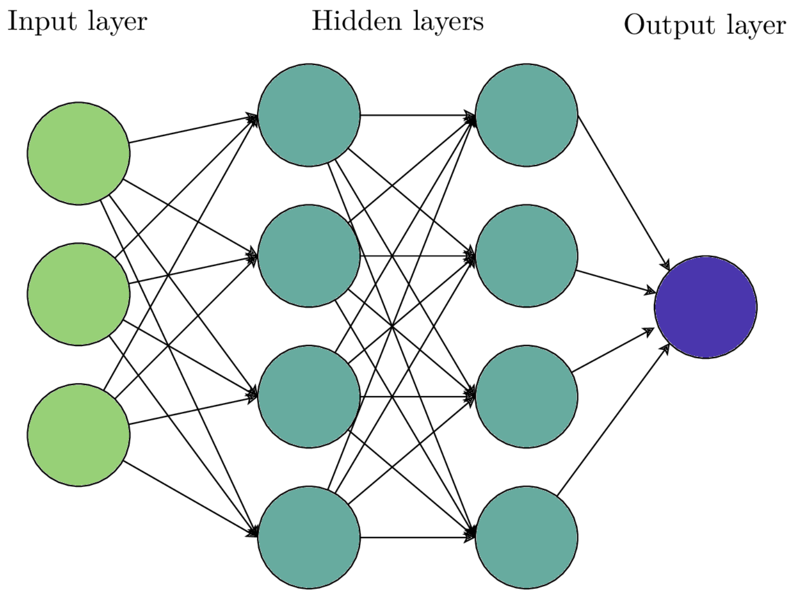


Figure 2.3: Multilayer feed-forward network has one or more layers of hidden units.

Activation functions

The activation function is usually either a hard threshold, called perceptron or logistic function in which case we called it sigmoid perceptron [35]. These activation functions provide the important property that the whole neural network can represent a nonlinear function. The logistic activation function also brings the advantage of being differentiable. Basically activation function decides the output of the neuron. In this section, two popular activation functions are described: ReLU and Leaky ReLU. You can find out more about activation functions in the book *Deeplearing book* [10] (Chapter 6).

ReLU

A rectified linear unit (ReLU) is an activation function recommended being used by default in most of the neural networks [10]. It is defined as:

$$g(x) = max(0, x) \tag{2.1}$$

Applying this function for linear transformation output produces non-linear transformation. Because the function is almost linear, a lot of properties that make linear models generalize well and easy to optimize using gradient-based methods are preserved. The graph of ReLU function is shown in Figure 2.4.

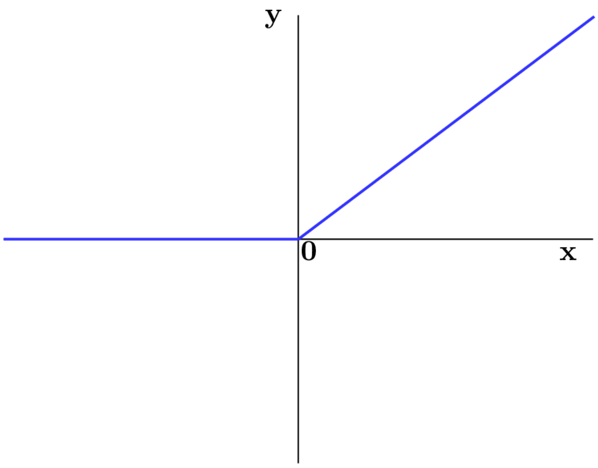


Figure 2.4: ReLU is an activation function which has a lot of properties that make linear models easy to optimize.

Leaky ReLU

A potential disadvantage of ReLU during the optimization is that whenever the unit is not active, the gradient is zero [27]. This could lead to cases, where a unit is never activated because a gradient-based optimization algorithm will not adjust the weights of a unit that never activates initially. This problem is called *vanishing gradient problem*. Besides vanishing gradient problems we might expect training to be very slow. Leaky rectified linear unit (Leaky ReLU) (see Figure 2.5) is nearly similar to ReLU but allows for small non-zero gradient when the unit is not active. It is defined as:

$$g(x) = \begin{cases} 0.1x & x < 0 \\ xfor & x >= 0 \end{cases} \tag{2.2}$$

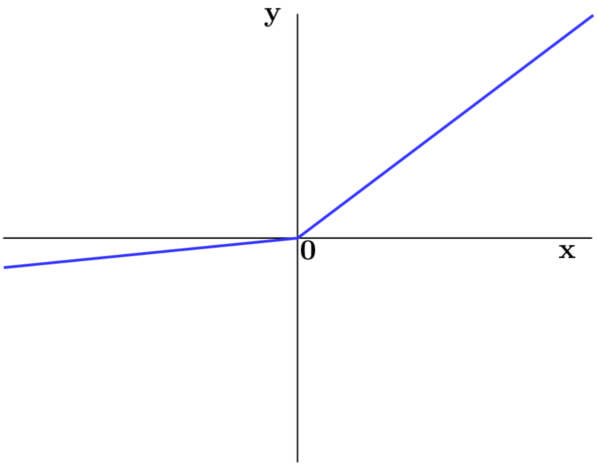


Figure 2.5: Leaky ReLU is an activation function which help to solve the vanishing gradient problem.

Learning in neural networks

In this section a learning process of the neural networks is described. We will describe what is a loss function and briefly mention two commonly used loss functions. Then we will explain how this function can be minimized using *gradient-descent* algorithm and how error is back-propagated through all layers in the network.

Loss function

In the learning process of neural networks, our goal is to minimize a *loss function*. The loss function is used to measure the difference between the output of the network from the desired output. During the optimization of the loss function, the learning parameters are tuned. One of the commonly used loss functions for regression is *mean squared error* (MSE), defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.3}$$

MSE is a sum of squared distances between the target variable and the predicted values. It is commonly used to measure error for *linear regression*. Say that there is linear function with input x and real-valued coefficients w_0 and w_1:

$$h_w(x) = w_1 x + w_0 \tag{2.4}$$

If we have training set of n values in the x,y plane, then task of finding h_w that best fits these data is called linear regression [35]. To fit this line we need to find values of the weights w_0 and w_1 that minimize the MSE loss.

For classification i.e when output is probability distribution, there is a commonly used loss function called *cross entropy loss*. It is defined as:

$$CE = -(y_i \log{(\hat{y}_i)} + (1 - y_i)\log{(1 - \hat{y}_i)}) \tag{2.5}$$

where y_i is the ground truth vector and $\hat{y}_i$ is predicted vector of probabilities. Neural networks which use cross-entropy as a loss function usually has softmax activation layer [2] as an output layer.

Optimization

Gradient descent algorithm is essential for the training of neural networks. The idea behind it is to calculate how each parameter needs to be changed to decrease the loss function [37]. Because the loss function can be differentiated with regard to each parameter, the gradient vector can be calculated. The parameters that locally minimize the lost function can be found by moving along the negative gradient for each parameter. The size of a step is defined by hyper-parameter called *learning rate*. Optimization of a loss function on the training data does not guarantee the good accuracy of prediction on the testing data. Optimization algorithms commonly used nowadays are *Stochastic gradient descent* (SGD) [19] or *Adam* [20].

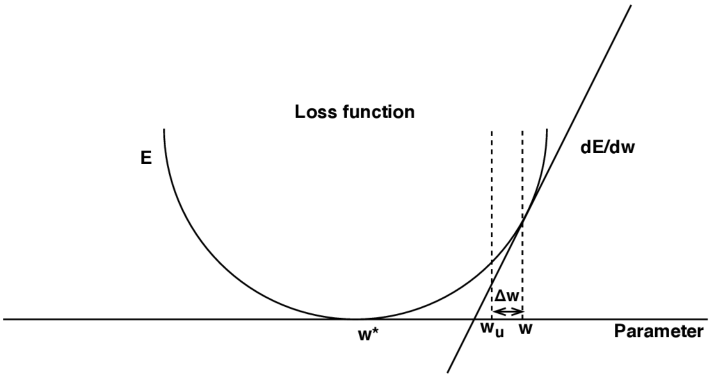


Figure 2.6: Optimization using gradient descent. Adapted from: [37]

Backpropagation

Backpropagation algorithm was introduced by [34] (Rumalhart et al. 1986). It solves the problem that comes with the addition of the hidden layers into the neural network. We are able to calculate the error of the network output and compare it with the desired output from training data. But we need not know how we need to update every weight in the network. We can do this by back-propagating the error.

Say that unit has a real-valued output y_j and we are trying to optimize loss function E. Then we start backward pass by computing the $\partial E/\partial y$ for each of the output units. Then we can apply the *chain rule* to compute a $\partial E/\partial x_j$.

$$\partial E/\partial x_j = \partial E/\partial y \cdot dy_j/dx_j \tag{2.6}$$

This means that we know how the loss will be affected by the change of the total input x. We also know that total input is a linear function of states of the lower level units and also a linear function of the weights on the connections so it is possible to compute how the change of these states and weights will affect the loss [34].

2.1 Convolutional neural networks

A convolutional neural network (CNN) is an architecture proven to be useful in computer vision applications. The motivation behind this is that it is useful to represent image regions with filter outputs [7]. Convolution operation takes advantage of three ideas that can help improve a machine learning system [10]. The first one is *sparse interactions.* This means that when the image of specific size is processed, neurons can extract elementary visual features such as oriented edges, endpoints, corners or similar features with kernels that occupy dramatically fewer pixels than original image [23]. This means that we have to store fewer parameters which reduces memory requirements and increasing statistical efficiency [10]. The second one is *parameter sharing.* This means that one parameter is used in more than one function in a model. And finally, parameter sharing has the consequence that the layer has a property called *equivariance to translation* which means that if the input changes, the output changes in the same way. In this section, a discrete convolution is explained. To learn more about the motivation behind the usage of convolution in convolutional neural networks, refer to book *Deep learning* [10] (Chapter 9).

Discrete convolutions

A discrete convolution is a linear transformation. A discrete convolution of two discrete signals $x[n]$ and $h[n]$ is defined as:

$$y[n] = \sum_{k:-\infty}^{+\infty} x[k]h[n-k] \tag{2.7}$$

In Figure 2.7 the example of discrete convolution is provided. The blue grid is called *input feature map*. It is common to have multiple feature maps stacked one onto another [4]. An example can be *channels* when we speak about images. The grey grid is called *kernel* of value slides across the feature map. At each location, the product is calculated between each element of the kernel and the input element it overlaps, and the results are summed up to obtain the output in the current location. The final output of this procedure is called *output feature map*. If there are multiple kernels, this procedure is repeated for each of them to form the same number of output feature maps. Example of the discrete convolution in Figure 2.7 is an instance of $2D$ convolution but it can be generalized to N-D convolutions.

