RWTH Aachen University
Software Engineering Group

Comparison of Deep Learning Architectures on Simulated Environments

Seminar Paper

presented by

Bergerbusch, Timo

1st Examiner: Prof. Dr. B. Rumpe

2nd Examiner:

Advisor: Evgeny Kusmenko

The present work was submitted to the Chair of Software Engineering

Aachen, May 30, 2018

The present translation is for your convenience only.
Only the German version is legally binding.

Statutory Declaration in Lieu of an Oath

Last Name, First Name	Matriculation No. (optional)

I hereby declare in lieu of an oath that I have completed the present Bachelor’s thesis/Master’s thesis* entitled

independently and without illegitimate assistance from third parties. I have use no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

The German version has to be signed.

Location/City, Date	Signature

*Please delete as appropriate

Official Notification:

Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whosoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 to 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.

(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

I have read and understood the above official notification:

The German version has to be signed.

City, Date	Signature

Abstract

The topic of autonomous driving using artificial intelligence increases in importance with the overwhelming amount of software usage within vehicles. For that *Convolutional Neural Networks* (CNNs), which try to figure out the importance of special areas of a single picture, have been shown to be promising.
In this paper we will give a general introduction to the topic of CNNs. We distinguish between the three main *deep learning languages* (DLLs) currently used and researched for autonomous driving agents: mediated perception, behaviour reflex and direct perception. Further we will compare different languages, which can be used to implement the different DLLs, based on the factors of usability, scope of functionality and the integration on a subject.
As a proof of concept we will train a CNN using the language *CNNArch* on the famous KITTI dataset in order to create a trained model. This model will then be tested on a test set created using either the simulation tool `MontiSim` or the open source racing game TORCS, containing multiple different challenging scenarios the agent has to manage.
Finally we evaluate the trained model on it's performance and try to reason, why it performed particularly good/bad, and give an overview based on the implemented test in order to state the similarities and differences of the languages.

Contents

Chapter 1

Preliminaries

The field of autonomous driving agents has rapidly increased in modern car manufacturing. Current research topic rises the agents from parking or lane keeping assistant fully autonomous driving agents obeying the traffic rules and having the ability to react to the volatile environment in a reasonable way.

For that machine learning techniques have proven themselves as an essential part. But in order to fulfil the security standards and create a sophisticated agent it has to be trained on hundred-thousands of scenarios each having a large set of data attached, for example sensor and camera data. The approach of *Convolutional Neural Networks* (CNNs) have been proven to be powerful enough to handle such many training iterations with a huge number of input variables, while maintaining the large learning capacity. [KSH12]

A CNN, as explained in Section 1.2, has a general structure, but can be altered to fit into the approach in various ways influencing the result. Therefore we introduce in **??** the three main approaches of using a CNN.

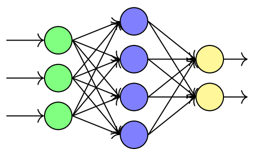
Further in Chapter 2 we see that there is the need of specific languages for the design and implementation of such agents. Two languages will be discussed and compared based on their suitability regarding the AlexNet, stated in Section 1.3.

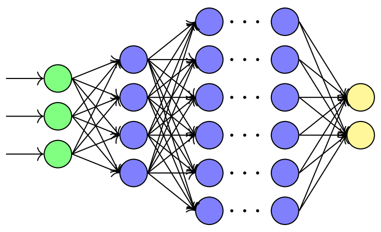
1.1 Neural Networks

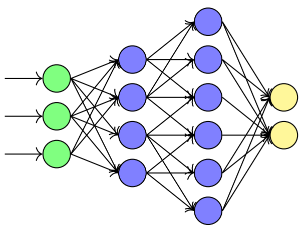
The neural networks are a construct adapted from biological processes. The general construct is very simple, but the expressiveness is very high but still not fully researched. A neural net is made out of neurons and has one very basic function: It takes a fixed number $n \in \mathbb{N}$ of the incoming values x_i, where the vector $(x_0, \dots x_n)^T$ is called a tensor, and multiplies them each with a specific weight w_i, where $0 \leq i \leq n$. Also every neuron contains a bias b, which is a general value subtracted from the sum, so $\sum_{i=0}^{n}(x_i \cdot w_i) - b$.

Often one applies an activation function to fix the value between 0 and 1. Such a function would be for example the sigmoid function or the ReLu, which are both non-linear functions. Without using a non-linear one gets restricted to linear regression and therefore reducing the ability to model more complex functions. This non-linear normalized value gets forwarded to the neurons of the next layer.

Those neurons are ordered in different groups often called layers, as seen in Figure 1.1a.


(a)


(b)


(c)

1. input layer (green):
   This layer gets fed with the input values of the problem, which can be for example sensor data or pixel color values.

2. hidden layer (blue):
   The hidden layer consists of neurons receiving the values from the previous layer, while not being obliged to have the same number of neurons (c.f. Figure 1.1a). Different hidden layer architectures can be distinguished to be deep (c.f. Figure 1.1b). This means, that there are multiple layers of neurons within the hidden layer itself. Also a variation within the hidden layer is the possibility of fully connectivity (c.f. Figure 1.1c). Thus some neurons don't forward their value to every neuron of the next layer.
   There is no rule of how to construct the best hidden layer, considering number of sub-layers, neurons per layer or the connectivity.

3. output layer (yellow): The output neurons contain the value the neural network produces. Depending on the neural networks purpose it can be for example a confidence value of a classification, like recognizing a stop sign, or the value of changing the steering wheel angle.

In order to train a neural network one has to define the behaviour the neural network should have. In an image classification example one should know what the correct class of a given image of a sign is, i.e. a speed limit sign.
A neural network can then be trained by giving it values for the input layer and comparing the values of the output layer with the solutions it should have resulted in. The difference can then be checked. Such a difference can be simply `true/false` or a value indicating how big the difference is. In the example of signs a classification of a "speed limit 70"-sign as "speed limit 50"-sign is still wrong, but as bad as a classification as a "stop"-sign.
Using this difference value the neural network can use linear algebra algorithms to adjust the weights w_i and biases b to improve the output iteratively.

Further information about the underlying training algorithms like gradient descend, newtons method, conjugate gradient or Levenberg-Marquardt algorithm is not given here in order to keep the paper in a justifiable length.

