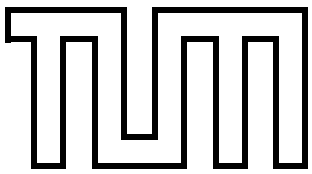


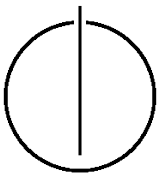
FAKULTÄT FÜR INFORMATIK

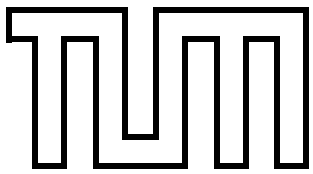
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Vehicle position estimation on surveillance dynamic vision Sensor

Armin Baur





FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Vehicle position estimation on surveillance dynamic vision Sensor

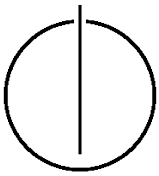
Bestimmung der Fahrzeugposition mit einem dynamischen Vision Sensor

Author: Armin Baur

Supervisor: Prof. Dr.-Ing. habil. Alois Christian Knoll

Advisor: Simon Gökstorp and Darjan Salaj

Date: August 15, 2021



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, August 15, 2021 Armin Baur

Acknowledgments

I consider the last seven months as an extremely insightful time on both an academic and a personal level. Also, I want to express my honest gratitude for my advisor Darjan Salaj and Simon Gökstrop, whose help and attention to detail were crucial for the quality of this thesis. Also, my thanks go to Patrick Ruoff and Julius Imbery for their help and work, which I could build on. Last but not least, I want to thank Ani for proofreading and the Global Communication Center for their advice on my academic writing style.

Abstract

Environmental knowledge of an autonomous vehicle is limited by its physical range of sensors and algorithmic performance, as well as by occlusions that degrade its understanding of the current traffic situation. Not only does it pose a significant safety hazard and limit driving speeds, it can also lead to awkward maneuvering. Smart infrastructure solutions can help to solve these problems. A smart infrastructure will fill the void between vehicle perceptions and extend its range in the shape of a digital model of the present traffic situation (the digital twin), by offering more accurate information on its environment.

This paper describes how a new synthetic dataset was created using Carla Simulator. This is based on the Unreal Engine and had to be modified to make this possible. This artificial dataset has made it possible to train an existing neural network called YOLOv5. This shows that the completeness as well as the functionality of the dataset is given. The dataset covers many possible applications.

The important features for further work are the annotation of the vehicles and the existence of a dynamic vision sensor event stream. The YOLOv5 network was modified to use the DVS event stream as input to detect vehicles and determine their position. Subsequently, the working network was modified and tested several times. Successfully, various activation functions were replaced with the leaky-integrate-and-fire neuron. Finally, the network was optimized in terms of runtime and memory consumption by means of pruning and quantization.

Zusammenfassung

Das Umgebungswissen eines autonomen Fahrzeugs ist durch die Reichweite seiner Sensoren und die Leistung seiner Algorithmen sowie durch Verdeckungen begrenzt, die sein Verständnis der aktuellen Verkehrssituation beeinträchtigen. Dies stellt nicht nur ein erhebliches Sicherheitsrisiko dar und begrenzt die Fahrgeschwindigkeit, sondern kann auch zu ungünstigen Manövern führen. Intelligente Infrastrukturlösungen können dazu beitragen, diese Probleme zu lösen. Eine intelligente Infrastruktur füllt die Lücke zwischen den Wahrnehmungen des Fahrzeugs und erweitert seine Reichweite in Form eines digitalen Modells der aktuellen Verkehrssituation (des digitalen Zwillings), indem sie genauere Informationen über seine Umgebung bietet.

In dieser Arbeit wird beschrieben, wie ein neues synthetisches Datensatz mithilfe von Carla Simulator erstellt wurde. Dieser baut auf der Unreal Engine auf und musste um dies zu ermöglichen modifiziert werden. Dieser künstliche Datensatz hat es ermöglicht ein bereits existierendes neuronales Netz namens YOLOv5 zu trainieren. Dies zeige, dass die Vollständigkeit sowie die Funktionalität des Datensatzes gegeben ist.

Der Datensatz deckt viele Anwendungsmöglichkeiten ab. Die für die fortsetzende Arbeit wichtigen Eingenschaften sind die Annotierung der Fahrzeuge sowie das Existieren von einem Dynamic Vision Sensor event stream. Das YOLOv5 Netzwerk wurde so verändert, dass es den DVS event stream als Eingabe verwenden kann um Fahrzeuge zu erkennen und ihre Position zu bestimmen. Daraufhin wurde das funktionierende Netzwerk mehrfach modifiziert und getestet. Erfolgreich wurden verschiedene Aktivierungsfunktionen mit dem Leaky-integrate-and-fire Neuron ersetzt. Auch wurde das Netzwerk zum Schluss mittels pruning und quantizierung in Richtung Laufzeit und Speicherverbrauch optimiert.

Contents

1 Introduction **2**
- 1.1 Problem . . . 2
- 1.2 Motivation . . . 4
- 1.3 Objectives . . . 4
- 1.4 Outline . . . 5

2 Background **6**
- 2.1 Dynamic Vision Sensor (DVS) . . . 6
- 2.2 Neural Networks . . . 9
- 2.3 Convolutional Neural Networks . . . 11
  - 2.3.1 Convolutional Layer . . . 12
  - 2.3.2 Pooling Layer . . . 13
  - 2.3.3 Fully-Connected Layer . . . 14
- 2.4 Spiking Neural Networks . . . 15
- 2.5 Model Architecture You Only Look Once (YOLO) . . . 17
  - 2.5.1 YOLOv5 Architecture . . . 19
  - 2.5.2 Input . . . 21
  - 2.5.3 Backbone: Focus Layer . . . 21
  - 2.5.4 Backbone: BottleneckCSP Layer . . . 22
  - 2.5.5 Backbone: SPP Layer . . . 23
  - 2.5.6 Neck . . . 24
  - 2.5.7 Activation functions . . . 26
- 2.6 Pruning . . . 26
  - 2.6.1 Structured vs. Unstructured . . . 27
  - 2.6.2 Parameter Scoring . . . 28
  - 2.6.3 Pruning Approaches . . . 29
- 2.7 Quantization . . . 29

3 Related Work **31**
- 3.1 Synthetic Data for Deep Learning . . . 31
- 3.2 Event-based Moving Object Detection and Tracking . . . 33

3.3 DavisDrivingDataset 2020 34
3.4 Ground Moving Vehicle Detection and Movement Tracking Based on the Neuromorphic Vision Sensor 36

4 Dataset **40**
4.1 CARLA Simulator 42
4.2 CARLA DVS Camera Simulation 43
4.3 Data generation 45
4.3.1 Open source contribution 46
4.3.2 Datageneration Details 46
4.3.3 Dataset 47

5 Methods **50**
5.1 YOLO 50
5.2 Pruning 50
5.3 Quantization methods 51
5.4 Spiking CNN Activations 51

6 Results **52**
6.1 Vehicle position estimation using Spiking Neural Networks . . 52
6.2 Improving the network generalization with pruning 53
6.3 Improving the network efficiency by quantization 55
6.4 Speedups 56

7 Conclusion **57**

A Sourcecode **58**
A.1 Convolution 58
A.2 Spiking Convolution 59

AI Artificial Intelligence

DVS Dynamic Vision Sensor

FCNN Fully Connected Neural Network

JSON JavaScript Object Notation

YOLO You Only Look Once

DAVIS Dynamic and Active-pixel Vision Sensor

SLAM Simultaneous Localization and Mapping

NN Neural Network

CNN Convolutional Neural Network

SNN Spiking Neural Network

AP average precision

MLP multi-layer perceptrons

ReLU Rectified Linear Unit

SCDEKF strong tracking center differential external Kalman filter

LIF leakyintegrate and fire

FoV field of view

IIS Intelligent infrastructure systems

Chapter 1

Introduction

This chapter introduces the topics of vehicle detection using a dynamic vision sensor. The research is motivated by the problem that Vehicle detection with frame based cameras can be improved by using a DVS. And therefor saving resources. The research proposes a new type of vehicle detection based on DVS and YOLO.

1.1 Problem

The available sensor ranges and object detection capabilities restrict an autonomous vehicle's and the ensuing scene's ambient perception. Even in the vehicle's immediate proximity, the existence of occlusions results in inadequate information about the vehicle's surroundings. The ensuing uncertainties endanger not just the autonomous car, but also other road users. It must lower its driving speed in order to drive safely, which slows down traffic. As a result of the car reacting spontaneously to unanticipated circumstances, driving comfort suffers. The Intelligent infrastructure systems (IIS) can help to alleviate these issues by giving extra information about each road user and the overall traffic condition to autonomous cars as well as conventional vehicles and drivers. For instance, an IIS may monitor and recognize road users from various higher-level views, therefore increasing its coverage beyond that of a single vehicle. When a car receives this new information, it can better comprehend its surroundings and plan safer and more convenient driving moves. Furthermore, an IIS with the aforementioned characteristics provides a range of services that aid in decision-making. However, developing such a system involves a number of problems, such as selecting the right hardware and sensors, as well as optimizing their deployment and utilization in a complicated software stack. The system's perception must be

dependable and strong under a variety of weather, lighting, and traffic circumstances. A combination of multimodal sensors, redundant road coverage with overlapping field of view (FoV), precise calibration and strong detection and data fusion algorithms is necessary to achieve this dependability. This architecture is already the result of our real-world experience with IIS Providentia shown in Figure 1.1.

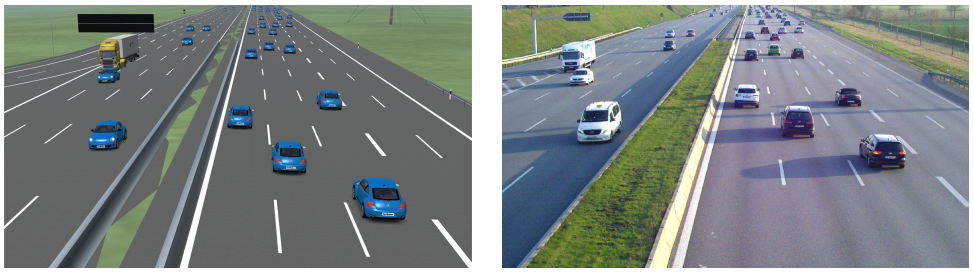


Figure 1.1: On the left side the Digital Twin is depicted. On the right side the original Image. [KSGK19]

A smart infrastructure will fill the void between vehicle perceptions and extend its range in the shape of a digital model of the present traffic situation (the digital twin), by offering more accurate information on its environment, shown in Figure 1.2.

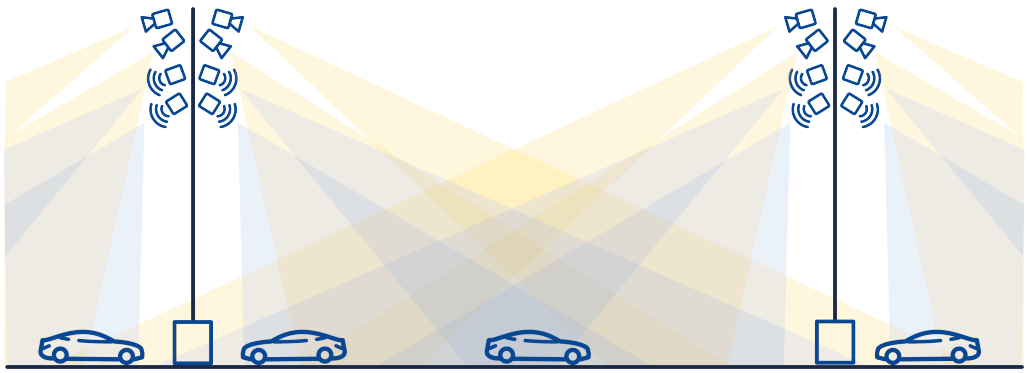


Figure 1.2: A Sample Construction of the Hardware needed to build this IIS [KSGK19]

It is already developed with a pipeline including object detection with a neural network called tiny YOLOv3 working on a radar system and RGB images. To predict the exact 6DOF Location of the Vehicles on a motorway and is published by [KSGK19]. This vehicle detection should be build for another scenario with more traffic participants in the City.

1.2 Motivation

The central motivation driving this thesis is the opportunity to enhance the Providentia Project. This is made possible to the largest part due to inventions in an event camera.

Since event cameras have great potential for robotics and computer vision in difficult scenarios for conventional cameras, such as low latency, high speed and high dynamic range. However, novel methods are needed to process the unconventional output of these sensors to unlock their potential. This Thesis aims to provide insight into the field of event-based image processing for vehicle detection. The focus is on the possibility of event cameras providing enough data to make accurate statements. This would result in many advantages. Apart from the obvious ones like less data flow and cheaper computation costs. There is also the safety aspect of the higher dynamic range, which also allows backlit shots without overexposure. Also, the lower power consumption is important to mention in this day and age. Not only the sensor but the complete pipeline needs less resources. Another advantage the sensor offers in our scenario is the very low motion blur it brings with it. So cars can be detected very well at night compared to normal cameras.

1.3 Objectives

In this work we want to clarify whether it is possible to detect and track vehicles with the help of the DVS event stream. Since the event data is very similar to the principle of the leakyintegrate and fire (LIF) neuron, the integration is also an intention to be tested. Thus, system design develops an application domain model in the one in the previous chapter, that maps the application domain models to the goals specified in this chapter. Since there is no dataset that represents our scenario, we have to create one here. This one will suffice as a synthetic one for our experiment. The question that arises here is whether the network is faster or requires less computing power compared to a similar RGB network.

1.4 Outline

This thesis covers seven chapters. This introductory chapter gives an overview of the thesis' topics, existing problems, possible solutions, the research objectives and this outline. Chapter 2 provides more information about the fundamental concepts used in this thesis. It contains sections about event cameras, neural networks, machine learning, and machine earning interpretability. As well as a detailed view of the most important part for this thesis, YOLOv5. Related work is discussed in Chapter 3. There, a variety of related studies are discussed and a gap in the literature is elaborated, which should be filled by this thesis. Chapter 4 follows the requirements elicitation and analysis templates by [BD09]. It defines the requirements of the LATEST system and introduces a correct, complete, consistent, and verifiable representation of the Data generation pipeline. Build upon an already existing System which is going to be improved for our thesis. The changes of the Neural Network Design of YOLOv5 is detailed in Chapter 5, which follows the system design document template by [BD09]. There, the design goals of the envisioned system are worked out and LATEST is decomposed into smaller subsystems. Chapter 5 covers detailed explanations of all optimization steps done to the Neural Network. This thesis' approach was evaluated in different steps and is described in Chapter 6. The sections included describe the pruning quantization and spiking objectives and the setups and results of its different phases. Additionally, the findings and limitations are discussed. Chapter 7 closes this thesis by providing a summary of the work performed as part of this thesis and gives suggestions for possible future work.

Chapter 2

Background

This chapter covers important background knowledge that is built upon in the rest of this thesis.

Dynamic Vision Sensor (DVS) also known as Dynamic and Active-pixel Vision Sensor (DAVIS) or event camera introduces this new type of camera in Section 2.1.

Section 2.2 covers the fundamental concept of Neural Network (NN). As well as Section 2.3 about Convolutional Neural Network (CNN) will make use of NN for image classification.

In the Section 2.4 Spiking Neural Network (SNN) the activation layer from usual NN will be replaced.

And afterwards used to rebuild of YOLO in Section 2.5 to be able to use it with SNN on the DVS data.

The generalization and regularization will be improved through 2 already known techniques called pruning in Section 2.6 and quantization in Section 2.7. They also offer interference speedup and a smaller network size.

2.1 Dynamic Vision Sensor (DVS)

The novel image sensors known as event cameras, which have the potential to alleviate most of the problems caused by the design principles of conventional cameras as mentioned in Section 1.1 when combined with new event-driven computer vision algorithms introduced in this thesis.

A DVS is a sensor that works radically differently from a conventional camera. By reporting asynchronous intensity changes over time or space rather than synchronous full image frames, they generate low bit-rate, information rich data streams that are free of the redundancy of video while also providing additional benefits such as high dynamic range, high temporal

resolution, and low latency. They are the result of bio-inspired silicon retina research in neuromorphic engineering, with the goal of replicating some superior qualities of biological vision.

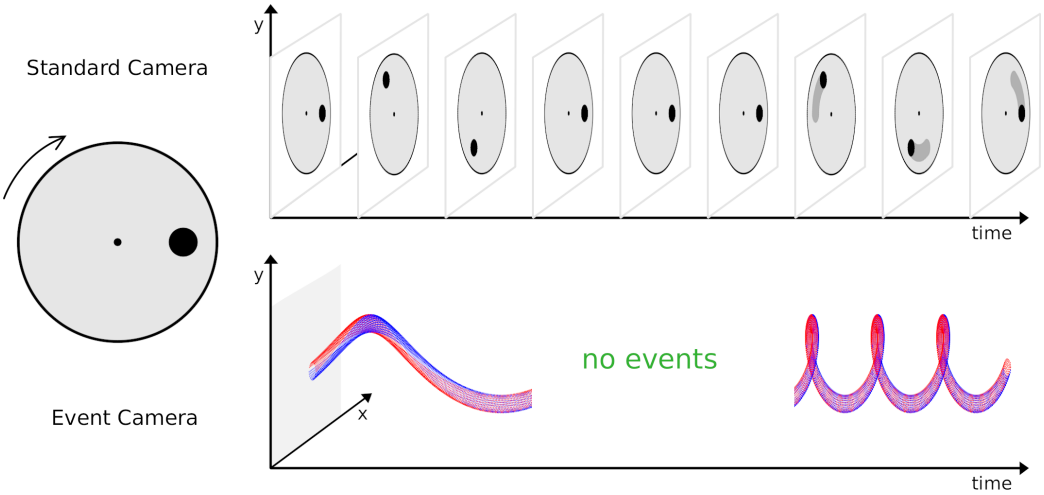


Figure 2.1: Event camera vs standard camera: Unlike the upper graph, which shows a sequence of video frames from a standard camera, the lower graph shows a stream of events from an event camera, which has no redundant data output (only informative pixels or no events at all), no motion blur, and a high dynamic range. The red and blue dots indicate positive and bad events, respectively, and this picture was inspired by the famous animation of [Kim17]

It is inspiring to look at the distinctions between event cameras and conventional cameras, as represented in Figure 2.1, in order to comprehend how event cameras work and comprehend how they could be advantageous for real-time computer vision applications.

