

Master's Thesis

# Domain Generalization for Vision Perception Models by Camera-Lidar Contrastive Pretraining

Nils Golombiewski

Advised by Dr. Kira Maag and Prof. Dr. Hanno Gottschalk  
with support from Christoph Hümmer and Manuel Schwonberg

August 2024

Technical University Berlin

## Abstract

Autonomous driving relies heavily on accurate scene understanding across diverse and unpredictable environments. A key challenge in this context is *domain generalization*—the ability of vision models to perform reliably under distribution shift, i.e. in scenarios that differ significantly from their training data, such as during adverse weather or lighting conditions or changes in locality. Traditional supervised learning approaches often struggle with this generalization due to the variability in real-world conditions. Current models for autonomous driving may excel in controlled environments, but tend to underperform in challenging scenarios, limiting their robustness and safety. Addressing these limitations is crucial for deploying autonomous systems that can handle the full spectrum of real-world conditions without requiring extensive retraining or additional labeled data.

Multimodal *vision-language models* like CLIP [71] and EVA-CLIP [90] exhibit remarkable zero-shot performance and domain generalization in various computer vision tasks. Hümmer et al. [50] showed that leveraging pretrained CLIP image encoders for autonomous-driving-related tasks can significantly improve domain generalization performance.

This thesis examines the impact of multimodal pretraining on domain generalization from another perspective: We examine the hypothesis that multimodal pretraining to align the representations of data pairs from different modalities can enhance an image encoder model’s domain generalization, provided that the co-modality representation is robust under domain shift. This certainly holds true (for some domain shifts) in the case of 3d lidar point clouds which are completely robust against shifts in lighting conditions and partially robust against shifts in wheather. We therefore explore the effect of multimodal pretraining on pairs of 2D camera images and 3D lidar point clouds, a common data combination in autonomous driving.

To test the hypothesis, we adopt a contrastive learning approach, similar to CLIP, and evaluate our hypothesis as follows. We compare three versions of the same image encoder architecture: all three are initially initialized with pretrained weights from unimodal pretraining for image classification on ImageNet. Two versions then undergo additional contrastive pretraining to align 2D and 3D representations: one with a pretrained model from the literature [82], and the other with our own contrastive pretraining on the A2D2 dataset [29]. All three models are subsequently trained under identical conditions for semantic segmentation on Cityscapes dataset [15] and evaluated—without further training—on the ACDC dataset [78], which allows testing of segmentation performance under adverse lighting and weather conditions (night, fog, rain, and snow) to assess domain generalization.

Our experiments reveal no significant improvement over traditional unimodal pretraining. Possible reasons include: (1) aligning images and lidar point clouds may not provide a sufficiently useful training signal for semantic segmentation, as we observe a decline

in segmentation performance with extended and more effective contrastive pretraining; and (2) the dataset size used for pretraining may be insufficient to yield the desired generalization, aligning with findings of Fang et al. [24] that suggest that data size and diversity, rather than the multimodal approach itself, drives enhanced generalization in CLIP models.

Despite these negative findings, multimodal self-supervised learning remains a promising approach, particularly for autonomous driving, as it has the potential to teach models complex representations of the physical world that are crucial for real-world applications without requiring manually labeled data. Future research could explore larger datasets, alternative modalities, or non-contrastive learning methods to further improve generalization in AI models.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Context and Motivation . . . . .	6
1.2	Overview of the Thesis Structure . . . . .	7
<b>2</b>	<b>Review of basic theory and related works</b>	<b>8</b>
2.1	The Learning Problem . . . . .	8
2.2	Neural Networks Basics . . . . .	10
2.2.1	Feed Forward Neural Networks . . . . .	10
2.2.2	Activation Functions . . . . .	12
2.2.3	Loss Functions . . . . .	15
2.2.4	Universal Approximation Property . . . . .	15
2.3	Training Neural Networks . . . . .	16
2.3.1	Stochastic Gradient Descent . . . . .	17
2.3.2	Backpropagation . . . . .	19
2.3.3	Regularization . . . . .	21
2.4	Deep Learning for Computer Vision . . . . .	22
2.4.1	Images and Point Clouds . . . . .	22
2.4.2	Lidar . . . . .	23
2.4.3	Semantic Segmentation . . . . .	24
2.5	Advanced Neural Network Architectures . . . . .	25
2.5.1	Convolutional Neural Networks . . . . .	26
2.5.2	Transformers . . . . .	28
2.5.3	Vision Transformers (ViTs) . . . . .	31
2.5.4	Domain Generalization . . . . .	31
2.6	Self-supervised Multimodal Learning . . . . .	33
2.6.1	CLIP . . . . .	33
2.6.2	Related Work on Camera-Lidar Contrastive Learning . . . . .	34
<b>3</b>	<b>Method</b>	<b>36</b>
3.1	Motivation . . . . .	36
3.2	An Argument for Multimodality . . . . .	36
3.3	Camera-Lidar Contrastive Learning . . . . .	37
<b>4</b>	<b>Numerical Experiments</b>	<b>39</b>
4.1	Overview . . . . .	39
4.1.1	Architectures . . . . .	39
4.1.2	Datasets . . . . .	40

4.1.3	Metrics . . . . .	41
4.2	Details and Numerical Results . . . . .	41
4.2.1	LIP-Loc and timm-ViT-S . . . . .	41
4.2.2	Finetuning Pipeline . . . . .	42
4.2.3	Contrastive Pretraining . . . . .	44
4.2.4	Pretraining data pipeline . . . . .	44
4.2.5	Pretraining architecture . . . . .	45
4.2.6	Pretraining Results . . . . .	46
<b>5</b>	<b>Discussion</b>	<b>53</b>
5.1	Useful Supervision . . . . .	53
5.2	All about Data? . . . . .	53
5.3	Outlook . . . . .	54

# 1 Introduction

## 1.1 Context and Motivation

Recent leaps in deep learning with large neural networks have raised public awareness for the progress in artificial intelligence.

One of the most transformative and prominent advancements within this field is the proliferation of large language models with multimodal capabilities that tie together the vision and language domain and perform tasks like automated image analysis, text prompt guided image creation and others that involve joint understanding of text and vision.

This success was enabled, in part, by new approaches in self-supervised representation learning: Unlike classic supervised methods that rely on the presence of annotated training sets, self-supervised learning allows models to learn meaningful representations from raw data, for example text, images and videos, or text-image pairs retrieved from the internet or compiled from other sources. This approach scales up the amount of available training data by orders of magnitude, as illustrated, for example, by the LAION datasets that contain billions of data points and thus training examples—a number that is impossible to achieve by human labeling efforts and supervised.

One of the most influential methods within this self-supervised learning framework is to train vision perception models by natural language supervision via contrastive learning, as exhibited by the CLIP model by OpenAI in 2021 [71].

The success of CLIP and its variants, like [10, 90, 55], has specifically enabled vision models to generalize better than before to unknown tasks and datasets. This capability of domain generalization is crucial, in particular when deploying intelligent machines in sensitive areas like autonomous driving or medical diagnosis.

From around the mid 2010s, deep learning models have been able to match or surpass human performance in narrow tasks like image classification [40] or board games [83]. Despite these successes, the challenge remains of generalizing deep learning models to perform reliably across diverse and unseen domains and in situations that differ significantly from their training.

This issue is particularly pertinent in real-world applications such as autonomous driving, where models must navigate safely in varying environmental conditions without retraining.

A possible explanation for the lack of ability of current deep learning models in this regard is that we still have not yet managed to teach them robust and general representations of the world that match those exhibited by humans or even animals. A striking illustration of this deficiency is given by the countless amounts of video and image data that have been fed into autonomous driving models, yet the goal of fully autonomous

driving eludes us. In contrast, an average human of 20 years can learn this task within only dozens of hours of practice—building upon a deep and rich world representation developed through the course of his lifetime and, possibly, human evolution.

The core problem addressed in this thesis is how to enhance the domain generalization capabilities of computer vision models in the context of semantic segmentation for autonomous driving. The primary hypothesis driving this research is that contrastive multimodal representation learning—where models are trained to align representations from different modalities, in our case camera images and lidar point clouds—can significantly improve the domain robustness of these models. This idea is directly inspired by the success of vision-language models, like CLIP, which have shown that learning representations across multiple modalities can lead to improved performance and generalization in diverse tasks.

## 1.2 Overview of the Thesis Structure

The thesis is organized as follows.

- **Chapter 2: Review of basic theory and related works** starts with an overview of the theoretical foundations of machine learning: statistical learning theory and neural networks. We also review some basics of neural network training: stochastic gradient descent and backpropagation as fundamental methods. We then focus on computer vision and introduce the required learning tasks and some more specialized model architectures before giving a more detailed introduction to the CLIP model and review some previous works about our thesis subject: camera-lidar contrastive pretraining.
- **Chapter 3: Method** starts with the motivation of our subject and presents a heuristic argument for its feasibility. We then lay out the general method of testing our hypothesis in detail.
- **Chapter 4: Numerical Experiments** introduces our experiment setting by giving an overview of the employed architectures, datasets and metrics. We then give a detailed presentation of our experiments and their results.
- **Chapter 5: Discussion and Outlook** This chapter discusses the implications of the experimental findings, critically evaluates the limitations of our approach suggests avenues for future research. The discussion also touches on the broader impact of the results and how they relate to ongoing debates in the field, such as the importance of data scale versus the benefits of multimodal learning.

## 2 Review of basic theory and related works

We start with an overview of the theoretical foundations of machine learning: statistical learning theory and neural networks. Then we review the basics of neural network training: stochastic gradient descent and backpropagation as fundamental methods as well as the concept of regularization. We proceed by focusing on computer vision and introduce the required learning tasks and some more specialized model architectures. Lastly, we present recent work related specifically to the subject of this thesis.

### 2.1 The Learning Problem

*Machine learning*<sup>1</sup> is a subfield of artificial intelligence, an area of research and development that studies and creates intelligent machines. What is learning? An informal definition is given by Tom Mitchell in [67]:

A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .

What distinguishes machine learning from other forms of artificial intelligence is that the goal is not to solve the task  $T$  explicitly by providing a handcrafted set of fixed instructions, but to develop a *learning algorithm* that produces a computational model to solve the task.

More formally, we can frame the learning task as a classical approximation problem: We want to approximate a *target function*  $f^* : \mathcal{X} \rightarrow \mathcal{Y}$  where  $\mathcal{X}$  is the domain that contains the *data* (input) and  $\mathcal{Y}$  is the set of possible outputs.<sup>2</sup> We refer to each  $x \in \mathcal{X}$  as a *data point* where  $\mathcal{X}$  is either  $\mathbb{R}^n$  or a subset thereof. The elements of  $\mathcal{Y}$  are called *labels*. We distinguish between *regression* tasks where  $\mathcal{Y} = \mathbb{R}$ , i.e. the labels represent real values, and *classification* tasks where  $\mathcal{Y} = \{1, \dots, c\}$ , i.e. the labels represent discrete classes.

To capture the notion of experience  $E$ , we assume a *data generating probability distribution*  $\mathcal{D}$  that produces a sequence of *training data*  $S = (x_i, y_i)_{i=1}^n \sim \mathcal{D}$  where  $y_i = f^*(x_i)$  is the true label for data point  $x_i$ .<sup>3</sup> By  $S$  we denote the *training set* which contains the training data. Since we have access to the true labels, we call this setting

---

<sup>1</sup>The term was first introduced by Arthur Samuel in 1959. [79]

<sup>2</sup>We require and tacitly assume the existence of a measurable space on our domain  $\mathcal{X}$ , in most cases the Borel- $\sigma$ -Algebra or the power set. We also assume all involved functions to be measurable.

<sup>3</sup>The probability measure  $\mathcal{D}$  is defined on the underlying measurable space on  $\mathcal{X}$ .



*supervised learning*. In *unsupervised learning*, the training data are unlabelled. In this case, the goal is to either learn the underlying distribution  $\mathcal{D}$  or some statistic of it. Throughout this thesis we will stick to the supervised setting.

The approximation task then is to *learn* a function  $h : \mathcal{X} \rightarrow \mathcal{Y}$ ,  $h \in \mathcal{H}$ , from the examples given by  $f^*(\mathcal{S})$  that approximates  $f^*$  well on both  $\mathcal{S}$  and on  $\mathcal{X} \setminus \mathcal{S}$ . Approximating  $f^*$  on  $\mathcal{X} \setminus \mathcal{S}$  can be interpreted as  $h$  performing well on new data, hence we also call it a *prediction task*.  $\mathcal{H}$  is called the *hypothesis set* and contains all candidate functions for the prediction task.

To evaluate the performance  $P$  of our prediction, we introduce the *loss function*  $\mathcal{L} : \mathcal{H} \times (\mathcal{X}, \mathcal{Y}) \rightarrow \mathbb{R}$ . Our goal is to minimize the *generalization error* or *risk*

$$\mathcal{R}(h) = \mathbb{E}_{(x,y) \sim \mathcal{D}}[\mathcal{L}(h, (x, y))] = \int_{\mathcal{X} \times \mathcal{Y}} \mathcal{L}(h, (x, y)) \, d\mathcal{D}(x, y).$$

Since we don't know the distribution  $\mathcal{D}$  nor  $f^*$ , we cannot compute the generalization error. Instead, we minimize the *empirical risk*, also called *empirical error* or *cost*

$$\hat{\mathcal{R}}_{\mathcal{S}}(h) = \hat{\mathcal{R}}_{\mathcal{S}}(h) = \frac{1}{|\mathcal{S}|} \sum_{(x,y) \in \mathcal{S}} \mathcal{L}(h, (x, y)) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(h, (x_i, y_i)).$$

We will sometimes also refer to it as the *loss over the training data* or similar. Using the empirical risk as a proxy for the risk can be justified under the assumption that the data samples are drawn i.i.d. according to  $\mathcal{D}$ . Then, for a fixed prediction  $h$ , we have

$$\mathbb{E}_{S \sim \mathcal{D}}[\hat{\mathcal{R}}_{\mathcal{S}}(h)] = \mathcal{R}(h),$$

meaning the training data  $S$  form a representative sample over  $\mathcal{X}$ .

The *approximation error*, defined as

$$\mathcal{E}_{\text{appr}}(\mathcal{H}) = \inf_{h \in \mathcal{H}} \mathcal{R}(h),$$

quantifies the best prediction for  $f^*$  that is possible with  $h \in \mathcal{H}$ . However, that is a theoretical quantity. There is also an *optimization error* introduced by the learning algorithm itself: In practice, the algorithm will often not find an optimal  $h \in \mathcal{H}$  even if the approximation error is negligible.

There is a non-trivial trade-off between the different sources of error. If  $\mathcal{H}$  is too small, the approximation error will be non-trivial and the model is said to *underfit*. If  $\mathcal{H}$  is large and the approximation error negligible, the model might *overfit* the training data and not generalize well (as illustrated in figure 2.1). To prevent overfitting, different methods of *regularization* are employed (see section 2.3.3).

*Deep learning* is a subfield of machine learning that is concerned with the study and development of a special class of hypothesis sets: deep neural networks. Over the last decade, neural networks in various forms have become the state of the art in a wide range of machine learning applications.

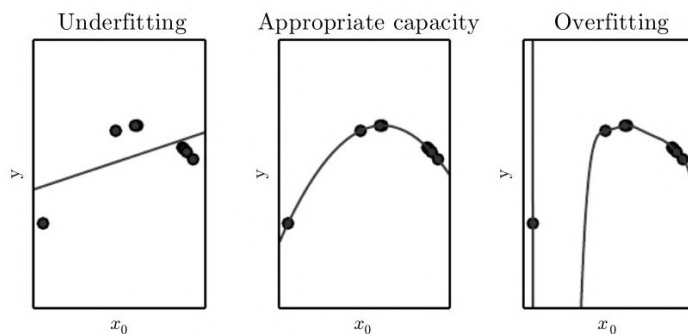


Figure 2.1: In this simple example, the data points follow a quadratic function: The linear approximation (left) has not enough degrees of freedom to minimize the empirical error and underfits, while a polynomial of degree 9 (right) overfits the training set: The empirical error is 0, but the expected error for new data will be significant. Figure from [32]

## 2.2 Neural Networks Basics

Artificial neural networks are inspired by the human brain and date back at least to 1943 [66]. Here, we treat them as mere “function approximation machines” [32, p. 169] to solve certain machine learning tasks.

There is also a distinction between neural networks as a class of mathematical functions and as a computational structure: a special type of directed acyclic graph. We refer to the underlying computational graph structure as the *architecture* of a neural network. The relationship between mathematical functions and neural network architectures is not one-to-one: Every well-defined neural network architecture or computational graph gives rise to a unique neural network function, but a neural network function may be implemented by multiple distinct architectures. In the following, we will mostly gloss over such subtleties and use the two notions interchangeably.

### 2.2.1 Feed Forward Neural Networks

The fundamental building block of a neural network is the (artificial) *neuron*, also referred to as *node* or *unit*. Each neuron or node is a composition of a (in general nonlinear) function and an affine-linear function (see figure 2.2). The formal definition can be given as follows.

**Definition 1.** An *artificial neuron* with *weights*  $w_1, \dots, w_n \in \mathbb{R}$ , *bias*  $b \in \mathbb{R}$  and *activation function*  $\rho : \mathbb{R} \rightarrow \mathbb{R}$  is defined as the function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  given by

$$f(x_1, \dots, x_n) = \rho \left( \sum_{i=1}^n x_i w_i + b \right) = \rho(x \cdot w + b),$$

where  $w = (w_1, \dots, w_n)$ ,  $x = (x_1, \dots, x_n)$  and  $\cdot$  is the dot product.