1.2 Convolutional Neural Network (CNN)

A CNN is a special class of deep feed-forward neural networks. On of the main design goals of a CNN is that they require a minimal amount of preprocessing. This is an important aspect, because they are often fed with images. Preprocessing high resolution images is very costly in terms of computational time. In the context of autonomous driving the time is even more crucial, since the driving agent needs to be able to react to spontaneous events.

Like most parts of neural networks, also the CNNs are inspired by biological processes. It is mainly based on the connectivity pattern of an animals visual cortex, where special neurons respond only to stimuli of their receptive field, represented as rectangles lying in the image. Partially overlapping guarantees a complete coverage of the field of view. Those rectangles are often called kernel or filters.[MMMK03]

The partitioning into those rectangles can also has the advantage, that the size of the input image is not relevant. If there would be a direct correspondence of a pixel to one input value then a change of the size would infer null values or additional input values, where the weights are not directly well suited. But with partitioning it into sub-rectangles of the image the values can be unified by selecting these rectangles relative to the size. This is only given, if there are no fully connected layers. Otherwise one has to perform other steps like cropping, scaling or padding.

These separation into those receptive fields has also the advantage that is reduces the effort to train a CNN. The weights and biases of neurons of each receptive field are equal. This is reasonable since for example a speed limit road sign should be identified independent whether it is located next to the road, like on a normal road, or above the road, like on an highway. [L$^+$15]

Further CNNs make strong and mostly correct assumptions about the nature of images, like stationary of statistics and locality of pixel dependencies. This leads to fewer connections and parameters, compared to a normal feed-forward neural net with similar sized layers, and therefore reduces the time it takes to be trained, while being only slightly worse in their best-performance. [KSH12]

TODO: mention pooling

1.3 AlexNet

The *AlexNet* is one of the best performing CNN architectures currently known. It is trained on the ImageNet subsets of `ILSVRC-2010` and `ILSVRC-2012`[1] and became famous because of its result being way ahead of all other competitors.

A highly optimized GPU implementation of this architecture combined with innovative features is publicly available. Those features lead to improve performance and reduce training time.[KSH12]

An important note is that the original test is dated back to 2012 and therefore was used with an overall GPU memory of 6GB, with which training took abound six days. With modern hardware like new GPUs, SLI usage or even clusters, the training can be done faster, or the model can be trained with more data to improve performance. The improvement cased only by hardware can be roughly grasped through [SCE$^+$17].

[1]Further information: http://www.image-net.org/challenges/LSVRC/

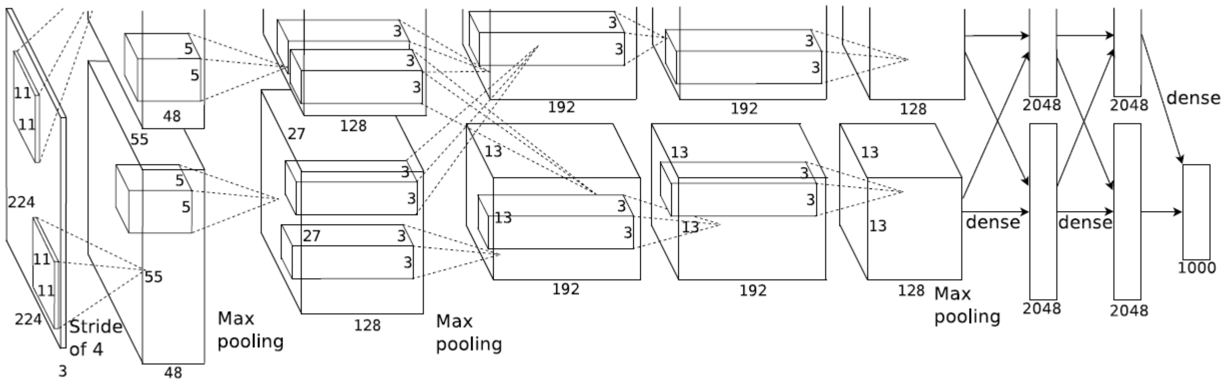


Figure 1.2: The AlexNet-architecture for two GPUs. It consists of 5 convolutional layers (at the beginning) and three fully-connected layers (at the end). The possibly multiple GPUs only communicate between two layers, but never within a layer.[KSH12]

Chapter 2

Deep Learning Languages

Constructing a CNN from scratch in any typical language like Java, C++, or Python is very elaborately and has a high error potential. Even libraries in any such language often encounter the problem of over-complication due to their own style and syntactical and semantic architecture. Therefore there is a need of specialized languages.

The Deep Learning Languages (DLLs) are part of the Domain Specific Languages (DSL). Their main goal is to provide an easy to understand, as less verbose but as expressive as possible way of describing a CNN with its different layers and connections. Also one wants to have simple build pre-sets.

For that we consider three deep learning languages and analyse them on the previously mentioned properties.

2.1 CNNArchLang

(The whole description is based on[TvWH17] and especially [Tim18])

One language for modeling CNNs is CNNArchLang. This language is developed at the Chair of Software Engineering, especially Thomas Michael Timmermanns, at the RWTH Aachen University and part of the MontiCar language family. The main purpose of its creation is the necessity of special properties not given by other CNN-languages: *C&C integration* and *type-safe component interface*. Its basic structure is very similar to python to improve understanding, based on familiarity with Python, and have an equal non-typed syntax.

One very huge advantage of CNNArchLang is that it's designed to be very simplistic and have less verbose than most other languages to model CNNs. It does so, by moving from defining a CNN by every single neuron to the definition via layers only. For that specific purpose many layers are already defined (c.f. Section 2.1.2). New layers can be constructed by combining predefined layers.
This slightly reduces the expressiveness, since the possibility of performing computations on single tensors is lost. Such low-level operations are used extremely rarely and are not a drastic disadvantage.

In contrast to other languages for deep learning CNNArchLang does not denote the connections of layers directly, but tries to model the data flow through the network. For that specific task it contains two main operators:

->: Serial Connection:
This orders two elements sequentially. This means it denotes the first elements output as the second elements input.

|: Parallelization:
This allows the split of the network into separate data streams, which can be computed in parallel.