Standard cameras record scenes at predetermined time intervals (global or rolling shutter) and output a series of image frames. As shown in Figure 2.1, if a stationary conventional camera stares at the spinning disc with the black dot on the left, we acquire a series of snapshots as indicated in the top spatial-temporal graph on the right. If the events are simply visualized by taking all in a certain, $t_\delta = 30ms$ an example image is shown in Figure 2.2 on the left side. The graph depicts some main properties of standard video frames: there are blind time intervals between frames, the sensor continues to send redundant data even when the disc is stationary (no new information produced), and we suffer from motion blur if the disc spins too fast (illustrated by the gray tails along the trajectory of the block dot).

In contrast to ordinary cameras, event cameras produce a stream of

asynchronous events (also known as spikes), each with a pixel location, polarity, and microsecond-precise date, indicating when individual pixels record log intensity changes of a pre-set threshold magnitude. Positive and negative changes, on the other hand, result in positive and negative events. By encoding only image changes, the bandwidth required to transmit, process, and store a stream of events is much lower than that required for standard video, removing the redundancy in continuously repeated image values; however, this stream should in theory contain all the information of standard video, at least up to scale, and without the usual frame-rate and dynamic range bounds. For example, observing the same spinning disc with a fixed event camera yields the stream of events depicted in the lower spatial-temporal graph on the right side of Figure 2.1 — red and blue dots represent positive and negative events, respectively. This graph also depicts some fundamental aspects of the event stream, including the near constant reaction to very rapid motion and how the output data-rate depends on scene motion, albeit in practice it is nearly always far lower than that of regular video. These qualities have the potential to overcome the limitations of traditional imaging sensors in real-world computer vision applications, such as low frame rate, high latency, low dynamic range, and high power consumption.

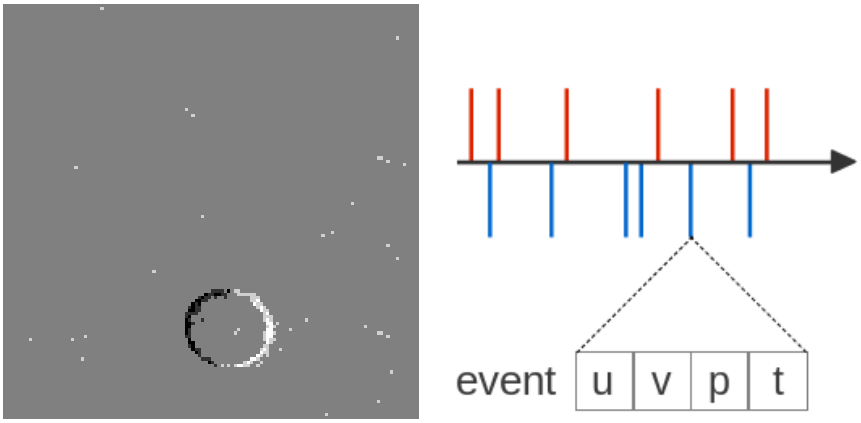


Figure 2.2: On the left side is an image-like visualization of accumulated events within a time interval, where white and black pixels represent positive and negative events, respectively. right is a series of events depicted as upward and downward spikes for positive and negative events. Each event is a tuple (hu, v, p, t) where u and v are the event's pixel coordinates, p is the event's polarity, and t is the event's timestamp in microseconds; [Kim17]

In summary, the event camera has the following advantages over a usual frame based one:

- more dynamic range 140db vs 60db
- no motion blur
- smaller data stream (sparse matrix)
- high temporal resolution (in the order of microseconds)
- less energy consumption

Each event is encoded as described in 2.2

2.2 Neural Networks

A typical NN is made up of many simple, connected processors called neurons, each of which generates a sequence of real-valued activations. The first one exists out of perceptrons. The perceptron is a simplified artificial neural network first introduced by [Ros58] shown in Figure 2.3

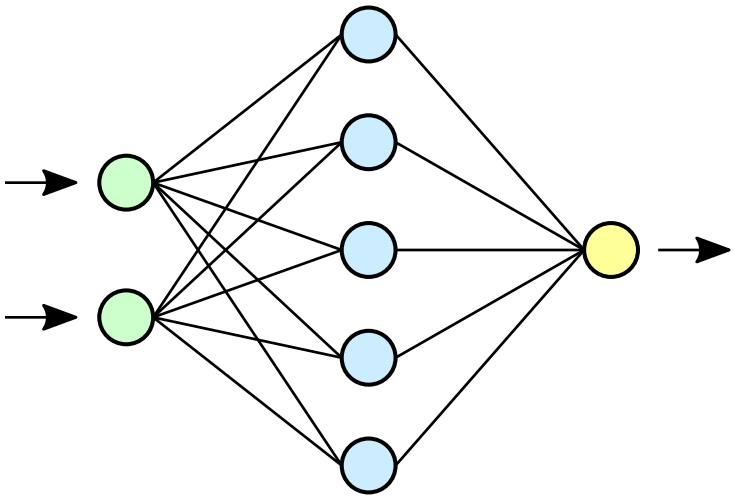


Figure 2.3: Simplified representation of an artificial neural network

In its basic version (simple perceptron), it consists of a single artificial neuron with adjustable weights and a threshold. Today, this term is used to refer to various combinations of the original model, distinguishing between single-layer and multi-layer perceptrons (MLP). Perceptron networks convert an input vector into an output vector and thus represent a simple associative memory.

Some neurons may have an impact on their surroundings by activating actions. Finding weights that cause the NN to exhibit desirable behavior,

such as driving a car or classify images, is the goal of learning or credit assignment. Depending on the problem and how the neurons are coupled, such behavior may necessitate extended causal chains of computational stages as shown in 2.3 and 2.5, with each stage transforming (sometimes in a non-linear way) the aggregate activation. As [Sch14] says, Deep Learning is concerned with appropriately attributing credit to the network's weights and biases across many such phases. The cell body of a single neuron is depicted in Figure 2.4.

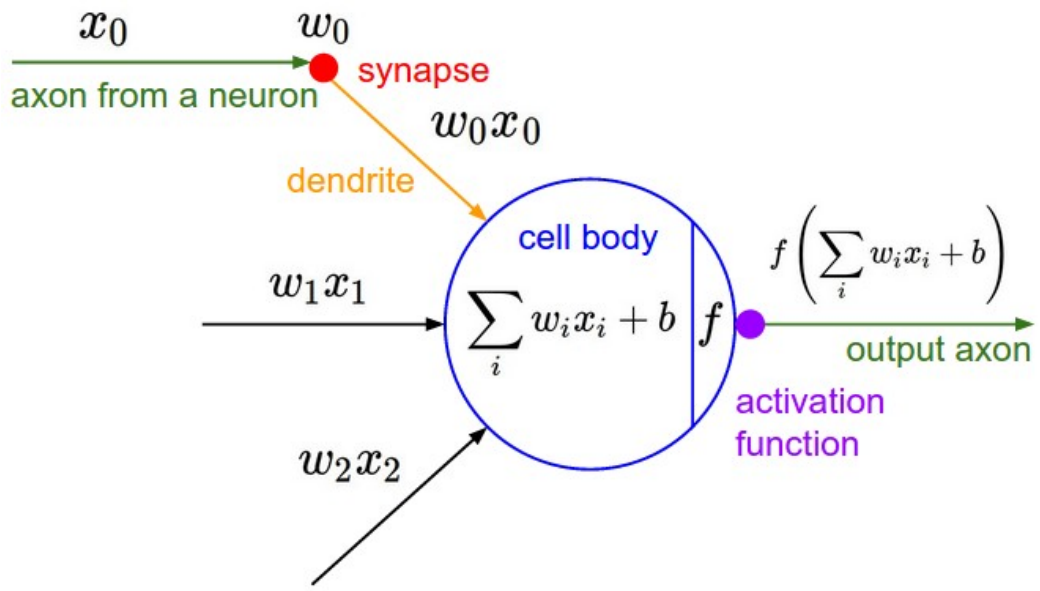


Figure 2.4: Visualization of a single neuron.

In this example, each input is multiplied by its associated weight in 2.1 - 2.3.

$$y_0 = x_0 * w_0 \tag{2.1}$$
$$y_1 = x_1 * w_1 \tag{2.2}$$
$$y_2 = x_2 * w_2 \tag{2.3}$$

Next, all the weighted inputs are added together with a bias b:

$$y_0 + y_1 + y_2 + b \tag{2.4}$$

Finally, the sum is passed into an activation function:

$$y = f(y_0 + y_1 + y_2 + b) \tag{2.5}$$

The activation function is important to reduce the image to a specific range. And to be able to easily back propagate the network. An easy example for

an activation function is the sigmoid shown in Figure 2.5. Which projects the input between $[0, 1]$.

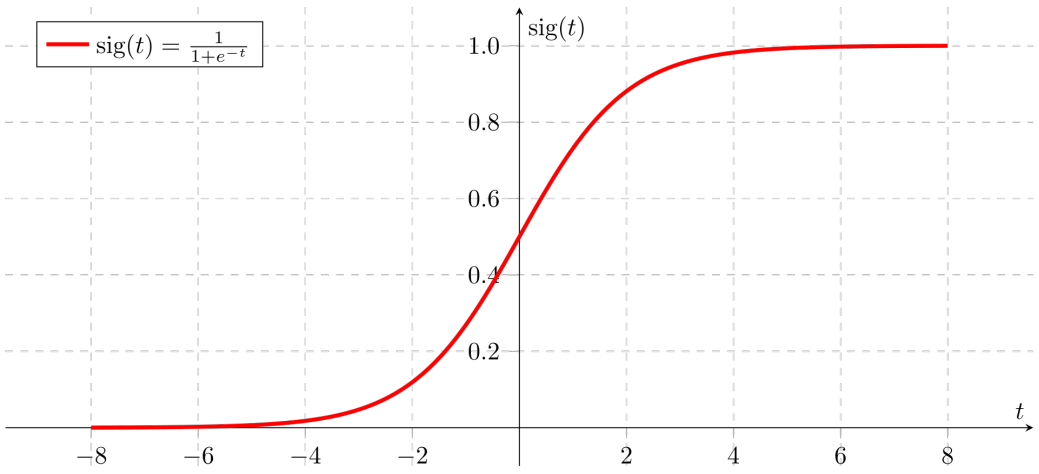


Figure 2.5: Visualization of the Sigmoid activation function.

2.3 Convolutional Neural Networks

A CNN is an artificial neural network. It is a concept inspired by biological processes in the field of machine learning [MMMK03]. Convolutional Neural Networks find application in numerous Artificial Intelligence (AI) technologies, primarily in machine processing of image or audio data. Basically, the structure of a classical CNN consists of one or more Convolutional Layers, followed by a Pooling Layer. In principle, this unit can repeat itself as often as desired; if there are enough repetitions, we then speak of Deep Convolutional Neural Networks, which fall into the area of Deep Learning shown in Figure 2.6.

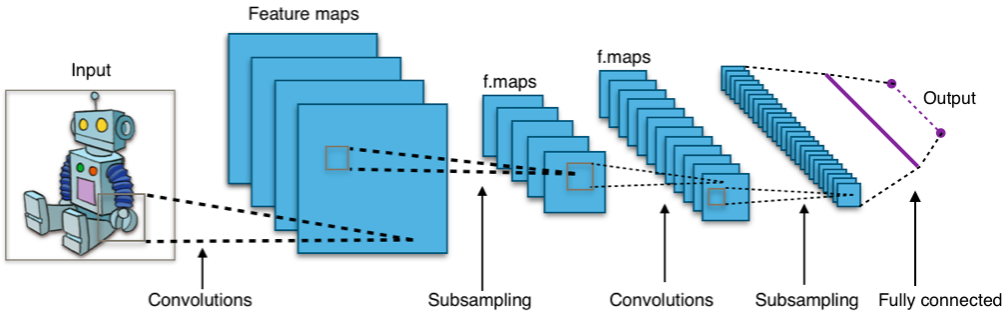


Figure 2.6: Structure of a typical CNN for image classification.

Architecturally, three major differences can be noted in comparison to the multilayer perceptron (for details, see section 2.3.1 Convolutional Layer):

- 2D or 3D arrangement of neurons.
- Split weights
- Local connectivity

2.3.1 Convolutional Layer

Usually, the input is a two- or three-dimensional matrix (e.g. the pixels of a grayscale or color image). Accordingly, the neurons are arranged in the convolutional layer.

The activity of each neuron is calculated via a discrete convolution (hence the addition convolutional). In this process, a comparatively small convolution matrix (filter kernel) is moved stepwise over the input. The input of a neuron in the convolutional layer is calculated as the inner product of the filter kernel with the currently underlying image section. Accordingly, neighboring neurons in the Convolutional Layer respond to overlapping areas (similar frequencies in audio signals or local environments in images) [LBBH98].

It should be emphasized that a neuron in this layer only responds to stimuli in a local environment of the previous layer. This follows the biological model of the receptive field. In addition, the weights for all neurons of a convolutional layer are identical (shared weights). This means that, for example, each neuron in the first convolutional layer encodes to which intensity an edge is present in a certain local area of the input. Edge detection as the first step of image recognition has high biological plausibility [HW68].It follows directly from shared weights that translation invariance is an inherent property of CNNs.

The input of each neuron, determined by discrete convolution, is now transformed by an activation function, usually Rectified Linear Unit (ReLU) shown in 2.6 [Aga18]

$$f(x) = max(0, x) \tag{2.6}$$

for CNNs, into the output, which is supposed to model the relative firing frequency of a real neuron. Since backpropagation requires the computation of gradients, a differentiable approximation of ReLU is used in practice 2.7:

$$f(x) = ln(1 + e^x) \tag{2.7}$$

Analogous to the visual cortex, in deeper convolutional layers both the size of the receptive fields (see Pooling Layer section 2.3.2) and the complexity of the recognized features increase.

2.3.2 Pooling Layer

For example, for object recognition in images, the exact position of an edge in the image is of negligible interest - the approximate localization of a feature is sufficient. There are different types of pooling. By far the most common is max-pooling shown in Figure 2.7, where from each 2×2 square of neurons in the convolutional layer, only the activity of the most active (hence "max") neuron is retained for further computational steps; the activity of the remaining neurons is discarded. Despite the data reduction (75% in the example), pooling usually does not reduce the performance of the network. On the contrary, it offers some significant advantages:

- Reduced space requirements and increased computation speed.
- Resulting possibility to create deeper networks that can solve more complex tasks
- Automatic growth of the size of the receptive fields in deeper convolutional layers (without the need to explicitly increase the size of the convolutional matrices)
- Preventive measure against overfitting

Alternatives such as mean pooling have been shown to be less efficient in practice [SMB10]. The biological counterpart to pooling is lateral inhibition in visual cortex.

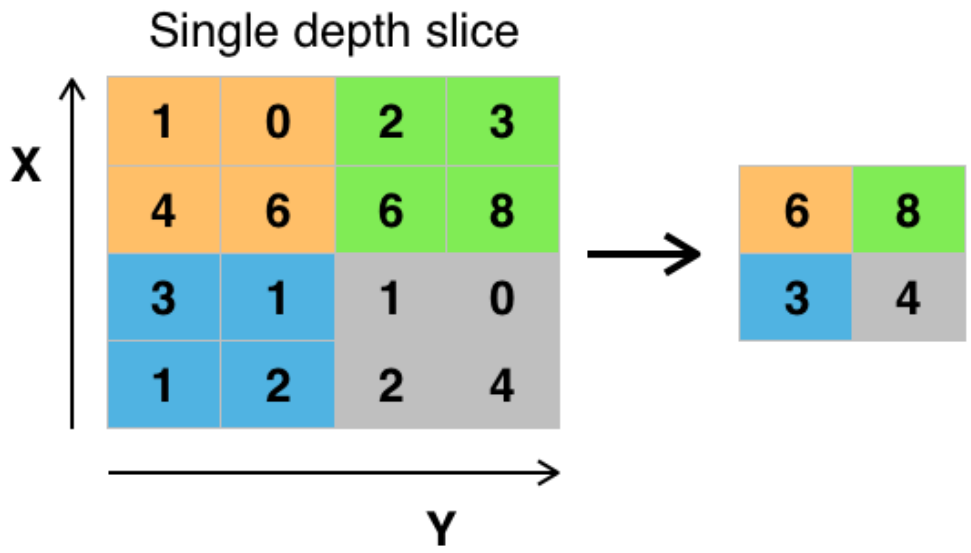


Figure 2.7: Max pooling with a 2×2 filter and step size = 2. The step size indicates how many pixels the filter moves per operation.

2.3.3 Fully-Connected Layer

After some repetitive units consisting of convolutional and pooling layers, the network can terminate with one (or more) fully-connected layers according to the architecture of the multilayer perceptron. This is mainly applied in classification and call Fully Connected Neural Network (FCNN). The number of neurons in the last layer then usually corresponds to the number of (object) classes that the network should distinguish. This, very redundant, so-called one-hot-encoding has the advantage that no implicit assumptions about similarities of classes are made. The output of the last layer of the CNN is usually transformed into a probability distribution by a softmax function, a translation- but not scale-invariant normalization over all neurons in the last layer.

2.4 Spiking Neural Networks

McCulloch and Pitts [MP43] made the first attempt to characterize the underlying activity of neurons in 1943. This first model included binary activation, excitatory and inhibitory inputs, unit weights, and a fixed threshold. This meant that the network's weights and thresholds had to be defined analytically and could not be learned. Despite these constraints, a network of McCulloch-Pitts neurons can mimic any logic function.

The perceptron model was developed in 1958 as an expansion of the McCulloch-Pitts model [Ros58]. This methodology enabled parameter learning and the use of effective minimization methods to tackle linear separation issues. The perceptron reduces the complicated dynamics of synapses to a weight factor. The weighted sum of the activation of input neurons is the definition of the neuron's membrane potential:

$$u = \sum_{i=0}^{N} w_i x_i + w_0 \tag{2.8}$$

where w_i is the synaptic weight to input neuron i, w_0 is the weight bias that determines the resting potential, and N is the number of input neurons. The output is defined as a membrane potential threshold function:

$$z = \begin{cases} 1 \text{ if } u \geq \theta \\ 0 \text{ if } u \leq \theta \end{cases} \tag{2.9}$$

Where z is the value of the threshold parameter.

The next generation of artificial neuron models replaced the threshold function at the output with nonlinear activation functions, typically sigmoid ortangens hyperbolicus [Sal18]. Networks of neurons having non-linear activations, as opposed to perceptrons, may differentiate data that is not linearly separable [Bis06].

The LIF model of spiking neurons is discussed futher below. Because of its physiologically realistic communication, the LIF neuron belongs to the third generation of neuron modes and is one of the most common formal spiking neuron models used in research. The information transferred through the networks of LIF neurons is encoded in the spike trains produced by the neurons.