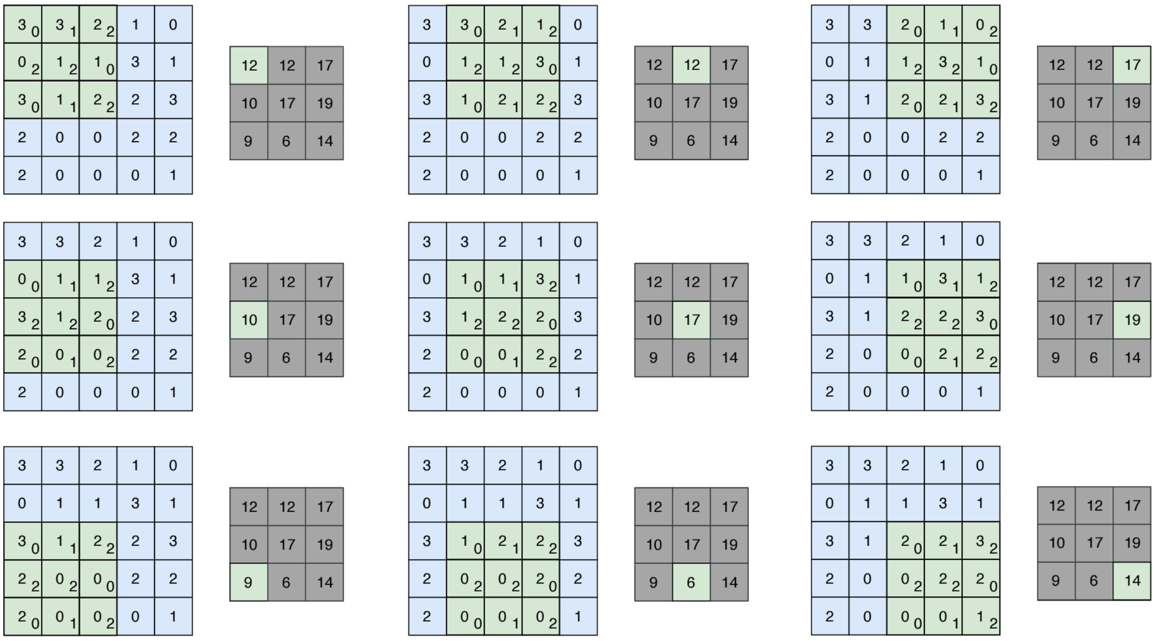


Figure 2.7: Example of convolution with one kernel. Adapted from: [4]

2.2 Layers in neural networks

In this section we will describe a layers commonly used in neural networks. Besides the basic layers we will touch also layers which helps neural networks to generalize better on testing data.

2.2.1 Fully connected layer

A fully connected layer is the layer where all neurons from adjoining layers are pairwise connected but there are no connections between neurons within the same layer. The output value is calculated as a dot product of the input vector and row of weight matrix [21]. Name fully connected is used because the output is calculated using all input elements (see Figure 2.8).

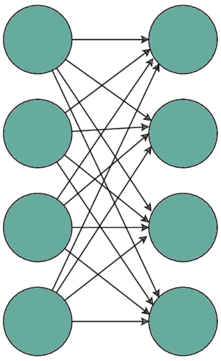
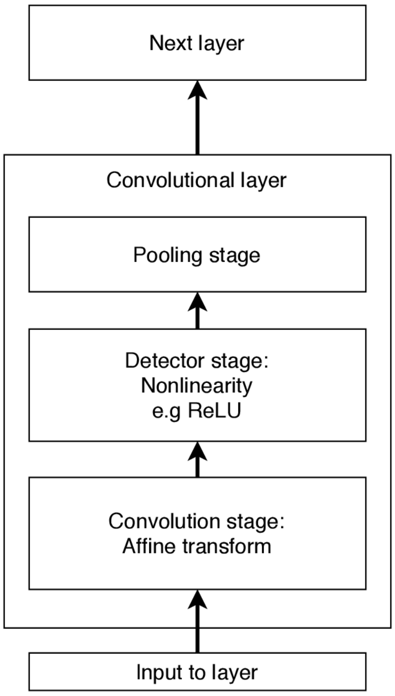


Figure 2.8: Fully connected layers.

2.2.2 Convolutional layer

Typical convolutional layer in neural networks is composed of three stages [10]. In the first stage, the layer performs several convolutions in parallel to produce a set of linear activations. In the second stage, a non-linear function such as ReLU is applied to each of the linear activations. And finally, in the third stage, a pooling function is used to modify the output layer in addition.

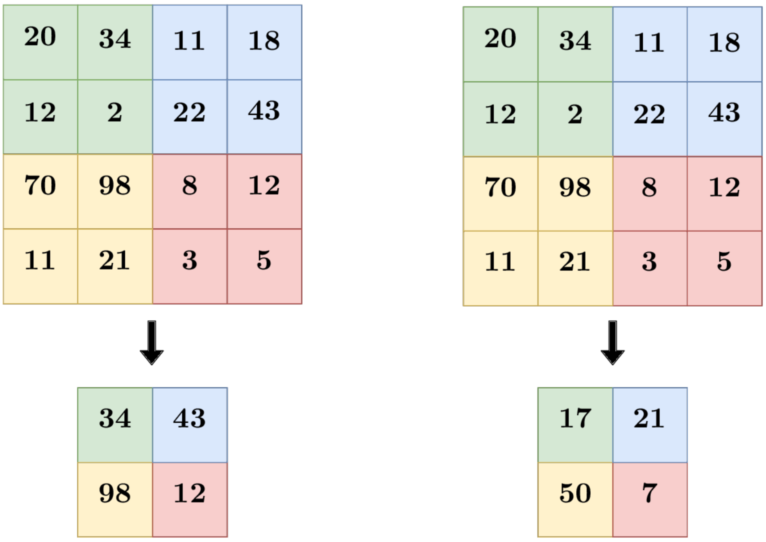


Next layer
Convolutional layer
Pooling stage
Detector stage: Nonlinearity e.g ReLU
Convolution stage: Affine transform
Input to layer

Figure 2.9: Convolutional layer. Adapted from: [10]

2.2.3 Pooling layer

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs [10]. The function of pooling layer is to gradually reduce the spatial size of the representation in order to reduce the number of parameters and the network computation [18]. Most commonly used form of pooling is max pooling (see Figure 2.10 (a)) and in past also average pooling (see Figure 2.10 (b)) but it was recently replaced by max-pooling which was proven to works better in practice. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations [18].



20	34	11	18
12	2	22	43
70	98	8	12
11	21	3	5

34	43
98	12

(a) Max pooling

20	34	11	18
12	2	22	43
70	98	8	12
11	21	3	5

17	21
50	7

(b) Average pooling

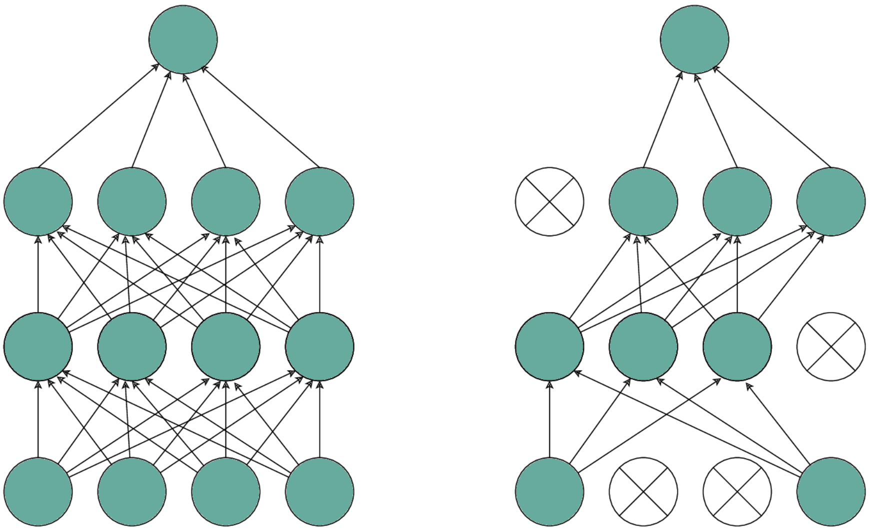
Figure 2.10: Different types of 2×2 *pooling.*

2.2.4 Dropout layer

Dropout layer introduced by [36] is a technique that tries to solve two issues related to neural networks. First one is a problem called *overfitting* when the network producing good results on training data but poor results on testing data. The second one is training using several different network architectures which also helps to generalize better. Dropout is referring to dropping of hidden units in neural networks [36]. Each unit is retained with certain probability p. Dropping of units with certain probability is used just during the training time. During the testing, the Dropout layer is ignored. On figure 2.11 the idea behind the dropout layer is shown.

2.2.5 Batch normalization layer

Batch normalization is a technique that dramatically reduces the training time of the deep neural networks [17]. By adding this layer to network architecture normalization is performed for each training *mini-batch*. This allows using of higher learning rates and less attention for weight initialization. Batch normalization also provides regularization and in some cases eliminating the need for Dropout [36].



(a) Standard multi-layer neural network.

(b) After applying dropout.

Figure 2.11: Neural network with dropout.

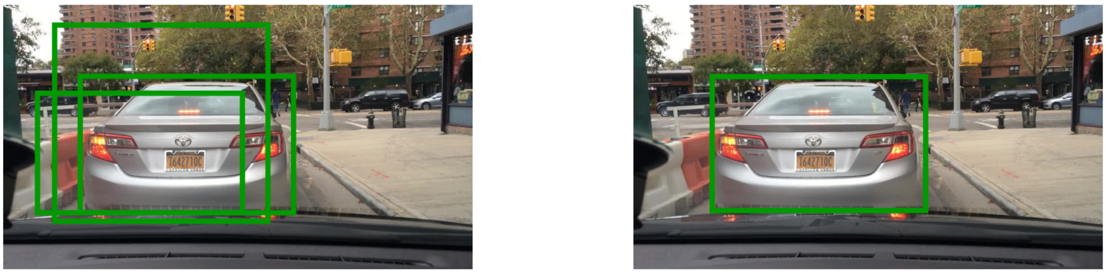
Chapter 3

Object detection methods

Object detection builds upon the success of object recognition classifiers. Object detection methods besides the classification of objects in the image, predicts also their position inside the image. This is usually done by predicting coordinates of bounding boxes surrounding the object and confidence of this prediction. Object detection is a more complex task than object recognition because objects inside the image are usually of various sizes and ratios.

3.1 History

Before the deep neural networks become popular, the most effective object detection method was Deformable Part Model (DPM) [5]. It uses deformable part models and detects objects across all possible locations and scales. This approach after integration with post-processing techniques, i.e bounding box predictions, non-maximum suppression (see Figure 3.1) and rescoring of detections using contextual information achieved state-of-the-art results on object detection tasks.



(a) Before non-maximum supression

(b) After non-maximum supression

Figure 3.1: In many of the object detection methods it is possible that one object is detected more than once. Non-maximum supression algorithm is addressing this issue. It first remove all predictions with confidence score lower than certain threshold. Then it takes a prediction with highest confidence score and remove all predictions that have intersection over union with this prediction higher than certain threshold (usually 0.5). Images are taken from BDD dataset [41].