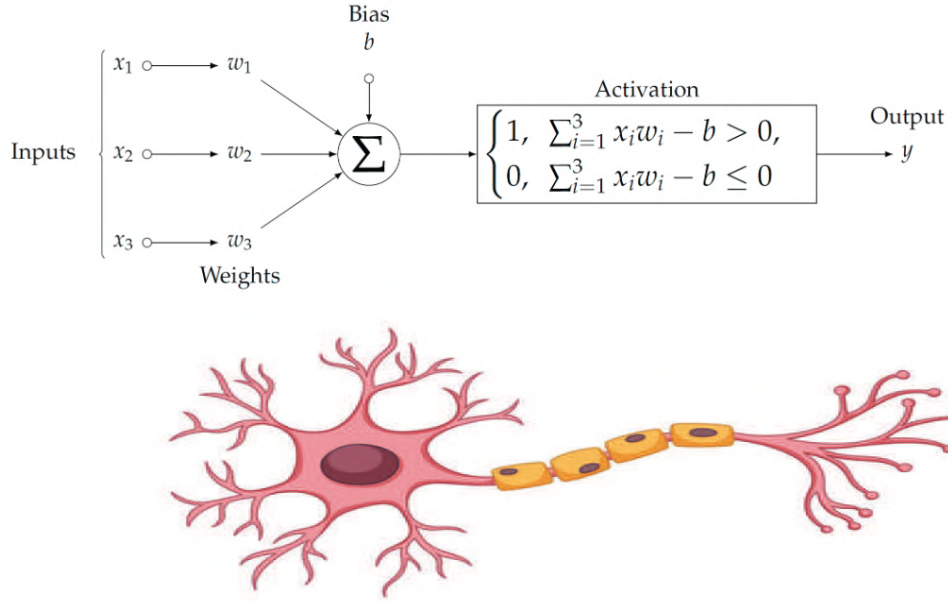


Figure 2.2: An artificial neuron, from the lecture notes to *Theoretical Foundations of Deep Learning* by Gitta Kutyniok.

A *feedforward neural network* consists of such neurons arranged in layers where connections are only allowed between consecutive layers and information flows only one way: in the direction from input to output. If neurons of consecutive layers of a network are all connected, i.e. no weights equal zero, it is called a *fully-connected neural network* or *multilayer perceptron* (MLP), illustrated in figure 2.3.

The layers of a network can be considered a vectorized version of neurons where the vector-valued affine-linear map consists of a weight matrix, a bias vector and the activation function is applied component-wise. The whole network is just a composition of such layers.

**Definition 2.** We call the first layer of a neural network the *input layer*, the last one the *output layer*, and all layers in between *hidden layers*. Let  $d \in \mathbb{N}$  be the dimension of the input layer,  $L$  the number of layers,  $N_0 := d$ ,  $N_l$  for  $l = 1, \dots, L-1$  the dimensions of the hidden layers,  $N_L$  the dimension of the output layer,  $\rho, \sigma : \mathbb{R} \rightarrow \mathbb{R}$  two activation functions that operate component-wise, and, for  $l = 1, \dots, L$ , let  $T^l$  be the affine-linear function

$$T^l : \mathbb{R}^{N_{l-1}} \rightarrow \mathbb{R}^{N_l}, \quad T^l(x) = W^l x + b^l,$$

with  $W^l \in \mathbb{R}^{N_l \times N_{l-1}}$  the weight matrix and  $b^l \in \mathbb{R}^{N_l}$  the bias vector of the  $l$ th layer. Then  $\Phi_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^{N_L}$ , given by

$$\Phi_\theta(x) = \sigma(T^L \rho(T^{L-1} \rho(\dots \rho(T^1(x)) \dots))), \quad x \in \mathbb{R}^d,$$

is called a *neural network* with *parameters*  $\theta = ((W^l, b^l))_{l=1}^L$ .

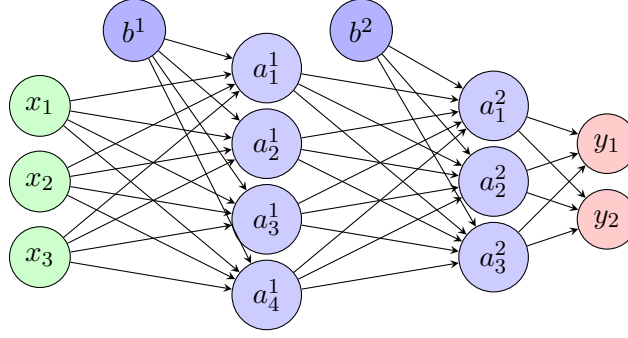


Figure 2.3: An multilayer perceptron with input  $x$ , output  $y$  and where  $a^1 = \rho(W^1x + b^1)$  and  $a^2 = \sigma(W^2a^1 + b^2)$  represent the activations of the two hidden layers with biases  $b^1$  and  $b^2$ .

The network consists of  $L$  layers in total (not counting the input layer) and  $L - 1$  hidden layers. For  $L = 2$  or 1 hidden layer, we call a network *shallow*, and for  $L \geq 3$  *deep*.

Note that in definition 2 we specified only two distinct activation functions  $\rho$  and  $\sigma$ . In general, each neuron in a neural network can have its own activation function. But it is common to use the same activation function  $\rho$  on all hidden layers and a separate activation function  $\sigma$  on the output layer.

### 2.2.2 Activation Functions

Our exposition follows [32] and also draws on some more recent information given in [68, 81, 70].

Without a nonlinear activation function, the expressive power of a neural network (section 2.2.4) is equal to that of an affine-linear function. Thus nonlinear activation functions are essential to represent nonlinear target functions.

Since the activation on the output layer determines the whole network's range, it is usually guided by the specific task. Often we want to attach some meaning to the final output of the network. In unbounded prediction tasks like regression the output represents unbounded real values so the output activation can be omitted. That is not applicable to classification tasks where we usually want to interpret the output as a probability of the input belonging to a certain class. Hence, a natural choice for classification tasks are the logistic function  $\sigma(x) = \frac{1}{1+e^{-x}}$  for binary problems, also referred to as *the sigmoid function*<sup>4</sup>, or the *softmax function* which maps an input vector from  $\mathbb{R}^n$  to  $(0, 1)^n$  to a probability distribution:

$$\text{softmax}(x_1, \dots, x_n) = \frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}, \quad \text{for } k = 1, \dots, n.$$

Because of this property, the softmax is the standard output activation for multiclass classification tasks.

<sup>4</sup>In general, a sigmoid function is defined as a bounded, differentiable, real-valued function on  $\mathbb{R}$  or a subset thereof that has a non-zero derivative at each point and exactly one inflection point.

The activation function in the hidden layers is not subject to such task-dependent restrictions so the choice of possible functions is much greater. Since the hidden layer activation plays a vital role in the model architecture, it is commonly chosen to optimize the training process and model performance.

Sigmoid functions like the logistic function or the hyperbolic tangent  $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$  have been among the first activation functions to be studied and employed in hidden layers. Their use was inspired by neurons in the brain as they are a differentiable approximation of binary step functions that indicate whether a neuron fires or not. With the rise of deep neural networks, sigmoid functions have largely fallen out of favor because they can negatively affect gradient-based optimization: As their derivative tends to zero for  $|x| \rightarrow \infty$ , gradients tend to zero as  $|x|$  rises. This *vanishing gradient problem* can cause neurons to permanently drop out and, in the worst case, learning to fail.

A partial solution to this problem is the *rectified linear unit* (ReLU), defined as  $\text{ReLU}(x) = \max(x, 0)$ . The ReLU is close to a linear function, fast to compute, suffers less from vanishing gradients and has been found to consistently outperform sigmoid functions [30]. Since the gradients for negative inputs vanish, dead neurons can still become an issue. Multiple variants have been proposed to address this problem by allowing some gradient flow also for negative input values, e.g. the *leaky ReLU* (LReLU) [64]

$$\text{LReLU}(x) = \begin{cases} x & x \geq 0, \\ \alpha x & x < 0, \end{cases}$$

the *exponential linear unit* (ELU) [12]

$$\text{ELU}(x) = \begin{cases} x & x \geq 0, \\ \alpha(e^x - 1) & x < 0, \end{cases}$$

where  $\alpha > 0$  is a fixed hyperparameter. Smooth approximations to the ReLU are the *Gaussian Error Linear Unit* (GELU) and the *Sigmoid Linear Unit* (SiLU) [42], defined as

$$\text{GELU}(x) = xF_{\mathcal{N}(0,1)}(x) = \frac{1}{2}x \left( 1 + \text{erf} \left( \frac{x}{\sqrt{2}} \right) \right), \quad (2.1)$$

$$\text{SiLU}(x) = x\sigma(x) = \frac{x}{1 + e^{-x}}, \quad (2.2)$$

where  $F_{\mathcal{N}(0,1)}$  is the standard normal distribution's cumulative distribution and  $\text{erf}$ , defined as  $\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ , the error function. Note that all these activation functions and their hyperparameters are fixed during training.

There has also been work on activation functions containing learnable parameters that are directly optimized during network training, for example the *parametric ReLU* (PReLU) [64] that is defined equivalently to the leaky ReLU, but with  $\alpha$  as a learnable parameter, or a parametrized version of the SiLU, called *Swish* function [72]:

$$\text{Swish}(x) = \frac{x}{1 + e^{-\beta x}},$$

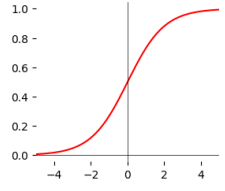
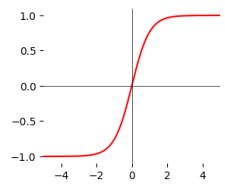
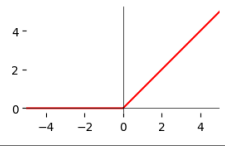
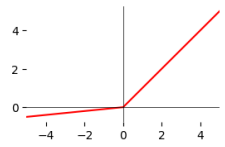
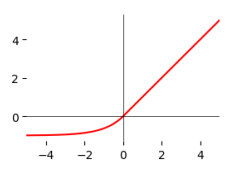
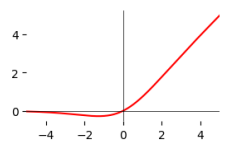
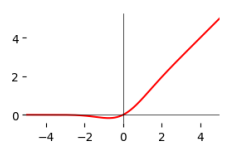
Name	$x \in \mathbb{R}^n$ (softmax), else $x \in \mathbb{R}$	Plot
Softmax	$\frac{e^{x_k}}{\sum_{i=1}^n e^{x_i}}, k = 1, \dots, n$	
Logistic/Sigmoid	$\frac{1}{1 + e^{-x}}$	
Hyperbolic Tangent (Tanh)	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	
Rectified Linear Unit (ReLU)	$\max(0, x)$	
Leaky ReLU (LReLU)	$\max(\alpha x, x)$	
Exponential Linear Unit (ELU)	$x \mathbb{1}_{x \geq 0} + \alpha(e^x - 1) \mathbb{1}_{x < 0}$	
Sigmoid Linear Unit (SiLU)	$\frac{x}{1 + e^{-x}}$	
Gaussian Error Linear Unit (GELU)	$\frac{1}{2}x \left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$	

Table 2.1: Some common activation functions.

where  $\beta$  is a learnable parameter.

There is a wide range of other proposed and employed activation functions in theory and practice. For the ones mentioned here, we give an overview in table 2.1.

### 2.2.3 Loss Functions

The loss function is central to the learning task as it defines the objective function that our learning algorithm aims to optimize (see section 2.3). For one dimensional regression tasks (where labels represent continuous real values) the most common loss function is the *quadratic loss*

$$\mathcal{L}(h, (x, y)) = (h(x) - y)^2.$$

The corresponding cost function is called *mean squared error*.

For binary classification tasks, the *0-1 loss*

$$\mathcal{L}(h, (x, y)) = \mathbb{1}_{\{h(x) \neq y\}}$$

indicates whether the assigned class label corresponds to the true label. However, this loss function suffers from non-differentiability, which discourages gradient-based optimization methods, and it can be computationally intractable. Learning can also be more efficient by assigning probabilities to class labels, as with a softmax output activation. This approach captures more information, as the loss function not only indicates whether the model assigns a correct label, but also how close it is to the correct label. This gives rise to the *cross-entropy loss* for classification with  $C$  classes,  $\mathcal{Y} = \{1, \dots, C\}$ :

$$\mathcal{L}(h, (x, y)) = -y^T \log(h(x)) = -\sum_{i=1}^C y_i \log(h(x)_i),$$

where  $y \in \mathbb{R}^C$  is a column vector encoding class  $i$  being the true label with 1 as the  $i$ th entry and all other entries being 0 (*one-hot-encoding*) and  $h(x)$  is a column vector containing the class probabilities predicted by the model  $h$ .

If we consider  $p$  to be the true distribution of labels on our data and  $h$  the empirical distribution given by our model, then minimizing the cross-entropy loss is equivalent to minimizing the *Kullback–Leibler divergence* between  $p$  and  $h$  or, equivalently, of applying *maximum likelihood estimation* to the loss as a function of  $h$ .

### 2.2.4 Universal Approximation Property

A natural question to ask about neural networks as a class of approximating functions is how expressive they are: How big is the class of functions that they can, in principal, approximate arbitrarily well?

Let  $\mathcal{H}$  and  $\mathcal{C}$  be classes of measurable functions on  $\mathbb{R}^n$  or some subset. We say that  $\mathcal{H}$  has the *universal approximation property* with respect to  $\mathcal{C}$  if for all  $f \in \mathcal{C}$  and each  $\epsilon > 0$  there is an  $h \in \mathcal{H}$  such that  $\|f - h\| < \epsilon$  where  $\|\cdot\|$  is some appropriate norm depending on context, e.g. the supremum norm or  $L_p$ -norm if the underlying function space is a subset of  $C(X)$  or  $L_p(\mu)$ , respectively. Thus  $\mathcal{H}$  is a class of universal approximators for

$\mathcal{C}$  exactly if  $\mathcal{H}$  is *dense* in  $\mathcal{C}$  with respect to the norm metric. Statements asserting such a property are commonly referred to as *universal approximation theorems*.

Some of the earliest of such results are due to Cybenko [16] and Hornik, Stinchcombe and White [45] which asserted that feedforward neural networks with only one hidden layer, arbitrary width and certain activation functions, including sigmoids, are universal approximators for the space of continuous real-valued functions on compact subsets of  $\mathbb{R}^n$ . Under some mild conditions, the universal approximation extends to the target function’s derivative [46]. Subsequently, Hornik extended these results to the  $L_p(\mu)$  spaces [47]. Then Leshno et al. proved the far-reaching generalization that “[an MLP] can approximate any continuous function to any degree of accuracy if and only if the network’s activation function is not polynomial” [59]. Note that this includes popular activation functions like the ReLU.

While encouraging, most of these early works were qualitative in nature and assumed arbitrarily wide shallow networks. Later works further explored the connection between the size of networks in width and depth and the resulting approximation errors. Within the last 10 to 20 years, special interest in the role of depth was raised due to the success of deep neural networks in many applications [53, 37].

However, compared to the practical exploits of deep learning, theoretical understanding remains largely rudimentary and today cannot explain why deep neural networks excel like they do in many learning tasks, why certain architectures perform better than others or how to construct optimal architectures for given tasks. Development in deep learning is mostly driven by practical experimentation rather than deductive insight [93, 5, 104, 2, 31].

## 2.3 Training Neural Networks

The common training strategy in deep learning is *empirical risk minimization (ERM)*: Given our model architecture  $\Phi_\theta$  and training data  $S$ , we minimize the cost  $\hat{\mathcal{R}}_S(\Phi_\theta)$ , i.e. we approximate

$$\theta^* = \arg \min_{\theta} \hat{\mathcal{R}}_S(\Phi_\theta),$$

usually by a gradient-based optimization algorithm (see section 2.3.2 and following sections).

To get an estimate of the generalization error, we either obtain more i.i.d. samples from  $\mathcal{X}$  or split-off a subset  $\mathcal{T}$  of  $S$  beforehand and compute the empirical error  $\hat{\mathcal{R}}_{\mathcal{T}}(\Phi_\theta)$ .  $\mathcal{T}$  is referred to as *test set* because the data points in  $\mathcal{T}$  are not used to learn the prediction  $\Phi_\theta$ , but only to evaluate its performance. The i.i.d. assumption justifies using the performance of  $\Phi_\theta$  on  $\mathcal{T}$  as an estimate for the generalization error.

The choice of neural network architecture involves selecting the model class itself as well as that of *hyperparameters*, some of which are specific to the training algorithm, some to the model. In order to supervise the training performance and either adjust hyperparameters along the way or switch to a different model altogether, a *validation set*  $\mathcal{V} \subset S$  is used: Similarly to the test set, the data from  $\mathcal{V}$  are not used for training.



However, since they guide the selection and specification of the learned model, it will be statistically dependent on the validation data. This difference between validation and test data is crucial. In order for the test data to allow an accurate estimation of the generalization error and thus an unbiased evaluation of the model's generalization performance with respect to unseen data, the test data may not guide the construction of the model in any way.

### 2.3.1 Stochastic Gradient Descent

Let  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be a differentiable function. The gradient of  $f$  at  $x = [x_1, \dots, x_d]^T \in \mathbb{R}^d$  is given by

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_d}(x) \end{bmatrix}.$$

Assume that we want to find the global minimum

$$\min_{x \in \mathbb{R}^d} f(x).$$

The gradient  $\nabla f(x)$  has the property of pointing in the direction of steepest ascent in any given point  $x$ . We can use this to find the minimum with a method called *gradient descent*: We choose an initial point  $x^0 \in \mathbb{R}^d$ . At each iteration  $k$ , we compute the gradient  $\nabla f(x^k)$  and update by letting

$$x^{k+1} = x^k - \eta_k \nabla f(x^k)$$

where  $\eta_k > 0$  is a sequence of hyperparameters called *step size* or, in a machine learning setting, *learning rate*. The negative gradient  $-\nabla f(x^k)$  points in the direction of steepest descent from  $x^k$  so we have  $f(x^{k+1}) \leq f(x^k)$  for a sufficiently small step size  $\eta_k$ . We repeat the update until a stopping criterion is met, such as  $\|\nabla f(x^k)\| < \epsilon$  for some  $\epsilon > 0$  or until a maximum number of iterations  $K$  is reached. The process is illustrated in figure 2.4.