The cell membrane is represented as an electrical circuit consisting of parallel resistor R Rand capacitor C parts driven by current I. The current I is viewed as dendritic incoming current. The current component discharged

across the resistor R is interpreted as ion leakage through open channels in cells, whilst the second current component charges the capacitor C. The capacitor voltage is defined by

$$\tau_m \frac{\delta u}{\delta t} = -u(t) + RI(t) \tag{2.10}$$

and is interpreted as membrane potential u. Where $\tau_m = RC$ is the neuron's membrane time constant [5].

An action potential (spike) is created when the membrane potential exceeds the threshold parameter. Spikes are formal events in this paradigm, defined by their firing time $t^{(f)}$, which is specified by the threshold criterion.

$$t^{(f)} : u\left(t^{(f)}\right) = \theta \tag{2.11}$$

The membrane potential is set to the resting potential immediately after the spike fires $u_r < \theta$ [5]. The LIF neuron, in general, can implement an absolute refractory time, which we do in our simulations. When the membrane potential $u = u_r$ approaches the threshold θ at $t = t^{(f)}$, we stop the dynamics 2.10 for the period of absolute refractory time Δ^{abs} and resume the dynamics at $t^{(n)} + \Delta^{\text{abs}}$ with the membrane potential $u = u_r$ [5].

Because the LIF model does not explicitly describe the shape of a spike, the driving current I of coupled LIF neurons is unknown. The driving current in a biological neural network is determined by the activation of presynaptic neurons. At the synapse, spikes generate current pulses that are proportional to the excitatory post synaptic potential in the receiving dendrite. The input current in this configuration is the weighted sum of received current pulses, where weights denote synaptic efficacies:

$$I_i I^{(t)} = \sum_i w_i \sum_f \alpha_i \left(t - t_{i,f}\right) \tag{2.12}$$

where kernel $\alpha_i(t)$ is the current pulse shape created by synapse i, t_{ij} is a list of input spike times arriving at synapse i, and w_i is the synapse's weight or efficiency. The kernel α is often a Dirac δ -pule, but it can alternatively be a double exponential kernel with rise and fall times τ_{r} and τ_x:

$$\alpha_i(t) = \frac{q}{\tau_s - \tau_r} \left(exp\left(-\frac{t - \Delta_i}{\tau_s}\right) - exp\left(-\frac{t - \Delta_i}{\tau_r}\right) \right) \Theta\left(t - \Delta_i\right) \tag{2.13}$$

where q is the total charge of a synapse, Δ_i is the latency of a synapse, and $\Theta(t)$is the Heaviside step function [GK02].

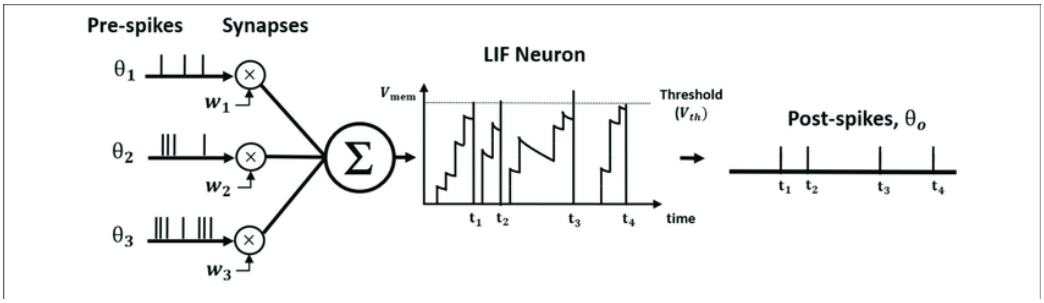


Figure 2.8: The illustration of Leaky Integrate and Fire (LIF) neuron dynamics. [LSP+20]

The kinetics of Leaky Integrate and Fire (LIF) neurons are depicted in 2.8. The synaptic weight modulates the pre-spikes, which are incorporated as the current influx in the membrane potential, which decays exponentially. The post-neuron fires a post-spike and resets the membrane potential whenever the membrane potential reaches the firing threshold.

2.5 Model Architecture You Only Look Once (YOLO)

The majority of CNN-based object detectors were largely applicable only for recommendation systems because of inaccuracy and speed. With the aim to overcome this and create a real-time object detection CNN that operates on a conventional GPUs, [RDGF16] introduced the YOLO architecture. Real-time object detector operation on conventional GPUs allows their mass usage at an affordable price. YOLO was also designed to be easily trained and used in production systems. A detector is composed of two parts, a backbone and a head. The backbone is pre-trained on ImageNet, and the head is used to predict classes and bounding boxes of objects. For those detectors running on CPU platform, the backbone could also be implemented as SqueezeNet or MobileNet. The head implementation is usually categorized as one-stage object detector or two-stage detector. Some detectors insert layers between backbone and head to collect feature maps from different stages [BWL20]. Figure 2.9 shows the architecture of such object detector consisting of several components.

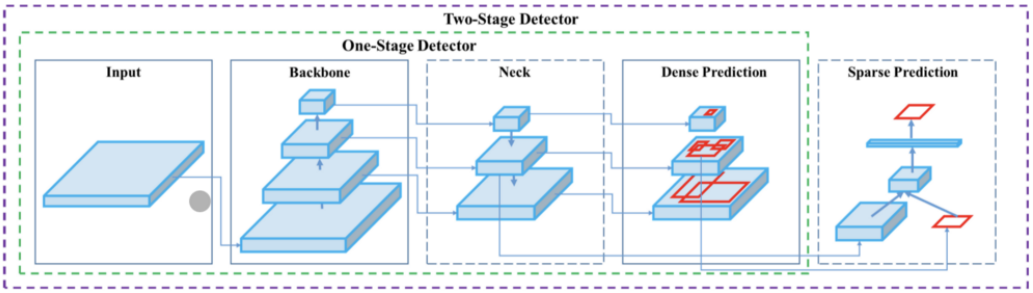


Figure 2.9: Object detection architecture from [BWL20]

- **Input**: Image, Patches, Image Pyramid
- The **Backbone** is primarily used to extract key features from an input image. Common backbones are: VGG16 [SZ15], ResNet-50 [HZRS15], SpineNet [DLJ+20], EfficientNet-B0/B7 [TL20], CSPResNeXt50 [WLY+19], CSPDarknet53 [WLY+19]
- The **Neck** is primarily used to build feature pyramids. Feature pyramids aid models in generalizing well when it comes to object scaling. It aids in the identification of the same object in various sizes and scales. Feature pyramids are extremely useful in assisting models to perform well on previously unseen results. Other versions, such as
  - Additional blocks: SPP [HZRS14], ASPP [CPK+17], RFB [LHW18], SAM [WPLK18]
  - Path-aggregation blocks: FPN [LDG+17], PAN [LQQ+18], NAS-FPN [GLPL19], Fully-connected FPN, BiFPN [TPL20], ASFF [LHW19], SFAM [ZSW+19]

  employ various forms of feature pyramid techniques.
- The **Head** is primarily responsible for the final detection phase. It uses anchor boxes to produce final output vectors with class probabilities, objectness ratings, and bounding boxes.
  Available heads are:
  - **Dense Prediction** (one-stage):
    - RPN [RHGS15], SSD [LAE+16], YOLO [RDGF16], RetinaNet [LGG+18] (anchor based)
    - CornerNet [LD19], CenterNet [DBX+19], MatrixNet [RKP19], FCOS [TSCH19] (anchor free)

- **Sparse Prediction** (two-stage):
  - Faster R-CNN [RHGS15], R-FCN [DLHS16], Mask RCNN [HGDG18] (anchor based)
  - RepPoints [YLH+19] (anchor free)

2.5.1 YOLOv5 Architecture

YOLOv5 is a single-stage object detector [JSB+21] which consists of three equal parts: the backbone, the neck, as well as the head. The only setting that is adjustable is the depth and width of the model. When V5l is executed, the following default parameters are used:

- A depth multiplier controls the value of the amount of detail in the model. To illustrate, the depth of v5s is 0.33, and the number of Bottlenecks is equal to the depth of v5l.
- The parameter, width multiple, controls the number of convolution kernels. The number of convolution kernels for V5s is set to 0.5, which implies that the width of V5s is 0.5, too. Of course, one can also set it to one and a quarter. For example, the first layer of the backbone in the YOLO V5 yaml is [[-1, 1, Focus, [64, 3]], and the width of V5s is 0.5, so this layer is actually [[-1, 1, Focus, [32, 3]].
- The column parameter is used to specify the input. -1 represents the input obtained from the upper layer, -2 represents the input obtained from the upper two layers (the same is true for head).
- Number column parameters: 1 means there is only one, 3 means there are three identical modules.

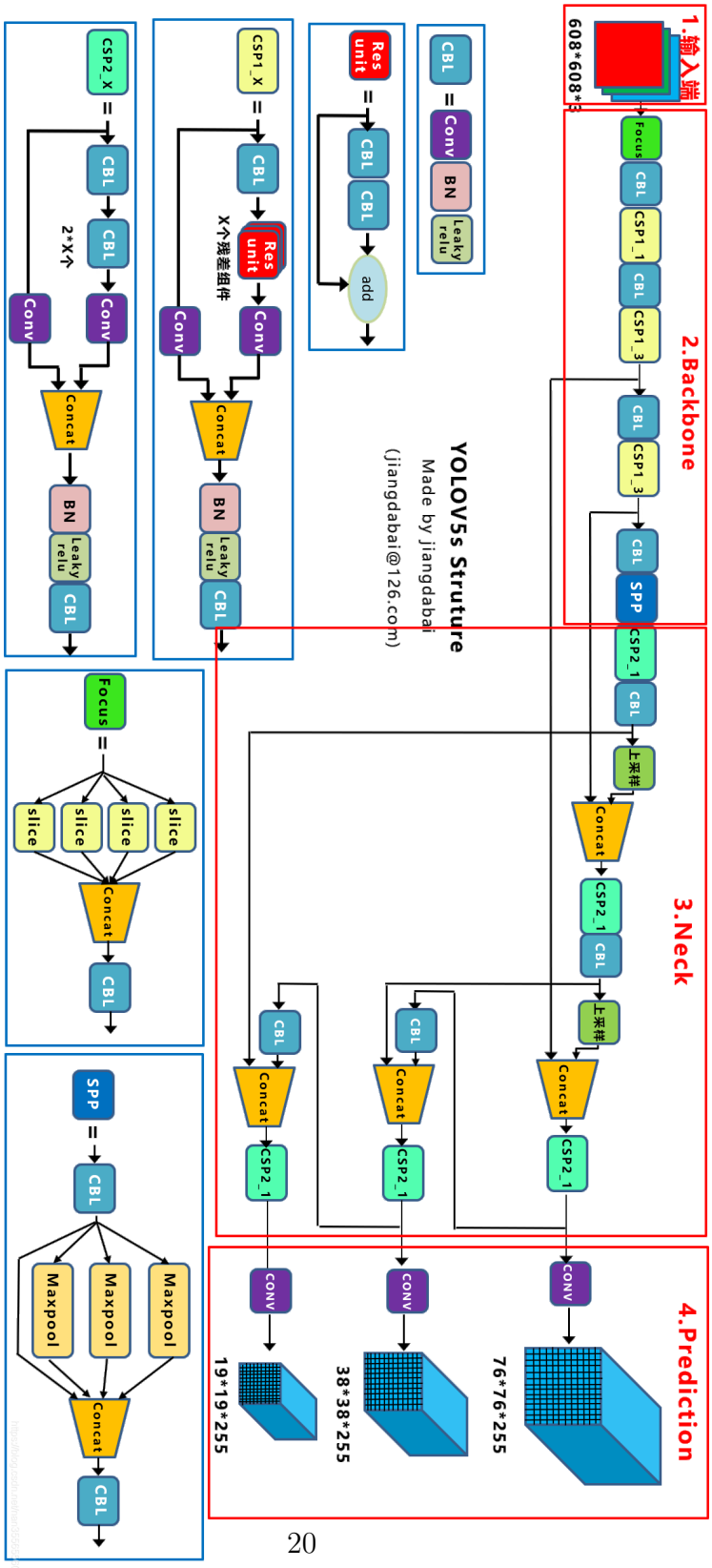


Figure 2.10: Yolov5 structure diagram. It shows all high level architectures as well as the 4 parts [Dab20]

The architecture can be divided into multiple higher levels shown in Figure 2.10 and explained further on.

1. Input: The innovational methods used are Mosaic data enhancement, adaptive anchor frame calculation, adaptive picture scaling.

2. Backbone: Uses a Focus structure at the entry and mainly consists out of CSP structures

3. Neck: The network for target detection inserts some layers between the backbone and the output layer, including the SPP module and the framework for FPN+PAN.

4. Prediction: The output layer's anchor framework mechanism is the same as that of Yolov4 and the key change is the GIOU Loss loss feature, replacing DIOU-CIOU Loss.

2.5.2 Input

The input differs from training to inference in the manner of data augmentation and enhancement. During training, the average precision (AP) of small targets is generally much lower than of medium or large targets [KWM+19], and thus the data gets modified by a mechanism calls Mosaic data enhancement during training. Mosaic data enhancement puts multiple images into one and by random scaling, random cropping, and random arrangement and is published by [ZCG+19]. But because we are using synthetic data this input feature is disabled.

2.5.3 Backbone: Focus Layer

The Focus layer is the first layer in yolov5, and it's based on the SpaceToDepth stem published in TResNet [RLN+20] as a faster and more efficient way to stem the input. In figures 2.11 and 2.13 the functionality is shown for Yolov5s. Using the Yolov5s structure as an example, the original $608 \times 608 \times 3$ image is input into the Focus structure, and the slicing operation is used to first become a $304 \times 304 \times 12$ feature map, and then after a convolution operation of 32 convolution kernels, the final change is a $304 \times 304 \times 32$ feature map. It should be noted that Yolov5s' Focus structure now employs 32 convolution kernels, whereas the number of the other three structures has increased. Pay careful attention first, and then the variations between the four systems will be clarified.

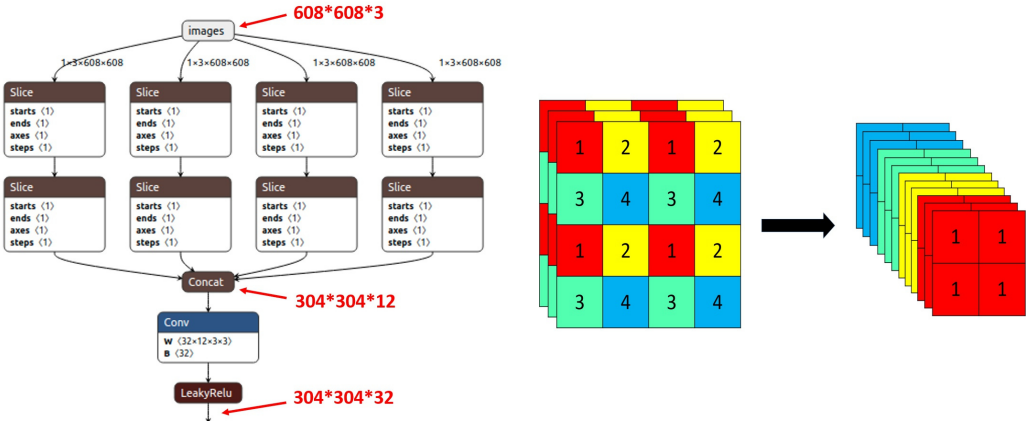


Figure 2.11: On the left side is the netron Visualisation of the Focus layer; And on the right side the simplified Illustrated one
[Dab20]

2.5.4 Backbone: BottleneckCSP Layer

BottleneckCSP is divided into two parts, Bottleneck and CSP.

Bottleneck is a classic residual structure composed of a 1x1 convolutional layer (conv+batch norm+leaky relu), a 3×3 convolutional layer, and finally a residual structure to add to the initial input. It should be noted that YOLO V5 controls the model's depth via depth multiple. For example, the depth of V5s is 0.33 and the depth of V5l is 1, implying that the number of Bottleneck in V5x's Bottleneck CSP is three times that of V5s, which was the model's first Bottleneck CSP. There are two CSP structures designed in Yolov5. Taking the Yolov5s network as an example, the CSP1_X structure is used in the Backbone network, and the other CSP2_X structure is used in Neck shown in figure 2.12.

The figure 2.12 shows the CSP structure of YOLO V5s. The original input is divided into two branches, both performing the convolution operation to reduce the number of channels by half, and then branch one performs the Bottleneck x N operation. At the end both branches will be concatenated. That the input and output of BottleneckCSP are the same size, and the purpose is to allow the model to learn more features.

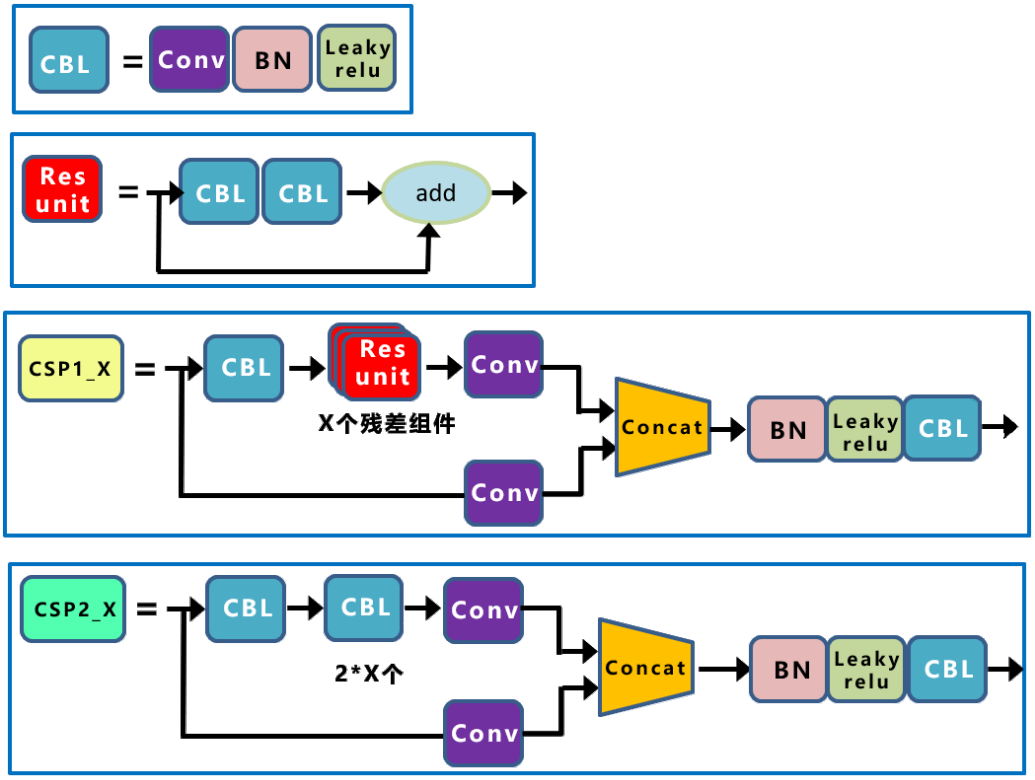


Figure 2.12: Detailed view of the BottleneckCSP [Dab20]

2.5.5 Backbone: SPP Layer

The figure 2.13 shows the SPP structure of YOLO V5s. The input of SPP is $512 \times 20 \times 20$. After a 1x1 convolutional layer, it outputs $256 \times 20 \times 20$. Then it is down-sampled through three parallel Maxpools. The result is added to its initial features to output $1024 \times 20 \times 20$. Finally, a 512 convolution kernel is used to restore it to $512 \times 20 \times 20$.