Since serial connection has the higher precedence one has to use brackets. Also to merge the splitted streams, created by |, one can use the operators: `Concatenate()`, `Add()` and `Get(index)`.

2.1.1 General Definitions

The general definitions of a CNN, which are the input, i.e. an image, maybe additional data in a specific file type, i.e. sensor data, and the output dimension, denoting the predictions or in our example the actions the car should perform. Those are the only typed values within the CNN model.

Such definitions can be modeled in CNNArchLang as presented in Figure 2.1.

```
def input Z(0:255)^{3, h, w} image[2]
def input Q(-oo:+oo)^{10} additionalData
def output Q(0:1)^{3} predictions
```

Figure 2.1: A general definition of a CNN using CNNArchLang

Further analyzed the definition can be broken down to the following components:

- Keyword: `def`
  Every input and output can be introduced using the keyword `def`
- Direction: `input/output`
  Every definition, being a part of the 2.1.1, has to be defined to be either an `input` or an `output`
- Range of numbers:
  One can define the input to have special constraints. For example only integer values are denoted by a `Z` representing $\mathbb{Z}$. The same for `Q` and $\mathbb{Q}$.
  Also the range has to be given via `(x:y)`, where `x` and `y` either are numbers or `-oo`(or `oo`) to denote ∞.
- Size:
  The size of for example the input or the number of classes is denoted by a matrix like notation using `^{size}`. For the input image (line 1) the size `^{3,h,w}` determines the input image to have 3 channels with an image width of `w` and image height of `h`. The others are just defined as 1×10 or 1×3 vectors/tensors.
- Naming:
  At the end of the line there has to be a name to identify the corresponding input/output.
  Also through the `[2]` behind the name `image` one can define it to be a fixed length array of images.

2.1.2 Predefined Layers and Functions

Different CNNs often use a similar basic set of layers, but arranging them differently. For that purpose there are some layers already defined by CNNArchLang to simplify the usage. There is for example the `FullyConnected`-layer with parameters for the number of units within and whether they should use a bias value (c.f. Section 1.1), the `Convolutional`-2D-layer with parameters for the kernel (rectangle) size, number of filters, the stride defining the distance of two rectangles, padding and the usage of biases. Further information on any of these parameters or other predefined layers can be found in [TvWH17].

Also there are already defined functions like the `Sigmoid`, `Softmax`, `Tanh` or `ReLu`. One important aspect is that every argument has to be named.

One important aspect is that CNNArchLang is not a framework itself. It is used to create the code to function in the MxNet (see Section 2.4).

2.2 Caffe

Caffe is a full deep learning framework, created by Yangqing Jia during hos PhD at UC Berkeley. It is a framework specially build to deal with multimedia input formats using state-of-the-art deep learning techniques. It comes as a BSD-license C++ library offering python and MATLAB bindings. One of its reasons why it's so well known and frequently used is because of its design based on expressiveness, speed and modularity.

Using Nvidias Deep Neural Network library cuDNN as a wrapper of the CUDA functionality, Caffe can use the GPU in order to process even faster and learn in a rate of 40 million pictures per day. The possibility of using multiple GPUs in SLI is not stated and therefore not taken into account. [Wik18] However the possibility of using a cloud system is mentioned. But whether it is a simple decentralization or the possibility to train the network using the combined power of multiple computers to train is not mentioned. [JSD+14]

Caffe tries to improve its readability by abstracting from the actual implementation using a graph-oriented way of network defining. For that Caffe uses two elements to represent the network:

- **Blob**:
  A 4-dimensional array storing data, like images, parameters or parameter updates. These blob are the communication between layers.
- **Layers**:
  A Layer as described in Section 1.1 and Section 1.2.

The whole model gets saved as a Google Protocol Buffer, which is a language-neutral structuration, with important key features like size-minimal and efficient serialization and efficient interfaces for C++ and Python. [JSD+14] [Var08] An example of a graph and its corresponding Protocol Buffer representation, written in prototxt, are given in Figure 2.2.

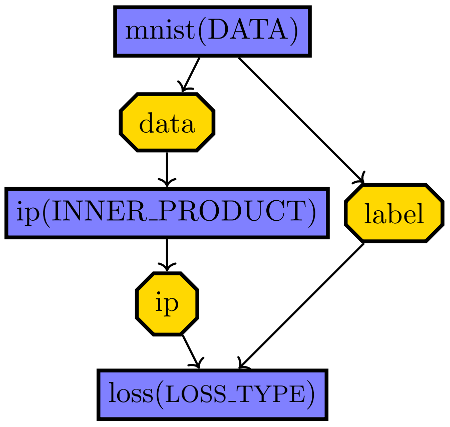
The biggest advantage of Caffe is the huge community providing a large set of presets, like layers or pre-trained nets and also a huge forum to ask other users about problems

```
name: "loss-net"
layers {
  name: "mnist"
  type: DATA
  top: "data"
  top: "label"
  data_param { ### }
}

layers {
  name: "ip"
  type: CONVOLUTIONAL
  bottom: "data"
  top: "conv"
  convolutional_param { ### }
}

layers {
  name: "loss"
  type: SOFTMAX_LOSS
  bottom: "ip"
  bottom: "label"
  top: "loss"
}
```

(a)



(b)

Figure 2.2: An example of a Softmax loss network. blue boxes are the different layers and the yellow boxes are the blobs. Note that in (a) the parameters are not mentioned since they don't add value for this example. There are things defined like kernel size, image scaling or image origin.

regarding ones project. This is such an advantage, because every module it is guaranteed to pass a coverage test. [JSD$^+$14]

Caffe is written using as plaintext named prototxt and therefore can be run via command line. One problem mentioned with that is that even though there are many nets already defined the creation of new nets often is highly verbose and repetitive. There are no shortenings. For example [Tim18] mentioned an example net written in CNNArchLang with 36 lines and in Caffe with 6700 lines. This is also due to the fact that even if a layer can be constructed as a composition of existing layers one often has to define the forward-, backward and gradient-updates.