One of the first deep learning approaches to object detection was *DetectorNet* [39]. DetectorNet replaced last softmax layer of AlexNet [22] with regression layer which predicts fixed sized object binary mask. After being resized to the image size, this binary mask represents one or several objects. If the particular pixel lies within the bounding box of an object of a given class it should have value 1, otherwise, it should have value 0.

3.2 Region proposals methods

Current successful object detection methods build on the idea to generate a large number of candidate boxes and use CNN as a feature extractor.

This approach was successfully used by R-CNN method [9]. R-CNN object detection system is composed of three modules. The first generates category-independent region proposals using selective search algorithm [40]. These proposals define a set of candidates boxes for the detector. The second module is large CNN that extracts fixed-size feature vectors from each candidate box. This CNN has the architecture of AlexNet [22]. As CNN expects input image of fixed size (227 × 227 pixels), for each region all pixels are warped in a tight bounding box around it to the required size. The third module is a set of class-specific linear SVMs [1].

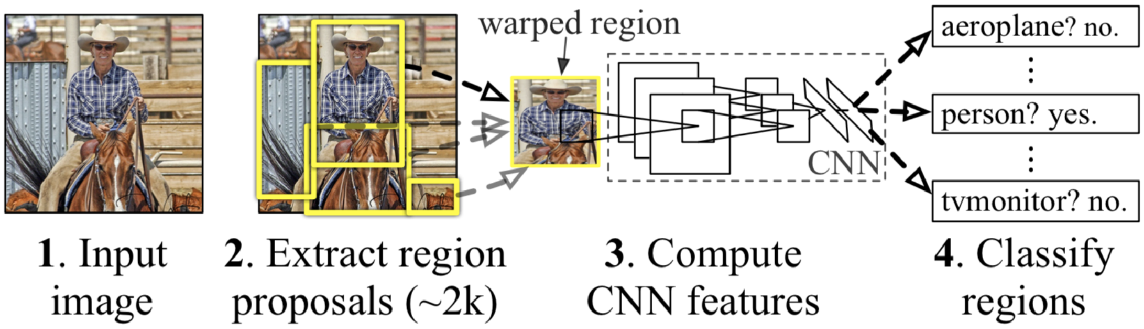


Figure 3.2: R-CNN. Taken from: [9]

Besides solid accuracy, R-CNN is slow because it performs forward pass of CNN for each region proposal without sharing any computations as stated in [8]. This issue was addressed and solved by the SPP-net introduced in article *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition* [14]. SPP-net is built upon R-CNN pipeline. It runs convolutional layers only once on the entire image regardless of a number of proposed regions. Then classifies each object proposal using a feature vector extracted from the shared feature map. This approach speeds up the R-CNN by maximum 100x during test time.

Fast R-CNN

Fast R-CNN [8] proposes a new approach that fixes the issues with R-CNN and SPP-net while improving speed and accuracy. First, it processes the entire image through convolutional and max-pooling layers to produce a convolutional feature map. Then for each object proposal, a region of interest (RoI) pooling layer extracts a fixed length feature vector. These vectors are then processed through fully connected layers. At the end of the network, there are two branches running in parallel: softmax branch predicting the class probability of an object and bounding box regression branch which predicts coordinates of a bounding box.

[1] SVM – Support vector machine, see [16] for more information

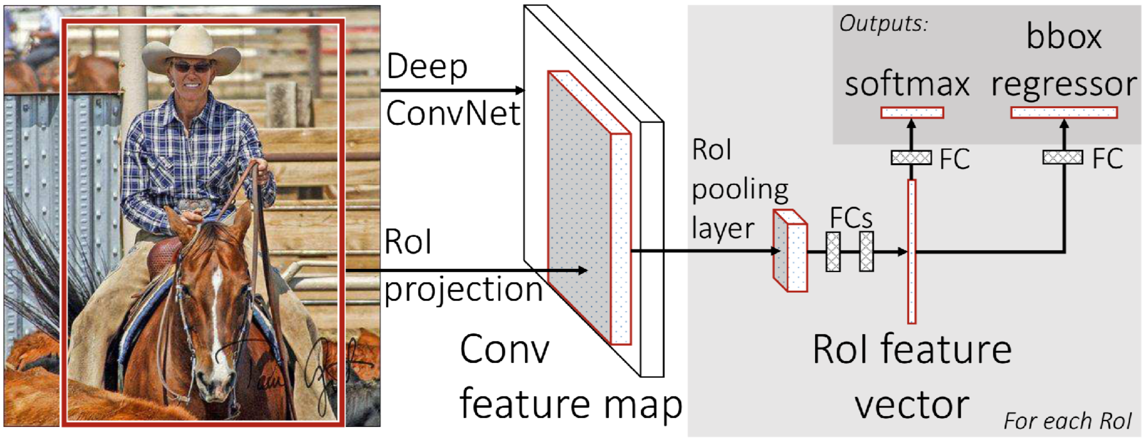


Figure 3.3: Fast R-CNN architecture. Taken from: [8]

Faster R-CNN

Faster R-CNN [33] replaces generating of region proposals boxes with a selective search algorithm, by *Region Proposal Network* (RPN). It has been shown that Selective search is a bottleneck of Fast R-CNN as it takes 2s per image on CPU. RPN is a fully convolutional neural network that shares full-image convolutional features with the detection network. This provides nearly cost-free region proposals. The input of RPN is an image and output is a set of region proposals each with a confidence score.

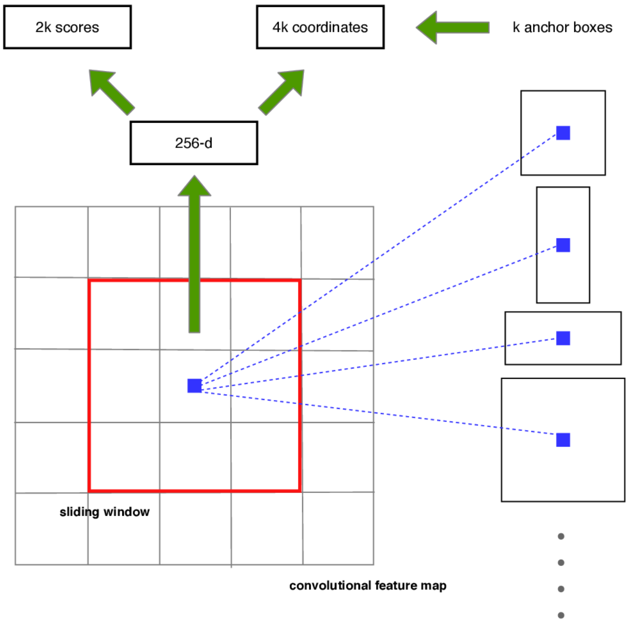
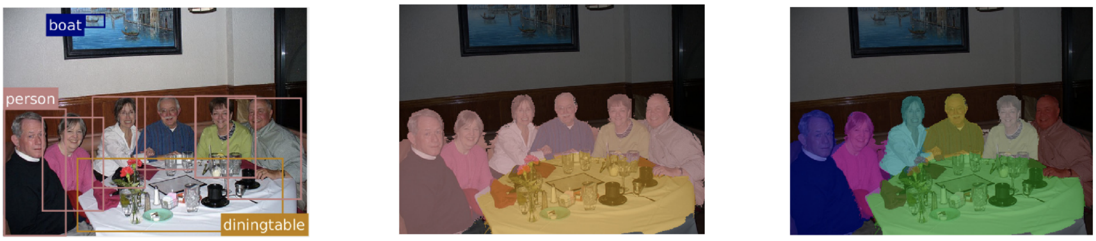


Figure 3.4: Region proposal network. Adpated from: [33]

Mask R-CNN

Mask R-CNN [13] is a method that extends the Faster R-CNN [33] by adding a branch for object mask prediction on each RoI in parallel with the classification and bounding box regression branch. This method is providing an object instance segmentation. This means that besides predicting the bounding box for every object in the image, it classifies every pixel to a fixed set of categories. The Mask R-CNN branch is a small fully convolutional layer [26] which give just small computational overhead to Fast RCNN framework. The key role in mask predictions has the introduction of *RoIAlign* layer. It is an improvement of RoIPool which is aligning the extracted features properly with the input.



(a) Object detection (b) Semantic segmentation (c) Instance segmentation

Figure 3.5: Mask R-CNN is providing instance segmentation (c) which is combination of object detection (a) and semantic segmentation (b). Taken from: [1]

3.3 One-stage detectors

One-stage detectors are generally faster and simpler than region proposals methods but sometimes in a tradeoff for best accuracy of detection. In this chapter, we will describe the most popular one-stage detector methods used for real-world use-cases mainly because of the real-time speed of detection.

3.3.1 YOLO (You only look once)

In this section YOLO object detection method introduced in article *You Only Look Once: Unified, Real-Time Object Detection* [30] is described. YOLO comes with a different approach to object detection. Instead of repurposing the object recognition classifiers for object detection it proposes a system that unifies object detection to a single neural network. This means that in one evaluation of an image it predicts a set of bounding boxes as well as class probabilities. Yolo reasons about image globally and it is extremely fast at the test time while keeping accuracy comparable with region proposal methods.

This system split an image equally into $S \times S$ grid. For each grid cell, it predicts B number of bounding boxes with confidence scores and C number of class probabilities. The cell is responsible for the detection of an object if the center of an object is present in this cell.

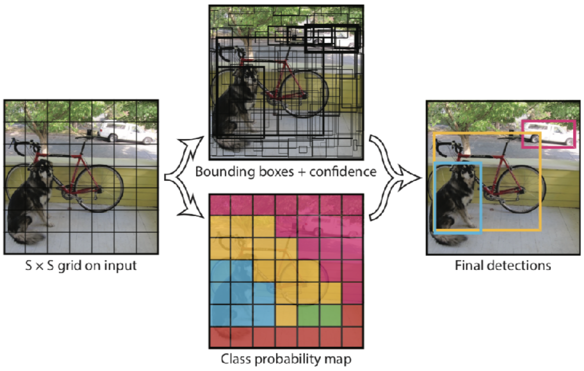


Figure 3.6: Yolo system. It first split the image into $S \times S$ grid. Then it predicts two bounding boxes with confidence score for each cell. It predicts also a class probability for each cell. Taken from: [30]

Network architecture

YOLO network is built from 24 convolutional layers followed by two fully connected layers. Convolutional layers are responsible for feature extraction from the image and fully connected layers for making the prediction of bounding box coordinates, confidence scores, and class probabilities. The final layer produces output tensor of size $(S \times S \times (B \cdot 5 + C))$ YOLO uses custom feature extractor which is inspired by GoogLeNet [38] model for image classification, but instead of inception modules used by GoogLeNet, it uses 1×1 reduction layers followed by 3×3 convolutional layers. YOLO also comes with a fast version which consists of 9 convolutional layers instead of 24 and fewer filters in those layers.