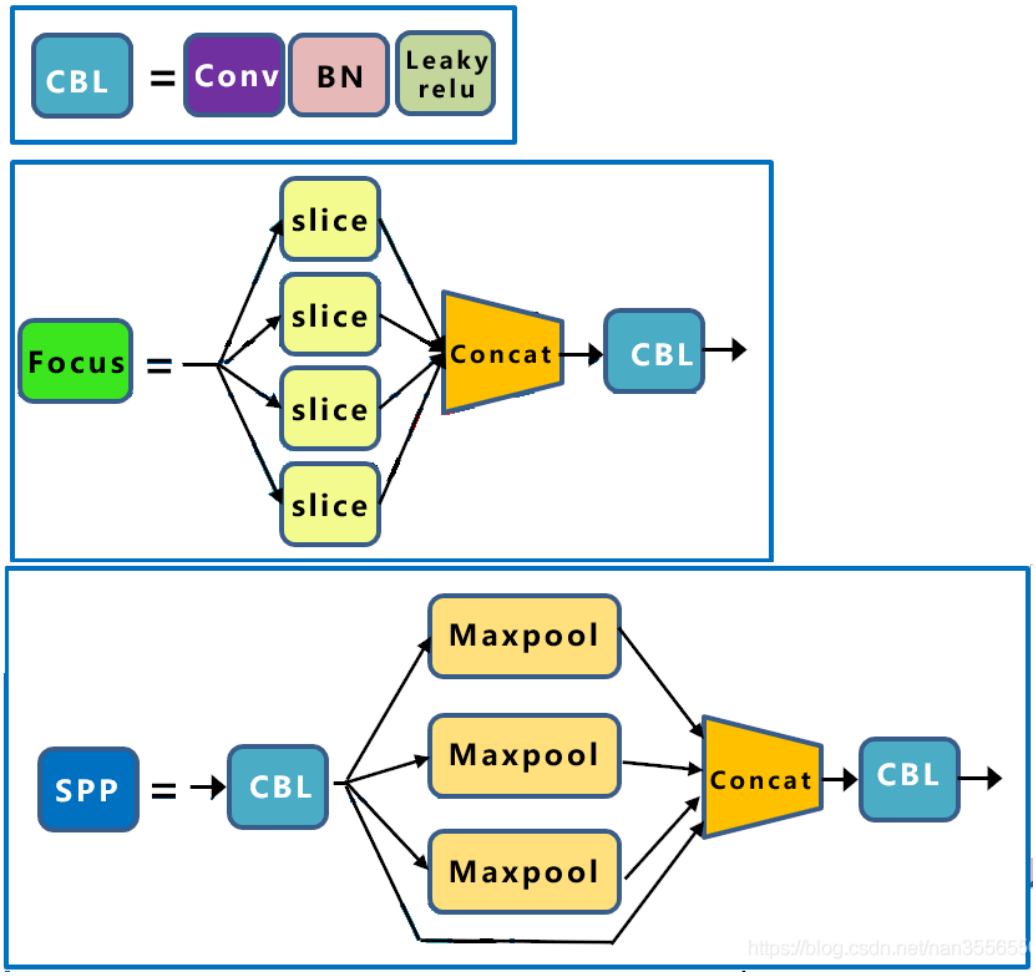


Figure 2.13: Detailed view of the SPP and Focus layers. [Dab20]

2.5.6 Neck

Yolov5's current Neck uses FPN+PAN structure. To detect objects, we can use a pyramid of the same picture at various scales shown in Figure 2.14 on the left. However, processing multiple scale images takes time, and the memory demand is too high to train end-to-end at the same time. As a result, we can only use it in inference to increase accuracy as much as possible. Alternatively, we can create a feature pyramid and use it to detect objects shown in Figure 2.14 on the right. Closer to the image layer, however, feature maps are composed of low-level structures that are ineffective for accurate object detection. The Feature Pyramid Network (FPN) is a feature extractor built with accuracy and speed in mind for such pyramid concepts. It replaces

detectors' feature extractors, such as Faster R-CNN, and generates multiple feature map layers (multi-scale feature maps) with higher quality information than the standard feature pyramid for object detection.

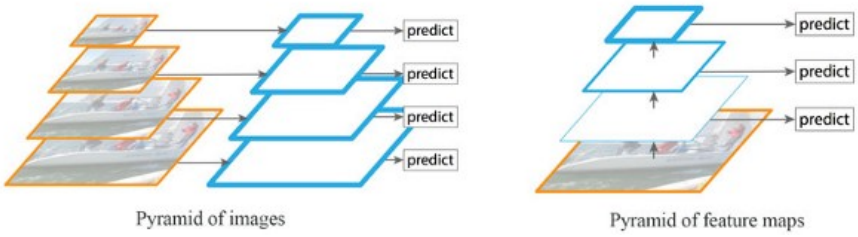


Figure 2.14: On the left a pyramid of images to detect different scaled objects. On the right a feature pyramid to do the same.
[LDG+17]

Furthermore, PAN is build on top of FPN to improve performance shown in Figure 2.15. Path augmentation and aggregation are used to boost efficiency. To render low-layer knowledge easier to spread, a bottom-up direction is supplemented. We devise adaptive feature pooling to allow each proposal to predict using information from all levels. The mask-prediction branch now has a complementary direction. This new structure results in acceptable efficiency. The upgrade, like FPN, is independent of the CNN system.

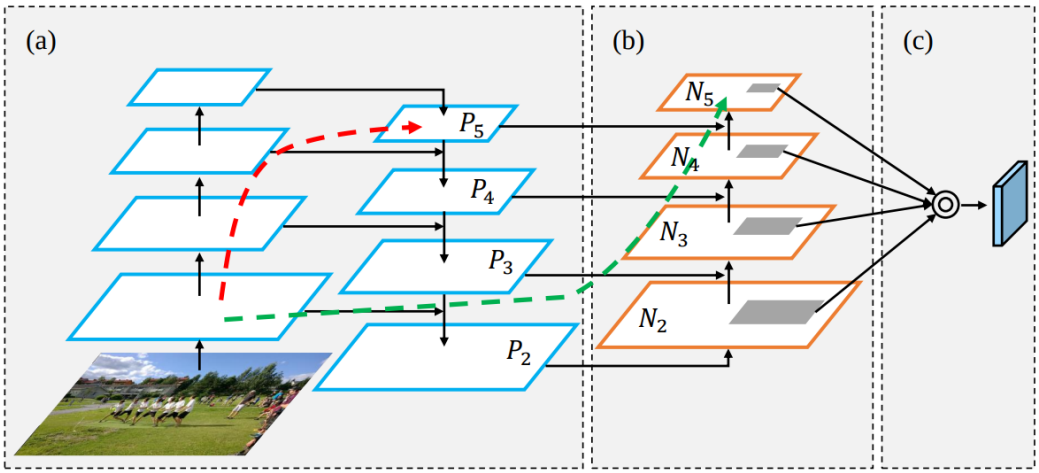


Figure 2.15: Illustration of PAN. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling.
[LQQ+18]

2.5.7 Activation functions

In any deep neural network, the selection of activation functions is important. Many new activation features, such as Leaky ReLU, mish [Mis20], swish [RZL17], and so on, have recently been published. In YOLO v5 the Leaky ReLU and Sigmoid activation functions are used. The Leaky ReLU activation function is used in the middle/hidden layers of YOLO v5, while the sigmoid activation function is used in the final detection layer. The SGD optimization feature is used by default but could be changed to ADAM easily. A compound loss is measured in the YOLO family based on objectness score, class likelihood score, and bounding box regression score. For the loss calculation of class likelihood and object score, Ultralytics used PyTorch's Binary Cross-Entropy with Logits Loss function. We also have the option of measuring the loss using the Focal Loss function [LGG+18]. The main difference in size and precision control of YOLOv4 and YOLOv5 is that YOLOv5 is reducing or extending each layer by a crop factor and YOLOv4 is reducing its overall layers. Therefor there will be less connections between the different stages.

2.6 Pruning

Deep learning methods at the cutting edge depend on over-parametrized models that are difficult to deploy. On the opposite, it is well known that biological neural networks make optimal use of sparse connectivity. It is critical to identify optimal techniques for compressing models by reducing their parameter count in order to minimize memory, battery, and hardware usage without compromising accuracy, deploy lightweight models on the computer, and ensure privacy through private on-device computation [Pag20].

The neural network (NN) Pruning is the process of shrinking a Neural Network by eliminating some of its parameters/weights. Pruning is often used to reduce the memory, computational, and energy bandwidths required for training and deploying NN models, which are infamous for their large model size, computational cost, and energy consumption. Pruning, and model compression in general, is desirable and often the only plausible way to deploy NN models on mobiles or edge devices, where memory, resources, and computational bandwidths are very small. But, one would wonder, why not use the cloud's potentially limitless virtual memory and computing power? While several NN models are currently running on the cloud, latency is not low enough for mobile/edge devices, limiting utility and necessitating data transfer to the cloud, which poses some privacy concerns [CR19].

A standard pruning method's workflow is as follows: A number of parameters of a trained model are zeroed out based on a pre-defined score. The network that remains is trained again (retraining). To approximate the accuracy of the original model with the sparse, pruned model, which now requires significantly lower memory, computational, and energy bandwidths than the original model due to the fewer parameters. To comprehend the developments in Pruning science, it is helpful to categorize new techniques as different forms across different dimensions (methods of classification) and their distinct strengths and weaknesses. The dimensions mentioned below are some of the most common.

2.6.1 Structured vs. Unstructured

Unstructured Pruning occurs when NN weights are pruned individually during the pruning process. Randomly zeroing out the parameters improves memory efficiency (models are stored in sparse matrices), but it does not always improve computational performance because we end up performing the same number of matrix multiplications as before. Since the matrix dimensions have not changed, they are simply sparse. While we could gain computational advantages by replacing dense matrix multiplications with sparse matrix multiplications, accelerating sparse operations on conventional GPUs/TPUs is not trivial. Structured Pruning, on the other hand, removes parameters in a structured manner to reduce the total computation required. For example, in a feedforward layer, some of the CNN channels or neurons are removed, resulting in a direct reduction in computation. Structured Pruning (pruning neurons) and Unstructured Pruning (pruning connections) are examples of pruning2.16.

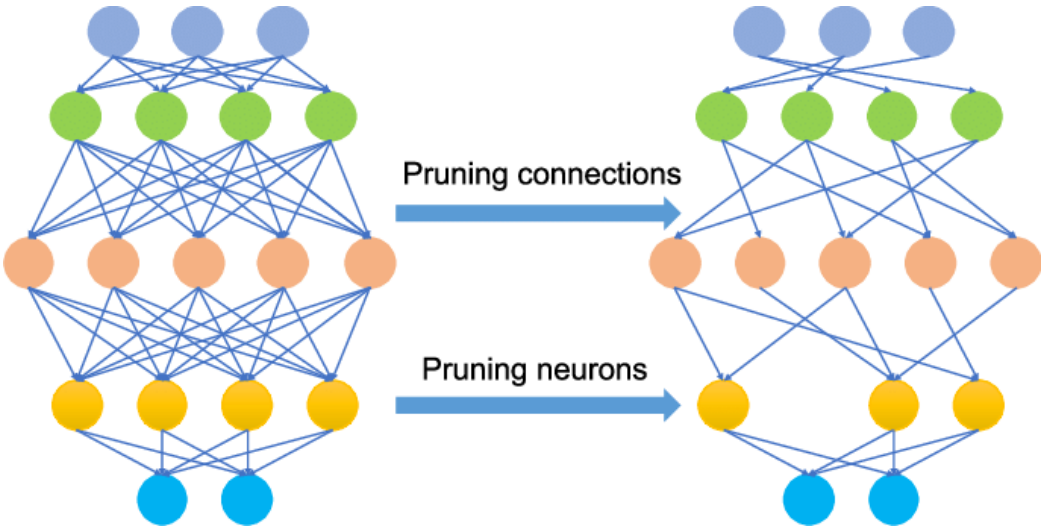


Figure 2.16: Neural Network Pruning differences between structured and unstructured.
[CR19]

2.6.2 Parameter Scoring

The method used to score each parameter, which is used to choose one parameter over another, can vary between pruning methods. The absolute magnitude scoring method is the industry norm, but a new scoring method may be developed to improve the efficiency of pruning. For example a mathematical norm could be used. A norm is applied to a vector. The norm of a vector maps vector values to values in $[0, \infty)$. Going a bit further, we define $\|x\|_p$ as a "p-norm". Given x, a vector with i components, a p-norm is defined as:

$$\|x\|_p = \left(\sum_i |x_i|^p \right)^{1/p} \tag{2.14}$$

The simplest norm conceptually is Euclidean distance. This is what we typically think of as distance between two points in space:

$$\|x\|_2 = \sqrt{\left(\sum_i x_i^2 \right)} = \sqrt{x_1^2 + x_2^2 + \ldots + x_i^2} \tag{2.15}$$

Another common norm is taxicab distance, which is the 1-norm:

$$\|x\|_1 = \sum_i |x_i| = |x_1| + |x_2| + \ldots + |x_i| \tag{2.16}$$

2.6.3 Pruning Approaches

Rather than pruning the desired amount all at once, which is known as One-shot Pruning, some methods, known as Iterative Pruning, repeat the process of pruning the network to some degree and retraining it until the desired pruning rate is reached. Previous research in the Iterative Pruning model demonstrated that learning or designing when and how much to prune helps. A pruning schedule defines the ratio of pruning after each epoch as well as the number of epochs to retrain the model once it is pruned. The following are the three steps of a standard iterative pruning approach:

1. train a large, over-parameterized model (pre-trained models are often available).
2. prune the trained large model based on a criterion.
3. fine-tune the pruned model to recover the lost results.

Another approach instead of fine-tuning would be Weight rewinding [RFC20] which leads to better results.

2.7 Quantization

Deep learning workloads must leverage available computation and memory resources more efficiently in order to perform well on a wide range of platforms, from cloud-scale clusters to low-power edge devices. One other popular method to enhance resource efficiency is to quantize the Neural Network by reducing the bitwidth e.g. replacing 32-bit floating point numbers with 8-bit integers. By representing the used spectrum form float32 into 8bit integer shown in Figure 2.17

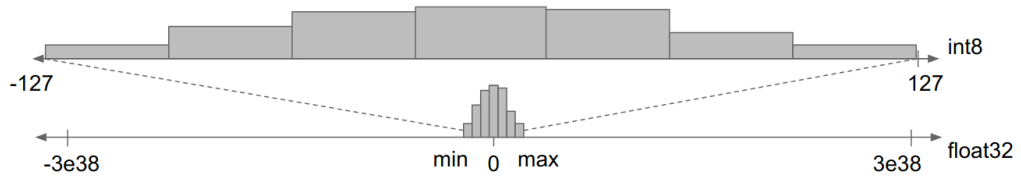


Figure 2.17: float32 quantized into int8
[AAB+15]

This enables the use of high-performance vectorized operations on a variety of hardware platforms, as well as a more compact model representation. In

comparison to normal FP32 models, PyTorch enables INT8 quantization, which allows for a 4x reduction in model size and a 4x reduction in memory bandwidth needs. When compared to FP32 computations, hardware support for INT8 computations is typically 2 to 4 times faster. In Figure 2.18 the representation of a 32–bit float number is shown. There are only 23 bits used for the decimal place. and thus the quantization will be 3 times less precise in the worst case as long as spectrum stays in $[-1, 1]$

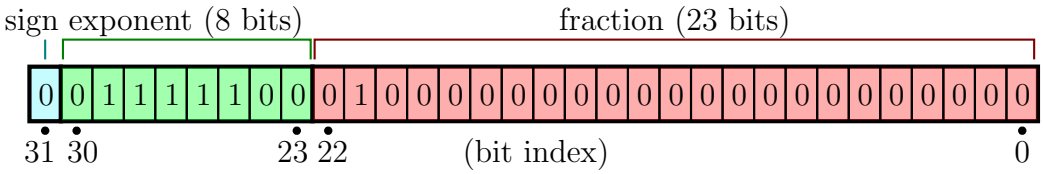


Figure 2.18: Binary representation of a 32-bit floating-point number. The value depicted, 0.15625, occupies 4 bytes of memory: 00111110 00100000 00000000 00000000

Quantization is primarily used to speed up inference, and quantized operators can only use the forward pass. There are different quantization approaches:

1. dynamic quantization (Weights have been quantized using activations that have been read/stored in floating point and then quantized for computation.)

2. static quantization (Weights have been quantified, activations have been quantified, and post-training calibration is necessary.)

3. quantization aware training (During training, weights were quantified, activations were quantified, and quantization numeric were modeled.)

Quantization can only be done in forward and not in backward pass. There for, the backward pass is done in FP32 and converted before being applied at the quantization aware training. This is called fake-quantization modules.

Chapter 3

Related Work

3.1 Synthetic Data for Deep Learning

Synthetic data is an approach to solving the data problem by producing artificial data from scratch or using advanced data manipulation techniques. Synthetic data can produce a potentially unlimited number of pixel-perfect labeled images and video clips, among other things. The goal is to make it possible for humans to create better data sets. This paper focuses on synthetic data rather than data augmentation. Synthetic data can be produced and supplied to machine learning models on the fly, during training.

The aim is to provide a more general overview of the field. They conclude with an analysis of some of the most popular synthetic data generators. In this paper, they look at the use of synthetic data to solve problems in computer vision, robotics, and other fields. Synthetic data can be used to resolve privacy or legal issues that make using real data prohibitively hard for some applications. They also examine how synthetic data is used to train new models.

In this paper, they look at the privacy side of synthetic data. They also discuss how to generate synthetic data with differential privacy guarantees. The paper concludes with a call for further work on synthetic data in finance and related fields. It includes directions for further research into the field. [Nik19]

They had a look into synthetic data for basic computer vision problems. They also examine how synthetic data can be used to improve general problems such as object detection or segmentation. In 1999, [FPC00] presented a synthetically generated world of images, with labeling derived from the corresponding 3D scenes, designed to train and evaluate computer vision algorithms. In 2011, [BSL+11] created a modern dataset for low-level vision

called Middlebury. [PMM+12] present the Tsukuba CG Stereo Dataset with synthetic optical flow maps. They show improvements in disparity and classification quality. This could lead to better understanding of where optical flow algorithms break down. The study was published in the open-to-read online journal arXiv.

