SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Monocular Depth Estimation by Domain Transfer Using Rigid Transformations

Dan Halperin

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY — INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Monocular Depth Estimation by Domain Transfer Using Rigid Transformations

Monokulare Tiefenschätzung durch Domänenübertragung unter Verwendung starrer Transformationen

Author:	Dan Halperin
Supervisor:	Prof. Dr.-Ing. Markus Lienkamp
Advisor:	Dominik Kulmer
Submission Date:	22.10.2024

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

Munich, 22.10.2024 Dan Halperin

Abstract

Depth estimation is a critical task in fields such as robotics, autonomous vehicles, and augmented reality, providing essential information for object detection, tracking, and path planning. Despite significant advancements using monocular RGB images or multisensor setups like LiDAR and Radar, challenges remain in developing solutions that are both accurate and computationally efficient, particularly for embedded systems with limited hardware resources.

This thesis explores various techniques that guide the depth estimation process using ground truth depth maps, beyond traditional loss functions. The proposed methods focus on aligning RGB and LiDAR modalities in a lower-dimensional latent space, facilitating a more effective and robust training pipeline. This research investigates several alignment strategies, ranging from minimizing Euclidean distances between the latent spaces to bridging their second-order structures. Additionally, in this thesis a novel approach for discovering linear transformations between the two manifolds, extending previous work that primarily focused on global-context latent spaces, will be proposed.

To address the limitations of sparse and noisy ground truth data in the NuScenes dataset, a specialized loss function and a mechanism to dynamically weight different losses is introduced, improving the visual predicted depth map. To lower the uncertainty, the dataset's inherent challenge of missing depth information in certain regions is mitigated by generating "quasi" semantic-segmentation maps, particularly for the "sky" regions.

The objective of this thesis is not to surpass state-of-the-art benchmarks, but to provide a comprehensive comparison of different latent space alignment techniques using an efficient model that balances performance with computational feasibility. The findings present valuable insights into the nuances of these methods and lay the groundwork for future research on a larger scale and with more advanced datasets, such as KITTI.

Contents

1 Introduction

Depth estimation is a challenging task due to the limited availability of ground truth data and the high uncertainty present across extensive regions of feature maps. It plays a critical role in various fields, such as robotics, autonomous vehicles, and augmented reality. For instance, in autonomous vehicles, depth estimation is essential for object detection, tracking, and path planning. This task can be approached using either monocular RGB images, which are straightforward to install on a moving platform, *"stereo"* cameras setup or by integrating information from multiple sensors, such as LiDAR and Radar, combined with visual cues. Many works have demonstrated outstanding performance in controlled environments where hardware constraints are not an issue. UniDepth [1] proposed an innovative method for learning the positional context of an image to guide the depth prediction pipeline, while CLIP2Depth [2] builds on the original CLIP [3] framework to enhance depth estimation using textual guidance. However, in the realm of embedded robotics, it is crucial to develop solutions that are not only accurate but also computationally efficient, given the limitations of available hardware.

A previous work, CamRaDepth [4], developed a multimodal model that integrates data from both RGB images and corresponding Radar cues, projected onto the 2D plane, to guide visual cues towards estimating a more accurate depth map. In that study, the best performance was achieved by occasionally replacing the Radar depth map with the ground truth LiDAR map as a data augmentation technique, with a certain probability p. This approach reduced the RMSE (Root Mean Squared Error) in meters from $4.0m$ to $3.76m$ on the test set, given the architecture used. Building on this concept, this thesis aims to explore how ground truth LiDAR data can guide RGB cues exclusively. Removing the Radar input eliminates the need for system calibration, making the setup cheaper. Therefore, maximizing performance while simplifying the setup is desirable.

This thesis explores various strategies for guiding the training pipeline using the ground truth depth map, beyond merely relying on the final loss function between the prediction and the ground truth. It investigates different techniques for aligning the two modalities in a lower-dimensional latent space, which is flexible and adaptable to various objectives. The experiments involve encoding both RGB and LiDAR data as inputs and evaluating methods that range from directly minimizing the Euclidean

distances using the MSE loss function to closing the gap between their second-order structures. Additionally, this work explores two novel approaches: Using *Relative Representations* and finding a *linear transformation* between the two manifolds. Those ideas have been investigated before [5], [6], but typically on a smaller scale and focusing on global-context latent spaces, represented as a single vector for the entire image. For depth estimation and reconstruction, however, the latent space is a 3D tensor of shape $C \times H \times W$, where each "pixel" represents the local context of the original image. This adds significant complexity to the problem, making the solution more challenging and interesting.

Moreover, this thesis proposes a novel mechanism for adaptively weighting the different loss functions, allowing for better control over their relative magnitudes during training. The following also introduces a new loss function tailored to the NuScenes dataset [7] to address a significant limitation of this dataset. The ground truth in NuScenes is a LiDAR point cloud projected onto the 2D image plane, which is prone to projection errors and only provides sparse ground truth for training. This introduces considerable uncertainty, resulting in visually poor predicted depth maps, as the model struggles to estimate depth in large regions of the image.

To mitigate this issue, "quasi" semantic-segmentation maps are first generated, since the NuScenes dataset does not include them. The focus is then put specifically on regions labeled as "sky", which in outdoor scenarios correspond to objects that are **infinitely** far away from the camera. Consequently, there is no depth information available for these regions from the LiDAR sensors, which operate by reflecting light rays. Using these segmentation maps, these regions are penalized to be as close to zero as possible, since this work employs an inverse-depth representation.

The goal of this thesis is not to surpass the state-of-the-art, but rather to compare different methods for aligning latent spaces. Given the limited hardware resources available in real-world applications, it is opted for a highly efficient model that enables running numerous experiments within a short timeframe while still producing competitive results. The focus is on exploring the nuanced differences between these approaches, which can later be applied to many more similar datasets, such as KITTI [8].

At the end, this thesis demonstrates an improvement in depth prediction using RGB images by learning linear rotations to a shared manifold with RGB-D features. This approach also enables 'stitching' the two modalities through the latent space. The promising zero-shot performance with RGB latent spaces, decoded by a decoder trained on RGB-D data, lays strong groundwork for future research in this field

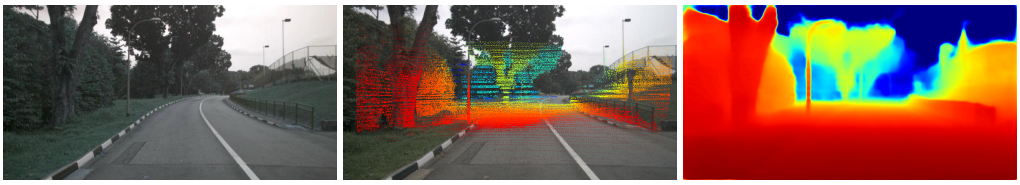
2 Related Work

This chapter reviews some previous works that this thesis builds upon, such as different approaches of depth prediction, interesting trials to align latent spaces and very powerful computational functions in neural networks.

2.1 Depth estimation vs depth prediction

In computer vision, "depth prediction" and "depth estimation" are two similar yet different challenges. Although often used interchangeably, "depth prediction" primarily involves machine learning models that anticipate depth values from visual data such as RGB images. On the other hand, "depth estimation" encompasses a wider array of techniques or modalities, including traditional stereo vision and advanced technologies like LiDAR, to infer depth information from visual cues and measurements.

Depth prediction is a very challenging task, given the limited capacity of the existing ground truth and high uncertainty in vast areas of the feature maps.



RGB GT - LiDAR point cloud Depth Prediction

Figure 2.1: Depth prediction is a challenging task, especially due to the lack of solid and dense ground truth information, and high uncertainty in vast areas of the feature maps. ***Source:*** *Scene* ***00453*** *from the NuScenes dataset* [7]

2.2 Depth completion for monocular imaging

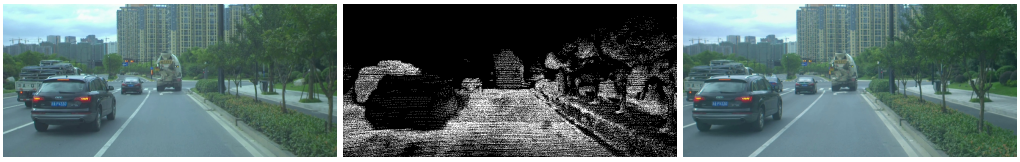
2.2.1 A stereo setup

A stereo setup, mimicking the binocular vision of biological eyes, involves the use of two or more cameras placed apart, capturing the same scene from slightly different

viewpoints. This configuration exploits the principle of triangulation, where the disparity between corresponding points in the images provides a direct and geometrically grounded cue for depth estimation. By comparing the relative positions of objects in the two images, the algorithm [9] can infer the distance and three-dimensional structure of the scene. This setup leverages the natural spatial separation between the cameras to compute depth with remarkable accuracy, making it a straightforward approach for capturing the depth information present in the scene.