Bounding box predictions

There are five predictions for each bounding box: x, y, w, h, c, where (x, y) are the coordinates of the center of a detected object relative to a grid cell, (w, h) are width and height relative to image size and c is the confidence score which represents *Intersection over union* (IOU) between the ground truth and predicted bounding box. The confidence score is formally defined as:

$$P(Object) * IOU_{prediction}^{truth} \tag{3.1}$$

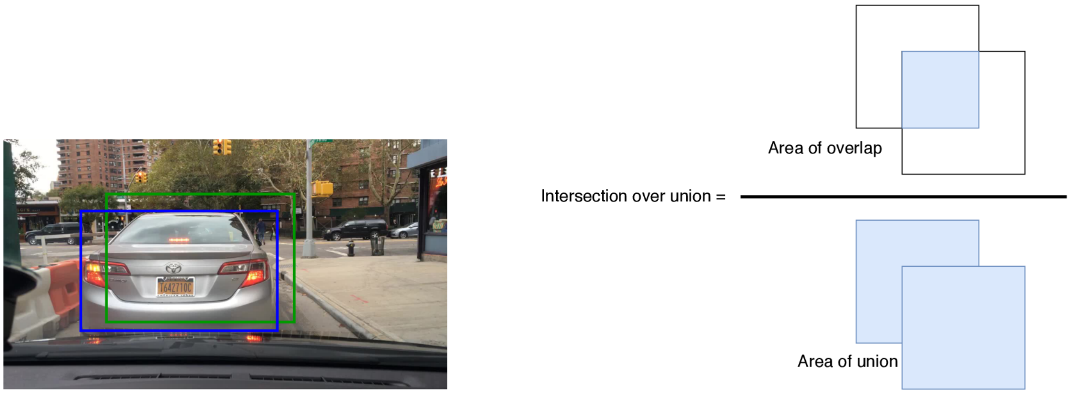
If there is no object present in the cell, the confidence score should be zero.

Class probabilities

One set of class probabilities is predicted for each cell. These probabilities are conditioned for a grid cell containing an object. Conditional class probabilities are at test time multiplied by confidence score:

$$P(Class_i|Object) * P(Object) * IOU_{prediction}^{truth} = P(Class_i) * IOU_{prediction}^{truth} \tag{3.2}$$

This encodes both the class probability and also how predicted bounding box fits the object



(a)

(b)

Figure 3.7: IOU is used to measure how much predicted bouning box overlaps with the ground truth bounding box. On Figure (a) ground truth rectangle (blue) overlaps predicted bounding box (green). Image is taken from BDD dataset [41]. On figure (b) is shown how IOU of two bounding boxes is computed.

Loss function

YOLO optimizes for sum-squared error in the output of the model because it is easy to optimize even though it does not maximize average precision. It treats localization error equally with classification error which might not be flawless. There is also a problem that in most of the images, there are many cells that do not contain the object. This pushes these cells confidence scores to zero, often overwhelming the gradient from cells containing objects. This can lead to model instability, leading to the early divergence of training.

To solve this problem of model instability, YOLO uses two hyperparameters in the loss function. The first is λ_{coord} which increases the loss from bounding box coordinate prediction and second is λ_{noobj} which reduces the loss of confidence predictions for boxes that do not contain objects. Authors of YOLO suggested to set them as $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$.

YOLO predicts several bounding boxes per grid cell. Only one bounding box predictor is responsible for each object during the training. This predictor is assigned according to which predictor has the highest current IOU with the ground truth. This leads to specialization among bounding box predictors - each predictor gets better to predict certain sizes, aspect ratios, or object classes.

Limitations

YOLO struggles with detection of small objects since every cell is predicting just two bounding boxes and one class probability per cell [30]. This means that if there are many objects in one cell YOLO will not detect all of them.

Since YOLO model learns to predict bounding boxes from training data it may struggle with generalizing predictions to new objects with different aspect ratios. Also YOLO model uses relatively rough features for predicting bounding boxes since the neural network architecture consists of many pooling layers.

The main source of error is incorrect localization. During the training loss function weights error in small bounding boxes equally, as in large bounding boxes, a however small error in small bounding box has a much higher impact on *IOU* than a small error in a large bounding box.

3.3.2 YOLOv2 and YOLOv3

The limitations mentioned in the previous section are addressed in the next versions of YOLO. The first set of improvements were proposed in the paper *YOLO9000: Better, Faster, Stronger* [31]. There was batch normalization added after every convolutional layer, which led to more than 2% increase in model mAP and removed the need for dropout in order to prevent overfitting. In YOLOv2 there is fine-tuned classification network at resolution 448 × 448, instead of original 224 × 224 used by ImageNet[22]. This increased mAP by more than 4% because filters are better adjusted for detection which is performed on higher resolution.

A significant change is removing fully connected layers and replacing them with *anchor boxes*. With the introduction of anchor boxes, there is also resize of input image resolution into 416 × 416. An odd number is chosen because of one cell in the center of the image. Large objects are usually in the center of the image so it is better to have just one cell in the center compared to four. In YOLOv2, class predictions are made for every anchor box rather than just for every cell as it is in YOLOv1. To help model learn sizes and aspect ratios of anchor boxes faster, YOLOv2 uses k-means clustering on training set bounding boxes to find good priors. You can read more about k-means clustering algorithm in [25].

Another improvement is the usage of *fine-grained features*. YOLOv2 is performing detection on 13 × 13 feature map. This can cause problems with detecting small objects. To solve this, a passthrough layer is added to bring 26 × 26 feature map from earlier layers. This layer concatenates high resolution features with low-resolution features by stacking outlaying features into different channels similar to the identity mappings in ResNet [15]. The detector runs on top of this expanded feature map. In order to predict on variety input dimensions, instead of fixed image size, every 10 batches it randomly changes input dimensions. Since YOLOv2 model downsamples by a factor 32, randomly chose dimensions are multiples of 32 ranging from 320 to 608. This model achieved state-of-the-art results on higher resolution while maintaining real-time speed. YOLOv2 also comes with new network classifier called *Darknet-19* (See Figure 3.8) which has 19 convolutional layers and 5 max-pooling layers.

In the so far last article from this series *YOLOv3: An Incremental Improvement* [32], feature extractor is changed to *Darknet-53* (See Figure 3.8). It is inspired by Darknet-19 and by Residual Networks [15]. Besides the change of architecture YOLOv3 perform detection on 3 scales of the extracted feature map.

3.3.3 Single Shot MultiBox Detector (SSD)

SSD was introduced in the article *SSD: Single Shot MultiBox Detector* [24] and it is a method for detecting images using a single deep neural network. SSD model predicts a fixed number of bounding boxes and scores for those boxes. Scores are representing the probability of object class instances presence. This is followed by non-maximum suppression to produce the final detections.

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		3 × 3/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		3 × 3	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	3 × 3	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		3 × 3/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	3 × 3	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		3 × 3/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	3 × 3	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	3 × 3	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		3 × 3/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	3 × 3	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	3 × 3	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

(a)

	Type	Filters	Size/Stride	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3/2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3/2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3/2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3/2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3/2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

(b)

Figure 3.8: Darknet-19 (a) architecture introduced in YOLOv2 and Darknet-53 (b) architecture introduced in YOLOv3 article with usage of residual connections. Adapted from [31] and [32]

Network architecture

Initial layers of the network are based on standard architecture used for image classification. Those are followed by convolutional feature layers, which decrease in size progressively and allow to perform detection on multiple scales in comparison with YOLOv1 [30]. There are multiple default bounding boxes associated with each feature map cell for multiple feature maps. At each feature map cell, the offsets are predicted relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. The network architecture is shown in Figure 3.9.

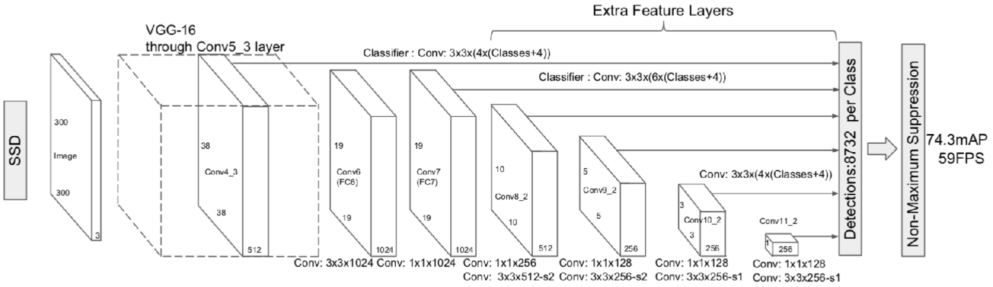


Figure 3.9: Single shot detector network architecture. Taken from: [24]

Chapter 4

Proposal

Training of object detection models consists of several steps. The first step is to prepare a dataset with annotated objects for training and testing. Our goal is to create and train detector of objects in the street (e.g cars, trucks or persons). A detector like this can be useful specifically for use cases like autonomous vehicles. Because of this, the training dataset should contain images taken at a different time of the day and with different weather conditions. Next step is to train a detector. This can be done in several ways, we can first train initial layers on a big annotated dataset like ImageNet [3] and then add more layers on top of the network to train detection on dataset annotated for detection. Some methods allow to skip this step and train classification and object detection from ground up. In this thesis we experimented with both of these approaches and results are presented in the next chapter.

In the systems for autonomous vehicles, besides the accuracy, the focus is also on achieving the real-time speed. This is the key aspect for these systems, since car needs to react very quickly for the events on the road. Because of this, we chose YOLO object detection method for our experiments, instead of the more accurate, but slower region proposals methods.

4.1 Network architectures

YOLOv1. First network architecture is inspired by YOLOv1. It is composed of 24 convolutional layers with Leaky ReLU as an activation function. Convolutional layers are followed by two fully connected layers with dropout after the first fully connected layer. Batch normalization is applied after every convolutional block. Convolutional blocks are displayed in Table 4.1. The input image size is 448×448. The output of the network is tensor $7 \times 7 \times (2 \cdot 5 + 10)$ as it predicts two bounding boxes per cell in a total of five predictions (x, y, w, h, c) and objects are classified into 10 classes.

	Type	Filters	Size/Stride
	Convolutional	64	7 × 7/2
	Maxpool		2 × 2/2
	Convolutional	192	3 × 3
	Maxpool		2 × 2/2
	Convolutional	128	1 × 1
	Convolutional	256	3 × 3
	Convolutional	256	1 × 1
	Convolutional	512	3 × 3
	Maxpool		2 × 2/2
4×	Convolutional	256	1 × 1
	Convolutional	512	3 × 3
	Convolutional	512	1 × 1
	Convolutional	1024	3 × 3
	Maxpool		2 × 2/2
2×	Convolutional	512	1 × 1
	Convolutional	1024	3 × 3
	Convolutional	1024	3 × 3
	Convolutional	1024	3 × 3/2
	Convolutional	1024	3 × 3
	Convolutional	1024	3 × 3
	Connected		
	Connected		

Table 4.1: YOLOv1 architecture. There is a batch normalization layer after every convolutional block and Leaky ReLU is used as the activation function.