If  $f$  is convex and the step size suitably chosen, the gradient descent algorithm converges to the global minimum [27, 6].

In deep learning, the objective function  $f$  is a deep neural network whose parameters we want to optimize over the training set. For large models, this operation can become very costly in terms of computational resources and large datasets often exceed the available memory of the computation device. Hence, we usually sample only a subset of the training data, called a *batch* or *minibatch*, in each gradient descent iteration. The resulting method is called *stochastic gradient descent* (SGD) and is the standard optimization algorithm in deep learning [92].

Let  $S$  be the training data,  $|S| = n$ ,  $\Phi_\theta$  a neural network with learnable parameters  $\theta$  and  $\mathcal{J}_S(\theta) = \hat{\mathcal{R}}_S(\Phi_\theta)$  the cost as a function of  $\theta$ , i.e.

$$\mathcal{J}_S(\theta) = \frac{1}{|S|} \sum_{(x,y) \in S} \mathcal{L}(\Phi_\theta, (x,y)),$$

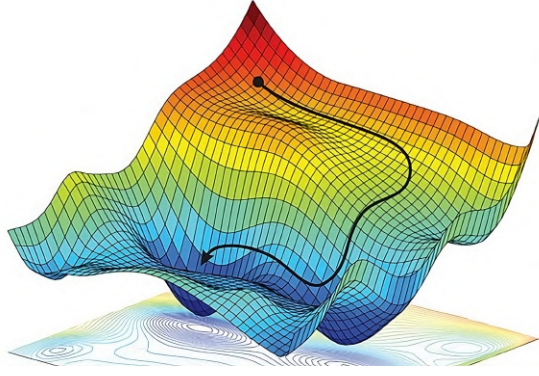


Figure 2.4: Gradient descent for a scalar function of two variables. Figure from [1].

where  $\mathcal{L}$  is the loss over one training example.

We then want to minimize  $\mathcal{J}_S$ . In each SGD iteration, we sample a minibatch  $S_k$  with a *batch size* of  $m$  training examples from  $S$ , where  $m$  is much smaller than  $n$ , and perform the parameter update

$$\begin{aligned}\theta^{k+1} &= \theta^k - \eta_k \nabla_{\theta} \mathcal{J}_{S_k}(\theta) \\ &= \theta^k - \eta_k \nabla_{\theta} \frac{1}{|S_k|} \sum_{(x,y) \in S_k} \mathcal{L}(\Phi_{\theta}, (x, y)) \\ &= \theta^k - \frac{\eta_k}{|S_k|} \sum_{(x,y) \in S_k} \nabla_{\theta} \mathcal{L}(\Phi_{\theta}, (x, y)),\end{aligned}\tag{2.3}$$

where  $y$  is the label corresponding to  $x$ . Note that in each iteration we only compute an estimate  $\nabla_{\theta} \mathcal{J}_{S_k}(\theta)$  of the true gradient  $\nabla_{\theta} \mathcal{J}_S(\theta)$  which is computationally much more efficient.

If we sample the minibatches  $S_k$  uniformly from  $S$  in each iteration,  $S_k$  is a sequence of i.i.d. random variables and we have  $\mathbb{E} [\nabla_{\theta} \mathcal{J}_{S_k}(\theta)] = \nabla_{\theta} \mathcal{J}_S(\theta)$ . In this case, SGD converges under suitable conditions [27, 33].

In practice, it is common to pick the batches  $S_k$  without replacement such that they cover all training data from  $S$  in one *epoch* of  $\lceil n/m \rceil$  iterations [80]—epoch is the common term for a training cycle over the whole training data. In theoretical analysis, this case is less understood, but convergence results exist as well [77, 36].

In mathematical optimization, gradient descent is usually considered too slow compared to higher-order methods like *Newton's method* [76, section 5.3.1]. But those require computation of the Hessian matrix of second partial derivatives in each iteration. In deep learning, dataset size and dimension of  $\theta$  can reach into the hundreds of millions or even billions. As a consequence, SGD is usually more efficient and sometimes the only choice that is computationally feasible [32, section 8.6.1].

In addition to computational efficiency, the randomization of descent directions in SGD provides a form of regularization and can prevent overfitting (see also section 2.3.3). Note also that in deep learning, as opposed to other optimization settings, the goal is not

really to minimize the training loss, but to generalize well to unknown data. Therefore, it is often not necessary to find a global minimum, but only to reduce  $\mathcal{J}_S$  sufficiently.

The choice of batch size is often not trivial: On the one hand, larger batches leverage parallel computation more effectively and can significantly speed up training, but they can reduce regularization which may hamper generalization [44]. Significant efforts have been made to increase efficiency of large batch training [34, 103].

SGD is also sensitive to the learning rates  $\eta_k$  which typically need to be scaled with the batch size to achieve optimal results. Various learning rate schedules and other optimizations exist and are often preferred over the basic form of SGD presented here [92]. Learning rate, batch size and other training specific hyperparameters are often optimized during the training and validation process [32, section 11.4].

### 2.3.2 Backpropagation

The following exposition closely follows [69].

At the core of SGD is the computation of the gradients  $\nabla_{\theta}\mathcal{L}(\theta; x, y) := \nabla_{\theta}\mathcal{L}(\Phi_{\theta}, (x, y))$  for a training example  $(x, y)$ . For SGD to perform at scale, we need an efficient method to compute those gradients for inputs and parameters of very high dimensionality. The standard way to perform this operation efficiently is called *backpropagation*. In definition 2 we have introduced the notion of a feedforward neural network where information flows one way from the input, a data point  $x$ , to the model output prediction's loss  $\mathcal{L}(\Phi_{\theta}, (x, y))$ . This computation is called the *forward pass*. During the *backward pass*, the chain rule of differentiation is applied backwards from the loss in order to approximate the gradient  $\nabla_{\theta}\mathcal{L}$ .

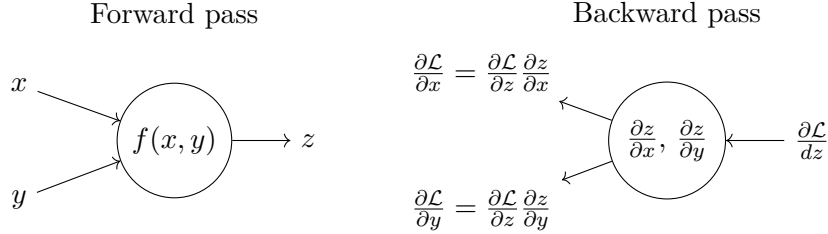


Figure 2.5: During the forward pass, the local gradients  $\frac{\partial z}{\partial x}$  and  $\frac{\partial z}{\partial y}$  can be computed and cached along with  $z = f(x, y)$ . In the backward pass, those local gradients are used to compute the global gradients  $\frac{\partial \mathcal{L}}{\partial x}$  and  $\frac{\partial \mathcal{L}}{\partial y}$  of the loss  $\mathcal{L}$  with respect to  $x$  and  $y$ .

Here we consider backpropagation for a feedforward neural network as per definition 2 where we assume  $\sigma = \rho$  (same activation function on each hidden and the output layer). As additional notation we introduce the *activation of layer  $l$*  as

$$a^l = \rho(T^l a^{l-1}) = \rho(W^l a^{l-1} + b^l)$$

and the *preactivation of layer  $l$*  as

$$z^l = T^l a^{l-1} = W^l a^{l-1} + b^l.$$

We adopt standard notation for vector and matrix entries:  $z_i^l, b_i^l, a_i^l$  are the preactivation, activation and bias of node  $i \in \{1, \dots, N_l\}$  in layer  $l, l = 1, \dots, L$ , and  $w_{ij}^l$  is the weight from node  $j$  in layer  $l-1$  to node  $i$  in layer  $l$ .

Consider the loss  $\mathcal{L}(\theta; (x, y))$  of the network  $\Phi_\theta$  for a fixed training example  $(x, y)$ . The goal of backpropagation is to compute the gradient  $\nabla_\theta \mathcal{L}$  which contains the partial derivatives

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^l} \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b_i^l}$$

for all  $i, j$  and  $l$ . Furthermore, we define the *error (of neuron  $i$ ) in layer  $l$*  as

$$\delta_i^l = \frac{\partial \mathcal{L}}{\partial z_i^l} \quad \text{and} \quad \delta^l = \frac{\partial \mathcal{L}}{\partial z^l},$$

respectively. The backpropagation algorithm computes those  $\delta^l$  layer by layer, going back from the last layer  $L$ . By definition we have  $a_i^l = \rho(z_i^l)$  and the chain rule yields

$$\delta_i^L = \frac{\partial \mathcal{L}}{\partial a_i^L} \rho'(z_i^L) = \frac{\partial \mathcal{L}}{\partial a_i^L} \frac{d}{dz_i^L} \rho(z_i^L)$$

and

$$\delta^L = \text{diag}(\rho'(z^L)) \nabla_{a^L} \mathcal{L}, \tag{2.4}$$

where  $\text{diag}(v)$  is a diagonal matrix with its diagonal given by the vector  $v$  and the scalar derivative  $\rho'$  is applied element-wise.

Next we want to obtain a recursive formula for  $\delta^l$  in terms of  $\delta^{l+1}$ . Again with the chain rule, we get

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \sum_{i=1}^{N_{l+1}} \frac{\partial \mathcal{L}}{\partial z_i^{l+1}} \frac{\partial z_i^{l+1}}{\partial z_j^l} = \sum_{i=1}^{N_{l+1}} \delta_i^{l+1} \frac{\partial z_i^{l+1}}{\partial z_j^l}. \tag{2.5}$$

Differentiating

$$z_j^{l+1} = \sum_i w_{ij}^{l+1} a_i^l + b_j^{l+1} = \sum_i w_{ij}^{l+1} \rho(z_i^l) + b_j^{l+1}$$

yields

$$\frac{\partial z_j^{l+1}}{\partial z_i^l} = w_{ij}^{l+1} \rho'(z_i^l)$$

and inserting the latter into equation (2.5), we obtain

$$\delta_j^l = \sum_{i=1}^{N_{l+1}} w_{ij}^{l+1} \delta_i^{l+1} \rho'(z_i^l)$$

and thus

$$\delta^l = \text{diag} \left( \rho' \left( z^l \right) \right) \left( W^{l+1} \right)^T \delta^{l+1}. \quad (2.6)$$

Combining equation (2.4) and equation (2.6), we obtain the formula

$$\delta^l = \text{diag} \left( \rho' \left( z^l \right) \right) \left( W^{l+1} \right)^T \dots \text{diag} \left( \rho' \left( z^{L-1} \right) \right) \left( W^L \right)^T \text{diag} \left( \rho' \left( z^L \right) \right) \nabla_{a^L} \mathcal{L} \quad (2.7)$$

that allows us to compute all  $\delta^l$  in sequence from that last layer backwards. Those, in turn, allow computing  $\nabla_{\theta} \mathcal{L}$  because another application of the chain rule yields

$$\frac{\partial \mathcal{L}}{\partial b_i^l} = \delta_i^l \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial w_{ij}^l} = a_j^{l-1} \delta_i^l.$$

In each forward pass, the input  $x$  is propagated forward through the network, computing and storing in memory all  $z^l$  and  $a^l$  along the way, starting from  $l = 1$ . Then the label  $y$  is used to compute the loss. Finally, in the backward pass, all the  $\delta^l$  are computed, starting from  $l = L$ , which gives the gradient  $\nabla_{\theta} \mathcal{L}$  for one training example  $(x, y)$ . This is done for each example in the current minibatch  $S_k$  until the cost is computed as weighted sum and the SGD parameter update performed as per equation (2.3).

As a final remark, we note that the backpropagation algorithm is not specific to neural networks, but a general and efficient method to evaluate partial derivatives of any function or expression that is given as a computational graph. In the general setup it is also known as the *reverse mode of automatic differentiation*.

### 2.3.3 Regularization

As mentioned previously, the goal of the learning algorithm is not to fit the model to the training set, but to achieve the best possible generalization to unknown data. Any change to the learning algorithm with the goal of improving generalization, but not necessarily training performance, is called *regularization*.

A common form of regularization is to amend the cost function  $\mathcal{J}(\theta)$  by introducing a penalty term  $\Omega(w)$  where  $w$  is the subsequence of the parameter vector  $\theta$  that contains all weights, but no biases:

$$\tilde{\mathcal{J}}(\theta) = \mathcal{J}(\theta) + \lambda \Omega(w).$$

$\lambda > 0$  is a new hyperparameter and adjusted during training. The most common choice for  $\Omega$  is

$$\Omega(w) = \frac{1}{2} \|w\|_2^2 = \frac{1}{2} w^T w$$

and called  $L^2$ -regularization or *weight decay*: Assume that there are no biases, i.e.  $\theta = w$ . Then we get

$$\tilde{\mathcal{J}}(w) = \mathcal{J}(w) + \frac{\lambda}{2} w^T w, \quad (2.8)$$

$$\nabla_w \tilde{\mathcal{J}}(w) = \nabla_w \mathcal{J}(w) + \lambda w. \quad (2.9)$$

The SGD weight updates change to

$$w^{k+1} = w^k - \eta \nabla_w \tilde{\mathcal{J}}(w) \quad (2.10)$$

$$= w^k - \eta (\nabla_w \mathcal{J}(w) + \lambda w) \quad (2.11)$$

$$= (1 - \lambda\eta)w - \eta \nabla_w \mathcal{J}(w), \quad (2.12)$$

where we assume a fixed learning rate  $\eta$ . The weight vector shrinks by a constant factor at each iteration, scaled by the learning rate  $\eta$ . This progressively reduces model capacity and can reduce overfitting.

Another common type of explicit regularization is *dropout*: A hyperparameter termed *dropout rate* determines a certain chance for each node to “drop out” at a particular training iteration, meaning that all incoming and outgoing weights for that node are set to zero for this step. Again, this reduces model capacity and encourages redundant representations which prevents overfitting and improves generalization in many cases.

If a validation set is used, the model performance on the validation set can be used to approximate generalization performance. *Early stopping* is a regularization method where training is halted once the validation performance stagnates or the training and validation loss start diverging, even if the validation performance is still improving, as these may be interpreted as signs of overfitting.

Other common forms of regularization are more implicit: As mentioned, SGD itself has a regularizing effect that is affected by the minibatch size and the learning rate because smaller minibatches and higher learning rates increase randomization in each SGD step which can prevent overfitting.

## 2.4 Deep Learning for Computer Vision

*Computer vision* in machine learning is a subfield concerned with extracting information from visual data. We start by introducing two data types that are relevant within the context of this thesis: *2D images* and *3D point clouds*. We then introduce some common computer vision tasks and discuss deep learning architectures that are commonly used to solve them.

### 2.4.1 Images and Point Clouds

2D images obtained by cameras are the most often considered input data type in computer vision. We can define the data domain as  $\mathcal{X} = D^{H \times W \times C}$  with a height of  $H$ , a width of  $W$  pixels and  $C$  *color channels* where  $C$  equals 1 for greyscale images and 3 for color images in an *RGB space*. The *depth*  $D$  represents different shades of grey or color in an image and can be given as a continuous scale ( $D = [0, 1]$ ) or discrete range ( $D \in \mathbb{N}$ ), the latter indicating the number of available shades of color. The elements of  $\mathcal{X}$  can thus be interpreted as generalized matrices with more than 2 dimensions which are called *tensors*.

A 3D point cloud  $\{p_i\}_{i=1}^n$  is a set of points defined within a three-dimensional coordinate system, where each point represents a specific location in 3D space. These point sets are called *clouds* and capture the spatial structure of an object, scene, or environment.

Unlike traditional 2D images, which capture information along two axes (height and width), 3D point clouds include a depth component, allowing for a more comprehensive representation of real-world objects or scenes.

Each point  $p_i$  is typically characterized by its Cartesian coordinates  $(x_i, y_i, z_i)$ , which define its position relative to a reference frame, but may contain additional features such as color or reflectance of the represented object. 3D point clouds can be obtained by estimating depth information from multiple 2D images [85] or by dedicated 3D scanning technology like *lidar*.

Point cloud data can be processed by neural networks in three distinct ways: Some models like the popular PointNet [26] directly process raw point sets of 3D coordinates. Another approach, called *voxelization*, is to map the points into a regular 3D *voxel grid*, which is similar to a 2D pixel image with one added dimension [102]. The grid is determined by its coordinate axes and the voxel size. The features of points that are mapped into the same voxel are aggregated. The voxel-representation is more compressed than the raw point set and allows to apply operations like 3D convolutions (section 2.5.1) that depend on a regular grid structure. The third common method employed is to project the points onto 2D *range images* [99]. It is the most compressed way to represent point clouds and allows application of 2D vision models.

### 2.4.2 Lidar

*Light detection and ranging* (lidar) is a remote sensing technology that utilizes laser light to measure distances and create three-dimensional representations of the environment. Since our objective involves lidar data, we'll give a brief introduction here, based on [101, 107].

The fundamental principle of lidar involves emitting laser pulses and measuring the time it takes for these pulses to return after reflecting off surfaces. Since the laser pulses travel with the constant speed of light, this time-of-flight measurement allows for precise distance computation.

This process is repeated millions of times per second, generating a 3D point cloud that represents the scanned area. Lidar technology is specifically relevant for the task of autonomous driving due to its independence on lighting conditions. It is often used in conjunction with other sensors like cameras to improve scene understanding and object detection [106, 98].

There is an ongoing debate within the scientific community and industry as to whether a combination of multiple sensors, like camera and lidar, or a camera-only setup offer more advantages in self-driving vehicles. On the one hand, lidar sensors offer unique capabilities that complement the camera images, on the other hand, they introduce additional complexities and some argue that reliance on pure camera vision alone provides a more unitary and scalable approach [96, 20].