Therefore, various modern datasets, such as KITTI [8], incorporate depth maps created in this fashion, aiming to present a solid ground for depth extraction.

In addition, modern research has already surpassed the performance of the straightforward reconstruction from a stereo-setup, by learning the process using machine learning algorithms, such as in [10], which is the best-performing method of such sort, on the "Stereo Depth Estimation on KITTI-2015" benchmark [11].

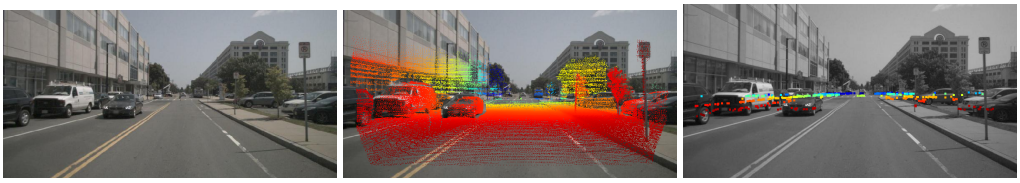


Left Image Depth map right image.

Figure 2.2: A reconstructed depth map from a stereo setup, by using triangularly.
Source: *Taken from the Driving-Stereo dataset [12]*

2.2.2 Multimodal setup

Another approach to predicting depth in a camera's view involves utilizing information from various sources. One common method is to combine data from other sensors, such as LiDAR or Radar, which use light rays or radio waves to measure depth in their surroundings. These sensors often produce 3D maps or point clouds, which can be processed in their original 3D form or projected onto the 2D image space (Figure 2.3). This additional information guides the depth prediction model to extract more meaningful features, enhancing performance significantly. However, such guidance requires the existence of such sensors, which is not yet a trivial setup for mass production. Also, besides their benefits, Radar depth maps are very limited in scope, while LiDAR struggles in difficult weather conditions. Therefore, an intriguing strategy for machine learning algorithms is to make use of this guidance during training, with the goal of enhancing model performance when dealing exclusively with RGB inputs during inference.



RGB LiDAR point cloud RADAR point cloud

Figure 2.3: Possible modalities of depth values. While "depth prediction" assumes only an RGB input, "depth estimation" relies on fine and meaningful features from other modalities or sensors. Note the density gap between the two. While the LiDAR point cloud is denser, the Radar sensor is much cheaper and more robust to bad weather conditions. ***Source:*** *Scene* ***20387*** *from the NuScenes dataset*

LiDAR. In recent years, LiDAR data have been used in different computer vision tasks because of their comparably high point cloud density and accuracy. This technology utilizes laser pulses to measure the time it takes for light to reflect off objects, and provides high-resolution 3D maps. However, the main drawbacks are the cost of use and the limited performance under bad weather conditions [13], [14]. Nevertheless, there are many works that use LiDAR data to their advantage in the task of depth completion. *BP-Net* [15] is the best published state-of-the-art approach for depth completion on the *KITTI* benchmark [16], that offers a published paper.

Radar. In contrast to LiDARs, Radar operates by emitting radio waves and measuring the time it takes for the waves to bounce back after hitting an object. This enables it to detect the presence, distance, and speed of objects. These sensors are much cheaper to deploy in mass production and are more robust in harsh weather conditions due to their larger wavelength [13], [14]. Therefore, although Radar does not provide the same accuracy and its point clouds are significantly sparser than those from LiDAR, there is ongoing research to try and apply the data to relevant vision tasks such as depth estimation. However, compared to research that relies on LiDAR data, the number of works that address this challenge is quite limited. Most of them implement fully convolutional models, such as the work of Long et al. [17] or Lin et al. [18].

2.2.3 Monocular imaging

While stereo and multimodal setups offer advantages in terms of depth estimation, Monocular depth prediction, using only a single camera, presents a more challenging scenario. Depth information needs to be inferred solely from the 2D visual cues present in the image. This complexity is outweighed by its advantages in terms of cost and ease of deployment, making it a desirable research area for applications in real-world

vehicles and robots.

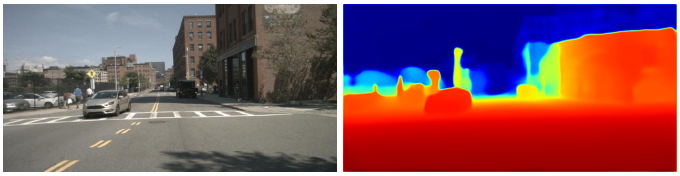


Figure 2.4: Monocular depth prediction. Attempting to achieve the most, while using the least.

2.3 Datasets and the Challenges in Depth Estimation

Depth estimation is a critical task in autonomous driving, and several datasets have become foundational in this field. Notably, NuScenes [7], KITTI [8], and NYU Depth v2 [19] are widely used benchmarks.

This thesis utilizes the NuScenes dataset, which offers a wide range of outdoor scenarios—from daytime to night, including challenging weather conditions. These scenarios are particularly interesting for research as they present greater challenges.

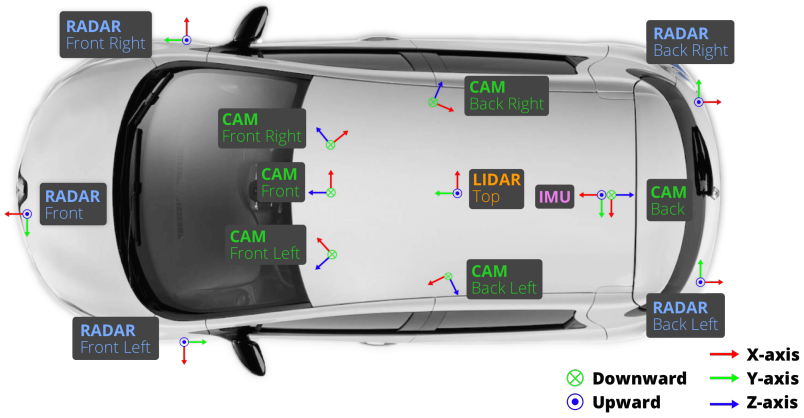


Figure 2.5: NuScenes setup on their vehicle. While cameras and Radar sensors are relatively cheap, they lack the incredible power of LiDAR in depth estimation, but the latter is still very expensive to deploy.

NuScenes is a large-scale autonomous driving dataset that captures data from various sensors, including a Velodyne VLP-32C LiDAR, six FLIR Blackfly cameras (1600×900

resolution, 12 Hz), and five Continental ARS 408-21 Radar sensors providing 360-degree coverage. The dataset includes 1000 driving scenes, each lasting 20 seconds, recorded in diverse weather conditions and times of day. However, when it comes to depth estimation, NuScenes provides only sparse ground truth depth maps, generated by projecting LiDAR point clouds onto 2D planes. This results in incomplete depth information, particularly in regions where LiDAR fails to capture data, such as the sky or distant objects.

One of the significant challenges in depth estimation is the inherent limitations and biases of the available data. Outdoor datasets like primarily consist of clear, daytime scenes, leading to biases that hinder model performance in more diverse real-world scenarios. These challenging edge cases include varying lighting conditions, weather patterns, and object occlusions caused by elements such as water droplets or fog on the camera lens. For instance, methods like STEPS [20] have explored the use of *Generative Adversarial Networks* (GANs) [21] to generate synthetic daytime images from nighttime scenes, thereby improving depth prediction under those conditions. Despite these advancements, the datasets still present limitations that must be addressed to enhance the robustness of depth estimation models. Additionally, the ground truth depth maps in these datasets, often generated using stereo vision or LiDAR, come with their limitations. While stereo-generated maps may accurately estimate depth in some regions, they can struggle with areas of occlusion or scene changes. On the other hand, LiDAR-based maps may miss infinitely distant objects, resulting in uncertainty in outdoor scenes, e.g., in the abundant sky regions. These challenges necessitate developing models that can handle these uncertainties and adapt to the limitations of the available data.

Another challenge in depth prediction arises from the sensitivity of deep learning models to variations in image properties. RGB data is particularly susceptible to changes in lighting, contrast, and image artifacts such as noise. This sensitivity can be especially problematic in challenging scenarios, such as nighttime images, where extensive dark regions may lack the necessary features for accurate depth estimation. To address this, it is essential to incorporate strategies like data augmentation and techniques for managing illumination changes, thereby improving the model's generalizability across different conditions. For example, the multimodel approach of using RGB images and Radar data, [4] to guide the model with some existing depth values from a commonly used sensor.