2.3 Caffe2

The framework Caffe2 is the successor of Caffe. Caffe2 is developed by Facebook and its current main usage is the phrase-wise translation in social networks. Keeping the modulartiy of Caffe in mind Caffe2 is also designed so that it can be up-scaled as well as mobile deployed. Also Caffe2 is designed in such a fashion that it can easily adapt to drastic changes like quantized computing. [Caf18]

Since the whole architecture is rewritten from scratch and regarding its now roughly one year of existence the library performs relatively well, but does not have the impact to outperform Caffe [Hei17]. One upside of the rewriting is that Caffe2 has a Python binding, like most other frameworks. One downside of the rewriting is that huge parts of the framework are not sufficiently documented. Caffe2 tries to improve but still has large whole within the documentation. [Tim18]

Caffe2 offers programs, which allow the user to convert Caffe and PyTorch models to Caffe2. This makes the switching to Caffe2 much easier, since the users don't have to rewrite their models. [Caf18]

The main difference between Caffe and Caffe2 in terms of designing a neural network is that in Caffe2 the user uses `Operators` as the basic units instead of layers. Even though they are similar to the layers of Caffe, they are more flexible and adaptable. Partly based on the popularity of Caffe, Caffe2 also has a huge community and a large set of preset `Operators`, which can be used. [Caf18]

2.4 MxNet

An other framework often mentioned in research is the MxNet. This deep learning framework is part of the Apache Software Foundation. Also it is said to be "Amazons deep learning framework of choice" [Yeg16] and featured to be a preset on the Amazon Web Services (AWS). [CL]

The MxNet tries to combine the advantages of imperative frameworks like numpy or MatLab with the advantages of declarative frameworks like Caffe, Caffe2 or Tensorflow.

The advantages of imperative and declarative approaches can best be understood using an example. Let the example be to compute: $a = b + c$

imperative:

Procedure: check the ability of b and c to be added. If so strictly compute the sum and declare a as the same type as b and c

Advantage: very straightforward, works well with typical structures, debugger and third party libraries

Usefull for: natural parameter updates and interactive debugging

declarative:

Procedure: compute the computation graph and store the values of b and c as data bindings

Advantage: perform computation as late as possible, leading to good optimization possibilities

Usefull for: specifying computation structure, optimization

By combining both approaches MxNet is able to provide a superset programming interface compared to Caffe. [CLL+15]

Also MxNet is able to reduce memory to a minimum by performing everything they can in-place and free or reuse as fast as possible. Thus the memory usage of MxNet is outperforming Caffe. [CLL+15] A new benchmark with Caffe2 has not been done yet.

An other very big upside of MxNet is the possibility to use not only multiple GPUs in an SLI connection, but also multiple computers or even server to train a neural network simultaneously. This results in an outstanding scalability. [CLL+15]

Similar to Caffe2, MxNet also allows the deployment of trained models to low-end devices using Amalgamation (c.f. [MxNa]) or the AWS.

Due to the fact that MxNet has found its way into Apache Incubator and therefore it is an Open-Source project, the creation of additional functions and nets is quite simple and is not bound to a given preset. Thanks to the community also a variety of nets constructed and some already pre-trained are free to use. [MxNb]

Chapter 3

Available Deep Learning Approaches

There are various approaches of autonomous driving agents making a variety of assumptions and differ in numerous options. But they can be mostly categorized into two major groups of approaches: mediated perception approaches and behavior reflex approaches. [CSKX15]

In this paper we further analyse a suggested third group, called direct perception, which can traced to [Gib79] in the mid 50's, but was sharply criticized by researchers of the other two groups, i.e. in [Ull80].

All these three groups differ in the way of interpreting the given sensor data and whether or not to create a some what bigger picture based on consequent data.

3.1 Mediated Perception

The mediated perception approach is a multi-component continuous process. Every component recognizes specific aspects for driving. For example traffic signs, lanes, other cars. Those components are then combined into one single world state representing the cars surrounding based on the sensor data. [GLSU13]
These world states are 3D models of the current world. Cars are identified using a classifier and then often surrounded by a 3D bounding box. An example can be seen in Figure 3.1. By comparing different frames generated one can estimate the speed and distance to those objects and derive an A.I. based precedence behavior. [GLSU13][CSKX15]

The often stated problems with such approaches are, that computing such a scene is costly in terms of computation time. Some information is irrelevant, redundant or even misleading due to inaccuracy of sensors. To perform a right turn the sensor information of the distance to a car left behind me is irrelevant, but becomes very important when taking a left turn.
Additionally many of the subtasks are still open research topics themselves. For example a reliable lane detection throughout various weather conditions or even a new road not having any drawn lines yet. [Aly08]
Also mediated perception approaches require very detailed information up front, like up-to-date maps.

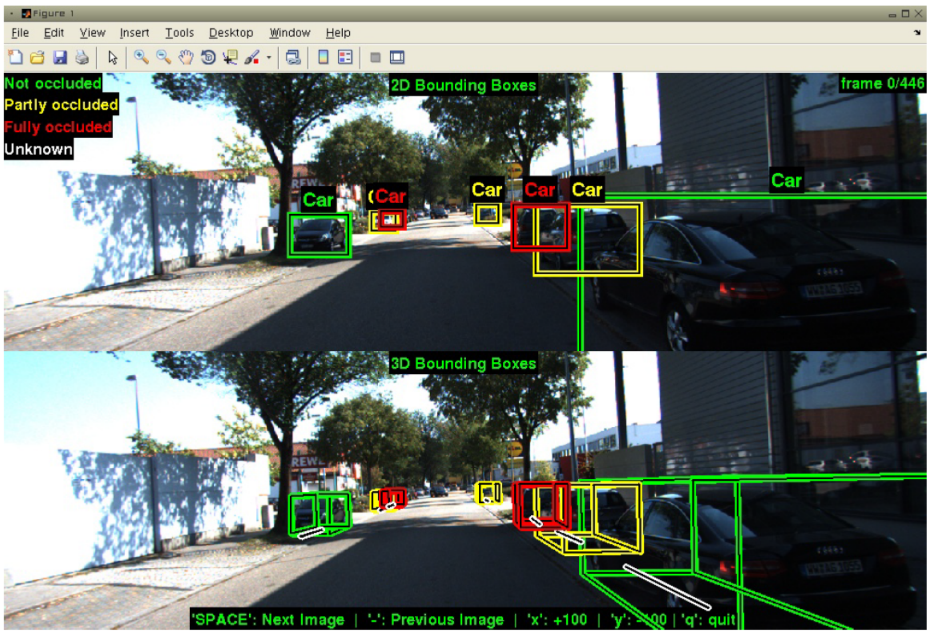


Figure 1
File Edit View Insert Tools Desktop Window Help
Not occluded
Partly occluded
Fully occluded
Unknown
2D Bounding Boxes
frame 0/446
Car
Car
Car
Car
Car
Car
3D Bounding Boxes
'SPACE': Next Image | '-': Previous Image | 'x': +100 | 'y': -100 | 'q': quit