The absence of lidar or other complementing sensors that naturally enhance perception under adverse light and weather conditions increases the need for cameras to perform well under such conditions. While efforts have been made to substitute lidar sensors [63, 48], deliberate efforts to increase the robustness of camera inference using lidar data at

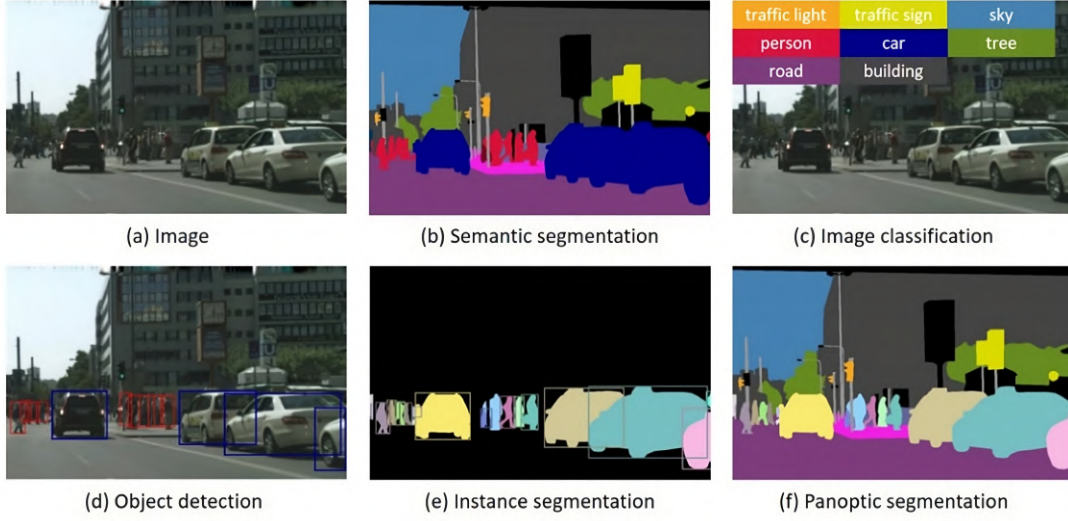


Figure 2.6: Overview of tasks in computer vision for a ground truth image (a). Image borrowed from [38].

training time has been mostly lacking so far. We will revisit this topic in section 2.6.2.

### 2.4.3 Semantic Segmentation

The most standard task in computer vision is *image classification* where whole images are assigned one of  $K$  class labels: We assume a true labeling function  $f^* : \mathcal{X} \rightarrow \mathcal{Y}$  where  $\mathcal{Y} = \{1, \dots, K\}$ . The case where  $f^*$  assigns labels on a pixel level is often called *dense prediction* and contains tasks like *object detection*, *depth estimation* and *semantic segmentation*. They are especially relevant to autonomous driving where deep understanding of traffic scenes is the basis for safe vehicle control. Today, deep neural networks are the prime method of solving those tasks [108, 38]. We will now introduce semantic segmentation as the main task treated in this thesis.

Semantic segmentation is a computer vision task that involves assigning a semantic class label to each pixel in an image, i.e.  $\mathcal{Y} = \{1, \dots, K\}^{H \times W}$  or, with one-hot-encoding,  $\mathcal{Y} = \{0, 1\}^{H \times W \times K}$  such that each pixel in  $H \times W$  is assigned a unit vector of length  $K$  to encode the true class label. The goal is to partition the image into semantically meaningful regions corresponding to objects, parts of objects or different background categories. This is of special interest in fields like autonomous driving, robotics and medical imaging. As opposed to the similar task of *instance segmentation*, which is the pixelwise labeling of distinct objects, semantic segmentation does not distinguish between different instances of the same class. *Panoptic segmentation* [54] is a combination of semantic and instance segmentation. We illustrate these together with the other mentioned common computer vision tasks in figure 2.6.

Given training data  $S = (x^n, y^n)_{i=1}^N$ , where  $x^n \in \mathcal{X}$  denotes an input image and  $y^n \in \mathcal{Y}$



the corresponding ground truth *segmentation map*, our training algorithm wants to find a model

$$\Phi_\theta : \mathcal{X} \rightarrow \hat{\mathcal{Y}} = [0, 1]^{H \times W \times K}$$

where  $\Phi_\theta(x^n) = \hat{y}^n = (\hat{y}_{i,j,k}^n) \in \hat{\mathcal{Y}}$  and  $\hat{y}_{i,j,k}^n = \mathbb{P}(k|(i, j), x^n)$  is the class posterior probability at pixel  $(i, j)$  for class  $k$  given input image  $x^n$ .

The standard loss function in semantic segmentation is a pixel-wise cross-entropy loss where the cross-entropy loss shown in section 2.2.3 is applied to each pixel and then averaged:

$$\mathcal{L}(y^n, \hat{y}^n) = -\frac{1}{HW} \sum_{i=1}^H \sum_{j=1}^W \sum_{k=1}^K y_{i,j,k}^n \log(\hat{y}_{i,j,k}^n)$$

where  $y^n \in \{0, 1\}^{H \times W \times K}$  is the ground truth one-hot-encoded segmentation map for an input image  $x^n$  with  $y_{i,j,k}^n = 1$  if pixel  $(i, j)$  belongs to class  $k$  and 0 otherwise. As in other classification tasks, the output activation is usually a pixelwise softmax.

The most common performance metric is the *mean intersection over union* (mIoU): Let  $A_k = \{(i, j) \in H \times W | \arg \max_l \hat{y}_{i,j,l}^n = k\}$  be the set of pixels assigned class label  $k$  by  $\Phi_\theta$  and  $B_k = \{(i, j) \in H \times W | y_{i,j,k}^n = 1\}$  be the corresponding ground truth set of pixels belonging to class  $k$ . The *intersection over union* for class  $k$  and a training example  $(x^n, y^n)$  is then defined as

$$\text{IoU}_k(x^n, y^n) = \frac{A_k \cap B_k}{A_k \cup B_k} \in [0, 1]$$

and the mean intersection over union over all  $K$  classes by

$$\text{mIoU}(x^n, y^n) = \frac{1}{K} \sum_{k=1}^K \text{IoU}_k(x^n, y^n)$$

which would again be averaged over a test or validation set to assess model performance.

## 2.5 Advanced Neural Network Architectures

The basic MLPs introduced in section 2.2.1 are commonly considered ill-suited for computer vision tasks like semantic segmentation. One reason is that their fully connected layers don't scale very well for high-dimensional input data like images: Even a small image of size  $100 \times 100$  with 3 color channels already has 30.000 features and thus each single neuron in the first hidden layer of an MLP would have to have 30.000 weights. With wider and deeper networks, the number of weights quickly becomes intractable even for moderately sized input images. This effect, often referred to as the *curse of dimensionality*, puts severe limitations on the scalability of MLPs and makes them inefficient for many problems.

In the following, we will introduce some more advanced neural network architectures that are employed for semantic segmentation and other dense prediction tasks: *convolutional neural networks* (CNNs) and *vision transformers* (ViTs). These architectures have been instrumental for deep learning to become the preferred method of tackling these tasks.

### 2.5.1 Convolutional Neural Networks

We base our following exposition closely on [32].

The design of CNNs takes inspiration from neurology which dates back to the 1960s. In their modern form, they were first conceived by LeCun [57] and successfully applied in the LeNet architecture [58]. In computer vision, CNNs have been the most popular architecture since the 2010s when ResNet [40], a very deep CNN of up to over 100 layers, first achieved human-level performance on the ImageNet Large Scale Visual Recognition Challenge [75].

CNNs are tailored to the vision domain and the regular 2D grid structure of pixel images. This adaptation to the input data structure is sometimes referred to as *inductive bias*. Their basic principle consists of sliding low-resolution matrices, so-called *kernels* or *filters*, over the image to detect local patterns like edges or corners.

This is done by performing *convolutions* between input image and kernels. These are defined by

$$F(i, j) = (I * K)(i, j) = \sum_m \sum_n I(i + m - 1, j + n - 1) K(m, n),$$

where  $I$  is the input image and  $K$  the kernel.<sup>5</sup> The kernel slides over the image and, in each position, an elementwise product between the kernel and a submatrix is performed, then the resulting elements are added up to obtain the entry of the feature map—this is equivalent to flattening both the kernel and submatrix to a vector and performing a dot product.

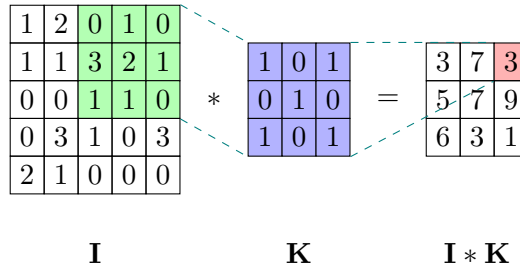


Figure 2.7: Convolution without padding reduces the input size of  $5 \times 5$  to an output size of  $3 \times 3$ .

The ranges of  $i$  and  $j$  are the height and width of the convolution’s output  $F$ , sometimes referred to as *feature map*, that is subject to implementation details. As an example consider figure 2.7 where  $I$  is a  $7 \times 7$  image,  $K$  a  $3 \times 3$  kernel and the convolution is applied only inside of  $I$ , which yields a shape of  $5 \times 5$  for  $I * K$ . This form of convolution reduces the dimension of the feature map compared to the input image which, in some settings, is undesirable. To prevent this reduction, the input image can be *padded*, most

<sup>5</sup>Note that, although the name *convolution* stems from the mathematical convolution operation  $(f * g)(t) = \int_{-\infty}^{\infty} f(s)g(t - s) ds$ , the two are not quite the same. In particular, the convolution operation defined here is not commutative.

commonly by enlarging the original input with zero-entries all-around, such that the resulting feature map will preserve the dimension of the input image, see figure 2.8.

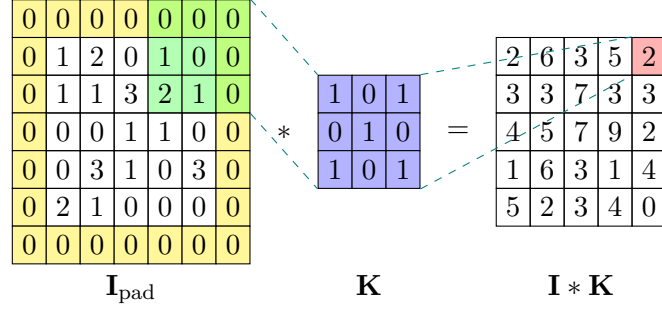


Figure 2.8: Convolution with zero-padding around the original input image to preserve the input dimension.

Note that in figure 2.7 and figure 2.8 we use a fixed kernel. The kernels in a CNN consist of learnable parameters, but the parameters for each kernel are shared across the whole image which dramatically reduces the total number of parameters compared to an MLP: Consider an input dimension of  $N \times H \times W \times C$  with batch size  $N$ , height  $H$ , width  $W$  and  $C$  the number of channels. We assume that the convolution preserves the dimension, then the output has dimension  $N \times H \times W \times C_F$  where  $C_F$  is the number of kernels or *feature map channels*. An MLP would require  $N \times H \times W \times C$  parameters to handle such an input batch of images. A CNN only needs  $C_F \times \text{dim}K$  where  $\text{dim}K$  is the dimension of the kernels which usually is between  $3 \times 3$  and  $7 \times 7$ .

Each convolutional layer is characterized by the size, number of its kernels and the stepsize, called *stride*, with which the kernels pass over the input image. After the convolutions, the activation is applied and these operations are commonly followed by a *pooling layer*: Pooling is a downsampling operation that reduces the spatial dimensions of the input. It works similar to convolutions by sliding a window, also specified by size and stride, over the image. The image values within that window are then aggregated, most commonly by taking the maximum or arithmetic mean, which is called *max pooling* or *average pooling*, respectively. Pooling helps in making the representations more compact and reduces the computational load, but also provides a form of translation invariance: small pixelwise variations in the input will not affect the representation.

A common structure for CNNs is to, layer by layer, successively reduce the image dimension while expanding the feature map dimension which leads to an ever narrower and deeper representation of the input. This part of the network is often referred to as an *encoder* or *feature extractor* as it transforms the input into a compressed representation, sometimes referred to as *features* or *encoding*. Such an encoder can be used as a *backbone* for multiple related tasks by stacking a *task head* on top of it: In image classification, for example, an MLP is often used to map the features provided by a vision encoder (e.g. a CNN) to the class labels. For dense prediction tasks like semantic segmentation, on the other hand, an output of the same dimension as the input is required to build

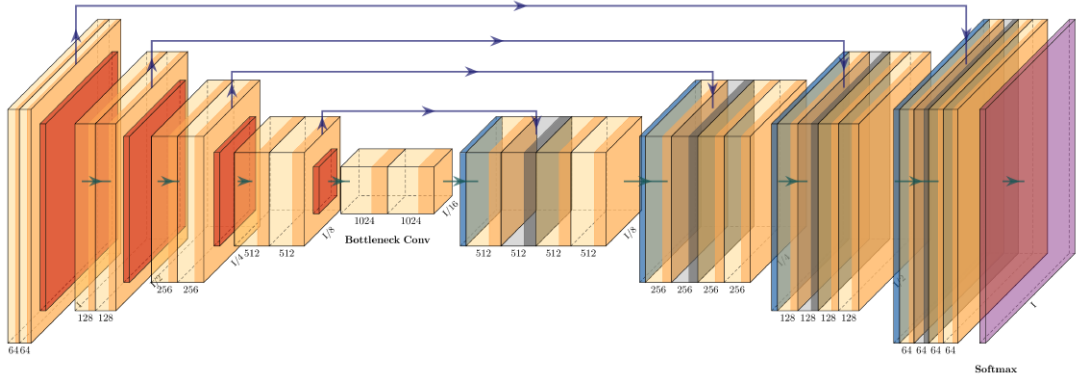


Figure 2.9: The U-Net architecture: A fully-convolutional neural network for semantic segmentation. The orange boxes represent convolutional layers with numbers below indicating their feature map dimensions. The red boxes represent max pooling layers, and the blue boxes the convolutional transpose layers. The arrows between encoder and decoder represent the skip connections. Image courtesy of [39].

a valid segmentation mask. This upsampling process is performed by the network part called the *decoder*: It decompresses the dense encoding into an output with dimension equal to the input. A classic example of this is the U-Net architecture [74]: A semantic segmentation network that consists of only convolutional layers, illustrated in figure 2.9.

The architecture has an encoder-decoder structure with the encoder implemented as a CNN with unpadded convolutional layers with a kernel size of  $3 \times 3$ , each followed by a ReLU activation and a  $2 \times 2$  max pooling operation with stride 2 for downsampling. At each downsampling step, the number of feature channels is doubled.

Every step in the decoder consists of upsampling the feature map followed by a  $2 \times 2$  *convolutional transpose* operation that halves the number of feature channels by spreading out the feature map via multiplication with a learned kernel. For details, we refer to the supplementary materials in [51]. So-called skip connections are used to concatenate the decoder feature maps with the correspondingly sized encoder feature maps. This is followed by two more  $3 \times 3$  convolutions with ReLU activations. At the final layer a  $1 \times 1$  convolution is used to map each 64-dimensional feature vector to the desired number of classes. In total, the network has 23 convolutional layers.

In the last years, CNNs have been challenged and in some scenarios eclipsed by another architecture: *vision transformers*. We will now discuss them in more detail since they provide the *backbone* for the main model used for this thesis.

## 2.5.2 Transformers

Transformers are sequence to sequence models and were initially introduced for natural language processing (NLP), specifically machine translation, in the seminal paper *Attention is All You Need* [94]. From 2023, they became known to the general public due to

their use in *large language models* (LLMs). Such transformers perform *auto-regressive text generation*, meaning they take a text input which is converted to a sequence of *tokens* and successively predict the next token after that sequence. Their key feature is the *attention* mechanism, first introduced in [3]: A method of directing the model’s focus in order to process high-dimensional input more efficiently. We will give an overview of the transformer architecture.

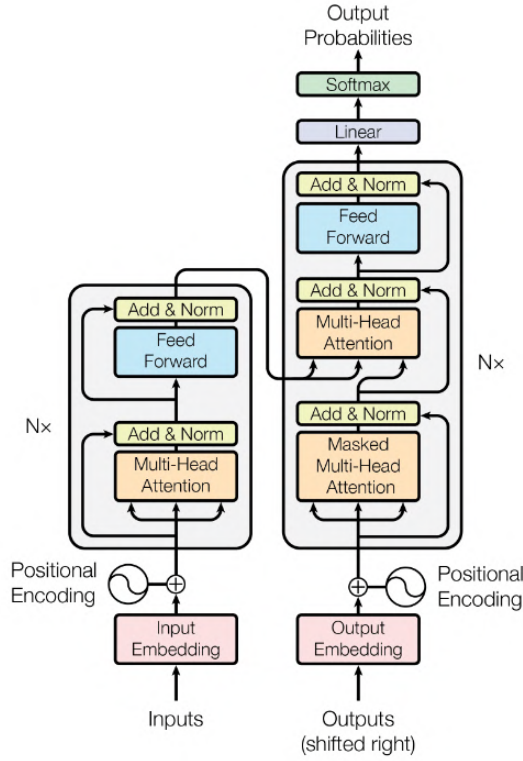


Figure 2.10: Original transformer architecture with encoder on the left and decoder on the right, from [94].

**Token embedding:** Tokens are minimal semantic units and obtained by indexing into a vocabulary of size  $c$  which turns raw text input into a sequence of token indices. These tokens are converted to  $d$ -dimensional *embeddings* via a  $c \times d$  *embedding matrix*  $E$  such that each row in  $E$  corresponds to a unique token embedding.  $E$  is initialized at random and learned during training. To preserve information about the order of tokens in the input sequence, a *positional encoding* is added to each input embedding.