In addition to the challenges posed by data quality and biases, the inherent limitations of single-image depth estimation must be considered. Unlike stereo vision, which leverages disparities between viewpoints to estimate depth, monocular depth estimation lacks these cues, making it a more complex and ill-posed problem. This necessitates the development of models that can utilize additional information, such as semantic

segmentation [22] or prior knowledge about the scene, to enhance depth prediction accuracy.

While alternative sensors like LiDAR and Radar offer distinct advantages, they also come with their own set of limitations. LiDAR, though unaffected by low-light conditions, produces sparse point clouds that may miss crucial depth information and lack detailed geometric structure. Additionally, LiDAR struggles to generate high-resolution and accurate points in adverse weather conditions. Radar, on the other hand, is more robust to challenging weather but generates extremely sparse point clouds with limited coverage in 3D space. These limitations highlight the need for methods that can integrate information from diverse sensors to achieve robust and accurate depth estimation across various real-world scenarios.

2.4 Modern Monocular Depth Estimation

The discussion on datasets highlights a crucial challenge: the lack of extensive and varied ground truth data for supervision tasks, limiting the ability of purely supervised models to accurately represent real-world scenarios. Moreover, models trained on such constrained datasets often struggle to adapt to new data or different distributions, resulting in poor performance when applied in unfamiliar domains.

To address this issue, recent research in reconstruction tasks like depth estimation and semantic segmentation has shifted focus towards achieving "Zero-shot" capabilities, aiming for models to demonstrate robustness across diverse domains. A notable example is Depth-Anything [23], presented at CVPR24, which not only addresses the scarcity of supervised ground truth data but also excels in transferring knowledge across vastly different domains, such as indoor and outdoor scenes and drastic changes in lighting conditions, which modern datasets struggle to mitigate. This is done by leveraging millions of unlabeled images and generating quasi-ground truth annotations through advanced methods.

However, this approach isn't unique to Depth-Anything. CLIP [3], a pioneering work by OpenAI, demonstrated the efficacy of manipulating latent spaces created by different modalities. By leveraging robust features extracted from text to guide image representations, CLIP showcased the ability to transcend the limitations imposed by the source image distribution. While features extracted from images are heavily influenced by the distribution of the source data, those derived from text exhibit greater resilience to changes in data distribution. This property of textual data allows the model to rely on less noisy representations, which in turn enhances performance in tasks such as image classification. CLIP's developers observed that this guidance at a lower level enables such models to perform well on unseen data and even across different distributions, as

it no longer solely relies on the distribution of the source data.

Another notable example of such manipulation is UniDepth [1]. This work currently stands as the state-of-the-art method for monocular depth estimation and presents a similar concept to the one outlined in this thesis. As described by its authors, regression tasks like depth estimation are inherently challenging and heavily influenced by the distribution of the data, making them highly sensitive to fundamental data properties. For instance, in RGB images, features are particularly sensitive to factors such as color schemes, camera intrinsic, contrast, and lighting conditions. To address these challenges, UniDepth employs a strategy where RGB latent data is projected into a 3D point cloud and adjusted to align with scene augmentations, thereby creating a robust model capable of handling sensitive feature domains. This process precedes decoding the latent space to produce a depth map. Additionally, the authors propose a geometric-invariant loss function that separates the camera and RGB gradients from the depth gradients. This approach tackles a significant issue where conventional models tend to overly rely on RGB features, which are less effective for learning accurate depth values for each pixel.

Another promising approach from CVPR24 is ECoDepth [24], which serves as another spinoff of CLIP. This method harnesses learned priors derived from a pre-trained Vision Transformer (ViT) model, which are then integrated with a diffusion backbone, and archives state-of-the-art results.

Another noteworthy method is Depth Pro [25], a model developed by Apple for monocular depth estimation. Unlike other approaches, Depth Pro excels in generating high-resolution depth maps from a single 2D image within an impressive 0.3 seconds. This method operates independently of camera metadata, making it highly versatile. It leverages sophisticated machine learning techniques to process visual data effectively, ensuring accurate depth estimation even in complex environments. Depth Pro has proven particularly useful for applications in augmented reality and autonomous systems, setting a new benchmark for speed and precision in the field of monocular depth estimation.

2.5 Visual Transformers

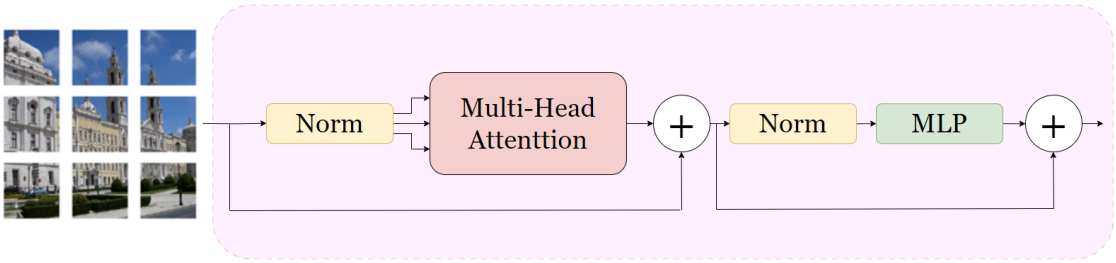


Figure 2.6: A visual transformer encoder block. It follows the same logic as its NLP counterpart, with embeddings as overlapping patches from the input feature map.

The transformer, a groundbreaking model introduced by Vaswani et al. [26], gained popularity because it could handle long-term dependencies, addressing key issues in previous state-of-the-art models like recurrent neural networks (RNNs) and long short-term memory (LSTM) [27].

Later, Dosovitskiy et al. [28] made a significant advancement by applying transformer architectures in computer vision. They replaced the traditional convolutional encoder with a transformer model in an autoencoder-like setup. Instead of sequential embeddings, input tokens became a set of patch embeddings obtained through convolutions from the feature map. To utilize these embeddings, the feature maps were flattened across spatial dimensions, resulting in tensors shaped $(B, H \times W, C)$, where B is batch size, C is the original tensor's channel count, and H and W are its spatial dimensions.

This innovative approach led to notable performance improvements across various vision tasks like depth estimation and segmentation. However, initial implementations faced limitations due to expensive operations, particularly in using fully connected layers and reshaping operations in the self-attention mechanism.

A GitHub repository collects numerous approaches to enhance the original ViT model, serving as a valuable resource for those interested in the topic [29]. Nonetheless, for this thesis, LVT (Light Visual Transformer) [30] was appealing. LVT offers a much lighter model in terms of parameter count and runtime while introducing novel self-attention modules. These modules aim to enrich embeddings with information from different scales or create stronger low-level features.

Moreover, the LVT repository provides a straightforward implementation in PyTorch, making it easy to use and modify compared to other complex implementations. Simply plug in your own code, and run. Its self-attention modules can serve as independent processing units, much like the default convolutional or fully connected layers but with greater capacity.

2.6 CBAM: Convolutional Block Attention Module

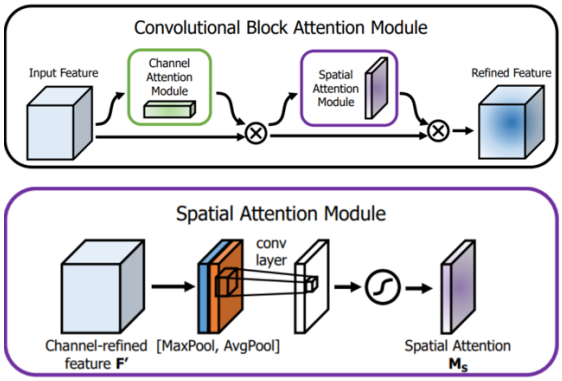


Figure 2.7: CBAM: find attention maps across the channels and spatially to find the center of focus.

The Convolutional Block Attention Module (CBAM) is a lightweight and effective attention mechanism designed to enhance the representational power of convolutional neural networks (CNNs) by focusing on important features while suppressing less relevant ones. CBAM introduces two types of attention mechanisms: *channel attention* and *spatial attention*.

The channel attention module refines feature maps by emphasizing more informative channels. It computes attention weights by globally pooling feature maps along spatial dimensions, using both max-pooling and average-pooling. These pooled features are passed through a small network consisting of fully connected layers, allowing the model to focus on channels that contribute the most to the task.

After channel attention, CBAM applies spatial attention, which focuses on important regions within the feature maps. This is achieved by pooling feature maps along the channel dimension (using max-pooling and average-pooling) and applying a convolution layer. The resulting attention map helps the model to attend to key spatial locations in the image.