During the training, the following loss function (4.1) is optimized. Loss function only penalizes classification error if there is an object in the cell and also penalizes localization error just if the predictor is responsible for the detection of that object [30].

$$\begin{aligned}\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2]\\ +\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2]\\ +\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2\\ +\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2\\ +\sum_{i=0}^{S^2}\sum_{c\in classes}\mathbb{1}_{i}^{obj}(p_i(c)-\hat{p}_i(c))^2\end{aligned} \tag{4.1}$$

Where 1_i^{obj} represents if an object is present in the cell i and 1_{ij}^{obj} represent that j-th bounding box predictor in cell i is responsible for the prediction of this object. Responsible means that bounding box predictor has the highest IOU with the ground truth among other predictors in the cell i [30].

In the second experiment with this network, we took advantage of the *transfer learning*. Transfer learning is a process when we take a model trained for a different task (in this case object classification) then freeze initial layers and train top layers for the new purpose. In this experiment, we used pretrained ResNet-50 from PyTorch model zoo [1]. In the official PyTorch documentation is reported that this model achieved accuracy 23.85 in Top-1 error in ImageNet classification challenge.

[1] PyTorch model zoo - https://pytorch.org/docs/stable/torchvision/models.html

To compare results of this network two more models were trained. First of them is a model based on *Tiny YOLOv3* architecture and the other one is based on *YOLOv3* architecture.

Tiny YOLOv3. Tiny YOLO network was created for extremely fast detection of objects. It is composed of just 13 convolutional layers. It is fast to train and achieving real-time detection time during the test time. The network architecture is shown in Figure 4.1.
YOLOv3. Last architecture that was used is YOLOv3. The reference implementation of the network shown in Figure 3.8 was used for the experiments. It is the most advanced version of YOLO so the most accurate results are expected.

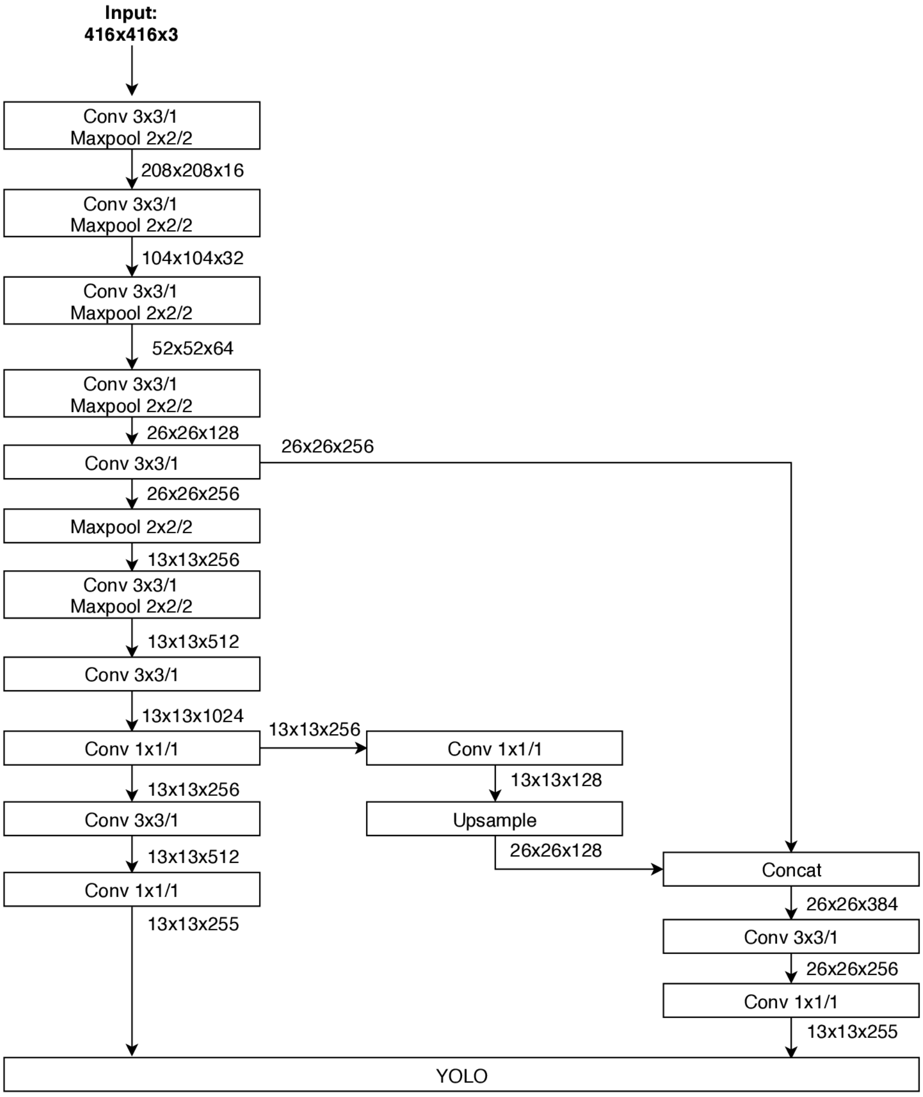


Figure 4.1: Tiny YOLOv3 architecture.

4.2 Dataset

For the training of the detector, Berkley Deep Drive (BDD) dataset was used [41]. This dataset contains images split to training, validation and test categories. Every image is annotated with bounding boxes around objects of 10 different classes. Images inside the dataset were taken during various weather conditions (see Figure 4.5) and time of the day (see Figure 4.6). Because dataset contains many of the scenes from the highway (See Figure 4.4) it contains on average just 1.2 persons per image in comparison there is on average 9.7 cars per image. Supporting toolset for pre-processing of data is available on GitHub [2]. The size of the validation set is 10000 images and the size of the training set is 69863 images.

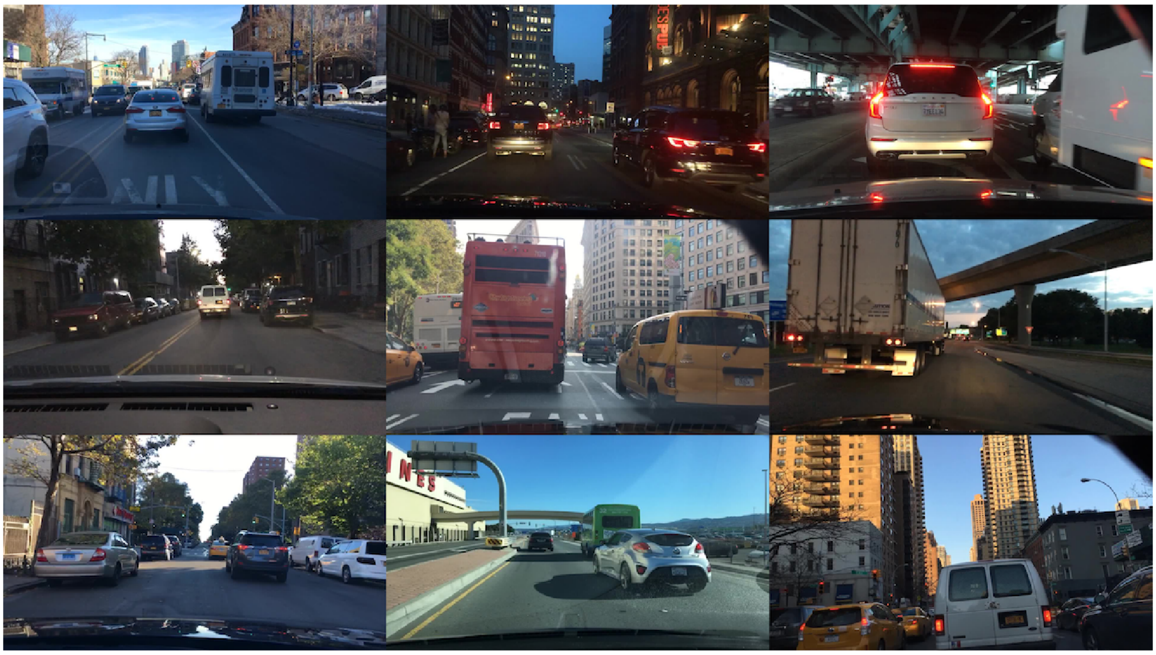
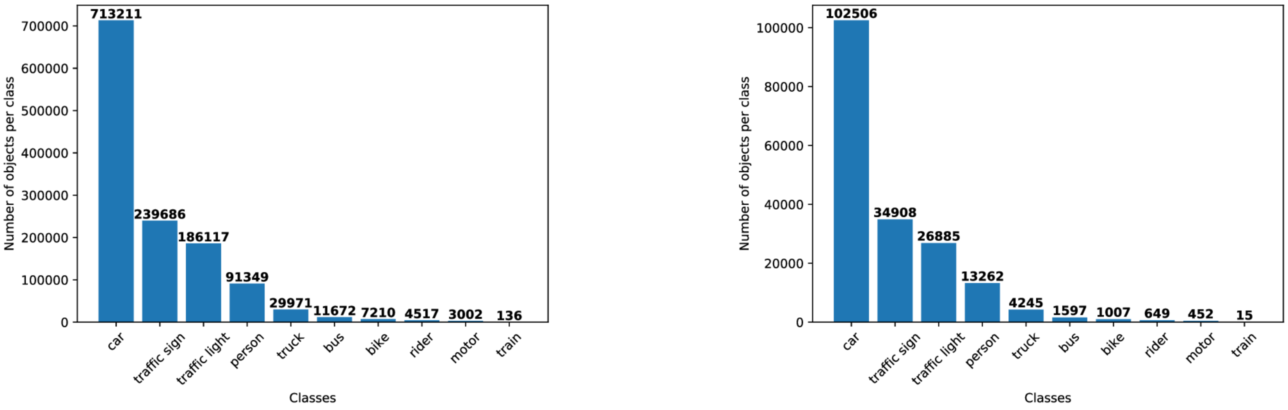


Figure 4.2: Samples from BDD dataset training data. Taken from: [6]

In [41] authors made experiments using Faster-RCNN on this dataset. Training was done separatly for three of the domains: *daytime*, *city* and *clear*. For *clear* domain they were able to achieve 34.0% mAP on out-domain testing and 36.6% mAP on in-domain testing. For *daytime* the results were 25.9% mAP out-domain and 36.6% mAP on in-domain testing and finally best accuracy was achieved by training on *city* domain 34.5% mAP out-domain and 42.0% mAP in-domain. For more detailed report see [41].

[2] https://github.com/ucbdrive/bdd-data



(a) Training dataset class distribution

(b) Validation dataset class distribution.

Figure 4.3: Dataset class distribution. Both the training and validation dataset have a similar class distribution. Classes: bike, rider, motor and train are not very well represented in the training dataset.

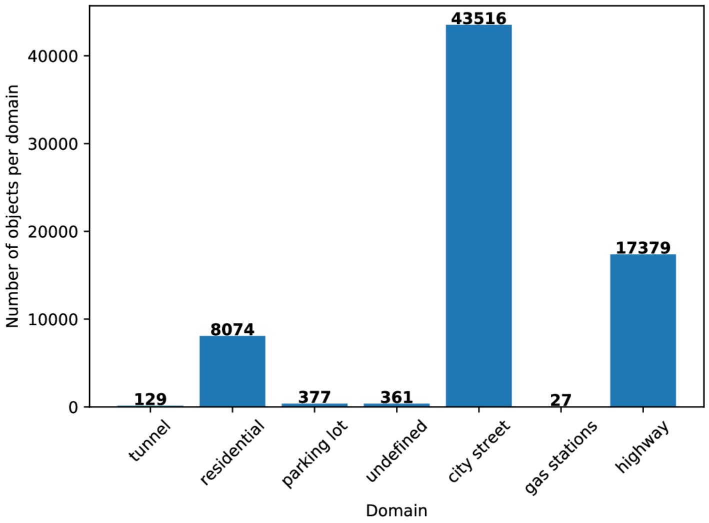


Figure 4.4: Scene diversity of training dataset. The most of the images are from the street or highway.