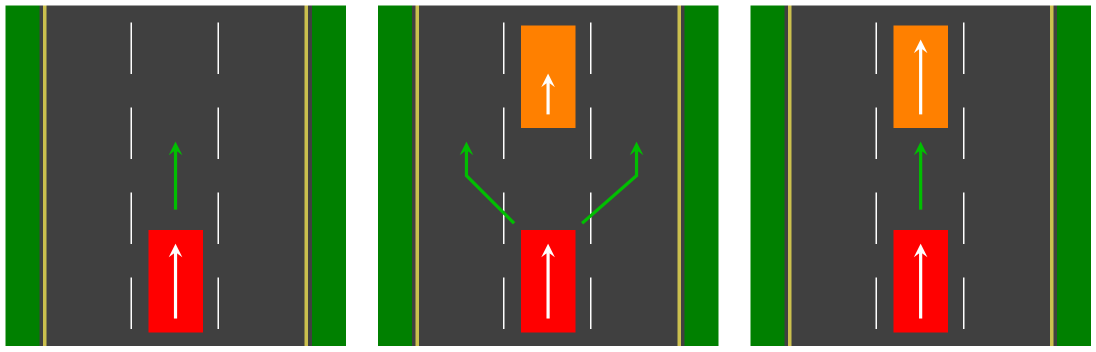
Figure 3.1: An example of a scene using 3D bounding boxes. This image is taken from the MATLAB delvopment kit of [GLSU13]

The approach of mediated perception is a reasonable and very sturdy way of handling such a complex task, but has its drawbacks regarding computational time and additional knowledge.

3.2 Behavior Reflex

The behavior reflex approach of constructing a reliable autonomous driving agent can be dated back to 1989 , where researchers tried to directly map a single frame to a decision of a steering angle. For such approaches a quite simple neural network were created.
The network ALVINN consisted of a single hidden layer, used back-propagation and is fed by two cameras: a 30×32 pixel video and a 8×32 pixel range finder retina. The input neurons fired depending on the blue color band of its pixel, because it is believed to infer the highest contrast between road and non-road. The difference in color of road and non-road was fed back to the network. The bias (activation level) is proportional to the proximity of the corresponding area, based on the likelihood and importance of having road in particular fields of the image.[Pom89]
For example having recognized that the road abruptly ends right in front of the car is more important than recognizing that there is a road in the top left corner.

Such systems, even though they are simple compared to the in **??** mentioned mediated perception approaches, have been proven to have the capability to perform simple tasks. It can elegantly be trained by having a human drive a car with the cameras equipped and forward the images to the neural network and adding the current steering angles as a label.[CSKX15]



(a) free lane (b) slower car in the same lane (c) equal fast car in same lane

Figure 3.2: The 3 scenarios causing problems with behavior reflex approaches. The red block is the agent, the orange block the other car, the white arrows indicate the velocity and the green arrows the logically deduced behaviors.

The problem with behavior reflex approaches is, that they reach their limits very early when adding more complex scenarios. Having simple alternations to the trained scenarios, which enforce a different behavior, is very hard to train to such a neural network.
For example comparing a simple straight 3 lane road with the car in the middle, as sketched in Figure 3.4. The system is confidently able to hold the angle and make small adjustments to stay in the lane (Figure 3.2a). But what if on the same road there is an other car in the middle lane in front of the agent, which is slower? Having quite the same input the system would have to overtake the car left or right (considering an american highway) (Figure 3.2b). Now also considering a car in front, which has the same speed. One can simple stay in the lane (Figure 3.2c). This maneuver is very hard to train to a simple neural network like ALVINN.

3.3 Direct Perception

The direct perception is the third group of approaches, which can be dated back to the 1954 and was initially mainly researched by James J. Gibson. [Gib54] The approach is based on analyzing a picture, not simply deducing a steering angle, or velocity change, like the behavior reflex approaches (cf. **??**), but also performing further computation without parsing it into a 3D world state model like the mediated perception approaches (cf. **??**). [CSKX15]
So it is a third paradigm, which can be interpreted as a hybrid of the two other paradigms. The approach tries to identify only the meaningful affordance indicators and make a decision based on those parameters.

The CNN gets trained to make a reasonable assumption about the distance of other vehicles, their velocity and lane marks. These assumptions are further called affordance indicators.

We further consider a design based on [CSKX15] and their way of training.

The original paper [CSKX15] stated the a total number of 13 indicators separated in two states to be sufficient for their design of a highway only autonomous agent. The states are: in line driving (following the lane) and on line driving (changing lanes). The values

```
always:
  1) angle: angle between the car’s heading and the tangent of the road
"in lane system", when driving in the lane:
  2) toMarking LL: distance to the left lane marking of the left lane
  3) toMarking ML: distance to the left lane marking of the current lane
  4) toMarking MR: distance to the right lane marking of the current lane
  5) toMarking RR: distance to the right lane marking of the right lane
  6) dist LL: distance to the preceding car in the left lane
  7) dist MM: distance to the preceding car in the current lane
  8) dist RR: distance to the preceding car in the right lane
"on marking system", when driving on the lane marking:
  9) toMarking L: distance to the left lane marking
  10) toMarking M: distance to the central lane marking
  11) toMarking R: distance to the right lane marking
  12) dist L: distance to the preceding car in the left lane
  13) dist R: distance to the preceding car in the right lane
```

Figure 3.3: The affordance indicators and their affiliation states

themselves can be categorized as: preceding car distances, distances to the lane markers and the steering angle. The indicators and their affiliation to the states can be seen in Figure 3.3.