While visual transformers (Section 2.5) have gained popularity for their powerful global attention mechanisms, they come with high computational costs, especially in terms of memory (VRAM) and processing time. Visual transformers rely on self-attention mechanisms that require flattening 2D spatial dimensions and computing pairwise interactions between all tokens, leading to quadratic complexity with respect to input size. This makes ViTs resource-intensive and harder to deploy on devices with limited memory.

In contrast, CBAM offers a more efficient solution for attention in CNNs. It introduces

only a small overhead in terms of both computation and memory, as the attention mechanisms are applied sequentially and locally. For systems with limited VRAM, such as embedded devices or environments with restricted GPU resources, CBAM is preferable because its lightweight nature allows us to use it at many more scales and waypoints in the architecture.

2.7 Dense-Net

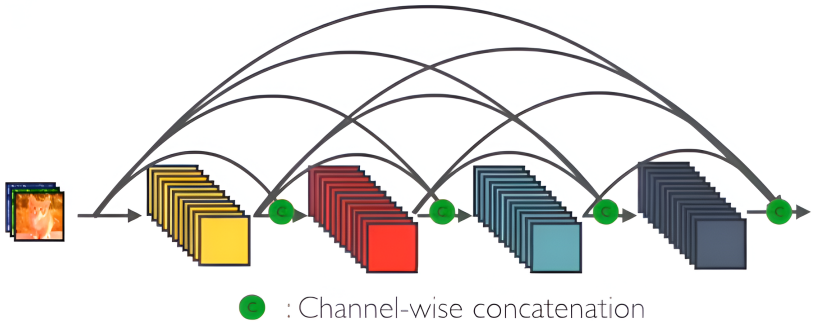


Figure 2.8: DenseNet - the input to the subnetwork is kept throught the different layers, thus allowing to learn and connect more fine-grained features.

DenseNet [31] is an architecture that improves feature reuse and gradient flow by connecting each layer to every previous one. Instead of adding information sequentially as in ResNet [32], DenseNet allows each layer to receive the feature maps from **all** earlier layers. This dense connectivity pattern also mitigates the vanishing gradient problem, but enhances feature propagation even more, and makes the network more parameter-efficient, as fewer redundant features need to be learned. This is a key advantage of DenseNet, because the network leverages previously learned features rather than recalculating them in each layer. The use of bottleneck layers and transition layers helps manage memory consumption by compressing feature maps between dense blocks.

However, the main drawback of DenseNet lies in its computational complexity. The dense connectivity pattern can lead to increased memory usage since each layer stores all feature maps. This makes the model slower to run and can be challenging for very large datasets or systems with limited memory. Additionally, while the model is efficient in terms of parameters, the dense connections may result in redundant computations, particularly in deeper architectures.

2.8 Guidance

A straightforward approach to supervised learning from RGB images is a challenging task, relying solely on an input that is usually noisy, low-resolution, and very biased to the camera settings. Therefore, many works in that domain, have pursued ways to introduce more meaningful features and knowledge into the training process.

The first approach would be of using multimodal models, such as combining processing branches of data from RGB images alongside data from depth-producing sensors, such LiDAR, Radar (Subsection 2.2.2) or even videos, which allow the creation of disparity maps out of a stereo setup between consecutive frames.

The following survey [33] discusses the very early approaches of fusing different modalities, such as early, late or deep fusion techniques. such a work, for example, would be [34] that uses later fusion, while also incorporating self-supervised methods, such as the completion of the input, after removing random parts of it. Such non-supervised mechanism enforces a form of regularization, making the features more robust and generic [35].

However, using LiDAR data, as useful as it is, is expansive to use, and therefore harder to deploy into real-world tasks. In addition to that, since it is a light-ray based sensor, its accuracy drops in bad weather conditions.

Therefore, some works explore, as described above, the fusion of RGB cues with Radar-guided features. "CamRaDepth" [4], my previous work, aimed to explore the utility of such fusion, while also aiming to introduce some guidance from quasi-created segmentation maps. While it definitely didn't perform as well as Lidar guidance on the NuScenes dataset (Section 2.4), that is composed of mainly daylight images, with a small amount of night or difficult weather conditions, it did allow some creative techniques to achieve promising performance.

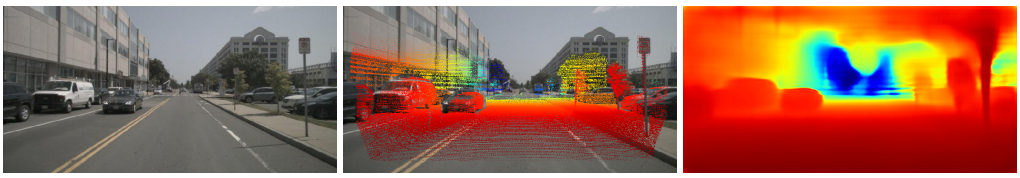
Another interesting work is given by [36], where the learning is guided by the projection of the image to the frequency-domain and encoding that DFT image with another encoder, thus using a wider range of robust features than just the original RGB cues, that have been proven to be very challenging for the task at hand.

Some studies like CLIP [3] have shown the potential of using features from other domains, but still rely on these features during inference. A variant called CLIP2DEPTH [2] uses a binning mechanism to associate each pixel with a depth bin named in text, leveraging the textual power from the original CLIP. However, this requires a hyperparameter for the number of bins, making training harder and increasing memory usage for the associated latent spaces.

2.9 Uncertainty in-depth prediction

In supervised training of a depth prediction model from RGB images, the quality of the resulting depth map is heavily influenced by the quality of the ground truth data. Commonly, Stereo-Setups or 3D LiDAR point cloud projections onto 2D image space are used to create these ground truth maps. However, both methods have their limitations, leading to uncertainties in the ground truth data. For instance, stereo-generated depth maps might lack information in regions with occlusions, while LiDAR-projected maps might not provide depth information for infinitely distant objects and suffer from errors in the projection itself. These uncertainties might not significantly impact indoor scenarios, but they can greatly affect the quality of depth maps in outdoor scenes. Outdoor scenes often contain far away objects and common occlusions, and they are subject to more extreme lighting conditions and weather patterns. For example, the sky, an infinitely distant object, does not reflect any surface, so LiDAR does not produce any depth information for it. Additionally, rain droplets on the camera lens create occlusions, resulting in wide areas with inconsistent RGB features, compared to neighboring pixels. However, sensors like LiDAR, which obtain their depth values by light ray reflections, can mitigate these issues to some extent.

To address these problems, the research community has proposed various methods to handle uncertainty in depth prediction. A common approach is to use probabilistic models, which can provide not just a single depth prediction, but also a measure of uncertainty associated with that prediction. This uncertainty can be used to identify regions in the depth map where the model's confidence is low, allowing for more robust decision-making in downstream tasks. For instance, the work of Kendall et al. [37] introduced a Bayesian neural network for depth prediction, which estimates the uncertainty in depth predictions by sampling from the posterior distribution of the network weights. This method provides a measure of uncertainty that can be used to improve the accuracy of depth predictions in challenging scenarios.



RGB LiDAR point cloud Prediction

Figure 2.9: Influence of uncertainty on depth prediction. ***Source:*** *Scene* ***20387*** *from the NuScenes dataset.*

3 Rigid Transformations

This chapter expands on the related work, delving into even greater detail. Given the remarkable works and research in this field, this section discusses the mathematical background of this thesis' innovative idea, aiming to push human knowledge further. The most novel approach of this thesis involves aligning latent spaces using linear transformations (rotations). Therefore, a thorough background on these fascinating functions is necessary.

3.1 Definition

Rigid transformations are transformations within the Euclidean space, that relate two bodies. Typically, these bodies are represented as sets of points in space, each of some dimension $n \in \mathbb{R}^D$. These transformations preserve distances and angles between points, thereby maintaining the structure of the bodies. The goal of rigid transformations is to align the source body X_s with the target body X_t, often by minimizing the distance between their centroids in space. This means that while the shape and size (scale) of geometric objects remain unchanged, their position and orientation may vary. However, perfect alignment is theoretically achievable only when both bodies share the same shape or structure.

Mathematically, a rigid transformation T can be defined as a combination of a rotation R and a translation $\mathbf{t}$:

$$X_t = T(\mathbf{x}) = R\mathbf{x} + \mathbf{t}$$

where $\mathbf{x} \in \mathbb{R}^D$ is a point in space, $R \in \mathbb{R}^{D \times D}$ is a rotation matrix, and $\mathbf{t} \in \mathbb{R}^D$ is a translation vector.