**Scaled dot-product attention:** An *attention head* consists of three matrices  $W_Q, W_K \in \mathbb{R}^{d_k \times d}, W_V \in \mathbb{R}^{d_v \times d}$  that project each embedded input token  $x_i \in \mathbb{R}^d$  to three vectors

$$q_i = W_Q x_i, \quad k_i = W_K x_i, \quad v_i = W_V x_i,$$

called the *query*, *key* and *value*. Then an *attention score* between tokens  $i$  and  $j$  is computed as dot product  $q_i k_j^T$  and converted to weights by a softmax over the second coordinate which are finally applied to the value vector  $v_j$ :

$$\text{Attention}(x_i; W_Q, W_K, W_V) = \text{softmax} \left( \frac{q_i^T k_j}{\sqrt{d_k}} \right) v_j.$$

We can also operate on the whole sequence: By setting

$$X = [x_1, \dots, x_n], \quad Q = W_Q X, \quad K = W_K X, \quad V = W_V x,$$

we obtain

$$\text{Attention}(Q, K, V) = \text{softmax} \left( \frac{Q^T K}{\sqrt{d_k}} \right) V.$$

The scaling term  $1/\sqrt{d_k}$  is an optimization in order to control gradients during training. This form of attention was introduced under the term *scaled dot-product attention* and is usually referred to as *self-attention*.

**Multihead attention:** Multiple attention heads can be computed in parallel and allow the model to focus on different parts of the input simultaneously. Each head performs the above calculations independently, and the results are concatenated and linearly transformed.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h) W_O \quad (2.13)$$

$$\text{where } \text{head}_i = \text{Attention}(Q W_i^Q, K W_i^K, V W_i^V), \quad (2.14)$$

$h$  is the number of attention heads, the projections  $W_i^Q, W_i^K \in \mathbb{R}^{d \times d_k}$ ,  $W_i^V \in \mathbb{R}^{d \times d_v}$  belong to  $\text{head}_i$  and the projection  $W_O \in \mathbb{R}^{h d_v \times d}$  is common to all  $h$  heads.

**Encoder-decoder transformer:** Figure 2.10 shows the original encoder-decoder transformer architecture for machine translation. Here the encoder provides the context by encoding the text to be translated. The decoder predicts the translated sequence token by token.

Each *transformer encoder* layer consists of a multi-head self-attention layer and an MLP on top of that. Several of such layers form a transformer encoder which produces hidden representations, called *encodings*, from either the input embedding and positional encodings (first layer) or the previous layer's encodings. Each *transformer decoder* layer has a *cross-attention* module between the self-attention and MLP. In cross-attention, the query weights are taken from a different token sequence than the key and value. This allows the decoder to condition its output on the input received by the encoder. The decoder eventually predicts the next token based on the input sequence. In order to prevent a token to attend to subsequent tokens, the input of the self-attention module in the decoder is masked.

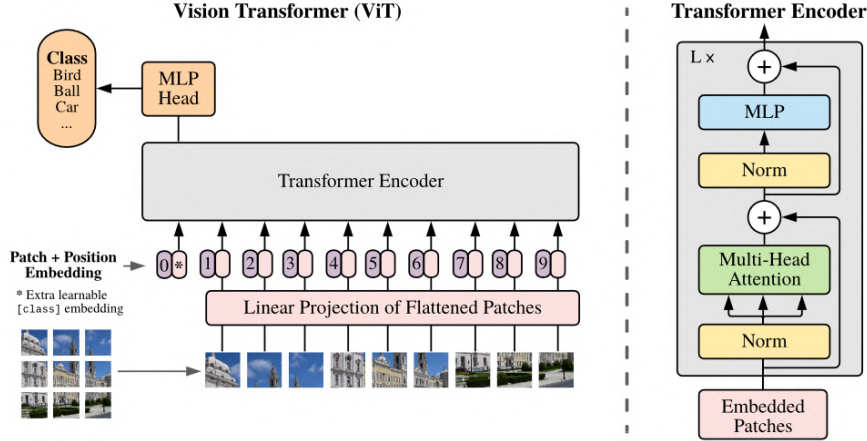


Figure 2.11: Vision transformer architecture, from [21].

### 2.5.3 Vision Transformers (ViTs)

The *vision transformer* (ViT), introduced in [21], adapts the transformer architecture to the task of image classification. The idea, as illustrated in figure 2.11, is to split an image into a sequence of smaller patches, treat each patch as a token, and then process these tokens using a standard transformer model.

The input image  $x \in \mathbb{R}^{H \times W \times C}$  is split into  $N = HW/P^2$  non-overlapping patches of size  $P \times P$  then each patch is flattened, transforming  $x$  into a sequence  $x_p = (x_p^1, \dots, x_p^N) \in \mathbb{R}^{N \times P^2 C}$  of  $N$  flattened patches. As in the standard transformer, each patch/token is projected into a  $d$ -dimensional embedding space by a learnable projection matrix  $E \in \mathbb{R}^{d \times P^2 C}$ . The *patch embeddings* are prepended by a *class token*  $x_{\text{class}}$  which encodes the class label and learnable *position embeddings*  $E_{\text{pos}} \in \mathbb{R}^{d \times (N+1)}$  are added to retain positional information about the patches. The resulting sequence

$$\left[ x_{\text{class}}, Ex_p^1, \dots, Ex_p^N \right] + E_{\text{pos}}$$

is fed into a standard transformer encoder as described above. The obtained encodings are then processed through an MLP that serves as task head for image classification.

ViTs have shown competitive performance on various vision benchmarks and have become the dominant backbone architecture in some, especially when pretrained on large datasets. They have successfully been adapted to more complex computer vision tasks like object detection and semantic segmentation.

### 2.5.4 Domain Generalization

It is crucial for deep learning models to perform well on new data that differs significantly from the training data. The domain a model operates on during training or inference is characterized by the data generating distribution (see section 2.1). The capability

of a model to perform consistently under a shift of that distribution is called *domain generalization*.

More formally: As before, let  $\mathcal{X}$  and  $\mathcal{Y}$  be our input and output space. A *domain* is characterized by a sample sequence (or set)  $\mathcal{S} = \{(x^n, y^n)\}_{n=1}^N$  of data points  $x^n$  and corresponding labels  $y^n$  and the corresponding data generating distribution  $\mathcal{D}$ , i.e.  $\mathcal{S} \sim \mathcal{D}$ . Domain generalization means that we have access to  $M$  training domains  $(\mathcal{S}_1, \mathcal{D}_1), \dots, (\mathcal{S}_M, \mathcal{D}_M)$  and want to learn a model  $\Phi_\theta : \mathcal{X} \rightarrow \mathcal{Y}$  that minimizes the error on an unseen target domain  $(\mathcal{S}^*, \mathcal{D}^*) \neq (\mathcal{S}_i, \mathcal{D}_i)$  for all  $i = 1, \dots, M$ :

$$\Phi_\theta \approx \arg \min_h \mathbb{E}_{(x,y) \sim \mathcal{D}^*} [\mathcal{L}(h, (x, y))]$$

where all  $(x, y) \in \mathcal{S}^*$ .

In real-world applications, the data encountered during deployment can differ significantly from the data used during training. This difference, known as *distribution shift*, can arise due to various factors. Many machine learning applications need to function across diverse environments without retraining. For example, an autonomous vehicle must operate reliably in different localities, weather and lighting conditions, even if it was trained in a limited set of environments. Collecting labeled high-quality training data for every possible target domain may be impractical in many situations. Domain generalization techniques allow models to leverage existing datasets and perform well in new domains without the need for extensive new data collection and labeling. Moreover, models that generalize well to different domains tend to perform better overall. Techniques that promote domain generalization often lead to the development of more robust and flexible models that can handle a wider variety of inputs and conditions.

To achieve domain generalization, several techniques and strategies are employed, among others:

- Regularization methods including the once discussed here improve generalization performance.
- *Data augmentation*: Introducing variability in the training data through transformations such as rotation, scaling, and cropping helps models learn invariant features.
- Domain-invariant features: Learning representations that are invariant to domain-specific characteristics can improve generalization (see section 2.6)

Another approach common in today’s deep learning practice is *transfer learning*: A *foundation model* is *pretrained* for one task on a very large dataset and then *finetuned* for other downstream tasks on smaller datasets. For example, most semantic segmentation models use an encoder-decoder architecture where the image encoder was often pretrained for the simpler task of image classification on internet scale data of hundreds of millions up to billions of images. The segmentation model uses this encoder as a feature extractor and attaches a task-specific decoder. This model with pretrained weights for the image encoder part is then finetuned on curated datasets for semantic segmentation.



## 2.6 Self-supervised Multimodal Learning

In section 2.1 we introduced the supervised learning framework where we train a model to assign true labels to data. This requires a set of annotated data that the model can learn from. This annotation usually requires human effort: Images need to be assigned a class, segmentation maps or bounding boxes for objects must be drawn. Though tremendous collective efforts have been made to create large annotated datasets in some domains, the almost 15M labeled data points in ImageNet-21k [73] being a prime example, this approach faces inherent limits: There are way more data in the world than can ever be sensibly annotated manually by human labour. A different approach is to forgo manual annotation and create a valid learning signal for the model from raw data in a fully automated way. This approach is called *self-supervised learning* and responsible for some significant advancements in recent deep learning [19, 8, 35, 41, 9, 71, 90].

*Multimodality* in deep learning refers to the ability of neural networks to process and integrate information from multiple types or modalities of data, such as text, images, audio, and video, which has gained significant importance in recent years. As a prominent example of a multimodal deep learning model we will introduce CLIP (Contrastive Language-Image Pre-training) [71].

The method behind self-supervised representation learning often consists of teaching a model to align different, but equivalent representations of input data, either across different modalities, like image-text pairs, or within a single modality, like two version of the same original image that have been altered by different transformations. An inherent problem with this approach is that there is the trivial solution of just zeroing out all representations in order to align them. One way to tackle this problem is called *contrastive learning*: We present the model with positive and negative examples, i.e. true and false pairs of representations and design a loss function in a way that encourages the model to pull the representations of positive pairs together and push the representation of negative pairs apart.

### 2.6.1 CLIP

CLIP is designed to learn visual concepts from natural language supervision, enabling it to understand and associate images with their textual descriptions. Figure 2.12 illustrates the model’s approach.

CLIP receives a batch of  $N$  pairs  $(I_k, T_k)$  of images  $I_k \in \mathbb{R}^{H \times W \times C}$  and text descriptions  $T_k$  that come from a dedicated dataset of 400 million such pairs collected from publicly available sources on the Internet. It is then trained to match the true image-text-pairings across a batch.

This happens as follows: Each image  $I_k$  from a batch is passed through an image encoder, CNN or vision transformer, and each text  $T_k$  through a transformer text encoder. The extracted features from the respective encoders are then projected by a learnable linear projection to a common  $d$ -dimensional embedding space and normalized. For each pair  $(I_k, T_k)$  in the batch, the model computes a similarity score in form of the dot product, resulting in  $N^2$  pairings for a batch size of  $N$ . Over these scores, it optimizes a

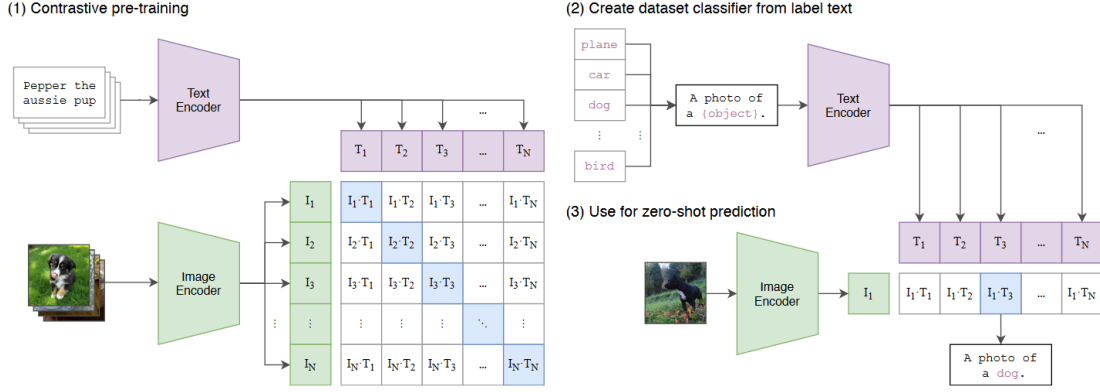


Figure 2.12: CLIP approach, from [71].

symmetric cross entropy loss such that the similarities (dot products) over the  $N$  true pairs are maximized and those over the  $N^2 - N$  false pairs are minimized.

CLIP excels at zero-shot learning, meaning it can generalize to various tasks on various datasets without additional training. CLIP has been evaluated on over 30 different datasets, demonstrating impressive zero-shot performance in tasks such as fine-grained object classification, geolocalization, action recognition in videos, and optical character recognition.

The reasons for the performance gains exhibited by vision-language models like CLIP are an open research question [24, 18, 49]. We will revisit this discussion in the method and discussion chapters.

### 2.6.2 Related Work on Camera-Lidar Contrastive Learning

Before introducing our method of multimodal representation co-learning between camera images and LiDAR point clouds, we mention some previous work on this topic that has been rather underexplored so far. There are, however, some previous works related to our approach.

Jiang and Saripalli have demonstrated the effectiveness of cross-modal representation learning between camera images and lidar point clouds with a custom Tuple-Circle loss function and network architecture based on ResNets and PointNet++ [52].

In LidarCLIP [43], lidar point clouds are mapped into a pre-existing CLIP embedding space: Using image-lidar pairs, a point cloud encoder (Sparse Single-Stride Transformer [23]) is trained for alignment with the CLIP image embeddings, relating text and lidar data with the image domain as an intermediary. Notably, this is achieved by optimizing a mean squared error or cosine similarity loss in order to allow for smaller batch sizes than is usually recommended for contrastive loss [8]. LidarCLIP shows remarkable performance in multiple tasks, including zero-shot classification, various retrieval tasks and even lidar-to-text and lidar-to-image generation.

Finally, LIP-Loc [82] applies a very CLIP-like method to the domains of image and

lidar point clouds on the task of cross-modal localization: They optimize a symmetric cross-entropy loss over similarity scores and, despite a relatively small batch size of 32, find it outperforms the triplet loss previously employed in this setting [105]. Instead of using a dedicated point cloud encoder, LIP-Loc first converts the point clouds to range images [99] and then obtains embeddings by processing them through the same vision transformer image encoder as the camera images.

## 3 Method

We first give a motivation for camera-lidar contrastive learning. Then we present a heuristic argument for the intrinsic value to contrastive multimodal representation learning. Finally, we propose a method to test this hypothesis.

### 3.1 Motivation

The image representations obtained by *vision-language models* (VLMs) like CLIP and its variants like OpenCLIP [10] and EVA-CLIP [90, 89] have demonstrated superior performance on various tasks, specifically in zero-shot settings, when compared to image encoders pretrained only in the image modality [50]. This success raises the question of why these multimodal foundational models outperform their unimodal counterparts across multiple tasks: Is it primarily the vast scale of their pretraining datasets that allows them to learn more robust representations? Or is there an intrinsic value attached to contrastive representation learning across multiple modalities?

Motivated by the success of vision-language-models, specifically to increase domain robustness in computer vision, the goal of this thesis is to explore contrastive multimodal representation learning between other modalities than image and text. For the autonomous driving setting, lidar point clouds and camera images are a suitable modality pairing because the combination of camera and lidar sensors is a common setup in available autonomous driving datasets. We attempt to replicate the learning of domain robust representations that improve domain generalization demonstrated by CLIP models for this, as to this objective, so far underexplored pairing of modalities.

### 3.2 An Argument for Multimodality

The following reasoning is adopted from [50], one of the first publications to leverage the domain generalization capacity of vision language models for the task of semantic segmentation in autonomous driving.

As in section 2.4, let  $\mathcal{X}$  be the space of pixel images of height  $H$  and width  $W$ . Our model  $\Phi$  is given by an encoder-decoder network for semantic segmentation as in section 2.4.3, i.e.  $\Phi = \Phi_D \circ \Phi_E$  with  $\Phi_E, \Phi_D$  the encoder and decoder, respectively, where the encoder  $\Phi_E$  maps each input image  $x$  to a high-dimensional feature representation  $\Phi_E(x) \in \mathcal{E}$ . We refer to  $\mathcal{E}$  as the embedding space.

Now, consider a source domain  $\mathcal{D}_S \subset \mathcal{X}$ , given by a dataset and data distribution (see section 2.5.4), and a target domain  $\mathcal{D}_T \subset \mathcal{X}$  where we assume that the domain shift from  $\mathcal{D}_S$  to  $\mathcal{D}_T$  is realized by a bijective map  $\phi : \mathcal{D}_S \rightarrow \mathcal{D}_T$ . As a standard example relevant

to the autonomous driving setting, we may imagine a given frame (traffic scene)  $x$  in daylight and  $\phi(x)$  being the same frame at night. Changes in weather conditions and locality are also a common challenge for autonomous driving systems [7, 25].

In multimodal representation learning, the model receives representations from multiple modalities to each datapoint as input. In VLMs, for example, there is a corresponding text description  $T(x)$  for each image  $x$ . In our setting, we have access to a lidar point cloud representation  $L(x)$  for each camera image  $x \in \mathcal{D}_S \cup \mathcal{D}_T$  and a lidar point cloud encoder  $\Psi_E$  that maps point clouds  $L(x)$  into the embedding space  $\mathcal{E}$ , shared between point cloud and image representations.

In order to leverage the multimodal representation to make the model  $\Phi$  more domain robust, the following two assumptions are sufficient.

First, the co-modal representation has to be invariant under the domain shift with high probability. For the domain shift from day- to nighttime and the lidar point cloud representation, this condition is perfectly met since their operating principle makes lidar sensors independent of lighting conditions and thus invariant under the day-night domain shift  $\phi$ :  $L(\phi(x)) = L(x)$  for all  $x \in \mathcal{D}_S \cup \mathcal{D}_T$ . This fact is another reason why lidar point cloud constitutes a suitable co-modality.

Second, the cross-modal alignment, in our case between camera image and lidar point cloud features, needs to be exact, i.e.  $\Phi_E(x) = \Psi_E(L(x))$  for all  $x \in \mathcal{D}_S \cup \mathcal{D}_T$ .