[OBZ+18] consider an unusual special case for this problem: underwater disparity estimation. Their work is interesting in the way they produce synthetic images on randomized synthetic surfaces. They then use rendering tools developed to simulate the underwater sensors and characteristic underwater effects such as back scattering.

Synthetic data can be used to solve problems such as object detection and segmentation. These problems include object detection for everyday objects, food, and retail items. Synthetic data also includes 3D-related labeling, depth maps, and volumetric 3D data. PartNet is a benchmark for 3D object and scene understanding, but the corresponding 3D models will no doubt be widely used to generate synthetic data developed by [MZC+18] Synthetic data can be created by reusing the work of 3D artists that went into creating virtual environments of video games.

Recent works use synthetic datasets of everyday objects in more complex ways. In [AMM+17] and [GMBK17] the authors place synthetic objects into indoor scenes with an eye toward home service robots. The aim is to make it possible to place these synthetic objects on real backgrounds. A long line of work has used synthetic data for object detection. In the next section, we will see how this progress helps to solve the basic computer problems: after all, recognizing synthetic objects is never the end goal.

The aim is to improve high-level computer vision with synthetic data for object detection. [HLWK17] propose to freeze the lower layers of a pretrained object detection architecture and only train the top layers on synthetic data. They report that freezing the layers helps significantly, improvement in their results. Synthetic data can be used to solve problems such as 3D pose, viewpoint, and depth estimation. Which was exactly our starting goal.

[SAS+18] propose a pipeline that combines detection-based masks by Mask R-CNN and semantic segmentation masks by DeepLab for background classes. NVIDIA researchers have created the first state of the art network for 6-DoF pose estimation on synthetic data. They trained a deep neural network and report that it can transfer 3D pose estimation from synthetic to real data with proper domain randomization and domain adaptation. Synthetic data helped improve 3D object pose estimation in [GAGM15] and multi-view object class detection.

The Render for CNN approach outperformed real data with a hybrid synthetic+real dataset on the view estimation problem. Synthetic data has

also been used for pedestrian detection by [MVGL10] as part of this thesis.

3.2 Event-based Moving Object Detection and Tracking

The computer vision community has started to derive inspiration from the neuromorphic community whose ideas are based on biological systems to build robust and fast algorithms which run on limited computing power. The research was carried out by researchers at the University of California, San Diego. A new time-image representation of the ego-motion estimation problem has been developed. The paper is highly parallelizable and can be easily ported to a GPU or an FPGA for even lower latency. It uses a novel event-only feature-less motion compensation pipeline.

A new event-based approach for 3D motion estimation and segmentation is proposed. The system would have no knowledge about its motion or the scene. It could be used to detect moving objects in challenging situations, such as driving a car at high speed. Inventor: We can't predict when an object will move - we can only track it.

The algorithm derives inspiration from 3D point cloud processing techniques, such as Kinect Fusion by [IKH+11]. It performs global motion compensation by fitting a 4-parameter motion model to the cloud of events in a small time interval. The algorithm then labels event clusters which do not conform to the motion model and labels them separately. The data from the DVS sensor is four-dimensional, with the additional, fourth component a binary value denoting the sign of intensity change.

VI-A, T is a discretized plane with each pixel containing the average timestamp of the events mapped to it by the event cloud. Equation 6 is a global error which takes into account local motion inconsistencies of the event cloud. A pipeline of algorithms is used to detect independently moving objects. Camera motion compensation and subsequent motion inconsistency analysis are used to compensate for global background motion. The algorithm was developed by the University of California, San Diego in the 1990s.

It has been used in a variety of applications, such as tracking drones. Algorithm 1 describes the approach of the detection and tracking of independently moving objects by observing inconsistencies of T. The results are shown in Algorithm 2, which shows how the algorithm is used to detect and track the objects that have been detected and tracked using a Kalman Filter. The detection algorithm presented in Subsec. VI-A runs in real-time (processing time $\leq \delta t$) to account for missing and wrong detections.

They emphasize the ability of our pipeline to perform detection at very high rates, and include several sequences where the tracked object changes its speed abruptly. The dataset was collected using the DAVIS240B bioinspired sensor equipped with a 3.3mm lens with a horizontal and vertical field of view of 80°. Most of the sequences were created in a hand-held setting, but some were recorded on a mobile phone. The aim of the experiment was to test the reliability of tracking in scenarios where detection is not possible for a small period of time-based sensing can fill a void in the area of robotic visual navigation.

The algorithm is based on event-based only motion estimation and clustering. They argue that it can be used to spot moving objects more accurately. And it's useful for 3D vision. [MFPA18]

3.3 DavisDrivingDataset 2020

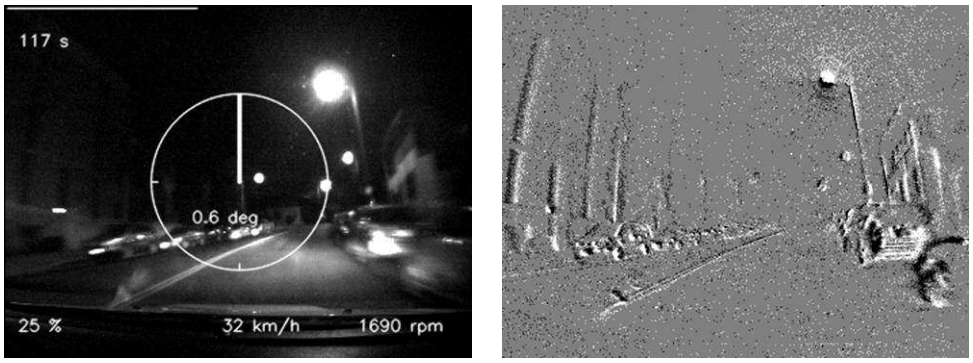


Figure 3.1: On the left side is the actual steering; On the right side is the visualized event data stream.
[HBN+20]

DDD20 is the longest event camera end-to-end driving dataset to date with 51h of DAVIS event+frame camera and vehicle human control data collected from 4000 km of highway and urban driving. One example Image collection is shown in 3.1. The next generation event camera called DAVIS could be used in driving applications. The system can handle dynamic range of 120dB vs. 60dB for handling uncontrolled lighting conditions. Samsung funded the project, which is now being expanded to include other partners.

DDD17, containing 12h of E2E labeled driving data, was used by [MLG+18] to compare the human steering with predictions using APS frames, APS frame differences, and DVS frames. DDD20 contains 51 h of recordings

collected from a 346 × 260-pixel DAVIS346 camera. The 'DAVIS Driving Dataset 2020' (DDD20) dataset will be released at http://sensorsini.uzh.ch/databases. We show that fusing APS and DVS data improves the predictions by a significant margin over either modality by itself. The original DDD17 Ford Mondeo dataset used a single mounting point.

The USB3.0 camera was mounted using a glass suction tripod fixed behind the windshield, just below the rear mirror, and aligned to point to the center of the hood. The 51 h of usable data were recorded under various weather, driving, road, and lighting conditions over about 60 days of intermittent recording. The car speed was uniformly distributed over the range of 0–130 km/h. Steering angles on straight roads were dominated by small deviations of ±10%. The original DDD17 dataset showed that a 50 ms DVS frame duration provided the optimum for DVS steering prediction.

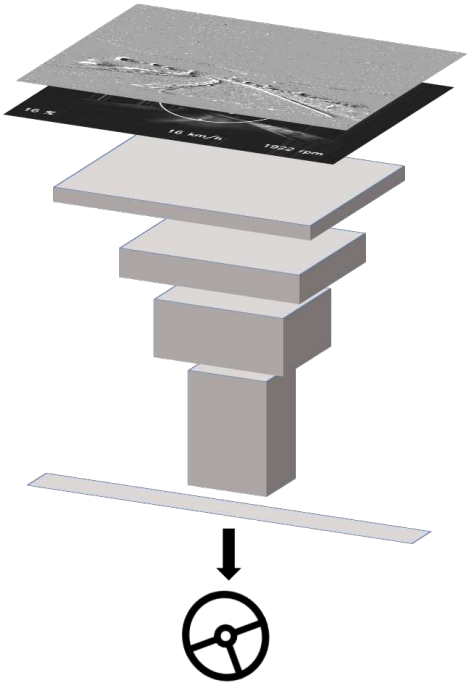


Figure 3.2: Illustration of the steering prediction network in the DDD20 paper. [HBN+20]

We used this data to study the steering angle prediction using our ResNet-32 network. The configuration for the convolutional neural network was similar to that used in [MLG+18]. Figure 3.2 shows the architecture of the steering angle network. In the cases of DVS-only and APS-only, the network is trained with a one-channel input.

The networks were trained for 200 epochs, using minibatches of 128 samples and using a Mean Squared Error (MSE) loss. Using DVS+APS results in the best steering prediction. All networks can successfully predict the steering wheel angle but the DVS +APS one is most accurate. It seems that the moving features exposed by DVS improve the steering predictions. The EVA for DVS+.APS is significantly better than for either DVS or APS alone.

DDD20 is the first open E2E driving dataset with over 50 h of recordings from a DAVIS event camera mounted on a vehicle driven over 4000 km. Results show that fused DVS and APS information best explains the steering variance under all driving conditions. [HBN+20]

3.4 Ground Moving Vehicle Detection and Movement Tracking Based on the Neuromorphic Vision Sensor

Autonomous vehicles need to be able to take initiative action to navigate traffic. This requires the autonomous vehicle to have some ability to detect and reason about the movement state of surrounding moving vehicles. The accuracy is good only for a short time ahead due to inertia and a varying movement. Several studies proposed from [BMH15], [ST12] and [SBM06] the use of machine learning methods for vision-based vehicle detection to improve detection accuracy. [BTDB11] presented a filter based on the dynamic Bayesian network that could estimate driving behaviors and anticipate future trajectories.

This approach is based on driving behavior estimation and classification, but drivers' behavior tends to be inherently multimodal. DVS uses an event-driven approach to capture pixel changes in the image. DVS do not have the concept of a "frame" and is not sensitive to light. Although the detection results are improved, the computational load resulting from vision computing remains high. This means that some monovision methods cannot meet the real-time requirement.

The main contribution of this article is to track moving objects by providing robust and accurate methods. The Otsu method is put forward to filter the scattered noise points, and the different moving objects were clustered with the event-points' depth information. This was used to generate high-resolution panoramic images of natural scenes. In the DVS sensor, the texture events of the moving objects occur obviously with the pixel value's change. The pixels with consistent colors or noise points usually occur at

a low frequency. By the Otsu method, the obvious and key features are highlighted and the noise pints are filtered of front-wheel steering. The state equation of the CTRV model is shown in [GBD10] and in [GKKÖ11] is necessary to use a multivariable Taylor expansion. After designing the Jacobian matrix (H) at the next page, and [SFR12] for the above.

strong tracking center differential external Kalman filter (SCDEKF) uses the principle of orthogonality [ZMN+20]. By introducing a fading factor in the state prediction covariance matrix, the output residuals are forced to have orchestrated or approximate orthogony. This allows the system to track the sudden switch of the system. A proposed method owns a low calculation load and high filter accuracy. The YOLO-V3 deep learning method is utilized to detect the moving objects.

The DVS and SCDEKF models are also established in Simulink. Because of the speed fluctuation, it is necessary to add process noise in the simulation process. The effectiveness of DVS is verified in two experimental scenarios. The results show that the common cam era has a distance-calculation delay problem and the DVS can detect the moving objects quickly.

In the experiments, two kinds of filter algorithms were adopted to conduct the vehicle position and movement estimations of Yolo-V3.

Model	Precision	Recall	mAP	FPS	Interface time	Parameters' file
Yolo-tiny	33.8	20.1	13.3	72	13.8	8.1M
Yolo-V3	39.2	23.5	15.7	67	14.9	20.8M

Table 3.1: NETWORK'S COMPARISON [LCSK20]

The comparison between YOLO and DVS is shown in 3.1.

3.3 shows the results of moving objects detection. Static trucks, moving vehicles, towers, and trees are all part of the highway setting. For moving-objects detection, common cameras and DVS cameras are utilized, and the results are shown in 3.3. YOLO has good object detection and classification capabilities, however it can't tell the difference between static and moving objects. The detection results of a DVS and K-means technique are shown in 3.3(b), and a car's front and rear sections are grouped into two objects, which is obviously incorrect. The detection results from a DVS and 3D-IMK approach are shown in 3.3(c), along with the accurate clustering findings. According to 3.3(b) and (c), the suggested 3D-IMK has a significant advantage in clustering, and the results are consistent with the simulation results. Although YOLO represents a significant step forward in increasing

the speed of object identification, 3.3(d) illustrates that YOLO still suffers from distance-calculation latency. One issue with YOLO detection is that it typically extracts photos corresponding to many candidate locations in advance, which takes up a lot of disk space. Each candidate region must go through the YOLO network calculation, which results in the identical feature extraction that is performed several times.

The campus scenario includes vehicles, bicycles, pedestrians, and trees. They detect the moving objects, calculate the angular velocity, and predict the turning centers of their moving trajectory by DVS and SCDEKF methods. Results show that the 3D-IMK can distinguish overlapped moving objects with a good effect. The SCDEKF method can achieve less delay effect and track the moving objects steadily. Compared with other computer vision algorithms, the DVS-based algorithms can detect moving objects with little delay and high accuracy. The overtaking simulation in Gazebo shows that the system can overtake the distance-calculation delay.

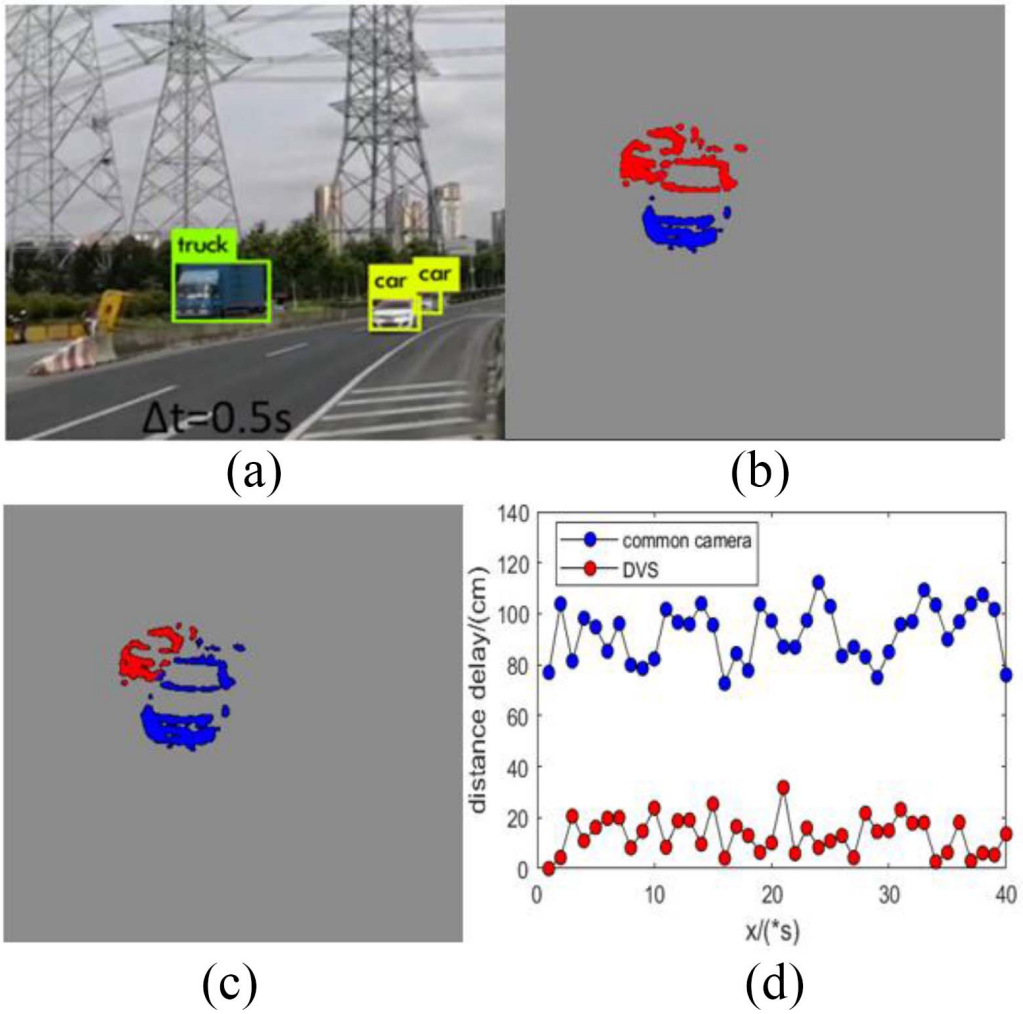


Figure 3.3: Detection experiment in the highway scenario. (a) Object detection by common camera. (b) DVS's clustering results by K-means. (c) DVS's clustering results by 3D-IMK. (d) Distance delay for different cameras.

[LCSK20]

Chapter 4

Dataset

Many modern AI problems stem from a lack of data. Either the available datasets are too limited, or the costs of manual labeling are prohibitively high, even though collecting unlabeled data is relatively simple. Synthetic data is an important approach to solving the data problem since it allows for the development of artificial data from scratch or the use of advanced data processing techniques to create novel and diverse training examples. The synthetic data generation method is better demonstrated by traditional computer vision problems, but it is also applicable in the problem domain this thesis tackles. Another significant challenge is the issue of domain transition. As shown in Figure 4.1, synthetic images do not look exactly like real images, and one must make them as photo realistic as possible (a common theme in synthetic data research is whether realism is really necessary [Nik19] and/or devise techniques that assist models in transferring from synthetic training sets to real test sets.