3.2 Rotation matrices

The rotation matrices R belong to the special orthogonal group $SO(D)$, which is the set of $D \times D$ matrices that are orthogonal ($R^\top R = I \rightarrow R^{-1} = R^\top$) and have a determinant of 1 ($\det(R) = 1$). $SO(D)$ is a Lie-group, which means it is not only a group of matrices, but also a smooth manifold - there is a continuous interpolation between all elements on the manifold. Also, that means that group operations (multiplication and inversion)

are smooth and differentiable functions. In addition, an n-dimensional rotation has $\frac{n(n-1)}{2}$ degrees of freedom.

Since the translation and scale components of the transformations are straightforward, in the following chapter we will be mainly dealing with rotations, that represent D-dimensional smooth manifolds of functions.

3.3 Rotations in 2D and 3D - $SO(2)$ and $SO(3)$

In 2D, the rotation matrix represents a single angle of rotation around the origin of the Euclidean 2D space (0, 0):

$$R_{2\times 2} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.1}$$

A point (x_1, y_1) in the Euclidean 2D space is transformed to (x_2, y_2) by the rotation matrix with an angle θ:

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_1\cos\theta - y_1\sin\theta \\ x_1\sin\theta + y_1\cos\theta \end{bmatrix} \tag{3.2}$$

The norm is preserved, i.e., $\sqrt{x_1^2 + y_1^2} = \sqrt{x_2^2 + y_2^2}$. From a 3D perspective, this rotation occurs around the z-axis.

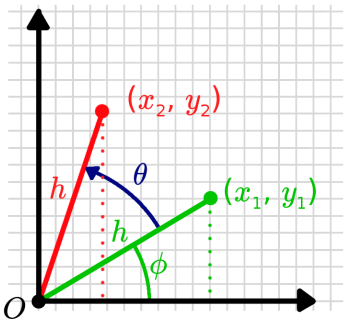


Figure 3.1: 2D rotations. The point (x_1, y_1), representing the vector $(1, 0)$ initially rotated by some angle ϕ, is transformed to (x_2, y_2) by the rotation matrix with an angle θ.

Therefore, a rotation matrix R in $SO(2)$ has a single **DoF!** (**DoF!**), the angle of rotation θ $\left(\text{DoF}(2) = \frac{2(2-1)}{2} = 1\right)$.

In 3D, rotations are more complex, with three degrees of freedom representing independent angles of rotation around the x, y, and z axes, known as "Euler angles".

Each rotation around one of the axes is represented by a 3×3 rotation matrix with a single degree of freedom θ_i, where $i \in \{x, y, z\}$, and θ_i is in the range $[0, 2\pi]$:

- Rotation around the x-axis: $R_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix}$
- Rotation around the y-axis: $R_{3\times3} = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix}$
- Rotation around the z-axis: $R_{3\times3} = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Given these three angles, any rotation matrix in $SO(3)$ can be represented as a composition of these three rotation matrices:

$$R_{3\times3}(\theta_x, \theta_y, \theta_z) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

This results in:

$$R_{3\times3}(\theta_x, \theta_y, \theta_z) =$$

$$\begin{bmatrix} \cos\theta_z\cos\theta_y\cos\theta_x - \sin\theta_z\sin\theta_y & -\cos\theta_z\cos\theta_y\sin\theta_x - \sin\theta_z\cos\theta_y & \cos\theta_z\sin\theta_y \\ \sin\theta_z\cos\theta_y\cos\theta_x + \cos\theta_z\sin\theta_y & -\sin\theta_z\cos\theta_y\sin\theta_x + \cos\theta_z\cos\theta_y & -\sin\theta_z\sin\theta_y \\ -\sin\theta_y & \cos\theta_y & 0 \end{bmatrix}$$

Since the rotations angles reside in a continuous space in $\mathbb{R}$, the interpolation between all rotations is smooth and differentiable. Also, since all rotation matrices are by definition orthonormal, their composition is also orthonormal. This is crucial, as the rotation must not change the scale of the input.

3.3.1 Quaternions

While Euler angles provide a straightforward way to represent rotations, they can suffer from issues such as **Gimbal Lock**, where the loss of one degree of freedom occurs due to the alignment of two rotation axes. To overcome these limitations, quaternions offer a more robust representation of 3D rotations.

A quaternion is a four-dimensional **complex number** of the form:

$$q = w + xi + yj + zk$$

where w, x, y, z are real numbers, and i, j, k are the fundamental quaternion units.

The quaternion representation of a rotation has several advantages:

- Avoidance of gimbal lock.
- Smooth interpolation (SLERP - spherical linear interpolation) between rotations.
- More compact and computationally efficient compared to rotation matrices (4 entries instead of 9).

A unit quaternion, which has a norm of 1, is used to represent rotations. The norm of a quaternion q is given by:

$$||q|| = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{3.4}$$

For q to be a unit quaternion, $||q|| = 1$. Using a normalized quaternion ensures that the quaternion represents a pure rotation without scaling. Also, as discussed later, this representation allows for robust learning of rotations through machine learning algorithms.

A notable property of quaternions is that a 3D rotation can be reconstructed from a unit quaternion $q = w + xi + yj + zk$:

$$R = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - wz) & 2(xz + wy) \\ 2(xy + wz) & 1 - 2(x^2 + z^2) & 2(yz - wx) \\ 2(xz - wy) & 2(yz + wx) & 1 - 2(x^2 + y^2) \end{bmatrix} \tag{3.5}$$

However, a **major drawback** of this representation is that each rotation matrix R can be represented by two different quaternions, q_1 and q_2. This phenomenon occurs because quaternions provide a "double-cover" of the special orthogonal group $SO(3)$. Specifically, if q_1 is a unit quaternion representing a certain rotation, then its negation, $q_2 = -q_1$, also represents the same rotation.

Mathematically, if

$$q_1 = w + xi + yj + zk$$

represents a rotation, then

$$q_2 = -q_1 = -w - xi - yj - zk$$

also represents the same rotation matrix R.

In practice, this means that using either q_1 or q_2 will result in the same rotation of a point $p = (p_1, p_2, p_3)$:

$$p' = q \cdot p \cdot q^{-1} = (-q) \cdot p \cdot (-q)^{-1} \tag{3.6}$$

Thus, the non-injective nature of the mapping from the quaternion space Q to the rotation space $SO(3)$ can lead to ambiguities in certain quaternion-based applications, and specifically when attempting to regress a rotation using deep-learning.

3.4 Rotations in n-dimensions $SO(n)$

Real-world problems often manifest in 3D space, and sometimes are reduced to 2D for simplicity. However, many mathematical challenges occur in high-dimensional spaces where the dimensionality far exceeds three components. For instance, the domain of images is represented in a space where each pixel contributes a dimension. While rotations in 2D and 3D are well understood, the concept becomes less intuitive in higher dimensions, despite the well-defined structure of $SO(n)$, the Special Orthogonal group in n-dimensions. Here, any orthonormal matrix R with a determinant of 1 belongs to $SO(n)$ and represents a rotation matrix.

However, representing rotations in higher dimensions is not as straightforward as in 2D and 3D, where methods like Euler angles or quaternions are commonly used. Furthermore, constructing rotations in higher dimensions is non-trivial due to the increased number of degrees of freedom, which can significantly complicate the process.

3.5 Straightforward Algorithm: Procrustes alignment

Assume two sets of points n-dimensional space, $X_s = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ and $X_t = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$, where X_s is the source set and X_t is the target set. The goal is to find a rigid transformation $T = s \odot (X_s \cdot R + t)$ that aligns the source set X_s with the target set X_t, where the two sets can differ in their orientation, position in space (poistion of their mean) and even scale (s).

The Procrustes alignment is a straightforward algorithm to retrive the optimal transformation between the two, such that it minimizes the Forbinius distance between the two sets of points:

$$\min_{R,t,s} \frac{1}{N} \sum_{i=1}^{N} ||s \odot (R\mathbf{x}_i + t) - \mathbf{y}_i||^2 \tag{3.7}$$

Where N is the number of points in the two sets. However, a perfect alignment is guaranteed only when the two sets share the same number of points shape or structure and that such transformation actually exists. In other words, both shapes are identical, but are placed differently in space due to some rigid transformation. The algorithm consists of the following steps:

Algorithm 1 Procrustes Alignment for Different Point Sets

Require: Two sets of points $X \in \mathbb{R}^{m \times d}$ and $Y \in \mathbb{R}^{n \times d}$, and a set of correspondences $C \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$

Ensure: Optimal rotation matrix R, translation vector T, and scale factor s

1: Extract corresponding points:

$$X_C = \{X_i \mid (i,j) \in C\}, \quad Y_C = \{Y_j \mid (i,j) \in C\}$$

2: Compute the centroids of X_C and Y_C:

$$\mu_X = \frac{1}{|C|} \sum_{(i,j)\in C} X_i, \quad \mu_Y = \frac{1}{|C|} \sum_{(i,j)\in C} Y_j$$

3: Center the point sets:

$$X'_C = X_C - \mu_X, \quad Y'_C = Y_C - \mu_Y$$

4: Compute the covariance matrix K:

$$K = {X'_C}^{\top} Y'_C$$

5: Perform Singular Value Decomposition (SVD) on K:

$$U, \Sigma, V^{\top} = \text{SVD}(K)$$

Algorithm 1 Procrustes Alignment for Different Point Sets (cont.)