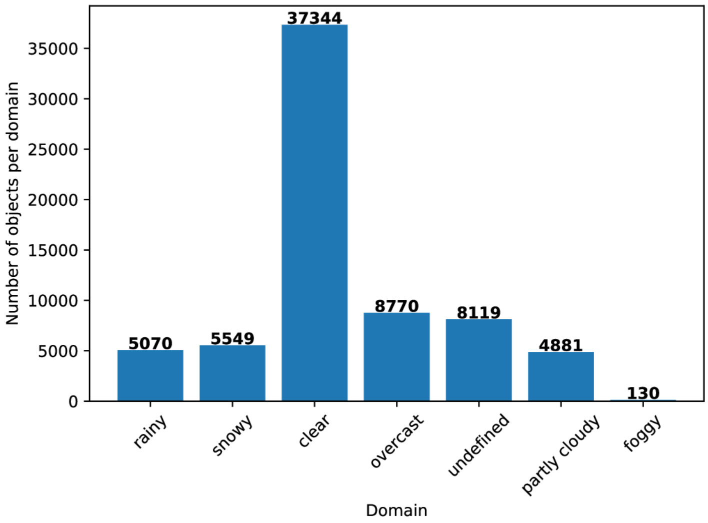


Figure 4.5: Weather diversity of training dataset. Images are mainly taken during the clear weather.

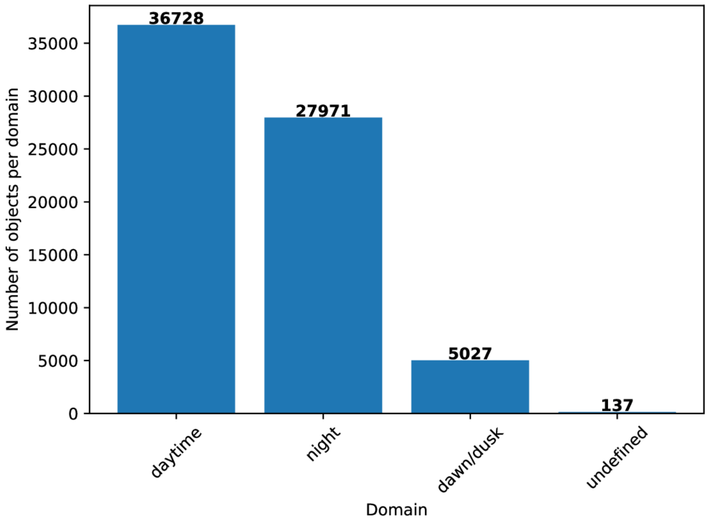


Figure 4.6: Time of the day diversity of training dataset. Images taken during the day are balanced with those taken during the night.

4.3 Object detection metrics

In our experiments, *average precision* metric is used to evaluate the performance of the resulting models. To be able to understand this metric we need to first explain what is *precision* and *recall*. Precision measures the percentage of correct predictions. Recall measures how good is the model in finding all *true positives*.

$$Precission = \frac{True\,positives}{True\,positives + False\,positives} \tag{4.2}$$

$$Recall = \frac{True\,positives}{True\,positives + False\,negatives} \tag{4.3}$$

Prediction is considered to be true positive if it has $IOU > 0.5$ with ground truth, otherwise, it is considered to be false positive. Prediction is also classified as false positive if there is a duplicate prediction of the same object. A predicted bounding box with $IOU > 0.5$, but the wrong classification is a false negative. The average precision is calculated as the area under the Precision-Recall curve. Mean average precision is the average of AP calculated for every class.

4.4 Metacentrum

Effective training of convolutional neural networks requires solid hardware. For the training of the models mentioned above, MetaCentrum GPU clusters were used. MetaCentrum[3] is the activity of CESNET focusing mainly on developing the grid infrastructure in the Czech Republic. It allows using of GPU clusters for educational and research purposes. For this thesis mainly clusters *konos* and *doom* were used. These clusters consists of four GPUs Nvidia GeForce GTX 1080 Ti or two Nvidia Tesla K20 5GB respectively.

[3] https://metavo.metacentrum.cz/

Chapter 5

Experiments and implementation

In this chapter, the implementation and results of the experiments are discussed. In total, we did two experiments with model based on YOLOv1 method, one with Tiny YOLOv3 and one with YOLOv3 based model. The implementation of YOLOv1 model is described in more details.

5.1 Implementation

The model based on YOLOv1 is implemented using Python programming language and it is available as Python package which can be installed via *pip*[1] and be used as configurable command line application. It supports various commands to train, test and validate the model on BDD dataset. PyTorch[2] framework is used for the implementation. PyTorch is an open-source tensor library which supports multiprocessing for faster data loading and provides a lot of functionality that comes handy when working with deep neural networks. It is written in Python and C++ programming languages and supports both GPU and CPU computing.

The models based on Tiny-YOLOv3 and YOLOv3 were trained using Darknet[3] framework implementation by AlexeyAB. Darknet is an open-source neural network framework written in C/C++ programming language. It is easy to configure for training on a custom dataset. Since Darknet supports different type of bounding box annotations than Berkley Deep Drive dataset a simple script for conversion was created and used to transform these annotations.

Implementation details

Dataset. For implementation of dataset PyTorch utils provides `Dataset` class. Then the `__get_item__` method is overriden with pre-processing of dataset into YOLOv1 format. This method returns one sample - tuple image, ground-truth tensor. For working with dataset there is useful `DataLoader` class from PyTorch utils which allows to iterate through dataset and get batches of specific size.

Data augmentation. Training of neural networks sometimes require more data than is available. One of the possible options for exapanding the dataset is data augmentation.

[1] https://pypi.org/

[2] https://pytorch.org/

[3] https://github.com/AlexeyAB/darknet

Dataset can be expanded by creating new images from existing by applying transformations such as shift rotation, flip, distort, shading or applying hue. PyTorch provides these transformations in package `torchvision`. For this model shading, and applying hue data augmentation was used.

Neural network. Neural Network is represented using PyTorch `nn.Module`. In constructor there are layers split into the blocks using `nn.Sequential`. The forward pass of the network is implemented in `forward` method.

Loss function. The loss function implements mathematical formula from equation 4.1. The loss function is similary as a neural network implemented as `nn.Module`. In a forward pass the value of the loss is calculated for the whole mini-batch. Backward pass is done automatically using PyTorch's auto grad mechanism.

Training and validation. Training and validation dataset is loaded using `DataLoader` class from PyTorch's utils. Training has configurable parameters like *learning rate* or *batch size*. Trained weights are saved in the file specified by command line argument using `torch.save` method. Every 1000 iterations there is backup model saved, which provides the ability to continue training from the latest checkpoint. The output of the validation is annotation files for every image in the validation dataset. This `.txt` files were then compared with ground-truth files using a python script to calculate mAP[4] of the model.

Visualization. For object detection application it important to be able to visualize the results of the detection. For this purpose `Visualization` class was created. It uses libraries like OpenCV[5] or Matplotlib[6].

5.2 Results of the experiments

In this section the experiments using different network architectures described in Chapter 4 are presented. First the results of experiments with YOLOv1 model are presented and then experiments with Tiny-YOLOv3 and YOLOv3 models. At the end of this chapter the results are compared using mAP metric.

5.2.1 Experiments using YOLOv1

In the first experiment YOLOv1 model was trained without taking advantage of the transfer learning. The network was trained for 20 epochs with batch size 64. Adam was used as the optimizer for YOLOv1 loss function with weight decay equals to 0.0005. The whole training dataset was used for training. Then the detection was performed on the image from test set (see Figure 5.1). As there is always two bounding box predictions per cell, we used non-maximum suppression algorithm to avoid duplicate detections.

[4] mAP calculation - https://github.com/Cartucho/mAP

[5] OpenCV - https://opencv.org/

[6] Matplotlib - https://matplotlib.org/

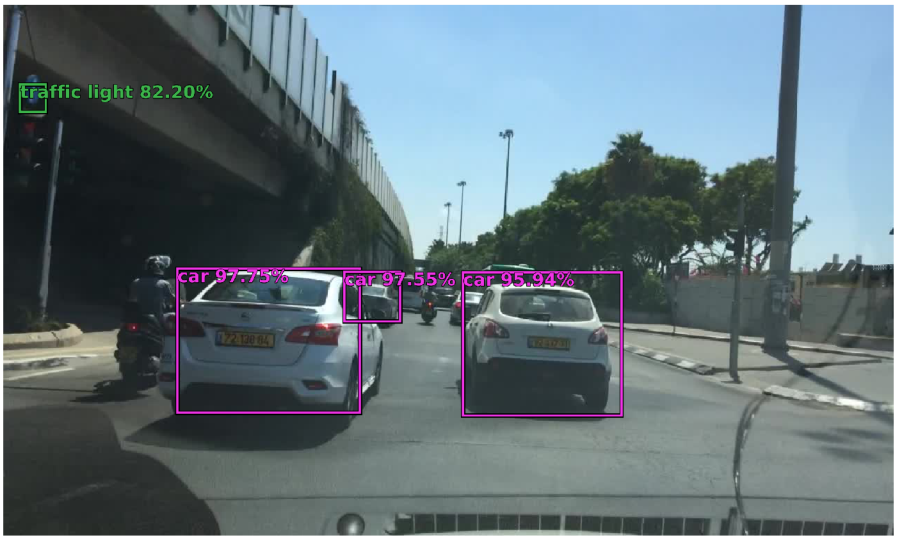


Figure 5.1: YOLOv1: Results after performing detection on the image from test set.

After the training, the validation was performed and model accuracy was computed using mAP metric. The network was able to learn to detect just 5 classes with accuracy comparable with the one reported in [41]. This is very likely caused by a high-class imbalance in training set and limitations of YOLOv1 to predict smaller objects and objects in flocks as it is mentioned in the section 3.3.1. The prediction time for one frame with resolution 1280 × 720 is 12.33s on the CPU Intel Core i5 7360U 2,3 GHz.