Based on the current state, all affordance indicators of the other state are not used, since the other state is defined to be inactive.
In order to identify the current state the host car is in, every state has their respective region, where they are active with an overlapping region for smooth transitioning.

Training is done by feeding the CNN with images of a current driving scenario as input and give estimated values to the mentioned affordance indicators active in the current state. Those values get forwarded to a controller dealing with the car. In the original approach the training was done via the TORCS game further explained in TODO

The controller is setup to take the affordance values and compute a decision from it in an imperative way. For example the steering can be computed as

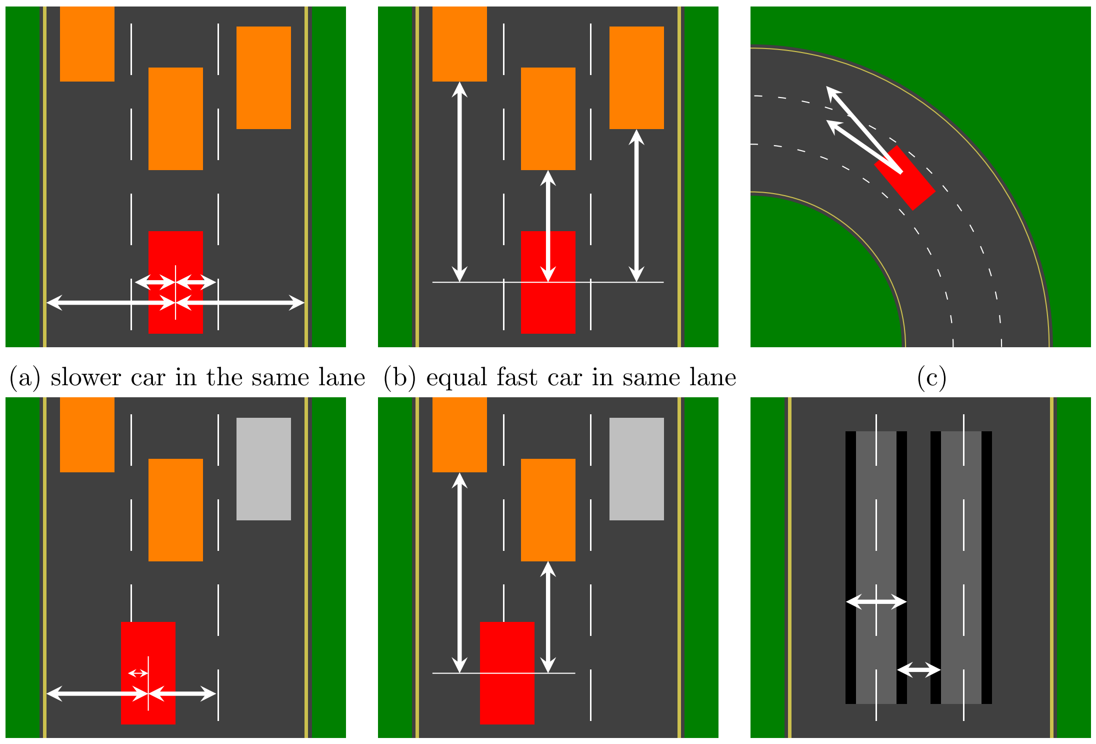
$$steerCmd = C \cdot (angle - \frac{dist\_center}{road\_width})$$

where $dist\_center$ is the center of the current state (line we are on or middle of the lane), a coefficient C suited for the current conditions, like weather or speed, and $angle \in [-\pi, \pi]$ as the affordance indicator.

Also using the controller to compute the accelerating and braking one can adjust the desired speed ($desired\_speed$) based on the affordance indicators. The $desired\_speed$ is set to the speed the agent wants to drive and can drop if a drastic steering motion is considered or accelerate to overtake a slower car.
In a one lane scenario with a preceding car driving slower than the $desired\_speed$ there is no space to overtake. For that the controller has the velocity control car-following model:

$$v(t) = v_{max}(1 - e^{-\frac{c \cdot dist(t)}{v_{max}} - d})$$

where v_{max} is the maximal speed the agent is allowed to drive, c and d are coefficients specific to external conditions, like a wet lane or the cars potential of accelerating and


(a) slower car in the same lane (b) equal fast car in same lane (c)

(d) free lane (e) slower car in the same lane (f) equal fast car in same lane

Figure 3.4: The 3 scenarios causing problems with behavior reflex approaches. The red block is the agent, the orange block the other car, the white arrows indicate the velocity and the green arrows the logically deduced behaviors. The names of the indicators is given in Figure 3.3

TODO: name the arrows

---

braking, and $dist(t)$ is the distance to the preceding car. This distance is given through the trained CNN.

TODO:

1. show graphic of how controller computes

Chapter 4

Comparison: CNNArchLang & Caffe

In this chapter we want to compare the AlexNet (c.f. Section 1.3) implemented using CNNArchLang and Caffe. For both we take an in depth look at the predefined architecture by their respective language. We do that, since both implementations were done by language experts and build up to be as efficient and precise in the language as possible.

Further we state the currently most used training databases and possible other ways to train the CNNs.

TODO: more explanation

4.1 Implementation

4.1.1 Caffe

The implementation of the AlexNet using Caffe is given partly in Figure 7.1. The whole net has a total number of 284 lines. Obviously the code written very verbosely. Every layer has to be explicitly specified even if they have a very similar structure to a previous layer.

For example comparing the pooling layer “pool1” from line 51 to 61 and the pooling layer “pool2” from line 99 to 109:

```
                ⋮
51 layer {
52   name: "pool1"
53   type: "Pooling"
54   bottom: "norm1"
55   top: "pool1"
56   pooling_param {
57     pool: MAX
58     kernel_size: 3
59     stride: 2
60   }
61 }
                ⋮
```

(a) “pool1”

```
                 ⋮
 99 layer {
100   name: "pool2"
101   type: "Pooling"
102   bottom: "norm2"
103   top: "pool2"
104   pooling_param {
105     pool: MAX
106     kernel_size: 3
107     stride: 2
108   }
109 }
                 ⋮
```

(b) “pool2”

Figure 4.1

The lines first 3 and last 6 lines are completely similar. The other 2 lines are just different in the name of the incoming and outgoing connections. Creating a huge and deep net would lead to a enormously large description file.