1: Compute the optimal rotation matrix R:

$$R = VU^\top$$

2: **if** $\det(R) \neq 1$ **then**
3: Flip the sign of the last column of R:
4: **end if**
5: Compute the optimal scale factor s:

$$s = \frac{\text{trace}(\Sigma)}{\text{trace}(X'^\top_C X'_C)}$$

6: Compute the optimal translation vector T:

$$T = \mu_Y - sR\mu_X$$

7: **return** R, s, T

```python
import torch

def procrustes_alignment_batch(X, Y):
    """
    Perform Procrustes alignment on batches of point sets X and Y.

    Parameters:
    X (torch.Tensor): Points of shape (batch_size, n, d).
    Y (torch.Tensor): Points of shape (batch_size, n, d).

    Returns:
    R (torch.Tensor): Rotation matrices (batch_size, d, d).
    s (torch.Tensor): Scale factors (batch_size,).
    T (torch.Tensor): Translation vectors (batch_size, d).
    """
    batch_size, n, d = X.shape

    # Compute centroids and center point sets
    mu_X = X.mean(dim=1, keepdim=True)
    mu_Y = Y.mean(dim=1, keepdim=True)
    X_prime = X - mu_X
    Y_prime = Y - mu_Y

    # Compute covariance matrix and SVD
    C = torch.matmul(X_prime.transpose(1, 2), Y_prime)
    U, S, V = torch.svd(C)

    # Compute rotation matrices
    R = torch.matmul(V, U.transpose(1, 2))

    # Adjust rotation matrices to ensure det(R) = 1
    det_R = torch.det(R)
    R *= (det_R < 0).unsqueeze(1).unsqueeze(2).expand_as(R)

    # Compute scale factors and translation vectors
    s = torch.sum(S, dim=1) / torch.einsum('bij,bij->b', X_prime, X_prime)
    T = mu_Y.squeeze(1) - s.unsqueeze(1) * torch.matmul(R, mu_X.squeeze(1).
        unsqueeze(2)).squeeze(2)

    return R, s, T
```

Listing 3.1: Compact PyTorch Implementation of Procrustes Alignment with Determinant Adjustment

3.6 Deep Learning on 3D Transformations

Rigid transformations, and rotations in particular, are fundamental concepts in computer vision and robotics. They describe the orientation and position of objects in 3D space. In the context of deep learning, learning and predicting 3D transformations is crucial for tasks such as object detection, pose estimation, and 3D reconstruction. However, deep neural networks often struggle to regress elements on the $SO(3)$ manifold due to their tendency to find the simplest solution to a problem rather than the most accurate one. Furthermore, these networks typically produce features in a Euclidean space, which does not align well with many computer vision tasks, that prefer working within specific manifolds or spaces. This section will discuss various representations for 3D rotations, their properties, and the challenges associated with regressing them, as explored in the insightful papers by the author of the PyTorch Library RoMa ([38]) and ([39]).

3.6.1 Regression to the $SO(3)$ Manifold

While $SO(3)$ represents a specific manifold, real-world problems are typically situated in three-dimensional space. Since neural networks usually produce features in Euclidean space, various techniques have been explored to project these features onto the $SO(3)$ manifold. An important property of such projection functions $F : \mathbb{R}^n \to SO(3)$ is differentiability, which is crucial for backpropagation during neural network training, as it requires the gradients of the loss function with respect to the network's parameters.

The Euclidean space and the $SO(3)$ manifold are not homeomorphic, meaning there is no continuous bijective function that can map features (or points in 3D Euclidean space, $v \in \mathbb{R}^3$) between them. Consequently, several approaches have been proposed to address this challenge:

- **Discretization:** Instead of regressing a precise or unique rotation, one can fix a set of k rotation anchors and classify the desired Euclidean features to the nearest anchor. For instance, in some works ([40]), it was demonstrated that rigid transformations could occur between latent spaces from the same distribution. Since the latent space manifold is flexible and unknown, anchors were used to help the network learn transformations between these latent spaces.

- **Differentiable Mapping:** A continuous mapping from Euclidean space to the $SO(3)$ manifold is preferred, as it leverages deep learning more effectively for the task at hand. However, for such regression to be feasible, the mapping functions must be differentiable to ensure proper behavior during backpropagation. The papers discussed describe various differentiable mappings to different rotation

representations, such as Euler angles, quaternions, 6-Degree-Of-Freedom (6D) parameters, and full 3×3 rotation matrices, each with its own advantages and drawbacks. Additionally, desirable properties for these functions include:

1. **Surjective:** For every function f on the regressed differentiable manifold X, there must exist a corresponding $y \in Y$ such that $f(x) = y$ for all $x \in X$.
2. **Differentiable:** Given a loss function $L : X_{N \times M} \to \mathbb{R}$, the derivative $\frac{dL}{dX} = \frac{dL}{df}\frac{df}{dX}$ must exist and be continuous.
3. **Connected Pre-images:** A desirable property of the mapping function f is that it should map connected and continuous inputs or "pre-images" on the manifold X. If two intermediate regressed latent spaces x_i and x_j are poorly connected, but $f(x_i) \sim f(x_j)$, learning a well-generalized mapping f becomes more challenging. Therefore, the model should ensure that the regressed pre-images are continuous and connected.
4. **Full-Rank Jacobian (optional):** The Jacobian matrix of a mapping function f contains all first-order partial derivatives of the output tensor Y with respect to the input tensor X:

$$\frac{\partial Y}{\partial X} = \begin{bmatrix} \frac{\partial Y_1}{\partial X_1} & \frac{\partial Y_1}{\partial X_2} & \cdots & \frac{\partial Y_1}{\partial X_n} \\ \frac{\partial Y_2}{\partial X_1} & \frac{\partial Y_2}{\partial X_2} & \cdots & \frac{\partial Y_2}{\partial X_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial Y_n}{\partial X_1} & \frac{\partial Y_n}{\partial X_2} & \cdots & \frac{\partial Y_n}{\partial X_n} \end{bmatrix}$$

   A full-rank Jacobian of shape $n \times m$ means that the matrix's rank, which is the number of linearly independent columns, equals $\min(n, m)$. In the context of the Jacobian, a full-rank matrix indicates that the mapping function f is invertible and bijective, meaning for every $x \in X$, there is a unique $y \in Y$ such that $f(x) = y$. This ensures the function behaves "nicely" on the manifold and is differentiable.

The following section will discuss different representations of rotations within the $SO(3)$ manifold. For detailed mathematical proofs and background, please refer to the original papers, where each representation is explored in depth. The methods are summarized in the table below, adapted from [38]:

Table 3.1: Satisfaction of properties of section 2 by $SO(3)$ mappings.

Domain	Pre Image Shape	Surjective	Full rank	Connected/convex pre-images
Euler angles	$v \in \mathbb{R}^3$	✓	×	×
Rotation vector	$v \in \mathbb{R}^3$	✓	×	×
Quaternion	$v \in \mathbb{R}^4 \setminus \{0\}$	✓	✓	✓
Gram-Schmidt (6D)	$\mathcal{M}_{3,2} \in \mathbb{R}^{3\times 2}$	✓	✓	✓
Procrutes (9D)	$\mathcal{M}_{3,3} \in \mathbb{R}^{3\times 3}$	✓	✓	✓

3.6.2 Loss Functions for 3D Rotations

According to [38], neural networks usually regress values in an N-dimensional Euclidean continuous space. To project these values onto a different manifold, a suitable projection function must be chosen carefully. Neural networks minimize their loss functions through backpropagation, and these functions guide the learning process. When learning 3D rotations, which lie on the $SO(3)$ manifold rather than in Euclidean space, one could choose to penalize either the preimage (the values predicted by the neural network in Euclidean space) or the image (the projected values on the $SO(3)$ manifold). The papers discussed [38], [39] typically do not explore loss functions applied to the preimages, since the corresponding ground truth is usually not available. However, this approach could be interesting to investigate. Given two matrices of size 3×3, the potential loss functions can be grouped into two categories:

Loss functions on the pre-images

These operate directly on the Euclidean space. The matrices in this space are not required to follow any normalization, and specifically, they are not necessarily orthonormal. Within this category, we can further divide the loss functions:

- Element-wise: This family of loss functions includes those that penalize individual elements independently, without considering the relationships between different elements in the matrix. Examples include MAE, MSE, Huber, BerHu [41], L1-smooth, etc. For example, the MSE is defined as:

$$\texttt{MSE}(\texttt{R}, \hat{\texttt{R}}) = \frac{1}{9N} \sum_{i=1}^{N} \sum_{j=1}^{9} (R_{ij} - \hat{R}_{ij})^2$$

  However, these functions ignore some critical properties. For instance, two Euclidean pre-images that are close in Euclidean space might be far apart on the $SO(3)$ manifold. Additionally, no relationship between the elements is considered.