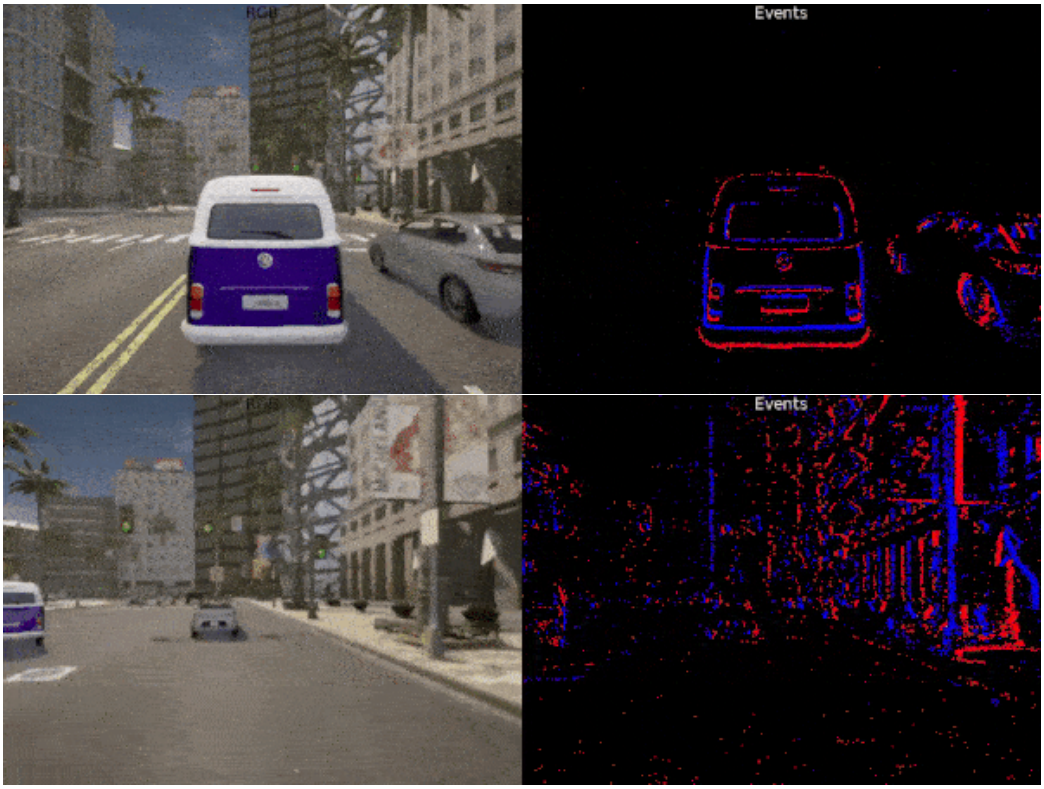


Figure 4.1: Example from a RGB image on the left and a DVS image on the right with displaying all events happened in the same time sequenced. [DRC+19]

This chapter details the system design and decisions for the generation of the synthetic dataset. There exist several well annotated recent datasets that are also using an event camera as well as RGB frames but do not provide similar annotations. Therefore Tobi [DHH20], Yuhuang Hu and Zhe He developed a realistic dynamic vision sensor event data synthesizer from real (or synthetic) conventional frame based video using an accurate DVS pixel model that includes DVS non idealities. Most of them are created by a DAVIS and are intended for Simultaneous Localization and Mapping (SLAM). It is created because of the lack off well annotated event camera datasets with the following properties:

- 3D annotation
- Instance segmentation
- 2D annotation
- synchronized RGB and event camera datastream

The Structure is build up on the System Design Document Template from "Object-Oriented Software Engineering. Using UML, Patterns, and Java" [BD09]. The system design's main objective is to work out the design goals of the envisioned system and to decompose it into smaller subsystems.

4.1 CARLA Simulator

CARLA is an open source framework based on the Unreal Engine [Epi19], which is able to render high realistic textures, shadows, weather conditions and illumination, as depicted in 4.1 developed by [DRC+17] CARLA was designed from the ground up for supporting autonomous driving systems production, training and validation. CARLA also offers open visual assets (urban layouts, houses and vehicles) in addition to open source code and protocols which were developed and can be used freely for this purpose. The platform of simulation supports versatile sensor suite specification, environment conditions, complete static and dynamic player control, mapping generation and much more [DRC+17]. The most highlighted features are:

- It offers scalability via a multi-client server architecture with multiple clients in the same or different nodes that manage various actors.
- CARLA provides a versatile API that enables users to monitor anything related to simulation, including the generation of traffic, pedestrian behavior, weather and sensors.
- The Autonomous Driving Sensor Suite comprises a wide range of sensor suites, including LIDAR, multiple cameras, DVS cameras, sensors and GPS.
- It is possible to disable its rendering for which graphics are not needed to provide a rapid execution of traffic simulation for planning, control and road behaviors.
- Users can easily build their own maps using software such as RoadRunner according to the OpenDrive format.
- As runnable CARLA agents, including an AutoWare agent and a Conditional Imitation Learner, we have Autonomous Driving baselines.
- And much more features we did not use.

Carla's Architecture is a Client-server Model. The client-server architecture is a computer paradigm in which the server hosts, provides, and

maintains the majority of the client's resources and services. One or more client computers are linked to a central server through a network or internet connection in this design. Because all requests and services are given through a network, client-server architecture is also known as a networking computer paradigm or client-server network. There for the Simulator runs as a server with Unrealengine and the Clients are able to steer a Car, get Sensor data, control the traffic, change the weather or even changing the Sensor properties on the fly.

4.2 CARLA DVS Camera Simulation

The DVS camera simulation outputs a stream of events. An event ($e = (x, y, t, pol)$ is triggered at a pixel x, y at a timestamp t when the change in logarithmic intensity L reaches a predefined constant threshold C (typically between 15% and 30%). It was developed and published by the University of Zurich [RGS18].

The generation of DVS pixel events is described by the following equation:

$$L(x, y, t) - L(x, y, t - \delta t) = polC \tag{4.1}$$

Where the value $t - \delta t$ represents the time when the last event occurred at the pixel and is true if the pixel changed from light to dark. The polarity is $+1$ when there is an increase in brightness, and -1 when there is a decrease in brightness. In the Figure 4.2 below, the following working principles are illustrated. The frame rate of the regular camera is the same regardless of whether or not there is motion in the picture. On the other hand, an event sensor needs microsecond response time to analyze and respond to changes in light levels. An event is created whenever there is positive or negative intensity change stronger than the contrast threshold C.

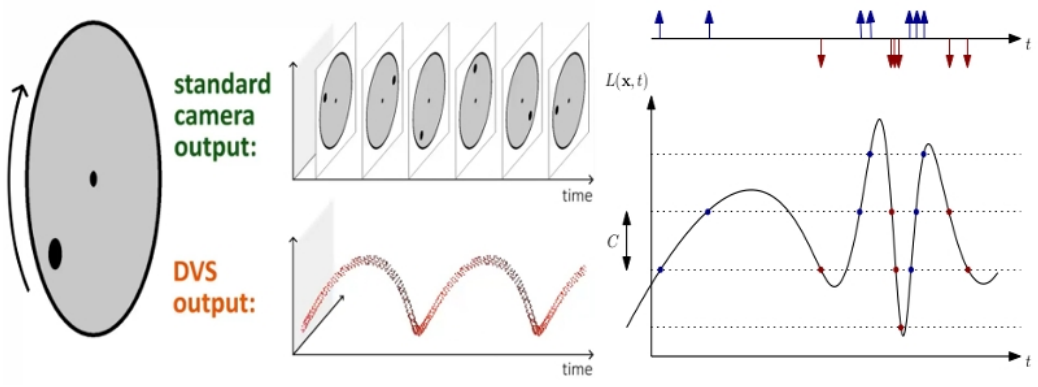


Figure 4.2: basic DVS functionality illustrated on a rotating dot [DRC+19]

A DVS is a full-form camera, which includes all of the characteristics found in the RGB camera. Even so, the operating principle of an Event camera is remarkably unique. The DVS camera is operating at irregular intervals between two consecutive conventional RGB images. The time resolution of a real event (a camera pixel which takes a millionth of a second) must be achieved using a real-time sensor (much higher frequency than a conventional camera). Formally, the number of events grows as the vehicle is moving faster. Thus, the sensor frequency should be proportional to the intensity of the scene. There is a trade off between the user's accuracy and computing cost. And because we did not have any concerning computational or time restrictions, we used a high accuracy for the generation of the DVS events. In Table 4.1 one can see all the attributes available as well as the ones we used as defaults.

Blueprint attribute	Type	Default	Description
positive_threshold	float	0.3	Positive threshold C associated to a increment in brightness change (0-1).
negative_threshold	float	0.3	Negative threshold C associated to a decrement in brightness change (0-1).
sigma_positive_threshold	float	0	White noise standard deviation for positive events (0-1).
sigma_negative_threshold	float	0	White noise standard deviation for negative events (0-1).
refractory_period_ns	int	0.0	Refractory period (time during which a pixel cannot fire events just after it fired one), in nanoseconds. It limits the highest frequency of triggering events.
use_log	bool	true	Whether to work in the logarithmic intensity scale.
log_eps	float	0.001	Epsilon value used to convert images to log: $L = log(eps + I/255.0)$ Where I is the grayscale value of the RGB image: $I = 0.2989 * R + 0.5870 * G + 0.1140 * B$

Table 4.1: Carla DVS Camera attributes and used properties [DRC+19]

4.3 Data generation

This section deals with the configuration and setup of the data generator. It meets the Object-Oriented Software Engineering Guidelines. And its Patterns are transferred from the UML, Patterns, and Java by [BD09] to C++,

Python and the C++ API for Python. The design's main objective is to achieve the goal of the system as best as possible, and then decompose the system into smaller subsystems. The following is a high-level summary of the software architecture and its relation to other parts of the thesis. If the system design goals are mentioned in the relevant section, then they will be achieved by the system's design. The subsystem decomposition defined in Section 4.3.1 serves to meet these design objectives. Section 4.3.2 presents the data management, which was carried out by latest. LATEST has good data access control as defined in Section 4.3.3.

4.3.1 Open source contribution

In order for the data set to serve as many purposes as possible and for us to have needed the segmentation instance as well, we decided to add exactly this sensor to the Carla Simulator. Segmentation is often the first step in image analysis. And consists in a complete segmentation that each pixel is assigned to at least one segment and thus has a context. In a coverage-free segmentation, each pixel is assigned to at most one segment. Thus, in a complete and coverage-free segmentation, each pixel is assigned to exactly one segment. The instance segmentation gives each instance on the image its own ID. This is divided into groups and instance ID. This was only made for vehicles and pedestrians because of some constrains given by the Unreal Engine and Carla Simulator. Which did not allow us to give more than 255 different segments at one time without changing the hole Architecture of Carla Sensor Engine. The exact assignment can be read in the appendix.

We had to add another sensor to the Carla framework. This sensor fulfills these properties.

4.3.2 Datageneration Details

The data creation pipeline is presented here. For this purpose, fixed camera positions were manually selected before the creation so that there are as many and alternating positions as possible. Also because so the traffic that was simulated, only in the field of view of the camera had to be calculated or spawned. For this one could fly in the selected 3D world by camera and determine the positions. These were then stored in a JavaScript Object Notation (JSON) file and later read in as a configuration file when the framework was started. During the data creation itself, each camera was processed sequentially. For this, in each step the cars that are needed, which can also reach the camera field of view in the recording time, were spawned in the vicinity of the field of view. The sequential processing was due to

the enormous power needed by the Carla engine to calculate the physics and thus our DVS events.

After each frame all data was labeled and stored. Thus, if the simulation was aborted, it could be continued directly. The following elements were stored per frame:

- 2D/ 3D annotation labels as JSON file
- DVS events array
- DVS event array as image
- Depth image
- Instance Segmentation
- RGB Image

Each set created by a camera position is stored in its own folder. This ensures that the different locations are not mixed during training. Its name is the camera seed as well as a hash from the extrinsic camera parameters. Each image name is generated by this hash and its frame number. An example path to an image would then look like this:

```
set_0_16452643/images/16452643_105945_dvs.png
set_0_16452643/labels/16452643_105945_dvs.txt
```

The converting of the DVS events to the DVS images is done by mapping a time sequence of them onto an image frame. We were given an event array which has the same recording time as the RGB images (20 ms). The negative and positive events are assigned an RGB color. For us this is blue for positive and red for negative. And these pixels were then drawn on the image area. We wanted to test if the DVS provides enough information for the task of vehicle position estimation. To do this we needed to benchmark against well known efficient existing model which is YOLO. So to do a direct comparison we used the DVS images. By using the DVS events stream instead of images there is a lot of potential to make the model/solution much more efficient!

4.3.3 Dataset

The data set itself finally consisted of

- 30 camera locations

- 900 Images per location
- 45 seconds per location
- splitted by camera location

All over we had 27000 Images splitted into train:test:val by a ratio of 3:1:1. One example dataset frame is visualized in Figure 4.3

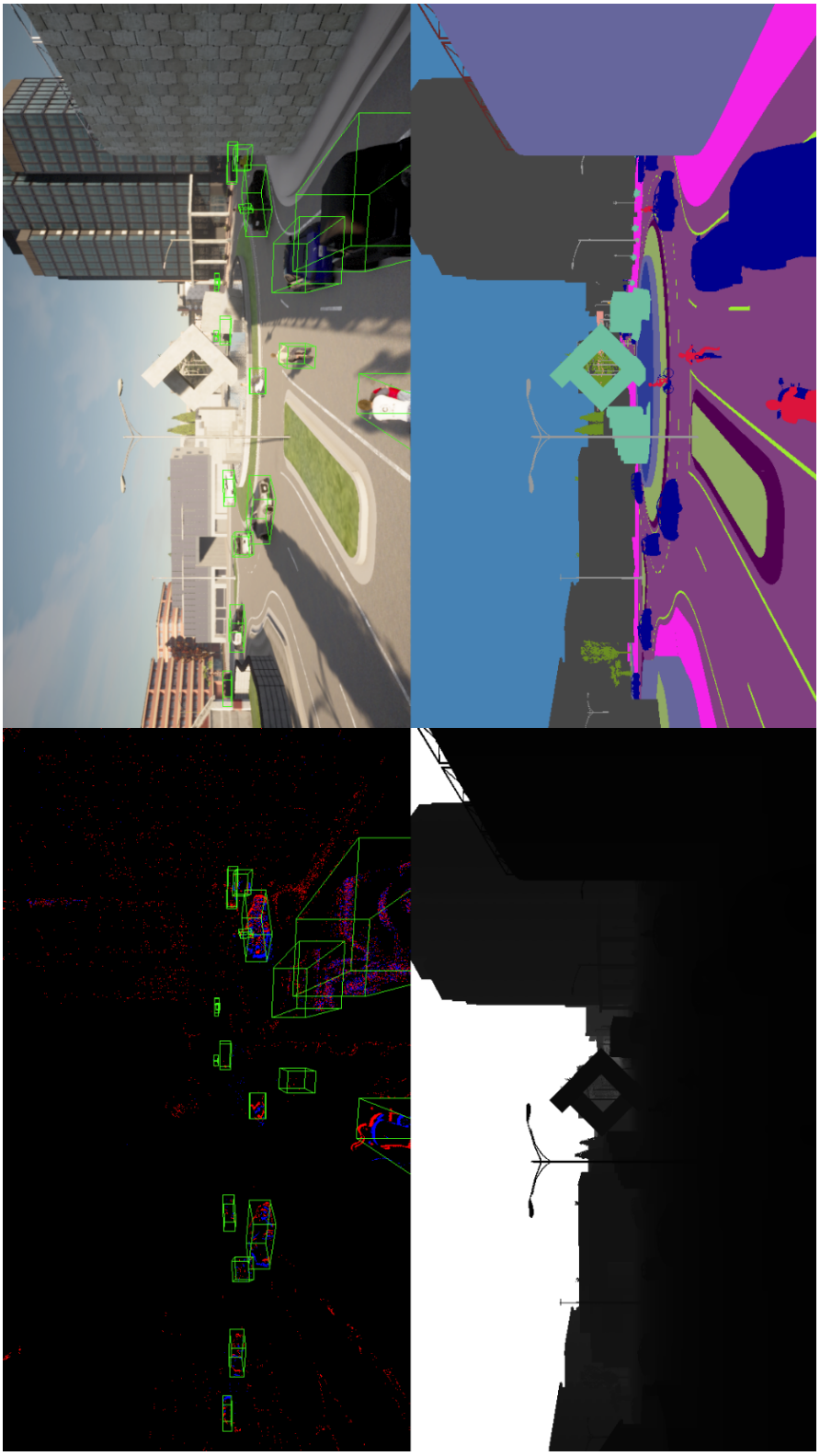


Figure 4.3: left top is the 3D annotated DVS Image, right top is the 3D annotated RGB Image, left bottom is the depth map, right bottom is the Semantic Segmentation Image

Chapter 5

Methods

5.1 YOLO

We used for both the DVS Image and the RGB Image YOLO the small YOLOv5 model with the following parameters:

```
Model Summary:
    191 layers,
    7.46816e+06 parameters,
    7.46816e+06 gradients
```

The hyperparameters we use are the default ones from the YOLOv5 source-code [Joc21] as well as the automatically applied finetuning from YOLOv5 is used.

5.2 Pruning

We applied pruning iteratively using pytorch's internal pruning functions as described in Chapter 2.6. We used LN_STRUCTURED with $n = 1$ explained in 2.15 and $n = 2$ explained in 2.16 on dimension 0 as well as L1_UNSTRUCTURED. For all methods we applied the iterative pruning from 0% to 99% of zeroes in different step sizes shown in equation 5.1.

$$amount \in [0, 10, 20, 30, 40, 50, 60, 70, 75, 78, 80, 90, 95, 97, 98, 99, 99.5] \quad (5.1)$$

```
torch.nn.utils.prune.ln_structured(
    module, name, amount, n, dim, importance_scores=None)

torch.nn.utils.prune.l1_unstructured(
    module, name, amount, importance_scores=None)
```

5.3 Quantization methods

We applied quantization with pytroch's internal functions described in Chapter 2.7 as quantization aware training. Therefor, it models the quantization errors in both the forward and backward passes using fake-quantization module before each layer and after each activation function. We had to modify all methods to be able to apply quantization aware training.

5.4 Spiking CNN Activations

For the Spiking CNN we had to add a state to the Network. To simplify this state conversion we used the framework Norse which was developed by [PP21]. Afterwards we only had to change the functions of the Convolutional Layer by changing the most used higher hierarchical Layers.

- Focus
- Convolution
- BottleneckCSP

This was done by adding the LIF parameter to the Constructor with the default parameters shown in 5.1 and replacing the activation function with a "LIFCell".

```
LIFPa = LIFParameters(
    tau_syn_inv=torch.as_tensor(1.0 / 5e-3),
    tau_mem_inv=torch.as_tensor(1.0 / 1e-2),
    v_leak=torch.as_tensor(0.0),
    v_th=torch.as_tensor(0.00001),
    v_reset=torch.as_tensor(0.0),
    method="tent",
    alpha=torch.as_tensor(100.0),
)
```

Figure 5.1: default LIF parameters

As well as adding a state parameter to each forward function, which keep track of its activations. The source code of the easiest conversions can be viewed and compared in A.2 and A.1.