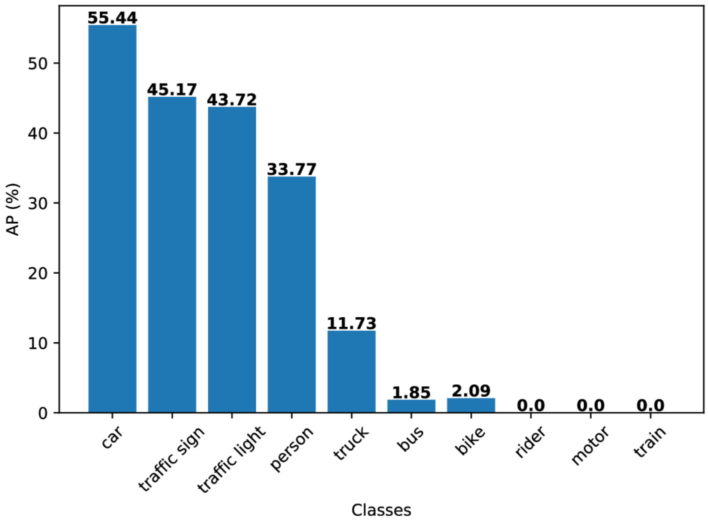


Figure 5.2: Results achieved using YOLOv1 network.

In the second experiment ResNet-50 architecture is used as a feature extractor. The 7 initial convolutional blocks were freezed. The eighth layer from ResNet was added on top with average pooling and YOLO layer composed from one linear layer and dropout layer. Model was trained for 20 epochs with batch size 64. Adam is used as an optimizer for YOLOv1 loss function with weight decay 0.0005. Whole training set is used for the training. Non-maximum suppresion algorithm is again used for removing the duplicate detections. After performing validation we can see that the results are just slightly worse than in the previous experiment. But thanks to pretrained feature extractor and optimization of fewer parameters the training process was more than 2× faster.

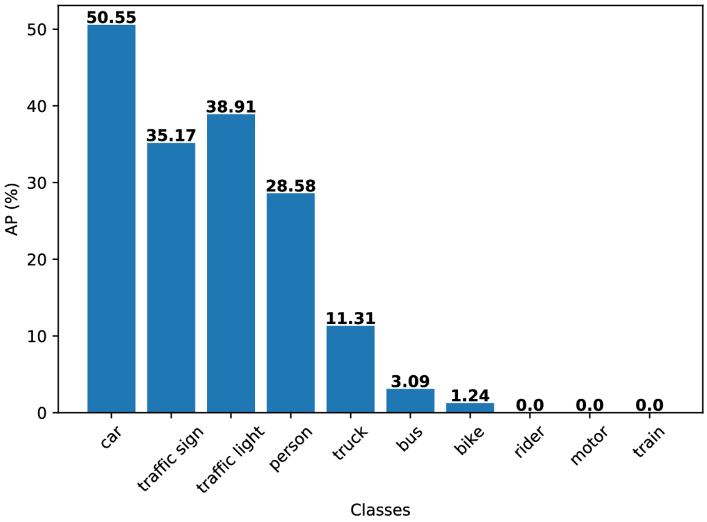


Figure 5.3: Results achieved with YOLOv1 network using ResNet-50 backbone.

5.2.2 Experiment using Tiny YOLOv3

In the third experiment a model based on Tiny-YOLOv3 architecture was trained using Darknet framework. Pre-trained feature extractor is used and the network was trained for 20000 iterations on the whole training dataset with batch size 64. SGD was used as an optimizer with momentum equals to 0.9 and 0.0005 weight decay. Validation was performed every 4 epochs on the validation dataset.

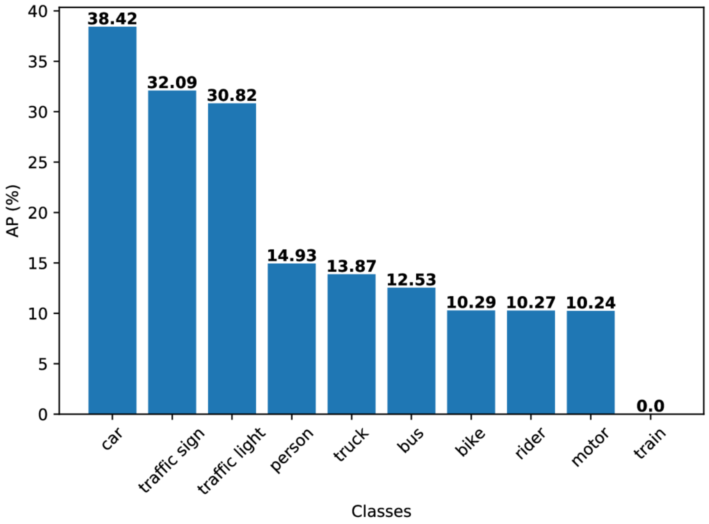


Figure 5.4: Results achieved with Tiny-YOLOv3 network.

This model was able to achieve comparable mAP with a much smaller size of the network and fewer parameters. This is likely because of improvements in YOLO method since version 1. Besides the good detection accuracy, Tiny-YOLOv3 performed also better in the speed of detection during the test time. Another thing to point out is that precision is more equally split among all classes unlike in the first experiment where the trained model is more imbalanced. The prediction time for one frame with resolution 1280 × 720 is 623.59ms on the CPU Intel Core i5 7360U 2,3 GHz.

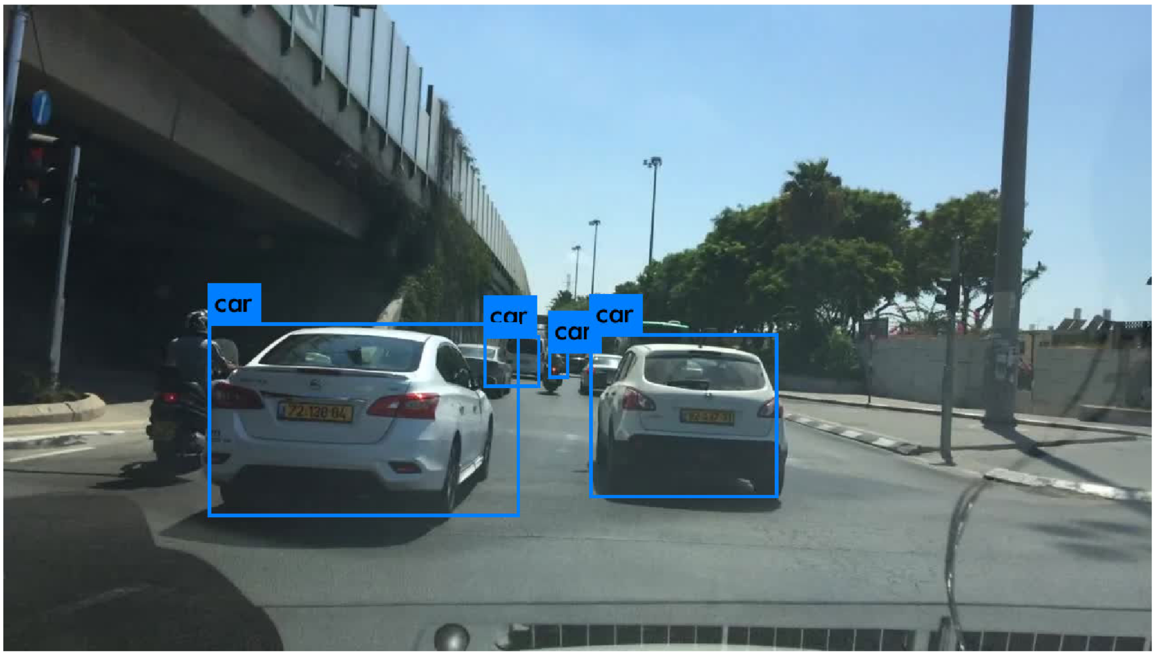


Figure 5.5: Tiny-YOLOv3: Results after performing detection on images from test set.

5.2.3 Experiment using YOLOv3

In the third experiment a model based on YOLOv3 architecture was trained using Darknet framework. Pre-trained feature extractor based on Darknet-53 architecture was used and the network was trained for 20000 iterations on whole training dataset with batch size 64. SGD was used as an optimizer with momentum equals to 0.9 and 0.0005 weight decay. Validation was performed every 4 epochs on the validation dataset.

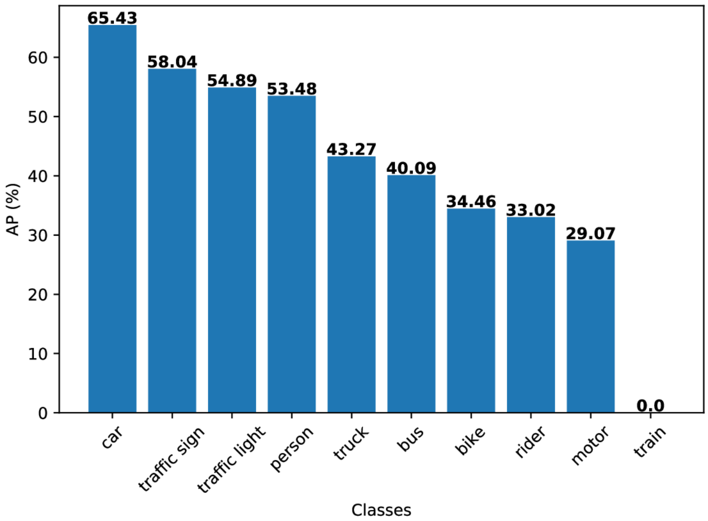


Figure 5.6: Results achieved with YOLOv3 network.

YOLOv3 with pre-trained weights from Darknet-53 achieved accuracy 41.18% mAP on the validation dataset which is more than two times better than two previous models. Since the mAP on the validation dataset was increasing just slightly in the last epochs, there is an assumption that the model could not learn more with these parameters of the training. The prediction time for one frame with resolution 1280 × 720 is 7.116s on the CPU Intel Core i5 7360U 2,3 GHz.

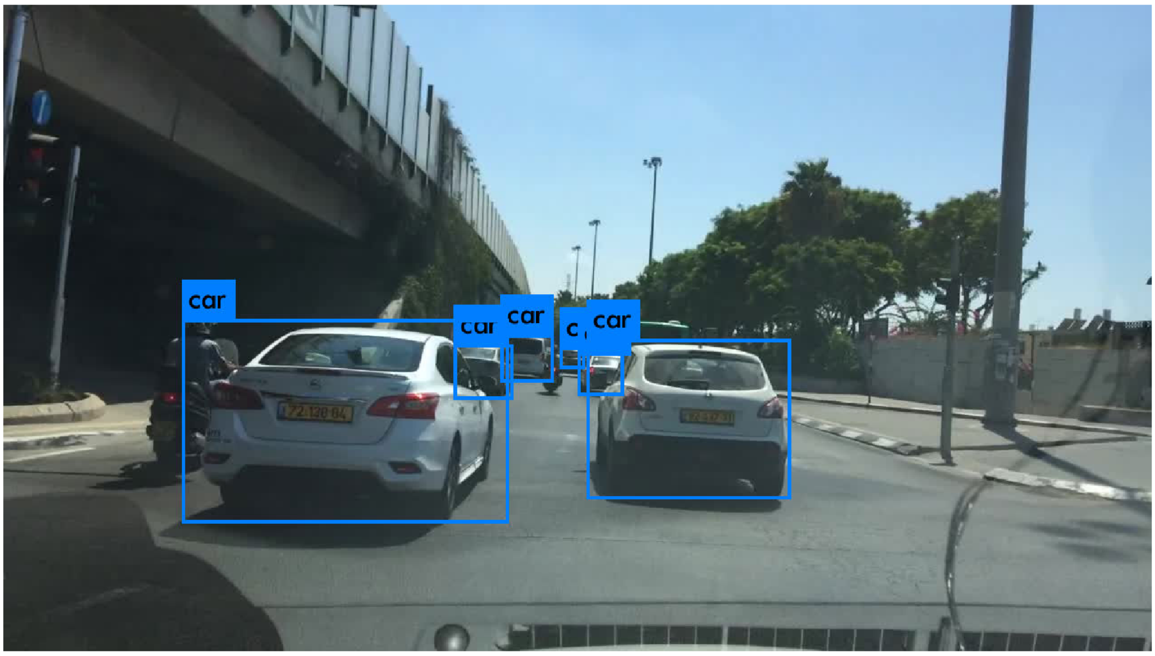


Figure 5.7: YOLOv3: Results after performing detection on images from test set.