Under these two assumptions, we may infer

$$\Phi_E(\phi(x)) = \Psi_E(L(\phi(x))) = \Psi_E(L(x)) = \Phi_E(x).$$

Applying the decoder  $\Phi_D$  to both sides yields

$$\Phi(\phi(x)) = \Phi(x),$$

in other words: The model  $\Phi$  is robust under the domain shift  $\phi$ .<sup>1</sup>

While the first assumption, domain-shift-invariance of lidar point cloud representation, is conceptually and empirically justified, the second assumption of exact feature alignment between images and point clouds is a lot harder to guarantee in practice. For one, it is not a trivial task to train image and point cloud encoders for exact alignment in the first place. Also, there might be a trade-off between this alignment objective and the objective of obtaining image representations that are suitable for the downstream task, semantic segmentation in this case.

### 3.3 Camera-Lidar Contrastive Learning

We adopt the contrastive pretraining approach demonstrated by CLIP [71] in order to specifically learn whether pretraining the image encoder of a semantic segmentation model for alignment with a lidar point cloud encoder can increase the image encoder’s representations with respect to domain generalization.

Our method consists of three stages:

---

<sup>1</sup>This is true only with the same probability that the first assumption is met. For the lidar point cloud representation and a domain shift only involving lighting conditions, this probability equals 1.

1. **Multimodal Pretraining** of an image encoder for alignment of pairs of camera images and lidar point clouds by optimizing a contrastive loss equivalent to CLIP.
2. **Finetuning** the image encoder for the task of semantic segmentation.
3. **Evaluation** of semantic segmentation performance on an unseen target domain to measure domain generalization.

To assess the significance of the multimodal pretraining stage, we compare two models of the same architecture: One undergoes both training stages outlined above (pretraining and task-specific finetuning) and the other one is directly finetuned.

## 4 Numerical Experiments

We will first introduce the experiment setting by giving an overview of the architectures, datasets and metrics, then proceed with a detailed presentation of our experiments and results.

### 4.1 Overview

#### 4.1.1 Architectures

**Encoder** We use the `vit_small_patch16_224` vision transformer model from Pytorch Image Models (timm) library [97] as image encoder for all experiments. This model was pretrained for the task of classification on ImageNet [17]. It is a more compact version of the standard vision transformer model introduced in section 2.5.3.

As lidar encoder, we use a MinkUNet34-W32 [11] with torchsparse backend [91] from MMDetection3D library [13] that was pretrained on SemanticKITTI dataset [4]. MinkUNet is an efficient 3D segmentation model based on the U-Net architecture presented in section 2.5.1. It uses a voxel representation of the point clouds and then performs sparse convolution operations. Figure 4.1 shows that the architecture is essentially equivalent to U-Net, except for the replacement of 2D pixel representation by 3D voxels and the regular 2D convolutions by 3D sparse convolutions.<sup>1</sup>

**Decoder** To make the image encoder backbone applicable to semantic segmentation, we attach a Segmenter [86] mask decoder head to the backbone. This purely transformer-based segmentation head relies only on the output embeddings from the image encoder

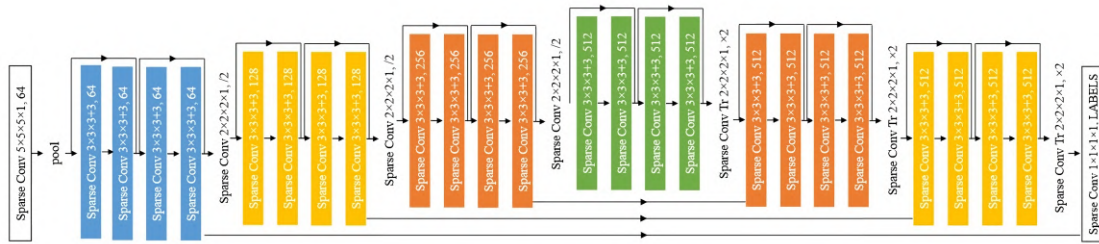


Figure 4.1: The MinkUNet architecture. Image from [11].

<sup>1</sup>The actual representation is even 4D due to an added time dimension as the model was designed to handle video input.

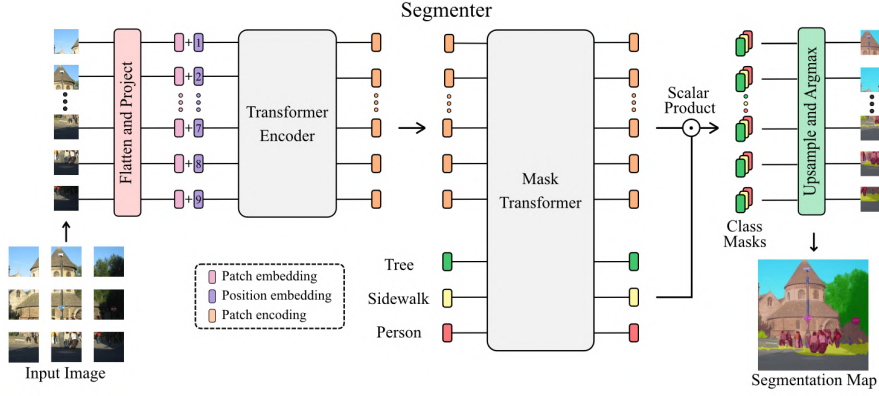


Figure 4.2: Segementer model. (Left) Encoder: The image patches are projected to a sequence of embeddings and then encoded with a transformer. (Right) Decoder: A mask transformer takes as input the output of the encoder and class embeddings to predict segmentation masks. Image and Caption from [86].

to build the segmentation maps and thus offers the benefit of being directly compatible with the vision transformer backbone. The architecture is illustrated in figure 4.2.

#### 4.1.2 Datasets

LIP-Loc pretrains on KITTI [28] and KITTI-360 [61] datasets. Released in 2012, KITTI is among the first and most popular datasets for autonomous driving, containing around 15,000 images of urban and suburban areas in good weather conditions. KITTI-360 is a large scale 3D video dataset comprising 300,000 images and laser point clouds of suburban scenes with consistent semantics in both 2D and 3D.

A2D2 dataset [29] contains 41,277 semantically annotated images, 38,481 of those with corresponding lidar point clouds. The image distribution is more focused on highways, country roads and suburban scenes, but also includes three German cities. It features various times of day and sunny to rainy weather conditions.

Finetuning for semantic segmentation is conducted on Cityscapes [15]. The dataset includes a training subset with 2,975 samples and an evaluation subset with 500 samples, overall 5,000 finely and 20,000 coarsely annotated images. Data were gathered in 50 different Germany cities and focus on urban traffic scenes.

To assess domain generalization, we perform evaluation on ACDC dataset [78] which features three adverse weather conditions and nighttime conditions and the same semantic classes as Cityscapes. The dataset consists of 4,006 images equally distributed between the four adverse conditions and another 4,006 images of the same scenes in good condition. Overall 5,509 images are annotated for panoptic segmentation.

Figure 4.3 allows to compare the distribution of the three datasets.



### 4.1.3 Metrics

We compare the mean intersection over union (mIoU) for 19 segmentation classes on ACDC dataset after the models have been pretrained on KITTI/KITTI-360 (LIP-Loc) or A2D2 (ours) and finetuned on Cityscapes. As a secondary metric we consider the *relative performance under domain shift* (rPD), defined as the quotient of mIoU on the target domain (ACDC) over mIoU on the source domain (Cityscapes):

$$\text{rPD} = \frac{\text{mIoU}_T}{\text{mIoU}_S}.$$

Our main interest is to observe how these metrics compare between an image encoder that was pretrained on the unimodal task of image classification on ImageNet and an image encoder that has passed through multimodal lidar-camera contrastive pretraining. The hope is that the domain invariant properties of lidar point clouds can guide the camera image encoder to increase domain robustness, as has previously been observed for vision-language models.

## 4.2 Details and Numerical Results

We conduct our experiments in two stages: In the first, we merely finetune the image encoder directly from timm or after additional pretraining by LIP-Loc on Cityscapes and evaluate that finetuned encoder on ACDC. Since the model has not seen ACDC with its very pronounced adverse conditions before, this is a challenging test for domain generalization. In the second stage, we take the same image encoder as LIP-Loc, but conduct our own contrastive pretraining.

### 4.2.1 LIP-Loc and timm-ViT-S

To avoid ambiguity, we clearly define the models used in our experiments as follows: *timm-ViT-S* refers to the `vit_small_patch16_224` image encoder from timm library. It is always assumed that this model is initialized with weights that were obtained by pretraining for image classification on ImageNet.

We use two LIP-Loc models to finetune and evaluate with our approach, both initialize the timm-ViT-S model and perform contrastive pretraining on top of it: Their main model, named `exp_largest` (ViT) in [82], we refer to as *LIP-Loc-KITTI*. It was trained on 18 out of 20 KITTI sequences. They also provide a model that has seen significantly more training data, named `exp_combined_vit` in the paper, which was trained on the same KITTI sequences plus 8 additional KITTI-360 sequences. We refer to this model as *LIP-Loc-KITTI+360*.

LIP-Loc uses the same contrastive loss function as CLIP and a batch size of 32. The batch size is of special significance in contrastive learning, as it determines the ratio of positive to negative pairs in a contrastive learning setting. Large batch sizes are usually thought to be necessary or at least beneficial for contrastive learning [43, 8]. As projection head, LIP-Loc uses an MLP with two hidden layers, GELU activation, dropout and layer

normalization. The exact training settings are not reported in the paper, but can be traced in the GitHub repository.<sup>2</sup>

LIP-Loc does not employ a dedicated lidar point cloud encoder, but instead converts the point cloud data to range images and encodes them also with a timm-ViT-S, just like the images.

#### 4.2.2 Finetuning Pipeline

We finetune those 3 backbone models with the Segmenter mask decoder head from MMSegmentation [14] and conduct all finetuning in this framework. This decode head has 2 layers and 6 attention heads which corresponds to the Seg-S model in [86]. The embedding dimension of 384 is given by the timm-ViT-S backbone. Our hyperparameter choices are based on [86]: We train with a batch size of 8 for 80,000 iterations on Cityscapes. As optimizer, we use SGD with a momentum of 0.9 and no weight decay, a base learning rate of  $10^{-2}$  and a polynomial decay schedule with minimum learning rate of  $10^{-4}$ . The input image size of  $224 \times 224$  is determined by the image encoder and standard image augmentations are applied: random resize, crop and flip as well as photometric distortions.

We train all 3 models in two modes: Either by training just the Segmenter head on top of a frozen timm-ViT-S backbone, i.e. the backbone parameters are no longer updated by SGD, or by fully finetuning the whole model. As expected, finetuning the whole model yields better segmentation performance, given as mIoU on Cityscapes validation set. The results for all three models shown in table 4.1 are comparable and top out just below 70 mIoU.

This is significantly lower than the around 80 mIoU [86] reports for the larger base and large ViT variants with the same patch size. The reason for this difference is not clear, since we used almost equal experiment settings, and the magnitude is unlikely explained by the smaller model capacity alone. Nevertheless, we continue with those settings without optimization as we are more interested in the relative performance of different models, rather than the absolute performance of our pipeline.

The results, exhibited in table 4.1, indicate a slight edge for the two LIP-Loc models in terms of segmentation performance and domain generalization, measured as absolute performance after domain shift. Other experiments we run show the same pattern: LIP-Loc models with a marginal lead on Cityscapes, but without showing improved relative performance under domain shift to ACDC. As an example, Table 4.1 shows two more models, indicated by “B64”, that have been trained for the same 80,000 iterations, but with a batch size of 64. Again, LIP-Loc performs marginally better than timm-ViT-S on Cityscapes, but with no gain on ACDC. On the contrary: Both models show significantly worse domain generalization which we interpret as simple overfitting due to the excessive number of training iterations, considering that Cityscapes is quite a small dataset with only 2,975 training examples.

Overall, the two LIP-Loc models have a marginal edge over the timm-ViT-S. However,

---

<sup>2</sup><https://github.com/Shubodh/lidar-image-pretrain-VPR>

<b>Experiment</b>	<b>mIoU (Cityscapes)</b>	<b>mIoU (ACDC)</b>	<b>rPD</b>
timm-ViT-S	68.01	37.93	0.56
timm-ViT-S (fb)	58.21	40.00	0.69
timm-ViT-S B64	69.50	33.24	0.48
LIP-Loc-KITTI	69.43	39.75	0.57
LIP-Loc-KITTI (fb)	60.42	38.40	0.64
LIP-Loc-KITTI B64	70.39	32.53	0.46
LIP-Loc-KITTI+360	68.79	40.10	0.58
LIP-Loc-KITTI+360 (fb)	60.28	38.95	0.65
CLCP (Lin) b128	67.79	38.23	0.56
CLCP (Lin) b256	68.76	37.94	0.55
CLCP (Lin) b512	68.27	39.73	0.58
CLCP (Lin) b768	68.68	37.07	0.54
CLCP (Lin) b768 (fb)	49.82	33.04	0.66

Table 4.1: Comparison of mIoU on Cityscapes and ACDC validation sets and rPD between those for timm-ViT, LIP-Loc models and our own initial pretraining runs, indicated by CLCP. For those we specify the batch size and (Lin) indicates that we use a linear projection head on the lidar encoder during pretraining. (fb) indicates models for which only the Segmenter task head was finetuned, while the ViT backbone remained frozen.

we judge the difference to be too small to yield any practical advantage and, at least for this specific setting, consider the effect of multimodal pretraining on domain generalization negligible. It is possible that the LIP-Loc models just perform better on Cityscapes because they have seen many more examples of traffic scenes during their pretraining on KITTI and KITTI-360, respectively. Considering the amount of LIP-Loc pretraining – 100 epochs on either KITTI (LIP-Loc-KITTI) or even KITTI plus a significant amount of KITTI-360 (LIP-Loc-KITTI+360) –, it is surprising to us how similar the models perform compared to timm-ViT-S. We conjecture that this is at least in part due to the fact that LIP-Loc uses range images and passes them through the same image encoder as the corresponding RGB images. We take this conjecture as motivation to use a different approach and employ a dedicated lidar encoder, thus giving somewhat more weight to the characteristics of the lidar point cloud modality.

### 4.2.3 Contrastive Pretraining

We establish our own pretraining pipeline in order to check whether we can reproduce the results obtained by the LIP-Loc encoder weights or even improve upon them under different conditions. Inspired by the LidarCLIP GitHub repository [43], we set up our own pretraining pipeline with Pytorch Lightning [22]. We will now introduce this pipeline in detail and present the numerical results. As the general approach closely mirrors CLIP we refer again to figure 2.12.

### 4.2.4 Pretraining data pipeline

Our dataset consists of the front center camera subset of A2D2 dataset: 28,637 RGB images and corresponding lidar point clouds. We split off 15 percent, around 4,285 images, as validation set and retain around 24,352 training examples. The dataset is arranged in 21 scenes of consecutive driving, where consecutive frames within a scene often lie only seconds apart. We split off the last 15 percent of images from every scene, rather than picking whole scenes, for validation. This way we align the distribution of training and validation data and preserve the diversity of the dataset, as the individual scenes are, overall, quite diverse.

We preprocess the original  $1,920 \times 1,208$  images by random crop to a resolution of  $896 \times 896$ , then resize them to the  $224 \times 224$  input size of the timm-ViT-S image encoder. The random crop is applied simultaneously to the point clouds. The random crop and resize serves as a basic data augmentation, but also compresses the input data and thus enables larger batch sizes during training. In determining an appropriate crop size, there are two competing goals: For one, we want the augmentation and data compression via cropping and resizing the images. But we also want to retain enough points for a meaningful scene representation: A2D2 contains a variety of scenes, thus the size of the point clouds varies greatly from only hundreds to several thousand points. If we crop too much of the original image and the corresponding point clouds, a lot of point clouds will end up with only few or even no points left. We experimented with smaller crop sizes, but found that  $896 \times 896$  strikes a good balance by retaining on average around 35

percent of lidar points and leaving only few very sparse point clouds. The random crop and resize is the only data augmentation we apply consistently for all experiments. More aggressive augmentations are used in ablations.

#### 4.2.5 Pretraining architecture

The two encoders for point clouds and images, respectively, provide features that we aggregate and project into the joint embedding space in order to obtain embeddings that can be aligned as to their cosine similarity (dot product). The model then optimizes the matching between corresponding embeddings across a batch.

**Lidar Encoder** To make the point clouds comparable to the corresponding images, we map their features into a joint embedding space for which we choose a dimension of 384 – the same as the image encodings from the timm-ViT-S.

To obtain global lidar encodings from the MinkUNet, we either just use a linear projection like CLIP or an MLP, like suggested in [8] and used by LIP-Loc. In order to avoid an information bottleneck by pooling the point cloud features to a dimension of 96, then projecting them up into the 384-dimensional embedding space, we apply the projection before average pooling. The same holds for the case when replacing the linear projection by an MLP with 2 hidden layers and GELU activation, which, except for the dimensions, is the same projection as LIP-Loc uses.

Note that the MinkUNet backbone is actually an encoder-decoder for semantic segmentation and extracts voxelwise features of dimension 96.<sup>3</sup>

We considered other choices for the lidar encoder, like PointNet [26], PointPillars [56] or state of the art models like Point Transformer v3 [100], PointGL [60] or DSVT [95]. Our choice was guided by the following considerations: For one, we want to use an encoder architecture that has a proven performance record when applied to autonomous driving datasets. The sparse point clouds obtained by lidar devices in outdoor settings are quite distinct from the more dense point clouds used for indoor scene understanding and 3D object detection. So we prefer an encoder that matches our setting. Another important aspect is the availability of pretrained weights, since we want to avoid having to train a lidar encoder from scratch. Unexpectedly, these requirements ruled out quite a lot of options, specifically the more recent models. In the end we choose the MinkUNet34-W32 model from MMDetection3D library also because it was straightforward to implement. It is still considered a state of the art architecture for 3D semantic segmentation [87] and fulfills our requirements with one drawback that it was not specifically designed for global feature extraction.