Chapter 6

Results

In this chapter we evaluate if the data generated by a DVS camera contains sufficient information for efficient vehicle position estimation. Given that the YOLOv5 architecture is able to learn and perform efficient object detection on RGB input images, we investigate if the same architecture is able to solve the same task using the DVS camera input images. To get the most valuable results, all scores are average numbers by training each network 10 times and only trained the altered layers from with scratch with all other layers frozen and fine-tuned it at the end.

6.1 Vehicle position estimation using Spiking Neural Networks

At the beginning, we altered different layers of YOLOv5 to spiking ones. These layers are:

- Focus
- Hole backbone
- All
- Prediction

And compared this to the results of YOLOv5 trained with different inputs shown in Table 6.1. The Network names which are not YOLOv5 in the Table 6.1 refers to the YOLOv5 with spiking modification.

Input	Network	precision	mAP 0.5-0.95	mAP 0.5	recall
RGB Image	YOLOv5	92	79	98	63
DVS Image	YOLOv5	96	81	99	65
DVS	Focus	82	82	99	1
DVS	Prediction	75	74	96	95
DVS	Backbone	70	69	90	85
DVS	All	30	28	19	18

Table 6.1: Metrics of different Networks compared to the original RGB Network. All Numbers are percentages.

Except for the improvement of the network by using the DVS images, one could expect the result to be like this. Due to the recall of 1 in the spiking focus network, it is to be expected that one object was recognized more the once. This did not happen with the normal Yolov5. For further investigations, the most successful spiking one was chosen.

6.2 Improving the network generalization with pruning

We pruned only the spiking focus network. In Figure 6.1 the pruning progress of different pruning values is shown. And only the important values are noted down.

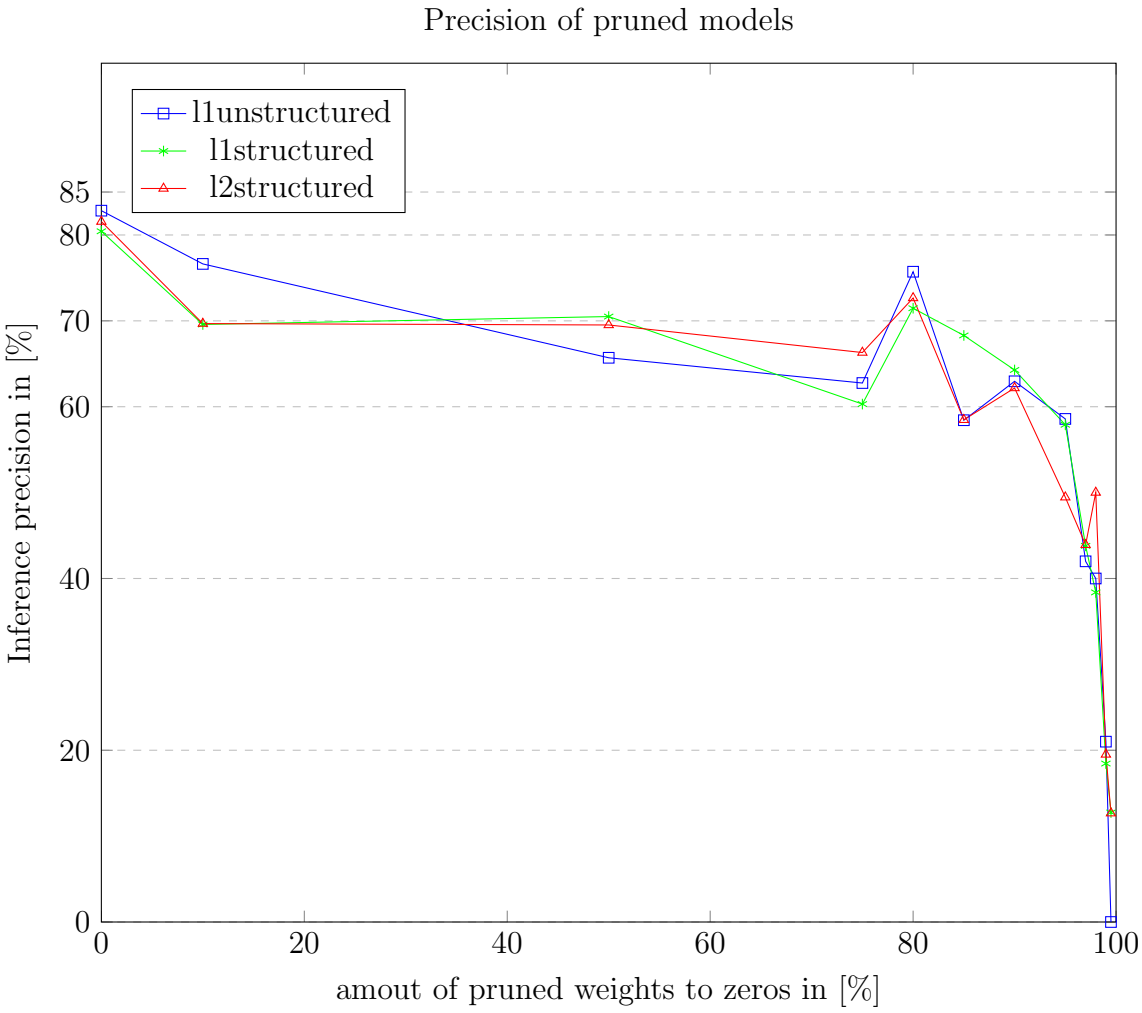


Figure 6.1: Caption

The resulting regularization could be best shown by comparing the validation metrics with its inference equivalent drawn in Figure 6.2 with a pruning rate of 78%.

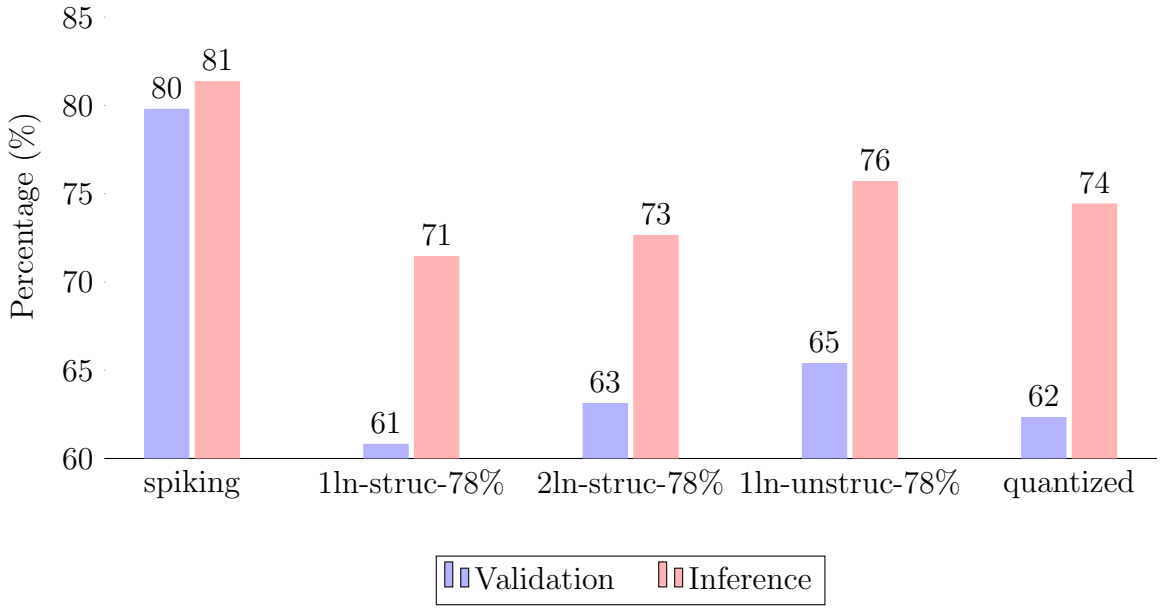


Figure 6.2: Compared Inference and Validation precision with different pruning methods on the spiking focus YOLOv5 network

With a residual size of 32% of the network and with only 6% loss of accuracy, pruning is successfully applied.

6.3 Improving the network efficiency by quantization

We applied quantization aware training on the best working YOLOv5 spiking Network from Table 6.1 the Focus one. In our case the Quantization aware training reduces the weight memory size by 4. Because we did exactly what is described in Section 2.7. We reduced the datatype from flot32 to int8 on the model weights. But still holds the following properties shown in Table 6.2. The inference comparison in shown in Figure 6.2

Input	Network	precision	mAP 0.5-0.95	mAP 0.5	recall
DVS	Focus	82	82	99	1
DVS	Quantized Focus	62	58	85	90

Table 6.2: Metrics of quantized Spiking Focus Layer YOLOv5 Network and for comparison the Spiking Focus Layer. All Numbers are percentages

6.4 Speedups

The Section 6.3 and 6.2 both improved the generalization of networks, but they also had minimal speed up, we should net forget in these results in Table 6.3. These accelerations were achieved on the device on which the training was performed. This can and will differ from device to device. Especially if the hardware was built for it

Input	Network	mins	relative
RGB Image	YOLOv5	125	1.00
DVS Image	YOLOv5	124	0.99
DVS	Focus	120	0.96
DVS	Prediction	116	0.93
DVS	Backbone	114	0.91
DVS	All	110	0.88
DVS	Pruned 78% Focus	95	0.76
DVS	Quantized Focus	90	0.72

Table 6.3: Inference duration compare with 1M runs per network.

Chapter 7

Conclusion

We have successfully created a dataset pipeline. With which a fully usable dataset was generated. This contains 2D, 3D, depth and segmentation labels for RGB images and DVS event stream.

This dataset was then tested and validated with YOLOv5 on RGB and DVS images. YOLOv5 on DVS Images was the corresponding point for all further researches. It was then shown that it is possible to detect vehicle positions only using the DVS event stream on a modified YOLOv5 Network. This network architecture was modified to be able to take an DVS event stream as input. And some layers were altered to spiking ones. All this was achieved with a loss of only absolute 10% in precision.

The network was then individually optimized using pruning and quantization, resulting in memory reduction and speedup. It was shown that the different pruning methods offered by PyTorch does almost perform the same for this type of network. A lost of roughly absolute 10% in precision compared to the inference from the spiking one was achieved. As well as a speedup from 25% to the original RGB YOLOv5. Quantization on the other hand did not work as well as pruning. It almost loses absolute 20% in precision by quantization aware training. And a speedup from 28% to the original one.

The network could be made even faster and smaller by choosing a spiking optimal network architecture for classification and then training the network from scratch. Because this synthetic dataset offers a lot of different labels, we could imagine one can train a 3D car detection network. Or other DVS based solutions. Maybe it would be interesting to test this network against real world data as well as fine tune it with them.

Appendix A

Sourcecode

A.1 Convolution

```
class Conv(nn.Module):
    # Standard Convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):
        # self, ch_in, ch_out, kernel, stride, padding,
        # groups, activation
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p),
                              groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = nn.Hardswish() if act is True
                    else (act if isinstance(act, nn.Module)
                              else nn.Identity())

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))
```

A.2 Spiking Convolution

```
class Conv_S(nn.Module):
    # Standard Spiking Convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1,
                 act=True, lifP=None):
        # self, ch_in, ch_out, kernel, stride, padding,
        #       groups, activation, LIFParameters
        super(Conv_S, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p),
                              groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        if not lifP:
            lifP = LIFPa # default LIFPa

        self.act = LIFCell(p=lifP, dt=0.001)

    def forward(self, input_tensor: torch.Tensor,
                state: Union[list, None] = None):
        if isinstance(input_tensor, tuple):
            state = input_tensor[1]
            input_tensor = input_tensor[0]

        return self.act(self.bn(self.conv(input_tensor)), state)

    def fuseforward(self, x, state: Union[list, None] = None):
        return self.act(self.conv(x), state)
```

List of Figures

1.1 On the left side the Digital Twin is depicted. On the right side the original Image. [KSGK19] 3
1.2 A Sample Construction of the Hardware needed to build this IIS [KSGK19] 3

2.1 Event camera vs standard camera: Unlike the upper graph, which shows a sequence of video frames from a standard camera, the lower graph shows a stream of events from an event camera, which has no redundant data output (only informative pixels or no events at all), no motion blur, and a high dynamic range. The red and blue dots indicate positive and bad events, respectively, and this picture was inspired by the famous animation of [Kim17] 7
2.2 On the left side is an image-like visualization of accumulated events within a time interval, where white and black pixels represent positive and negative events, respectively. right is a series of events depicted as upward and downward spikes for positive and negative events. Each event is a tuple (hu, v, p, t) where u and v are the event's pixel coordinates, p is the event's polarity, and t is the event's timestamp in microseconds; [Kim17] 8
2.3 Simplified representation of an artificial neural network 9
2.4 Visualization of a single neuron. 10
2.5 Visualization of the Sigmoid activation function. 11
2.6 Structure of a typical CNN for image classification. 11
2.7 Max pooling with a 2×2 filter and step size = 2. The step size indicates how many pixels the filter moves per operation. . . . 14
2.8 The illustration of Leaky Integrate and Fire (LIF) neuron dynamics. 17
2.9 Object detection architecture from [BWL20] 18
2.10 Yolov5 structure diagram. It shows all high level architectures as well as the 4 parts 20

2.11 On the left side is the netron Visualisation of the Focus layer; And on the right side the simplified Illustrated one 22
2.12 Detailed view of the BottleneckCSP 23
2.13 Detailed view of the SPP and Focus layers. 24
2.14 On the left a pyramid of images to detect different scaled objects. On the right a feature pyramid to do the same. 25
2.15 Illustration of PAN. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. 25
2.16 Neural Network Pruning differences between structured and unstructured. 28
2.17 float32 quantized into int8 29
2.18 Binary representation of a 32-bit floating-point number. The value depicted, 0.15625, occupies 4 bytes of memory: 00111110 00100000 00000000 00000000 30

3.1 On the left side is the actual steering; On the right side is the visualized event data stream. 34
3.2 Illustration of the steering prediction network in the DDD20 paper. 35
3.3 Detection experiment in the highway scenario. (a) Object detection by common camera. (b) DVS's clustering results by K-means. (c) DVS's clustering results by 3D-IMK. (d) Distance delay for different cameras. 39

4.1 Example from a RGB image on the left and a DVS image on the right with displaying all events happened in the same time sequenced. 41
4.2 basic DVS functionality illustrated on a rotating dot 44
4.3 left top is the 3D annotated DVS Image, right top is the 3D annotated RGB Image, left bottom is the depth map, right bottom is the Semantic Segmentation Image 49

5.1 default LIF parameters 51

6.1 Caption 54
6.2 Compared Inference and Validation precision with different pruning methods on the spiking focus YOLOv5 network . . . 55

List of Tables

3.1 NETWORK'S COMPARISON [LCSK20] 37

4.1 Carla DVS Camera attributes and used properties 45

6.1 Metrics of different Networks compared to the original RGB Network. All Numbers are percentages. 53
6.2 Metrics of quantized Spiking Focus Layer YOLOv5 Network and for comparison the Spiking Focus Layer. All Numbers are percentages 56
6.3 Inference duration compare with 1M runs per network. 56

Bibliography

[AAB+15] Abadi, Martín ; Agarwal, Ashish ; Barham, Paul ; Brevdo, Eugene ; Chen, Zhifeng ; Citro, Craig ; Corrado, Greg S. ; Davis, Andy ; Dean, Jeffrey ; Devin, Matthieu ; Ghemawat, Sanjay ; Goodfellow, Ian ; Harp, Andrew ; Irving, Geoffrey ; Isard, Michael ; Jia, Yangqing ; Jozefowicz, Rafal ; Kaiser, Lukasz ; Kudlur, Manjunath ; Levenberg, Josh ; Mané, Dandelion ; Monga, Rajat ; Moore, Sherry ; Murray, Derek ; Olah, Chris ; Schuster, Mike ; Shlens, Jonathon ; Steiner, Benoit ; Sutskever, Ilya ; Talwar, Kunal ; Tucker, Paul ; Vanhoucke, Vincent ; Vasudevan, Vijay ; Viégas, Fernanda ; Vinyals, Oriol ; Warden, Pete ; Wattenberg, Martin ; Wicke, Martin ; Yu, Yuan ; Zheng, Xiaoqiang: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. `https://www.tensorflow.org/`. Version: 2015. – Software available from tensorflow.org

[Aga18] Agarap, Abien F.: Deep Learning using Rectified Linear Units (ReLU). In: CoRR abs/1803.08375 (2018). `http://arxiv.org/abs/1803.08375`

[AMM+17] Alhaija, Hassan A. ; Mustikovela, Siva K. ; Mescheder, Lars M. ; Geiger, Andreas ; Rother, Carsten: Augmented Reality Meets Computer Vision : Efficient Data Generation for Urban Driving Scenes. In: CoRR abs/1708.01566 (2017). `http://arxiv.org/abs/1708.01566`

[BD09] Brügge, Bernd ; Dutoit, Allen: Object-Oriented Software Engineering Using UML, Patterns, and Java. In: Learning 5 (2009), Nr. 6, S. 7

[Bis06] Bishop, Christopher M.: Pattern recognition and machine learning. 1. USA, New York : Springer, 2006

[BMH15] Barrios, Cesar ; Motai, Yuichi ; Huston, Dryver: Trajectory Estimations Using Smartphones. In: IEEE Transactions on Industrial Electronics 62 (2015), Nr. 12, S. 7901–7910. http://dx.doi.org/10.1109/TIE.2015.2478415. – DOI 10.1109/TIE.2015.2478415

[BSL+11] Baker, Simon ; Scharstein, Daniel ; Lewis, J. P. ; Roth, Stefan ; Black, Michael J. ; Szeliski, Richard: A Database and Evaluation Methodology for Optical Flow. In: International Journal of Computer Vision 92 (2011), Mar, Nr. 1, 1-31. http://dx.doi.org/10.1007/s11263-010-0390-2. – DOI 10.1007/s11263–010–0390–2. – ISSN 1573–1405

[BTDB11] Berthelot, Adam ; Tamke, Andreas ; Dang, Thao ; Breuel, Gabi: Handling uncertainties in criticality assessment. In: 2011 IEEE Intelligent Vehicles Symposium (IV), 2011, S. 571–576

[BWL20] Bochkovskiy, Alexey ; Wang, Chien-Yao ; Liao, Hong-Yuan M.: YOLOv4: Optimal Speed and Accuracy of Object Detection. 2020

[CPK+17] Chen, Liang-Chieh ; Papandreou, George ; Kokkinos, Iasonas ; Murphy, Kevin ; Yuille, Alan L.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. 2017

[CR19] Chen, Jiasi ; Ran, Xukan: Deep Learning With Edge Computing: A Review. In: Proceedings of the IEEE PP (2019), 07, S. 1–20. http://dx.doi.org/10.1109/JPROC.2019.2921977. – DOI 10.1109/JPROC.2019.2921977