4.1.2 CNNArchLang

The implementation of the AlexNet using CNNArchLang can be seen in Figure 7.2. The complete script has 43 lines and defines the same net construction as the 284 line definition in Caffe. This shows the efficient language design used in the creation of CNNArchLang (c.f. Section 2.1).
The two pooling operations of Figure 4.1 can be located in line 32 using the python like syntax of definition and the sequential connection -> (c.f. Section 2.1).

Using those techniques CNNArchLang is able to write even complex neural network using few lines of code. This and the syntax itself create an easy to read program.

4.2 Training

In order to train a CNN, independent of the underlying approach (c.f. Chapter 2), one has obtain a huge database of input images and the labels, i.e. actions the CNN should have computed. For autonomous driving the training has to be very intensively done. Otherwise the car driven by the agent will take damage by just slight changes of the circumstances.
Also different scenarios have to be trained. Only training the driving on a road without others cars and simply following the lane is a very disparate task than overtaking a slower car.
For that the following sources of such databases are currently state-of-the-art.

4.2.1 KITTI Dataset

The KITTI dataset is a 6 hour recoding by the Karlsruhe Institute of Technology. They mounted various cameras and laser scanners on a VW Passat and drove around the german city Karlsruhe.
During those hours they collected a total amount of 180 GB of data. This data includes images, in different channels,from the drivers point of view, sensor data of distances, steering angle, acceleration/braking, current speed, GPS coordinates and others.
While other test sets are often developed using a very specific setup for a corresponding approach, the KITTI dataset has such a high variety of data captured that it is used by many approaches. It has become one of the default datasets to compare different approaches on. [GLSU13]

4.2.2 TORCS

A game called **T**he **O**pen **R**acing **C**ar **S**imulator or short TORCS is a racing game specially designed for artificial intelligence research. It is designed to be modular in order to retrieve every kind of data one needs for their approach. Also it offers a documented API to create for example a driving agent.

The possibility to collect any kind of data is what makes this game so popular in current research. The advantage over the KITTI dataset is that if one needs a very specific value not included within the KITTI dataset the approach can not be tested with it. Neglecting if such a value is meaningful in terms of the ability to collect it during real driving, there would be an other specialized set collecting only the data this specific approach needs and maybe not collecting the data one needs to compare it to an other approach.

The disadvantage of the training via TORCS and training via KITTI is that TORCS only uses artificial images, other artificial agents and is completely exempt from any kind of data noise like rain disturbing sensor data or sunlight blending the camera. Also other cars behave different in a game than in real life. The KITTI includes some of those problems. To which extend is arguable. [WEG$^+$00]

4.2.3 MontiSim

TODO

Chapter 5

Evaluation of Direct Perception

In this chapter we take a look back at the direct perception approach and evaluate its performance. For that we compare the, via TORCS (c.f. Section 4.2.2), trained version to a behavior reflex approach (c.f. Section 3.2). In addition to that there is also a comparison of the direct perception approach, trained via KITTI (c.f. Section 4.2.1), to a mediated perception approach stated in [GLW$^+$14]. [CSKX18] Finally we take a look at possible scenarios causing problems which may emerge.

The directed perception as stated in [CSKX15] and discussed in Section 3.3 is designed to handle highway driving tasks, such as driving in a lane, overtaking slower cars, detecting the lane configuration and breaking to avoid a collision.

The behavior reflex approach was able to follow empty lanes perfectly, but was completely unable to have a acceptable behavior considering other cars. Neither a sufficient speed regulation nor the task of staying in lane was observable. The agent left the track various times and had multiple collisions. On the other hand the direct perception approach manages to change lanes smoothly to avoid collisions and stay in lane. Due to the speed regulation in the controller, stated in Section 3.3, the agent is able to perform an emergency brake if necessary. So considering those scenarios the direct perception outperforms the behavior reflex approach.

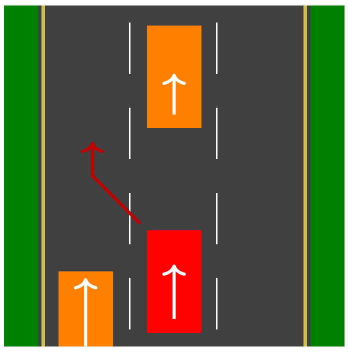


Figure 5.1: A scenario that has to be considered to create a full autonomous driving agent

In order to compare with the state-of-the-art mediated perception approach, the training is done using the KITTI dataset and also combine two CNNs for near and far perception, both using the direct perception approach. It shows that the direct perception approach is able to perform roughly as good, even though they restrict themselves to the cars closest to the host car. So the direct perception is sufficient for real world examples as well. [CSKX18] [CSKX15]

Despite the two mentioned comparisons there are still others that need further investigation. Considering more complex tasks, which mediated perception approaches are able to handle, the direct perception needs to prove itself. Considering classification tasks like road signs, pedestrians detection or traffic light detection including its current light showing still have to be done in order to create a sufficient agent for real-life cars. Also more complex scenarios like busy intersections have to be solved. Also scenarios as sketched in Figure 5.1, where the overtaking can not take place since a car left and possibly a bit behind of the host is even faster. A number of those scenarios can be managed by a sophisticated controller, but this would again take more time and provides less flexibility.

So concluding the direct perception definitely is state-of-the-art and has the potential to found the base of a sophisticated autonomous driving agent, if considered as a predictor of distances they call affordance indicators. But whether or not the direct perception can handle a complete real-life scenario is still up to prove.