- Loss functions that consider relationships:
  - Cosine Similarity: Given two rotation matrices R_1 and R_2, flatten the matrices into vectors. The cosine similarity is then defined as:

$$\texttt{CosineSimilarity}(R_1, R_2) = \frac{1}{N} \sum_{i=1}^{N} \frac{R_1 \cdot R_2}{\|R_1\|_2 \cdot \|R_2\|_2} \tag{3.8}$$

    This is a common technique for aligning two vectors in a latent space. A result of 1 indicates that the vectors are perfectly aligned in direction (linearly dependent), while a result of -1 indicates that they are perfectly aligned in the opposite direction. A result of 0 means the vectors are orthogonal. This loss function considers the cosine angle between the flattened rotation vectors of R_1 and R_2. It measures how close the two rotations are but does not enforce any orthogonality constraints. Additionally, it does not account for the scale of these matrices, which might cause issues when projecting onto the $SO(3)$ manifold.

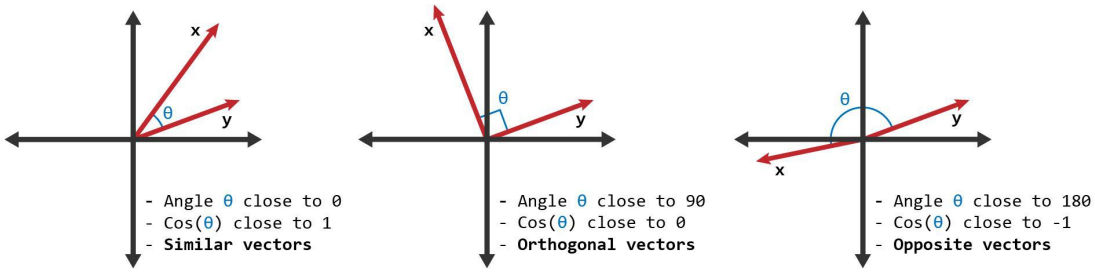


Figure 3.2: Different angles in 2D. Alignment of two vectors results with a very small angle θ between them.

  - Orthogonality Norm: Given two rotation matrices R_1 and R_2, the orthogonality norm is defined as:

$$\texttt{Orthogonality}(R_1, R_2) = \frac{1}{9N} \sum_{i=1}^{N} \sum_{j=1}^{9} \left[(R_1 \cdot R_2^T - I_{3\times3}) \right]_{ij} \tag{3.9}$$

    This loss function not only computes the cosine angle between corresponding rows of R_1 and R_2 but also enforces that the matrices R_1 and R_2 are orthonormal. This constraint narrows down the vast space of potential random pre-images to a more well-defined set that can later be projected onto the $SO(3)$ manifold.

Loss functions on the $SO(3)$ manifold

These loss functions operate directly on the rotation matrices after they have been projected onto the $SO(3)$ manifold. The key advantage of these loss functions is that they respect the geometric properties and constraints of the $SO(3)$ manifold, ensuring that the rotations learned by the neural network are valid and meaningful:

- Geodesic distance: Given two rotation matrices R_1 and R_2, the geodesic loss is defined as:

$$\texttt{Geodesic}(R_1, R_2) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{\text{tr}(R_1^T R_2) - 1}{2}\right) \tag{3.10}$$

  This loss function calculates the geodesic distance between the two rotation matrices R_1 and R_2 on the $SO(3)$ manifold. The geodesic distance is the shortest path between two points on a manifold, and in the case of $SO(3)$, it represents the angle of rotation between the two matrices. This loss function ensures that the rotations learned by the neural network are valid, as they respect the geometric properties and constraints of the $SO(3)$ manifold. However, since the `arccos` function is not differentiable at certain points, it can be challenging to optimize using gradient-based methods. In practice, the geodesic loss is often approximated by the `Chordal` loss, which is a smoother approximation that is easier to optimize [42].

- Chordal distance: The chordal loss computes the chordal distance between the two rotation matrices R_1 and R_2 on the $SO(3)$ manifold. The chordal distance is a smooth approximation of the geodesic distance, making it easier to optimize using gradient-based methods. This loss is a good choice for training neural networks to learn 3D rotations, as it respects the geometric properties and constraints of the $SO(3)$ manifold while being differentiable and easier to optimize.

$$\texttt{Chordal}(R_1, R_2) = \frac{1}{N}\sum_{i=1}^{N}(\|R_1 - R_2\|_F) = \frac{1}{N}\sum_{i=1}^{N}\left(\|R_1^T R_2 - I_{3\times 3}\|_F\right) \tag{3.11}$$

- RoMa's Geodesic Distance: According to the identity [38]:

$$\|R_2 - R_1\|_F = 2\sqrt{2}\sin\left(\frac{\alpha}{2}\right)$$

  The paper proposes using an angular version, which is to be minimized:

$$\texttt{RoMa}(R_1, R_2) = \frac{1}{N}\sum_{i=1}^{N} 2\sin^{-1}\left(\frac{\left(\|R_1^T R_2 - I_{3\times 3}\|_F\right)}{2\sqrt{2}}\right) \tag{3.12}$$

3.6.3 Euler Angles

Euler angles are a fundamental representation of 3D rotations, using a set of three angles to describe the rotation of a rigid body in three-dimensional space. These angles, typically denoted as α, β, and γ with values in the range $[-\pi, \pi)$, correspond to rotations around the x, y, and z axes, respectively. The rotation matrix R can be expressed as a composition of three rotation matrices $R_x(\alpha)$, $R_y(\beta)$, and $R_z(\gamma)$:

$$R = R_z(\gamma)R_y(\beta)R_x(\alpha) \in SO(3).$$

As shown in Equation 3.3. While Euler angles are surjective ($\forall x \in X, f(x) \in Y$), they do not ensure a continuous mapping, and the pre-images are not necessarily connected. This is because multiple sets of Euler angles can describe the same rotation. For example, both $r_1 = [0, \frac{\pi}{2}, 0]$ and $r_2 = [-\frac{\pi}{2}, \frac{\pi}{2}, -\frac{\pi}{2}]$ represent the same rotation. Additionally, the Jacobian of Euler angles can suffer from "rank deficiency," a problem known as "Gimbal Lock."

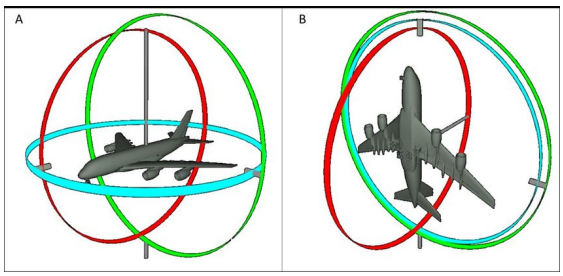


Figure 3.3: Illustration of gimbal lock, a phenomenon in which the axes of a gimbal become aligned, resulting in a loss of one degree of freedom. A known issue that could lead to undefitting in deep learning settings.

Gimbal Lock occurs in 3D rotational systems when the axes of two of the three gimbals (rotational axes) align, leading to the loss of one degree of freedom. In essence, Gimbal Lock makes certain rotations impossible because the orientation system loses one degree of freedom. In deep learning applications involving 3D transformations, such as pose estimation or orientation prediction, Gimbal Lock poses a significant challenge, and could easily result with convergence to suboptimal set of points or even undefitting, as it can result in multiple sets of Euler angles representing the same orientation, complicating optimization as the model receives conflicting signals for the same output. Moreover, it can make the model more sensitive to small angular changes, which can destabilize training. This instability arises because gradients near Gimbal Lock can either saturate or explode, raising even more difficulties on the optimization process of the model.