---

<sup>3</sup>We briefly considered extracting only the encoder features which have dimension 256 and are spatially more condensed. But it is challenging to track which features belong to which item in a batch through multiple layers of sparse 4D convolutions and considerable effort would have had to be spent for implementing, debugging and testing a working solution. So we opted for the simple out-of-the-box approach of processing the voxelized features.

**Image Encoder** The situation for the image encoder is way simpler: We just use the same timm-ViT-S as LIP-Loc used which allows for valid comparisons. This encoder already provides global image encodings of dimension 384. We still send them through a linear projection into the embedding space, rather than using the unaltered encodings as embeddings. This approach is more flexible as it allows for varying embedding dimensions and the computational overhead is negligible.

**Loss** We implement a loss function equivalent to CLIP [71, page 5] with a learnable temperature parameter, always initialized as the same 0.07.

#### 4.2.6 Pretraining Results

All models that we pretrain ourselves are referred to as CLCP for camera-lidar contrastive pretraining. For our initial pretraining runs, we opt to train only the image encoder, keeping the lidar encoder frozen – except for the projection to the embedding space. There are two reasons for this: For one, we don’t need the lidar encoder after pretraining. Training with the lidar encoder backbone frozen is much more efficient because it is the bigger of the two models with 38 M vs. 22 M parameters for the image encoder. Not training the lidar encoder significantly speeds up the training and enables bigger batch sizes – up to 768. There is also the concern that co-adaptation between lidar and image encoder might diminish the perception quality given by the pretrained weights in both encoders over time – this turned out to be unjustified, as we show below.

We guessed initial hyperparameters without any optimization, choosing a base learning rate of  $10^{-4}$  and AdamW optimizer with  $\beta = (0.9, .0999)$ ,  $\epsilon = 10^{-8}$  and a weight decay of  $10^{-5}$ . As does CLIP, we used a cosine annealing schedule [62] with minimum learning rate of  $10^{-6}$  and restarts every 10 epochs. We pretrain batch sizes of 128, 256 and 768 for 100 epochs, while the run with batch size 512 terminated at epoch 69 due to the early stopping criterion of no improvement in validation loss for 5 epochs.

We proceed to finetune all models on Cityscapes with the same settings exhibited in section 4.2.2 and to evaluate them on ACDC without further finetuning. The results, shown in table 4.1, are comparable to the LIP-Loc models and the timm ViT – with a slightly worse segmentation performance overall. We also checked the effect of freezing the image encoder backbone during finetuning, as we did for timm and LIP-Loc models. The performance drop for our model is even more pointed. We attribute this to the fact that that our camera-lidar contrastive pretraining does, in a way, too much to the ImageNet initialized encoder, diminishing its capabilities without enough compensation. This finding is also evidenced by our further experiments and we will revisit the point in section 5.1.

After these initial findings, we proceed by first optimizing the pretraining pipeline in order to facilitate faster runs. With these improved training settings, that yield a much faster convergence, we go on with further experiments. As training optimizations, we experiment with a one cycle learning rate schedule like LidarCLIP [43] after testing for highest convergent learning rate [84], but find the cosine schedule consistently more performant. We test the latter in two base settings used in [62]: a constant learning

rate cycle of 10 epochs or a doubling learning rate cycle starting at 1 epoch – both prepended by a linear warm up phase of 5 epochs. The variant with doubling cycles yields consistently faster convergence and a lower minimum validation loss so we continue with this learning rate schedule. We use a batch size of 256 for all further pretraining as the larger sizes of 512 and 768 do not fit in memory for one GPU with the full unfrozen lidar encoder – the bigger batch sizes don’t seem to offer significant advantages except faster training as per our initial experiment results, table 4.1. Base and minimal learning rate as well as weight decay we optimize by a grid search. Our final hyperparameter settings for all pretraining runs are shown in table 4.2.

Hyperparameter	Value
Batch size	256
Base learning rate	$10^{-3}$
Minimal learning rate	$10^{-5}$
Weight decay	$10^{-2}$
Learning rate schedule	Cosine with restarts
Warm-up period	5 epochs
Initial cycle length ( $T_0$ )	1
Cycle length multiplier	2
AdamW $\beta$	(0.9, 0.98)
AdamW $\epsilon$	$10^{-8}$

Table 4.2: Hyperparameters for camera-lidar contrastive pretraining (CLCP).

With these optimizations we run more pretrainings under different combinations of the following ablations: We unfreeze all lidar encoder weights, making it fully trainable, switch to an MLP projection head on the lidar encoder or apply additional image augmentations.

Figure 4.4 shows the difference in training efficiency between our initial run with batch size 256 and the optimized version. We stopped training for the optimized models at 67 epochs right at the schedule learning rate bump, as the models achieved a sufficiently low loss at this point. Figure 4.4 also illustrates the greatly improved convergence of the fully trained model versus the one with frozen lidar encoder. Although the significant gap between validation and train loss might indicate overfitting, we find that the val-train loss gap is significantly lower for the fully trained model with overall much higher capacity.

Our next step was to further increase model capacity by switching to the MLP projection head on the lidar encoder. We trained this variant also with frozen and unfrozen lidar encoder backbone and compare it to the equivalents with linear projection head in figure 4.5. The switch to the MLP projection head only impacts learning significantly if the lidar backbone remains frozen – clearly, the MLP head is more powerful than the simple linear projection in this case. However, when unfreezing the whole lidar encoder, there is no discernable difference left – compare dark blue to left line. For the MLP variant we can also confirm the earlier finding that training the whole model not only improves convergence, but also narrows the gap between train and validation loss.

This result is encouraging towards the contrastive alignment goal: The fully trainable model exhibits the capability of matching lidar and image embeddings without losing generality, at least for the task of matching point clouds and images, as is shown by reduced validation loss and its even narrowing gap to the train loss.

Our last pretraining variation is to incorporate additional image augmentations into the pipeline: We apply with a probability of  $p = 0.9$  a suite of pixelwise augmentations on the input images: Either a moderate color jitter ( $p = 0.7$ ) or a significant darkening ( $p = 0.2$ ), or conversion to grayscale ( $p = 0.1$ ). On top of those, we randomly apply with chance  $p = 0.5$  one variety of noise, chosen between Gaussian blur, Gaussian noise and glass blur with equal probability. The effect of augmentations is illustrated on a random image sample in figure 4.6.

Overall, figure 4.7 shows a negligible effect of those augmentations with respect to the validation loss, but a significant increase in train loss: As expected, the model has a harder time learning the augmented images, but performs equally on the unaugmented validation data. This finding is consistent between both model variants: MLP lidar encoder with or without frozen backbone.

After performing these ablations in pretraining, we are left with the choice of which models, and in what state, we take through finetuning and final evaluation. Due to the cyclic learning rate schedule with increasing cycle length, we obtain local minima for train and validation loss at epochs 11, 19, 35 and 67. At this point we halt training for most runs because the optimized models have basically converged to a train loss near zero. We choose a variety of models with different characteristics and checkpoints from the above-mentioned epochs for finetuning. The results in table 4.3 show that the segmentation performance across both Cityscapes and ACDC deteriorates with additional training. Specifically, none of the optimizations we perform in the pretraining stage has any benefit for domain generalization on ACDC – maybe except that there is a faint signal that the fully trained linear model (Lin) performs slightly better than the equivalent with frozen lidar backbone (Lin, fb). A possible explanation is that the image encoder in the fully trained model does not have to work as hard to align its embeddings to the lidar embeddings because the lidar encoder participates in that task. Overall, there is a very robust linear relationship visible: Increase the contrastive pretraining, and the segmentation performance drops. This finding is robust over the different settings: linear or MLP head on the lidar encoder, lidar backbone frozen or not, image augmentations or not – we find none of those makes a noticeable difference in either segmentation performance per se or domain generalization. In fact, these metrics seem very robustly correlated as is attested by the near constant rPD across experiments with only few outliers. Overall, the pretraining optimizations do not improve domain generalization, but rather decrease it, mediated by the general segmentation task performance.



<b>Experiment</b>	<b>mIoU (Cityscapes)</b>	<b>mIoU (ACDC)</b>	<b>rPD</b>
timm-ViT-S	68.01	37.93	0.56
timm-ViT-S (fb)	58.21	40.00	0.69
CLCP Ep19 (Lin, fb)	62.30	31.06	0.50
CLCP Ep35 (Lin, fb)	59.21	27.21	0.46
CLCP Ep19 (Lin)	63.52	33.48	0.48
CLCP Ep35 (Lin)	60.58	31.07	0.51
CLCP Ep67 (Lin)	58.11	29.28	0.50
CLCP Ep19 (MLP, fb)	62.59	33.96	0.55
CLCP Ep35 (MLP, fb)	59.09	29.92	0.51
CLCP Ep19 (MLP, aug)	62.35	30.57	0.50
CLCP Ep35 (MLP, aug)	59.32	28.94	0.49
CLCP Ep67 (MLP, aug)	57.11	28.02	0.49
CLCP Ep67 (MLP)	58.38	28.70	0.49

Table 4.3: Ablations results compared to timm-ViT. Ep: Epoch of pretraining checkpoint. MLP/Lin: MLP vs. linear projection in lidar encoder. fb: Lidar encoder frozen in pretraining. aug: Image augmentations.



Figure 4.3: Samples from A2D2, Cityscapes and ACDC datasets.

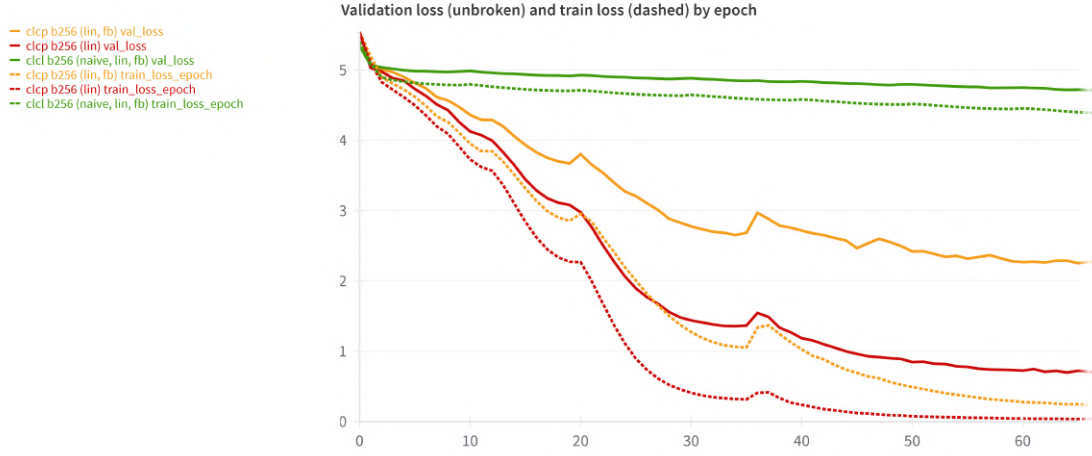


Figure 4.4: CLCP under naive initial (green) and optimized hyperparameters (orange) with frozen lidar encoder backbone (fb). The whole model pretraining is shown in red.

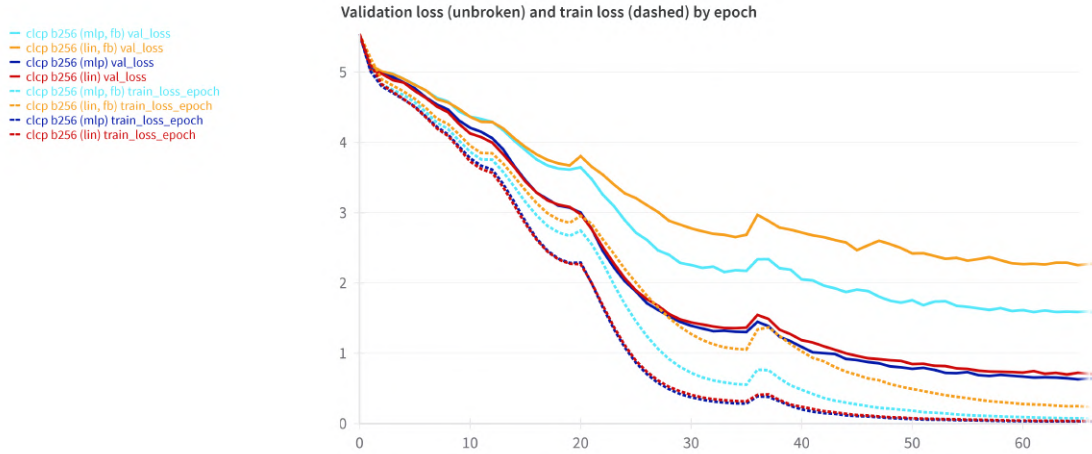


Figure 4.5: CLCP with MLP lidar encoder head (mlp) with backbone frozen (fb, light blue) and unfrozen (dark blue) and the linear projection head (lin) with frozen (fb, orange) and unfrozen (red) lidar backbone.

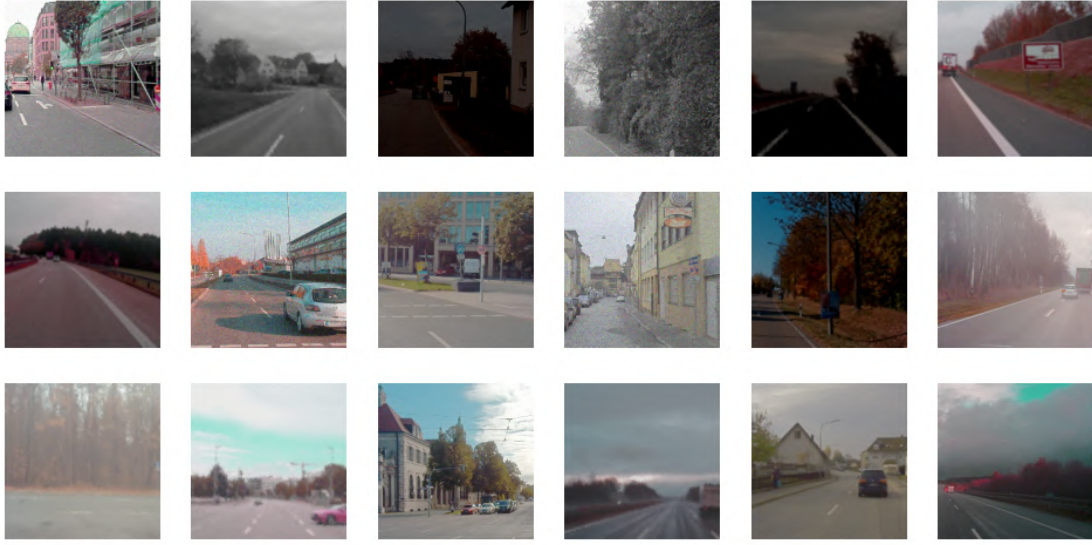


Figure 4.6: Illustration the applied image augmentations on a random sample from the training data.

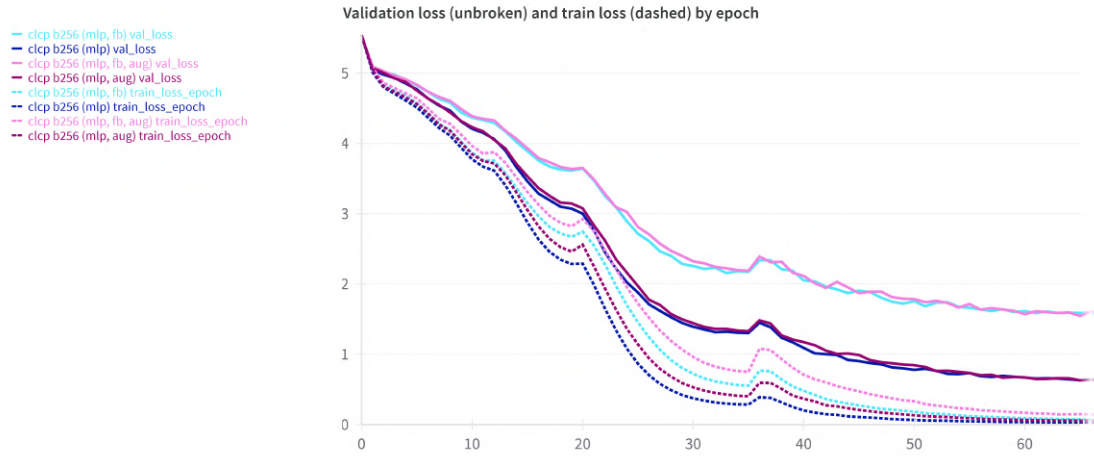


Figure 4.7: The MLP variants from before with frozen (light blue) or unfrozen (dark blue) lidar backbone, and the same variants trained with (light purple) and without (dark purple) additional image augmentations.

## 5 Discussion

In this thesis, we investigated the applicability of contrastive multimodal representation learning to the modality pairing of camera images and lidar point clouds in an autonomous driving setting, but could not show a significant advantage compared to standard unimodal pretraining methods. What might be the reasons for why the approach failed?

### 5.1 Useful Supervision

Although we managed to minimize the contrastive loss between lidar point cloud and camera image embeddings, our findings suggest that this did not help the domain generalization in semantic segmentation. Even worse: Across all model variants and training scenarios we explored, the semantic segmentation performance in the supervised setting on Cityscapes, as well as the domain generalization on ACDC deteriorated, the more effective the contrastive pretraining was.

The lack of a useful learning signal in pure image-point-cloud alignment constitutes a difference from the more prominent vision-language setting: The natural language supervision is inherently beneficial with respect to downstream tasks that involve connecting the vision and language domain, like image classification, image and text retrieval or geolocalization. Such an inherent benefit is harder to assert for the alignment with lidar point clouds.