5.2.4 Experiments summary

In the Table 5.1 is a summary of all experiments. There is mAP of top 5 classes that model learned to detect with best average precission and also overall mAP for all classes. There is a big difference between these two, mainly because of the class imbalance in the training set so models were able to learn to detect just few classes with a solid accuracy.

Model	Top-5 class mAP	overall mAP
YOLOv1	37,96 %	19.38 %
YOLOv1 ResNet-50	32,94 %	16.89 %
Tiny YOLOv3	26,03 %	17.35 %
YOLOv3	**55,02 %**	**41.18 %**

Table 5.1: Summary of the results aquired after performing testing on the validation dataset of size 10000 *images.*

Chapter 6

Conclusion

This thesis provides brief introduction into neural networks and specifically convolutional neural networks. Then it describes methods commonly used nowadays for the problem of object detection. The main goal of this thesis was to implement method for object detection of objects in the street. For this purpose a model based on YOLOv1 object detection method is created and trained on the Berkley DeepDrive dataset. The model was able to achieve good accuracy just on the few object classes which are most represented in the training dataset. Accuracy is also limited because of the limitations in the YOLOv1 design. To compare these results with other methods, two more models were trained. The first of them was based on Tiny-YOLOv3 architecture. This model achieved lowest accuracy from all of the tested models, but besides that it performs very well in speed of the detection. Detection of one image took around 600ms on the CPU. Last model which was trained is based on YOLOv3 architecture and achieved best accuracy from all of the tested models.

The most valuable thing which I have learned is how to implement object detection model based on a deep neural network from the ground up. This gave me better insight to how deep neural networks work, than using just existing solutions.

For the future advancement I suggest to expand the training dataset with images of classes that are under-represented in the current dataset. One of the possible improvements of the model can be more advanced data augmentation and speed optimization.

Bibliography

[1] Arnab, A.; Torr, P. H. S.: Pixelwise Instance Segmentation with a Dynamically Instantiated Network. *CoRR*. vol. abs/1704.02386. 2017. 1704.02386.
Retrieved from: http://arxiv.org/abs/1704.02386

[2] Dahal, P.: *Classification and Loss Evaluation - Softmax and Cross Entropy Loss*. [Online; visited 15.05.2019].
Retrieved from: https://deepnotes.io/softmax-crossentropy

[3] Deng, J.; Dong, W.; Socher, R.; et al.: Imagenet: A large-scale hierarchical image database. In *In CVPR*. 2009.

[4] Dumoulin, V.; Visin, F.: A guide to convolution arithmetic for deep learning. 2016. cite arxiv:1603.07285.
Retrieved from: http://arxiv.org/abs/1603.07285

[5] Felzenszwalb, P. F.; Girshick, R. B.; McAllester, D. A.; et al.: Object Detection with Discriminatively Trained Part-Based Models. *IEEE Trans. Pattern Anal. Mach. Intell.*. vol. 32, no. 9. 2010: pp. 1627–1645.
Retrieved from:
http://dblp.uni-trier.de/db/journals/pami/pami32.html#FelzenszwalbGMR10

[6] Fisher Yu, Y. C. F. L. M. L. V. M. T. D., Wenqi Xian: *Berkeley DeepDrive*. [Online; visited 15.05.2019].
Retrieved from: https://bdd-data.berkeley.edu/

[7] Forsyth, D. A.; Ponce, J.: *Computer Vision: A Modern Approach*. Prentice Hall. 2003.
Retrieved from: http://www.cs.berkeley.edu/~daf/book.html

[8] Girshick, R.: Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. Washington, DC, USA: IEEE Computer Society. 2015. ISBN 978-1-4673-8391-2. pp. 1440–1448. doi:10.1109/ICCV.2015.169.
Retrieved from: http://dx.doi.org/10.1109/ICCV.2015.169

[9] Girshick, R. B.; Donahue, J.; Darrell, T.; et al.: Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. vol. abs/1311.2524. 2013. 1311.2524.
Retrieved from: http://arxiv.org/abs/1311.2524

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

[11] Haykin, S.: *Neural Networks: A Comprehensive Foundation (3rd Edition)*. 01 2007. ISBN 0131471392.

[12] Haykin, S. S.: *Neural networks and learning machines*. Pearson Education. third edition. 2009.

[13] He, K.; Gkioxari, G.; Dollár, P.; et al.: Mask R-CNN. *CoRR*. vol. abs/1703.06870. 2017. 1703.06870.
Retrieved from: http://arxiv.org/abs/1703.06870

[14] He, K.; Zhang, X.; Ren, S.; et al.: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *CoRR*. vol. abs/1406.4729. 2014.
Retrieved from:
http://dblp.uni-trier.de/db/journals/corr/corr1406.html#HeZR014

[15] He, K.; Zhang, X.; Ren, S.; et al.: Deep Residual Learning for Image Recognition. *CoRR*. vol. abs/1512.03385. 2015. 1512.03385.
Retrieved from: http://arxiv.org/abs/1512.03385

[16] Hearst, M. A.: Support Vector Machines. *IEEE Intelligent Systems*. vol. 13, no. 4. July 1998: pp. 18–28. ISSN 1541-1672. doi:10.1109/5254.708428.
Retrieved from: http://dx.doi.org/10.1109/5254.708428

[17] Ioffe, S.; Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. ArXiv e-prints. 2015.

[18] Karpathy, A.: *CS231n Convolutional Neural Networks for Visual Recognition*. [Online; visited 15.05.2019].
Retrieved from: http://cs231n.github.io/convolutional-networks/

[19] Kiefer, J.; Wolfowitz, J.: Stochastic Estimation of the Maximum of a Regression Function. *Ann. Math. Statist.*. vol. 23, no. 3. 09 1952: pp. 462–466. doi:10.1214/aoms/1177729392.
Retrieved from: https://doi.org/10.1214/aoms/1177729392

[20] Kingma, D. P.; Ba, J.: Adam: A Method for Stochastic Optimization. *CoRR*. vol. abs/1412.6980. 2015.

[21] Kota Ando, M. I. T. A. M. M., Shinya Takamaeda-Yamazaki: A Multithreaded CGRA for Convolutional Neural Network Processing. *Circuits and Systems*. 2017.

[22] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 2012. pp. 1097–1105.

[23] LeCun, Y.; Bottou, L.; Bengio, Y.; et al.: Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, vol. 86. 1998. pp. 2278–2324.
Retrieved from:
http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665

[24] Liu, W.; Anguelov, D.; Erhan, D.; et al.: SSD: Single Shot MultiBox Detector. *CoRR*. vol. abs/1512.02325. 2015. 1512.02325.
Retrieved from: http://arxiv.org/abs/1512.02325

[25] Lloyd, S. P.: Least squares quantization in pcm. *IEEE Transactions on Information Theory*. vol. 28. 1982: pp. 129–137.

[26] Long, J.; Shelhamer, E.; Darrell, T.: Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015. ISSN 1063-6919. pp. 3431–3440. doi:10.1109/CVPR.2015.7298965.

[27] Maas, A. L.; Hannun, A. Y.; Ng, A. Y.: Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[28] Mcculloch, W.; Pitts, W.: A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*. vol. 5. 1943: pp. 127–147.

[29] Mehrotra, K.; Mohan, C.; Ranka Preface, S.: Elements of Artificial Neural Nets. 01 1997.

[30] Redmon, J.; Divvala, S. K.; Girshick, R. B.; et al.: You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. vol. abs/1506.02640. 2015. 1506.02640.
Retrieved from: http://arxiv.org/abs/1506.02640

[31] Redmon, J.; Farhadi, A.: YOLO9000: Better, Faster, Stronger. *CoRR*. vol. abs/1612.08242. 2016. 1612.08242.
Retrieved from: http://arxiv.org/abs/1612.08242

[32] Redmon, J.; Farhadi, A.: YOLOv3: An Incremental Improvement. *CoRR*. vol. abs/1804.02767. 2018. 1804.02767.
Retrieved from: http://arxiv.org/abs/1804.02767

[33] Ren, S.; He, K.; Girshick, R. B.; et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, edited by C. Cortes; N. D. Lawrence; D. D. Lee; M. Sugiyama; R. Garnett. 2015. pp. 91–99.
Retrieved from:
http://dblp.uni-trier.de/db/conf/nips/nips2015.html#RenHGS15

[34] Rumelhart, D. E.; Hinton, G. E.; Wilson, R. J.: Learning representations by back-propagating errors. *Nature*. vol. 323. 1986: pp. 533–536.

[35] Russell, S.; Norvig, P.: *Artificial Intelligence: A Modern Approach*. Series in Artificial Intelligence. Prentice Hall. third edition. 2010.
Retrieved from: http://aima.cs.berkeley.edu/

[36] Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; et al.: Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*. vol. 15, no. 1. 2014: pp. 1929–1958.

[37] Svarer, C.: Neural Networks for Signal Processing. 1995.

[38] Szegedy, C.; Liu, W.; Jia, Y.; et al.: Going Deeper with Convolutions. *CoRR*. vol. abs/1409.4842. 2014. 1409.4842.
Retrieved from: http://arxiv.org/abs/1409.4842

[39] Szegedy, C.; Toshev, A.; Erhan, D.: Deep Neural Networks for Object Detection. 2013.
Retrieved from: http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf

[40] Uijlings, J. R. R.; van de Sande, K. E. A.; Gevers, T.; et al.: Selective Search for Object Recognition. *International Journal of Computer Vision*. vol. 104, no. 2. 2013: pp. 154–171.
Retrieved from:
http://dblp.uni-trier.de/db/journals/ijcv/ijcv104.html#UijlingsSGS13

[41] Yu, F.; Xian, W.; Chen, Y.; et al.: BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *CoRR*. vol. abs/1805.04687. 2018. 1805.04687.
Retrieved from: http://arxiv.org/abs/1805.04687

Appendix A

The contents of the attached storage media

- **doc** - Latex source files and thesis pdf.
- **models** - Trained neural networks.
- **data** - Manual on how to obtain the dataset.
- **src** - All source files used for this thesis.
- **poster.pdf** - Poster from the assignment.
- **README.md** - Setup manual.