[Dab20] Dabai, Jiang: In-depth explanation of the core basic knowledge of Yolov5 of the Yolo series. https://blog.csdn.net/nan355655600/article/details/107852288, Januar 2020

[DBX+19] Duan, Kaiwen ; Bai, Song ; Xie, Lingxi ; Qi, Honggang ; Huang, Qingming ; Tian, Qi: CenterNet: Keypoint Triplets for Object Detection. 2019

[DHH20] Delbruck, Tobi ; Hu, Yuhuang ; He, Zhe: V2E: From video frames to realistic DVS event camera streams. In: arxiv (2020), Juni. http://arxiv.org/abs/2006.07722

[DLHS16] Dai, Jifeng ; Li, Yi ; He, Kaiming ; Sun, Jian: R-FCN: Object Detection via Region-based Fully Convolutional Networks. In: Lee, D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Luxburg, U. (Hrsg.) ; Guyon, I. (Hrsg.) ; Garnett, R. (Hrsg.): Advances in Neural Information Processing Systems Bd. 29, Curran Associates, Inc., 2016

[DLJ+20] Du, Xianzhi ; Lin, Tsung-Yi ; Jin, Pengchong ; Ghiasi, Golnaz ; Tan, Mingxing ; Cui, Yin ; Le, Quoc V. ; Song, Xiaodan: SpineNet: Learning Scale-Permuted Backbone for Recognition and Localization. 2020

[DRC+17] Dosovitskiy, Alexey ; Ros, German ; Codevilla, Felipe ; Lopez, Antonio ; Koltun, Vladlen: CARLA: An Open Urban Driving Simulator. In: Proceedings of the 1st Annual Conference on Robot Learning, 2017, S. 1–16

[DRC+19] Dosovitskiy, Alexey ; Ros, German ; Codevilla, Felipe ; Lopez, Antonio ; Koltun, Vladlen: Carla Simulator DVS Camera Documentation. `https://carla.readthedocs.io/en/latest/ref_sensors/#dvs-camera`, 2019. – Accessed: 2020-09-7

[Epi19] Epic Games: Unreal Engine. `https://www.unrealengine.com`. Version: April 2019

[FPC00] Freeman, William T. ; Pasztor, Egon C. ; Carmichael, Owen T.: Learning Low-Level Vision. In: International Journal of Computer Vision 40 (2000), Oct, Nr. 1, 25-47. `http://dx.doi.org/10.1023/A:1026501619075`. – DOI 10.1023/A:1026501619075. – ISSN 1573–1405

[GAGM15] Gupta, Saurabh ; Arbeláez, Pablo A. ; Girshick, Ross B. ; Malik, Jitendra: Inferring 3D Object Pose in RGB-D Images. In: CoRR abs/1502.04652 (2015). `http://arxiv.org/abs/1502.04652`

[GBD10] Gindele, Tobias ; Brechtel, Sebastian ; Dillmann, Rüdiger: A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments. In: A probabilistic model for estimating driver behaviors and vehicle trajectories in traffic environments, 2010, S. 1625 – 1631

[GK02] Gerstner, Wulfram ; Kistler, Werner: Spiking Neuron Models: An Introduction. USA : Cambridge University Press, 2002. – ISBN 0521890799

[GKKÖ11] Gadepally, V. ; Kurt, A. ; Krishnamurthy, A. ; Özgüner, Ü.: Driver/vehicle state estimation and detection. In: 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC) (2011), S. 582–587

[GLPL19] Ghiasi, Golnaz ; Lin, Tsung-Yi ; Pang, Ruoming ; Le, Quoc V.: NAS-FPN: Learning Scalable Feature Pyramid Architecture for Object Detection. 2019

[GMBK17] Georgakis, Georgios ; Mousavian, Arsalan ; Berg, Alexander C. ; Kosecka, Jana: Synthesizing Training Data for Object Detection in Indoor Scenes. In: CoRR abs/1702.07836 (2017). http://arxiv.org/abs/1702.07836

[HBN+20] Hu, Yuhuang ; Binas, Jonathan ; Neil, Daniel ; Liu, Shih-Chii ; Delbrück, Tobi: DDD20 End-to-End Event Camera Driving Dataset: Fusing Frames and Events with Deep Learning for Improved Steering Prediction. In: CoRR abs/2005.08605 (2020). https://arxiv.org/abs/2005.08605

[HGDG18] He, Kaiming ; Gkioxari, Georgia ; Dollár, Piotr ; Girshick, Ross: Mask R-CNN. 2018

[HLWK17] Hinterstoisser, Stefan ; Lepetit, Vincent ; Wohlhart, Paul ; Konolige, Kurt: On Pre-Trained Image Features and Synthetic Images for Deep Learning. In: CoRR abs/1710.10710 (2017). http://arxiv.org/abs/1710.10710

[HW68] Hubel, D. H. ; Wiesel, T. N.: Receptive fields and functional architecture of monkey striate cortex. In: The Journal of Physiology 195 (1968), Nr. 1, 215-243. http://dx.doi.org/https://doi.org/10.1113/jphysiol.1968.sp008455. – DOI https://doi.org/10.1113/jphysiol.1968.sp008455

[HZRS14] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: Lecture Notes in Computer Science (2014), 346–361. http://dx.doi.org/10.1007/978-3-319-10578-9_23. – DOI 10.1007/978–3–319–10578–9$_2$3. – $-ISBN9783319105789$

[HZRS15] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. 2015

[IKH+11] Izadi, Shahram ; Kim, David ; Hilliges, Otmar ; Molyneaux, David ; Newcombe, Richard ; Kohli, Pushmeet ; Shotton, Jamie ; Hodges, Steve ; Freeman, Dustin ; Davison, Andrew ; Fitzgibbon, Andrew: KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology. New York, NY, USA : Association for Computing Machinery, 2011 (UIST '11). – ISBN 9781450307161, 559–568

[Joc21] Jocher, Glenn: yolov5. `https://github.com/ultralytics/yolov5`, 2021

[JSB+21] Jocher, Glenn ; Stoken, Alex ; Borovec, Jirka ; NanoCode012 ; ChristopherSTAN ; Changyu, Liu ; Laughing ; tkianai ; yxNONG ; Hogan, Adam ; lorenzomammana ; AlexWang1900 ; Chaurasia, Ayush ; Diaconu, Laurentiu ; Marc ; wanghaoyang0106 ; ml5ah ; Doug ; Durgesh ; Ingham, Francisco ; Frederik ; Guilhen ; Colmagro, Adrien ; Ye, Hu ; Jacobsolawetz ; Poznanski, Jake ; Fang, Jiacong ; Kim, Junghoon ; Doan, Khiem ; Yu, Lijun: ultralytics/yolov5: v4.0 - nn.SiLU() activations, Weights & Biases logging, PyTorch Hub integration. `https://doi.org/10.5281/zenodo.4418161`

[Kim17] Kim, Hanme: Real-time visual SLAM with an event camera. In: Real-time visual SLAM with an event camera, 2017

[KSGK19] Krämmer, Annkathrin ; Schöller, Christoph ; Gulati, Dhiraj ; Knoll, Alois C.: Providentia - A Large Scale Sensing System for the Assistance of Autonomous Vehicles. In: CoRR abs/1906.06789 (2019). `http://arxiv.org/abs/1906.06789`

[KWM+19] Kisantal, Mate ; Wojna, Zbigniew ; Murawski, Jakub ; Naruniec, Jacek ; Cho, Kyunghyun: Augmentation for small object detection. 2019

[LAE+16] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott ; Fu, Cheng-Yang ; Berg, Alexander C.: SSD: Single Shot MultiBox Detector. In: Lecture Notes in Computer Science (2016), 21–37. `http://dx.doi.org/10.1007/978-3-319-46448-0_2`. – DOI 10.1007/978–3–319–46448–0_2. – –$ISBN$9783319464480

[LBBH98] Lecun, Y. ; Bottou, L. ; Bengio, Y. ; Haffner, P.: Gradient-based learning applied to document recognition. In: Proceedings of the IEEE 86 (1998), Nr. 11, S. 2278–2324. `http://dx.doi.org/10.1109/5.726791`. – DOI 10.1109/5.726791

[LCSK20] Liu, X. ; Chen, G. ; Sun, X. ; Knoll, A.: Ground Moving Vehicle Detection and Movement Tracking Based on the Neuromorphic Vision Sensor. In: IEEE Internet of Things Journal 7 (2020), Sep., Nr. 9, S. 9026–9039. http://dx.doi.org/10.1109/JIOT.2020.3001167. – DOI 10.1109/JIOT.2020.3001167. – ISSN 2327–4662

[LD19] Law, Hei ; Deng, Jia: CornerNet: Detecting Objects as Paired Keypoints. 2019

[LDG+17] Lin, Tsung-Yi ; Dollár, Piotr ; Girshick, Ross ; He, Kaiming ; Hariharan, Bharath ; Belongie, Serge: Feature Pyramid Networks for Object Detection. 2017

[LGG+18] Lin, Tsung-Yi ; Goyal, Priya ; Girshick, Ross ; He, Kaiming ; Dollár, Piotr: Focal Loss for Dense Object Detection. 2018

[LHW18] Liu, Songtao ; Huang, Di ; Wang, Yunhong: Receptive Field Block Net for Accurate and Fast Object Detection. 2018

[LHW19] Liu, Songtao ; Huang, Di ; Wang, Yunhong: Learning Spatial Fusion for Single-Shot Object Detection. 2019

[LQQ+18] Liu, Shu ; Qi, Lu ; Qin, Haifang ; Shi, Jianping ; Jia, Jiaya: Path Aggregation Network for Instance Segmentation. 2018

[LSP+20] Lee, Chankyu ; Sarwar, Syed ; Panda, Priyadarshini ; Srinivasan, Gopalakrishnan ; Roy, Kaushik: Enabling Spike-Based Backpropagation for Training Deep Neural Network Architectures. In: Frontiers in Neuroscience 14 (2020), 02, S. 119. http://dx.doi.org/10.3389/fnins.2020.00119. – DOI 10.3389/fnins.2020.00119

[MFPA18] Mitrokhin, Anton ; Fermuller, Cornelia ; Parameshwara, Chethan ; Aloimonos, Yiannis: Event-Based Moving Object Detection and Tracking. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2018), Oct. http://dx.doi.org/10.1109/iros.2018.8593805. – DOI 10.1109/iros.2018.8593805. ISBN 9781538680940

[Mis20] Misra, Diganta: Mish: A Self Regularized Non-Monotonic Activation Function. 2020

[MLG+18] Maqueda, Ana I. ; Loquercio, Antonio ; Gallego, Guillermo ; García, Narciso ; Scaramuzza, Davide: Event-based Vision meets Deep Learning on Steering Prediction for Self-driving Cars. In: CoRR abs/1804.01310 (2018). http://arxiv.org/abs/1804.01310

[MMMK03] Matsugu, Masakazu ; Mori, Katsuhiko ; Mitari, Yusuke ; Kaneda, Yuji: Subject independent facial expression recognition with robust face detection using a convolutional neural network. In: Neural Networks 16 (2003), Nr. 5, 555-559. http://dx.doi.org/https://doi.org/10.1016/S0893-6080(03)00115-1. – DOI https://doi.org/10.1016/S0893–6080(03)00115–1. – ISSN 0893–6080. – Advances in Neural Networks Research: IJCNN '03

[MP43] McCulloch, Warren S. ; Pitts, Walter: A logical calculus of the ideas immanent in nervous activity. In: The bulletin of mathematical biophysics 5 (1943), Dec, Nr. 4, 115-133. http://dx.doi.org/10.1007/BF02478259. – DOI 10.1007/BF02478259. – ISSN 1522–9602

[MVGL10] Marín, Javier ; Vázquez, David ; Gerónimo, David ; López, Antonio M.: Learning appearance in virtual scenarios for pedestrian detection. In: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, S. 137–144

[MZC+18] Mo, Kaichun ; Zhu, Shilin ; Chang, Angel X. ; Yi, Li ; Tripathi, Subarna ; Guibas, Leonidas J. ; Su, Hao: PartNet: A Large-scale Benchmark for Fine-grained and Hierarchical Part-level 3D Object Understanding. In: CoRR abs/1812.02713 (2018). http://arxiv.org/abs/1812.02713

[Nik19] Nikolenko, Sergey I.: Synthetic Data for Deep Learning. 2019

[OBZ+18] Olson, Elizabeth A. ; Barbalata, Corina ; Zhang, Junming ; Skinner, Katherine A. ; Johnson-Roberson, Matthew: Synthetic Data Generation for Deep Learning of Underwater Disparity Estimation. In: OCEANS 2018 MTS/IEEE Charleston, 2018, S. 1–6

[Pag20] Paganini, Michela: PYTORCH PRUNING TUTORIAL. https://pytorch.org/tutorials/intermediate/pruning_tutorial.html. Version: 2020

[PMM+12] Peris, Martin ; Martull, Sara ; Maki, Atsuto ; Ohkawa, Yasuhiro ; Fukui, Kazuhiro: Towards a simulation driven stereo vision system. In: Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), 2012, S. 1038–1042

[PP21] Pehle, Christian ; Pedersen, Jens E.: Norse - A deep learning library for spiking neural networks. http://dx.doi.org/10.5281/zenodo.4422025. Version: Januar 2021. – Documentation: https://norse.ai/docs/

[RDGF16] Redmon, Joseph ; Divvala, Santosh ; Girshick, Ross ; Farhadi, Ali: You Only Look Once: Unified, Real-Time Object Detection. 2016

[RFC20] Renda, Alex ; Frankle, Jonathan ; Carbin, Michael: Comparing Rewinding and Fine-tuning in Neural Network Pruning. 2020

[RGS18] Rebecq, Henri ; Gehrig, Daniel ; Scaramuzza, Davide: ESIM: an Open Event Camera Simulator. In: Billard, Aude (Hrsg.) ; Dragan, Anca (Hrsg.) ; Peters, Jan (Hrsg.) ; Morimoto, Jun (Hrsg.): Proceedings of The 2nd Conference on Robot Learning Bd. 87, PMLR, 29–31 Oct 2018 (Proceedings of Machine Learning Research), 969–982

[RHGS15] Ren, Shaoqing ; He, Kaiming ; Girshick, Ross ; Sun, Jian: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C. (Hrsg.) ; Lawrence, N. (Hrsg.) ; Lee, D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Garnett, R. (Hrsg.): Advances in Neural Information Processing Systems Bd. 28, Curran Associates, Inc., 2015

[RKP19] Rashwan, Abdullah ; Kalra, Agastya ; Poupart, Pascal: Matrix Nets: A New Deep Architecture for Object Detection. 2019

[RLN+20] Ridnik, Tal ; Lawen, Hussam ; Noy, Asaf ; Baruch, Emanuel B. ; Sharir, Gilad ; Friedman, Itamar: TResNet: High Performance GPU-Dedicated Architecture. 2020

[Ros58] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. In: Psychological Review 65 (1958), Nr. 6, 386–408. http://dx.doi.org/10.1037/h0042519. – DOI 10.1037/h0042519. – ISSN 0033–295X

[RZL17] Ramachandran, Prajit ; Zoph, Barret ; Le, Quoc V.: Searching for Activation Functions. 2017

[Sal18] Salaj, Darjan: Spike-based LSTM-like Modules in Neural Networks. 2018

[SAS+18] Saleh, Fatemeh S. ; Aliakbarian, Mohammad S. ; Salzmann, Mathieu ; Petersson, Lars ; Alvarez, Jose M.: Effective Use of Synthetic Data for Urban Scene Semantic Segmentation. In: CoRR abs/1807.06132 (2018). http://arxiv.org/abs/1807.06132

[SBM06] Sun, Zehang ; Bebis, G. ; Miller, R.: On-road vehicle detection: a review. In: IEEE Transactions on Pattern Analysis and Machine Intelligence 28 (2006), Nr. 5, S. 694–711. http://dx.doi.org/10.1109/TPAMI.2006.104. – DOI 10.1109/TPAMI.2006.104

[Sch14] Schmidhuber, Jürgen: Deep Learning in Neural Networks: An Overview. In: CoRR abs/1404.7828 (2014). http://arxiv.org/abs/1404.7828

[SFR12] Sung, Cynthia ; Feldman, Dan ; Rus, Daniela: Trajectory clustering for motion prediction. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems IEEE, 2012, S. 1547–1552

[SMB10] Scherer, Dominik ; Müller, Andreas ; Behnke, Sven: Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In: Diamantaras, Konstantinos (Hrsg.) ; Duch, Wlodek (Hrsg.) ; Iliadis, Lazaros S. (Hrsg.): Artificial Neural Networks – ICANN 2010. Berlin, Heidelberg : Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–15825–4, S. 92–101

[ST12] Sivaraman, Sayanan ; Trivedi, Mohan M.: Real-time vehicle detection using parts at intersections. In: 2012 15th International IEEE Conference on Intelligent Transportation Systems, 2012, S. 1519–1524

[SZ15] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015

[TL20] Tan, Mingxing ; Le, Quoc V.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 2020

[TPL20] Tan, Mingxing ; Pang, Ruoming ; Le, Quoc V.: EfficientDet: Scalable and Efficient Object Detection. 2020

[TSCH19] Tian, Zhi ; Shen, Chunhua ; Chen, Hao ; He, Tong: FCOS: Fully Convolutional One-Stage Object Detection. 2019

[WLY+19] Wang, Chien-Yao ; Liao, Hong-Yuan M. ; Yeh, I-Hau ; Wu, Yueh-Hua ; Chen, Ping-Yang ; Hsieh, Jun-Wei: CSPNet: A New Backbone that can Enhance Learning Capability of CNN. 2019

[WPLK18] Woo, Sanghyun ; Park, Jongchan ; Lee, Joon-Young ; Kweon, In S.: CBAM: Convolutional Block Attention Module. 2018

[YLH+19] Yang, Ze ; Liu, Shaohui ; Hu, Han ; Wang, Liwei ; Lin, Stephen: RepPoints: Point Set Representation for Object Detection. 2019

[ZCG+19] Zoph, Barret ; Cubuk, Ekin D. ; Ghiasi, Golnaz ; Lin, Tsung-Yi ; Shlens, Jonathon ; Le, Quoc V.: Learning Data Augmentation Strategies for Object Detection. 2019

[ZMN+20] Zhang, Ling ; Mao, Dongwei ; Niu, Jiong ; Wu, QM ; Ji, Yonggang: Continuous tracking of targets for stereoscopic HFSWR based on IMM filtering combined with ELM. In: Remote Sensing 12 (2020), Nr. 2, S. 272

[ZSW+19] Zhao, Qijie ; Sheng, Tao ; Wang, Yongtao ; Tang, Zhi ; Chen, Ying ; Cai, Ling ; Ling, Haibin: M2Det: A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network. 2019