Apart from scaling, which we’ll discuss below, a valid approach to remedy this problem could be to introduce an additional learning signal into the training pipeline that prevents the task performance from deteriorating under the contrastive pretraining. A straightforward way to achieve this could be to use annotations for either images or point clouds or both and construct a contrastive loss that balances the cross-modal alignment goal with the segmentation or another dense prediction task.

### 5.2 All about Data?

Another limitation of our approach is the lack of scale compared to state-of-the-art vision-language models. Those are trained on hundreds of millions or billions of samples where we only used thousands. It could be that a much larger scale and diversity is simply needed in order for an encoder to learn domain-robust representations: [24] presents compelling evidence that “CLIP’s robustness is determined almost exclusively by the training distribution”, in other words: the size and diversity of its training data. This limitation would be harder to overcome for the camera-lidar pairing, as internet scale data of suitable image-point-cloud pairs simply do not exist and will probably not in

the foreseeable future. However, it might still be possible to significantly expand the training data by relying on bigger datasets, like [88, 65], or to combine multiple datasets to extend data scale and diversity.

This ties in with the argument presented in section 3.2 which relied on the assumption that the representations of both modalities in the joint embedding space match perfectly across the source and target domain: Practically, this assumption can only be met if the model has a chance to learn the cross-modal alignment not only in the source, but also the target domain. It would make sense that models like CLIP and EVA-CLIP with their vast pretraining datasets are much closer to meeting that assumption than our setting permits.

On the other hand, this raises questions about a notion of domain generalization that assumes that the model has never seen the target domain at all: For large VLMs, or large pretraining datasets in general, it could be argued that this assumption is violated: If pretrained on datasets of such scale and diversity, a model might encounter the target domain in the pretraining phase, even though it never sees the target domain during finetuning. For camera-lidar models, this leaking of target domain data into the pretraining phase is much less likely, which, in return, could be considered especially challenging, because the available datasets are smaller and less diverse by a large margin compared to the training sets of SOTA VLMs. However, it bears mentioning that even off-the-shelf vision backbones today are mostly initialized with pretrained weights – in the case of timm-ViT these weights were trained on ImageNet-21k, which itself consists of close to 15M images and can hardly be considered small, except compared to the 400M image-text-pairs that CLIP was trained on or the multiple billions of pairs in the case of EVA-CLIP.

### 5.3 Outlook

Those limitations withstanding, multimodal pretraining in a self-supervised fashion holds great potential, specifically with regards to autonomous driving: In order to move consistently and safely through the world, a deep understanding of our physical surroundings is necessary. This can probably be learned neither by supervised, nor vision-language methods, as supervised methods lack the scale and language the inherent capacity to learn the deep and complex representations that are required for real-world perception and interaction. In addition to the suggestions already discussed, it might be worthwhile to look beyond contrastive methods, as has been shown to be effective by approaches like [35] and [9]. Non-contrastive methods are potentially more efficient as they dispense with the large amount of negative examples a model encounters in contrastive training. From a broader perspective, one could argue that this approach is also closer aligned with how we as humans learn to perceive the world: by observing and imitating success, rather than observing and avoiding failure.

Although the specific approach attempted in this thesis failed to achieve its objective of improving domain generalization in vision perception by contrastive methods, self-supervised learning could scale up the amount of available training data by orders of

magnitude by largely dispensing with the need for annotated data. This could facilitate significant advancements on enabling deep learning models to learn more complex and generalizable representations of the physical world, which is especially relevant for fields like autonomous driving and robotics, but also in artificial intelligence as a whole.

# Bibliography

- [1] Alexander Amini et al. *Spatial Uncertainty Sampling for End-to-End Control*. 2019. arXiv: 1805.04829 [cs.AI].
- [2] Sanjeev Arora. “The Quest for Mathematical Understanding of Deep Learning”. In: *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*. Vol. 182. Schloss Dagstuhl, Leibniz-Zentrum für Informatik. 2020, p. 1.
- [3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].
- [4] Jens Behley et al. “Semantickitti: A dataset for semantic scene understanding of lidar sequences”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 9297–9307.
- [5] Devansh Bisla. “Understanding Generalization in Deep Learning: An Empirical Approach”. PhD thesis. New York University Tandon School of Engineering, 2022.
- [6] Sébastien Bubeck et al. “Convex optimization: Algorithms and complexity”. In: *Foundations and Trends® in Machine Learning* 8.3-4 (2015), pp. 231–357.
- [7] Oliver De Candido, Xinyang Li, and Wolfgang Utschick. “An Analysis of Distributional Shifts in Automated Driving Functions in Highway Scenarios”. In: *2022 IEEE 95th Vehicular Technology Conference: (VTC2022-Spring)*. 2022, pp. 1–7.
- [8] Ting Chen et al. “A simple framework for contrastive learning of visual representations”. In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.
- [9] Xinlei Chen and Kaiming He. “Exploring simple siamese representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 15750–15758.
- [10] Mehdi Cherti et al. “Reproducible scaling laws for contrastive language-image learning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 2818–2829.
- [11] Christopher Choy, JunYoung Gwak, and Silvio Savarese. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 3075–3084.
- [12] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. arXiv: 1511.07289 [cs.LG].



- [13] MMDetection3D Contributors. *MMDetection3D: OpenMMLab next-generation platform for general 3D object detection*. <https://github.com/open-mmlab/mmdetection3d>. 2020.
- [14] MMSegmentation Contributors. *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*. <https://github.com/open-mmlab/mms segmentation>. 2020.
- [15] Marius Cordts et al. “The Cityscapes Dataset for Semantic Urban Scene Understanding”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.
- [16] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals, and Systems* 2.198 (1989), pp. 303–314.
- [17] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [18] Benjamin Devillers et al. *Does language help generalization in vision models?* 2021. arXiv: 2104.08313 [cs.AI].
- [19] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].
- [20] Ben Dickson. *Is camera-only the future of self-driving cars?* 2022. URL: <https://www.autonomousvehicleinternational.com/features/is-camera-only-the-future-of-self-driving-cars.html> (visited on 07/24/2024).
- [21] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: 2020.
- [22] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019.
- [23] Lue Fan et al. “Embracing single stride 3d object detector with sparse transformer”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 8458–8468.
- [24] Alex Fang et al. “Data Determines Distributional Robustness in Contrastive Language-Image Pre-training (CLIP)”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 6216–6234.
- [25] Angelos Filos et al. “Can autonomous vehicles identify, recover from, and adapt to distribution shifts?” In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3145–3153.
- [26] Alberto Garcia-Garcia et al. “Pointnet: A 3d convolutional neural network for real-time object class recognition”. In: *2016 International joint conference on neural networks (IJCNN)*. IEEE. 2016, pp. 1578–1584.
- [27] Guillaume Garrigos and Robert M. Gower. *Handbook of Convergence Theorems for (Stochastic) Gradient Methods*. 2024. arXiv: 2301.11235 [math.OC].

- [28] Andreas Geiger et al. “Vision meets robotics: The kitti dataset”. In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237.
- [29] Jakob Geyer et al. *A2D2: Audi Autonomous Driving Dataset*. 2020. arXiv: 2004.06320 [cs.CV].
- [30] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. “Deep sparse rectifier neural networks”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 315–323.
- [31] VA Golovko. “Deep learning: an overview and main paradigms”. In: *Optical memory and neural networks* 26 (2017), pp. 1–17.
- [32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [33] Robert Mansel Gower et al. “SGD: General analysis and improved rates”. In: *International conference on machine learning*. PMLR. 2019, pp. 5200–5209.
- [34] Priya Goyal et al. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018. arXiv: 1706.02677 [cs.CV].
- [35] Jean-Bastien Grill et al. “Bootstrap your own latent-a new approach to self-supervised learning”. In: *Advances in neural information processing systems* 33 (2020), pp. 21271–21284.
- [36] M. Gürbüzbalaban, A. Ozdaglar, and P. A. Parrilo. “Why random reshuffling beats stochastic gradient descent”. In: *Mathematical Programming* 186.1 (Mar. 2021), pp. 49–84. ISSN: 1436-4646.
- [37] Boris Hanin. “Universal function approximation by deep neural nets with bounded width and relu activations”. In: *Mathematics* 7.10 (2019), p. 992.
- [38] Shijie Hao, Yuan Zhou, and Yanrong Guo. “A Brief Survey on Semantic Segmentation with Deep Learning”. In: *Neurocomputing* 406 (2020), pp. 302–321. ISSN: 0925-2312.
- [39] HarisIqbal88. *PlotNeuralNet*. <https://github.com/HarisIqbal88/PlotNeuralNet>. 2018.
- [40] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.
- [41] Kaiming He et al. “Momentum contrast for unsupervised visual representation learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 9729–9738.
- [42] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].
- [43] Georg Hess et al. “LidarCLIP or: How I Learned To Talk to Point Clouds”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. Jan. 2024, pp. 7438–7447.

- [44] Elad Hoffer, Itay Hubara, and Daniel Soudry. “Train longer, generalize better: closing the generalization gap in large batch training of neural networks”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
- [45] K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.198 (1989), pp. 359–366.
- [46] K. Hornik, M. Stinchcombe, and H. White. “Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks”. In: *Neural Networks* 3.198 (1990), pp. 551–560.
- [47] Kurt Hornik. “Approximation capabilities of multilayer feedforward networks”. In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080.
- [48] Yue Hu et al. “Collaboration Helps Camera Overtake LiDAR in 3D Detection”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 9243–9252.
- [49] Yu Huang et al. “What makes multi-modal learning better than single (provably)”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 10944–10956.
- [50] Christoph Hümmer et al. *VLTSeg: Simple Transfer of CLIP-Based Vision-Language Representations for Domain Generalized Semantic Segmentation*. 2023. arXiv: 2312.02021 [cs.CV].
- [51] Daniel Jiwoong Im et al. *Generating images with recurrent adversarial networks*. 2016. arXiv: 1602.05110 [cs.LG].
- [52] Peng Jiang and Srikanth Saripalli. “Contrastive Learning of Features between Images and LiDAR”. In: *IEEE 18th International Conference on Automation Science and Engineering (CASE)*. Aug. 2022, pp. 411–417.
- [53] Patrick Kidger and Terry Lyons. “Universal Approximation with Deep Narrow Networks”. In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, July 2020, pp. 2306–2327.
- [54] Alexander Kirillov et al. “Panoptic segmentation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 9404–9413.
- [55] Andreas Koukounas et al. *Jina CLIP: Your CLIP Model Is Also Your Text Retriever*. 2024. arXiv: 2405.20204 [cs.CL].
- [56] Alex H Lang et al. “Pointpillars: Fast encoders for object detection from point clouds”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 12697–12705.
- [57] Yann LeCun et al. “Backpropagation applied to handwritten zip code recognition”. In: *Neural computation* 1.4 (1989), pp. 541–551.

- [58] Yann LeCun et al. “Learning algorithms for classification: A comparison on handwritten digit recognition”. In: *Neural networks: the statistical mechanics perspective* 261.276 (1995), p. 2.
- [59] M. Leshno et al. “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function”. In: *Neural Networks* 6.198, 199 (1993), pp. 861–867.
- [60] Jianan Li, Jie Wang, and Tingfa Xu. “Pointgl: a simple global-local framework for efficient point cloud analysis”. In: *IEEE Transactions on Multimedia* (2024).
- [61] Yiyi Liao, Jun Xie, and Andreas Geiger. “KITTI-360: A Novel Dataset and Benchmarks for Urban Scene Understanding in 2D and 3D”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.3 (2023), pp. 3292–3310.
- [62] Ilya Loshchilov and Frank Hutter. *SGDR: Stochastic Gradient Descent with Warm Restarts*. 2017. arXiv: 1608.03983 [cs.LG].
- [63] Junyi Ma et al. “Cam4DOcc: Benchmark for Camera-Only 4D Occupancy Forecasting in Autonomous Driving Applications”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 21486–21495.
- [64] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. PMLR. 2013.
- [65] Jiageng Mao et al. *One Million Scenes for Autonomous Driving: ONCE Dataset*. 2021. arXiv: 2106.11037 [cs.CV].
- [66] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biophysics* 5 (1943), pp. 115–133.
- [67] Tom M. Mitchell. *Machine Learning*. McGraw-Hill New York, 1997.
- [68] Andrew Nader and Danielle Azar. “Evolution of activation functions: an empirical investigation”. In: *ACM Transactions on Evolutionary Learning and Optimization* 1.2 (2021), pp. 1–36.
- [69] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.
- [70] Chigozie Nwankpa et al. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. 2018. arXiv: 1811.03378 [cs.LG].
- [71] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, 2021, pp. 8748–8763.
- [72] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: 1710.05941 [cs.NE].

- [73] Tal Ridnik et al. *ImageNet-21K Pretraining for the Masses*. 2021. arXiv: 2104.10972 [cs.CV].
- [74] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III* 18. Springer. 2015, pp. 234–241.
- [75] Olga Russakovsky et al. “Imagenet large scale visual recognition challenge”. In: *International journal of computer vision* 115 (2015), pp. 211–252.
- [76] Andrzej P Ruszczyński. *Nonlinear optimization*. Vol. 13. Princeton university press, 2006.
- [77] Itay Safran and Ohad Shamir. “Random Shuffling Beats SGD Only After Many Epochs on Ill-Conditioned Problems”. In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 15151–15161.
- [78] Christos Sakaridis, Dengxin Dai, and Luc Van Gool. “ACDC: The Adverse Conditions Dataset With Correspondences for Semantic Driving Scene Understanding”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 10765–10775.
- [79] Arthur L. Samuel. “Some studies in machine learning using the game of checkers”. In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.
- [80] Ohad Shamir. “Without-replacement sampling for stochastic gradient methods”. In: *Advances in neural information processing systems* 29 (2016).
- [81] Siddharth Sharma, Simone Sharma, and Anidhya Athaiya. “Activation Functions in Neural Networks”. In: *International Journal of Engineering Applied Sciences and Technology* 04 (May 2020), pp. 310–316.
- [82] Sai Shubodh et al. “LIP-Loc: LiDAR Image Pretraining for Cross-Modal Localization”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*. Jan. 2024, pp. 948–957.
- [83] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.
- [84] Leslie N Smith and Nicholay Topin. “Super-convergence: Very fast training of neural networks using large learning rates”. In: *Artificial intelligence and machine learning for multi-domain operations applications*. Vol. 11006. SPIE. 2019, pp. 369–386.
- [85] Ioannis Stamos. “Automated registration of 3D-range with 2D-color images: an overview”. In: *2010 44th Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2010, pp. 1–6.

- [86] Robin Strudel et al. “Segmenter: Transformer for semantic segmentation”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 7262–7272.
- [87] Jiahao Sun et al. *An Empirical Study of Training State-of-the-Art LiDAR Segmentation Models*. 2024. arXiv: 2405.14870 [cs.CV].
- [88] Pei Sun et al. “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.
- [89] Quan Sun et al. *EVA-CLIP-18B: Scaling CLIP to 18 Billion Parameters*. 2024. arXiv: 2402.04252 [cs.CV].
- [90] Quan Sun et al. *EVA-CLIP: Improved Training Techniques for CLIP at Scale*. 2023. arXiv: 2303.15389 [cs.CV].
- [91] Haotian Tang et al. “TorchSparse++: Efficient Training and Inference Framework for Sparse Convolution on GPUs”. In: *IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2023.
- [92] Yingjie Tian, Yuqi Zhang, and Haibin Zhang. “Recent Advances in Stochastic Gradient Descent in Deep Learning”. In: *Mathematics* 11.3 (2023). ISSN: 2227-7390.
- [93] Usman Ahmad Usmani and Mohammed Umar Usmani. “Theoretical Insights into Neural Networks and Deep Learning: Advancing Understanding, Interpretability, and Generalization”. In: *2023 World Conference on Communication & Computing (WCONF)*. IEEE. 2023, pp. 1–8.
- [94] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017.
- [95] Haiyang Wang et al. “Dsvt: Dynamic sparse voxel transformer with rotated sets”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 13520–13529.
- [96] Peide Wang. “Research on Comparison of LiDAR and Camera in Autonomous Driving”. In: *Journal of Physics: Conference Series* 2093.1 (Nov. 2021), p. 012032.
- [97] Ross Wightman. *PyTorch Image Models*. <https://github.com/rwightman/pytorch-image-models>. 2019.
- [98] Danni Wu, Zichen Liang, and Guang Chen. “Deep learning for LiDAR-only and LiDAR-fusion 3D perception: A survey”. In: *Intelligence & Robotics* 2.2 (2022), pp. 105–129.
- [99] Tao Wu et al. “Detailed analysis on generating the range image for lidar point cloud processing”. In: *Electronics* 10.11 (2021), p. 1224.
- [100] Xiaoyang Wu et al. “Point Transformer V3: Simpler Faster Stronger”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2024, pp. 4840–4851.

- [101] Yutian Wu et al. “Deep 3D Object Detection Networks Using LiDAR Data: A Review”. In: *IEEE Sensors Journal* 21.2 (2021), pp. 1152–1171.
- [102] Yusheng Xu, Xiaohua Tong, and Uwe Stilla. “Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry”. In: *Automation in Construction* 126 (2021), p. 103675.
- [103] Yang You et al. *Large Batch Optimization for Deep Learning: Training BERT in 76 minutes*. 2020. arXiv: 1904.00962 [cs.LG].
- [104] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.
- [105] Zhipeng Zhao et al. *Attention-Enhanced Cross-modal Localization Between 360 Images and Point Clouds*. 2022. arXiv: 2212.02757 [cs.CV].
- [106] Huazan Zhong et al. “A survey of LiDAR and camera fusion enhancement”. In: *Procedia Computer Science* 183 (2021), pp. 579–588.
- [107] Jingmeng Zhou. “A Review of LiDAR sensor Technologies for Perception in Automated Driving”. In: *Academic Journal of Science and Technology* 3.3 (2022), pp. 255–261.
- [108] Zhengxia Zou et al. “Object detection in 20 years: A survey”. In: *Proceedings of the IEEE* 111.3 (2023), pp. 257–276.