Chapter 6

Conclusion

Conclusion of differences and similarities between the frameworks

Property	CNNArchLang	Caffe	Caffe2	MxNet
General Information				
is full framework	✗	✓	✓	✓
SLI usage	–	✗[1]	✗[1]	✓
mult. computers	–	✓	✓	✓
Nets Supported				
typical CNNs	✓	✓	✓	✓
arbitrary CNNs	✗	✓	✓	✓
Recurrent NNs	✗	✓	✓	✓
Constructs				
predefined NNs	✓	✓	✓	✓
pretrained NNs	✗	✓	✗[2]	✓
simple arbitrary net creation	✓	✗	✗	✓
predefined functions	✓	✓	✓	✓
simple function creation	✓	✗	✗[2]	✓
low-level operations	✗	✗	✗	✓
Language bindings				
C++	✗	✓	✓	✓
Python	✓	✗[3]	✓	✓
MatLab	?	✓	✗	✓
Others	?	?	?	R, Go, Julia, Perl JavaScript, Scala

[1] depending on CUDA version/installation

[2] not clearly stated, but also not denied

[3] added lately

Possible Criteria

- generality

- expressiveness
- modularity
- is Framework?
- Installation
- Error Handling
- for equal task?
- low-level computations

Also a general conclusion based on results and [Grz17]

Chapter 7

Appendix

```
name: "AlexNet"
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape: {
      dim: 10
      dim: 3
      dim: 227
      dim: 227
    }
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv1"
  top: "conv1"
}
layer {
  name: "norm1"
  type: "LRN"
  bottom: "conv1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "norm1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "norm2"
  type: "LRN"
  bottom: "conv2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "norm2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv3"
  top: "conv3"
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
          ⋮
```

Figure 7.1: The Caffe implementation of the AlexNet. This is only the first 169 lines. The whole net has a size of 284 lines of code. The main reason of this is the highly verbose Protocol Buffer prototxt style. Further explanation in Section 2.2. [Caf]

```
architecture Alexnet(img_height=224, img_width=224, img_channels=3, classes=10){
    def input Z(0:255)^{img_channels, img_height, img_width} data
    def output Q(0:1)^{classes} predictions

    def split1(i){
        [i] ->
        Convolution(kernel=(5,5), channels=128) ->
        Lrn(nsize=5, alpha=0.0001, beta=0.75) ->
        Pooling(pool_type="max", kernel=(3,3), stride=(2,2), padding="no_loss") ->
        Relu()
    }
    def split2(i){
        [i] ->
        Convolution(kernel=(3,3), channels=192) ->
        Relu() ->
        Convolution(kernel=(3,3), channels=128) ->
        Pooling(pool_type="max", kernel=(3,3), stride=(2,2), padding="no_loss") ->
        Relu()
    }
    def fc(){
        FullyConnected(units=4096) ->
        Relu() ->
        Dropout()
    }

    data ->
    Convolution(kernel=(11,11), channels=96, stride=(4,4), padding="no_loss") ->
    Lrn(nsize=5, alpha=0.0001, beta=0.75) ->
    Pooling(pool_type="max", kernel=(3,3), stride=(2,2), padding="no_loss") ->
    Relu() ->
    Split(n=2) ->
    split1(i=[0|1]) ->
    Concatenate() ->
    Convolution(kernel=(3,3), channels=384) ->
    Relu() ->
    Split(n=2) ->
    split2(i=[0|1]) ->
    Concatenate() ->
    fc(->=2) ->
    FullyConnected(units=10) ->
    Softmax() ->
    predictions
}
```

Figure 7.2: The AlexNet implemented in CNNArchLang. This is the whole program to describe the whole net. Further explanation in Section 2.1. [TvWH17]

Bibliography

[Aly08] Mohamed Aly. Real time detection of lane markers in urban streets. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 7–12. IEEE, 2008.

[Caf] Caffe model zoo - alexnet. `https://github.com/BVLC/caffe/blob/master/models/bvlc_alexnet/deploy.prototxt`. [Online; accessed 30-May-2018].

[Caf18] Caffe2. What is caffe2?, 2018. [Online; accessed 28-May-2018].

[CL] Tianqi Chen and Mu Li. Mxnet: Flexible and efficient library for deep learning.

[CLL$^+$15] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015.

[CSKX15] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 2722–2730. IEEE, 2015.

[CSKX18] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. http://deepdriving.cs.princeton.edu/, 2018.

[Gib54] James J Gibson. A theory of pictorial perception. *Audiovisual communication review*, 2(1):3–23, 1954.

[Gib79] James J Gibson. *The ecological approach to visual perception.* Psychology Press, 1979.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[GLW$^+$14] Andreas Geiger, Martin Lauer, Christian Wojek, Christoph Stiller, and Raquel Urtasun. 3d traffic scene understanding from movable platforms. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):1012–1025, 2014.

[Grz17] Adam Grzywaczewski. Training ai for self-driving vehicles: the challenge of scale. Technical report, Tech. rep., NVIDIA Corporation. URL https://devblogs. nvidia. com/parallelforall/training-self-driving-vehicles-challenge-scale, 2017.

[Hei17] Heise Verlag. Machine learning: Facebook erweitert einsatz von caffe2, 2017. [Online; accessed 25-May-2018].

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[L+15] Yann LeCun et al. Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, page 20, 2015.

[MMMK03] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, and Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*, 16(5-6):555–559, 2003.

[MxNa] Amalgamation: Making the whole system a single file. `https://mxnet.incubator.apache.org/faq/smart_device.html`. [Online; accessed 25-May-2018].

[MxNb] Mxnet model gallery. `https://github.com/dmlc/mxnet-model-gallery`. [Online; accessed 25-May-2018].

[Pom89] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989.

[SCE+17] Vivienne Sze, Yu-Hsin Chen, Joel Einer, Amr Suleiman, and Zhengdong Zhang. Hardware for machine learning: Challenges and opportunities. In *Custom Integrated Circuits Conference (CICC), 2017 IEEE*, pages 1–8. IEEE, 2017.

[Tim18] Thomas Timmermanns. Modelling languages for deep learning based cyber-physical systems. 2018.

[TvWH17] Thomas Timmermann, Michael von Wenckstern, and Malte Heithoff. Cnnarchlang. https://github.com/EmbeddedMontiArc/CNNArchLang, 2017.

[Ull80] Shimon Ullman. Against direct perception. *Behavioral and Brain Sciences*, 3(3):373–381, 1980.

[Var08] Kenton Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*, 72, 2008.

[WEG+00] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 4, 2000.

[Wik18] Wikipedia contributors. Caffe — Wikipedia, the free encyclopedia, 2018. [Online; accessed 11-May-2018].

[Yeg16] Serdar Yegulalp. Why amazon picked mxnet for deep learning. *URL: https://www.infoworld.com/article/3144025/cloud-computing/why-amazon-picked-mxnet-for-deep-learning.html*, 2016.