Learning Euler angles involves regressing 3 degrees-of-freedom (DoF) vector for each rotation. Since Euler angles are represented in Euclidean space, simple regression loss functions such as Mean Squared Error (MSE) or Mean Absolute Error (MAE) can be used on the preimages, if available: Given a ground truth $y = \{\alpha, \beta, \gamma\}$ and predicted angles $\hat{y} = \{\hat{\alpha}, \hat{\beta}, \hat{\gamma}\}$, the loss function can be defined as:

$$\texttt{MSE}(\hat{y}, y) = \frac{1}{3N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{3.13}$$

where N denotes the number of rotations predicted.

3.6.4 Rotation Vector

Also known as "Rodrigues Parameters" or "Exponential Coordinates", the rotation vector is another representation of 3D rotations. This method uses a 3-dimensional vector to represent the rotation. The direction of the vector indicates the axis of rotation, while its magnitude corresponds to the angle of rotation. Formally, the rotation vector r is defined as

$$r = \theta u \tag{3.14}$$

where θ is the angle of rotation and u is a unit vector (normalized to lie on the unit sphere) representing the axis of rotation. The rotation matrix R can be derived from the rotation vector using Rodrigues' formula:

$$R = I + \sin(\theta)K + (1 - \cos(\theta))K^2, \tag{3.15}$$

where K is the skew-symmetric matrix corresponding to the unit vector u:

$$K = \begin{bmatrix} 0 & -u_z & u_y \\ u_z & 0 & -u_x \\ -u_y & u_x & 0 \end{bmatrix}.$$

This representation is surjective and offers several advantages over Euler angles. It provides a compact representation with only 3 degrees of freedom (2 for the unit vector, as the third component is constrained by $\|u\| = 1$, and 1 for the angle θ). It avoids the "gimbal lock" issue and allows for convenient interpolation between instances on $SO(3)$. However, rotation vectors can suffer from angle ambiguity and have been shown to be less effective globally in predicting desired rotation matrices in deep learning scenarios. The rotation vector can be regressed in Euclidean space as a preimage, using a loss function similar to Equation 3.13.

3.6.5 Quaternion

As described in Subsection 3.3.1, the "quaternion" is a common and powerful tool for representing rotations in 3D space. Every 4-dimensional vector is, by definition, a quaternion. Therefore, it is straightforward to learn using deep learning, as the image is achieved by simply normalizing the regressed preimage.

$$q' = \frac{q}{||q||} \tag{3.16}$$

Where $||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}$, q is the regressed preimage and q' is the normalized quaternion.

Therefore, it could be simply regressed given one of the regression loss function, such as MAE or MSE:

$$\texttt{MSE}(\hat{q'}, q') = \frac{1}{4N}\sum_{i=1}^{N}(q'_i - \hat{q'_i})^2 \tag{3.17}$$

Since quaternions represent orientations in space, we can leverage this fact to minimize the distance between them using the geodesic distance. The geodesic distance is the shortest path between two points on a curved surface, such as a sphere. To compute the angular distance between two quaternions, we look for the angle θ between them in 4D space. This approach is crucial because simply taking the Euclidean difference between quaternions in their vector form does not accurately reflect the angular difference between the rotations. The angle between two vectors $q_1, q_2 \in \mathbb{R}^N$ is defined as follows:

$$\cos(\frac{\theta}{2}) = q_1 \cdot q_2 \tag{3.18}$$

And therefore, the geodesic distance between two **unit** quaternions $q_1, q_2 \in \mathbb{R}^4$ is defined as:

$$\theta = 2\arccos\left(q_1 \cdot q_2\right) \tag{3.19}$$

as $\|q_1\| = \|q_2\| = 1$.

In addition, as previously mentioned, a significant drawback of using quaternion representation for rotations is that both a quaternion q and its opposite $-q$ represent the same rotation. This is due to the fact that both quaternions yield the same rotation outcome, as described in Equation 3.6. Although this doesn't affect the application of rotations, it can introduce ambiguity when computing the angle between vectors using Equation 3.19, potentially resulting in either q or $-q$ as the regressed quaternion.

To address this issue, the author of [38] introduced a loss function that avoids this ambiguity, as detailed in Equation 3.21.

For unit quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$, the squared Euclidean distance between them is:

$$\min \|q_2 \pm q_1\| = 2 \sin\left(\frac{\theta}{4}\right)$$

Which into consideration the ambiguity. For simplicity, let's take $\|q_1 - q_2\|$:

$$\|\mathbf{q}_1 - \mathbf{q}_2\|^2 = (\mathbf{q}_1 - \mathbf{q}_2) \cdot (\mathbf{q}_1 - \mathbf{q}_2) = \|\mathbf{q}_1 - \mathbf{q}_2\|^2 = \mathbf{q}_1 \cdot \mathbf{q}_1 - 2(\mathbf{q}_1 \cdot \mathbf{q}_2) + \mathbf{q}_2 \cdot \mathbf{q}_2$$

Since $\mathbf{q}_1$ and $\mathbf{q}_2$ are **unit**-quaternions:

$$\|\mathbf{q}_1 - \mathbf{q}_2\|^2 = 1 - 2(\mathbf{q}_1 \cdot \mathbf{q}_2) + 1 = 2 - 2(\mathbf{q}_1 \cdot \mathbf{q}_2)$$

Using the relationship of Equation 3.18:

$$\|\mathbf{q}_1 - \mathbf{q}_2\|^2 = 2 - 2\cos\left(\frac{\theta}{2}\right)$$

Applying the trigonometric identity $1 - \cos(x) = 2\sin^2\left(\frac{x}{2}\right)$:

$$\|\mathbf{q}_1 - \mathbf{q}_2\|^2 = 2 \cdot 2\sin^2\left(\frac{\theta}{4}\right)$$

And finally, by taking the square root we get:

$$\|\mathbf{q}_1 - \mathbf{q}_2\| = 2\sin\left(\frac{\theta}{4}\right) \tag{3.20}$$

Now, solving for θ:

$$\sin\left(\frac{\theta}{4}\right) = \frac{\|\mathbf{q}_1 - \mathbf{q}_2\|}{2} = \frac{\min \|\mathbf{q}_1 \pm \mathbf{q}_2\|}{2}$$

Therefore:

$$\frac{\theta}{4} = \arcsin\left(\min \frac{\|\mathbf{q}_1 \pm \mathbf{q}_2\|}{2}\right)$$

Which eventually gives:

$$\theta = 4 \cdot \arcsin\left(\frac{1}{2} \min \|\mathbf{q}_1 \pm \mathbf{q}_2\|)\right) \tag{3.21}$$

Finally, it is important to note that this representation works only for

$$Q = \{q | q = \frac{x}{\|x\|}, x \in \mathbb{R}^4 \setminus \{0\}\}$$

so it doesn't satisfy the pre-image connectivity constraint. However, it doesn't impose a real problem in practice.

3.6.6 6D — Gram-Schmidt Orthonormalization

Zhou *et al.* [43] demonstrated that the Gram-Schmidt orthonormalization process can be used to project a 6-DoF pre-image onto SO(3). This method transforms a set of linearly independent vectors into an orthonormal set, ensuring that the vectors are both orthogonal and normalized, with each vector having a unit length.

The process begins with two given vectors, which are normalized to produce the first two orthonormal vectors. These vectors are associated with 6 degrees of freedom (DoFs). Interestingly, [43] demonstrated that this task could also be accomplished using only 5 DoFs through stereographic projection [44], although this approach has been observed to yield poorer training performance in practice.

Subsequently, the algorithm generates a third vector that is orthogonal to the first two, resulting in a 3×3 orthonormal matrix. However, this process does not guarantee a determinant of 1. To address this, one can use the method outlined in Algorithm 1 to adjust the determinant, thereby transforming the matrix from a reflection to a proper rotation.

Algorithm 2 Gram-Schmidt Orthonormalization for Three Vectors

Require: Two linearly independent vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ in $\mathbb{R}^3$
Ensure: Three orthonormal vectors $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$

1: $\mathbf{u}_1 \leftarrow \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}$ ▷ Normalize the first vector
2: $\mathbf{u}_2 \leftarrow \mathbf{v}_2 - \langle \mathbf{v}_2, \mathbf{u}_1 \rangle \mathbf{u}_1$ ▷ Subtract projection onto $\mathbf{u}_1$
3: $\mathbf{u}_2 \leftarrow \frac{\mathbf{u}_2}{\|\mathbf{u}_2\|}$ ▷ Normalize the second vector
4: $\mathbf{u}_3 \leftarrow \mathbf{u}_1 \times \mathbf{u}_2$ ▷ Compute the cross product to find the third orthonormal vector
5: **return** $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$

The algorithm outlined in Algorithm 2 takes two linearly independent vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ in $\mathbb{R}^3$ and produces three orthonormal vectors $\mathbf{u}_1$, $\mathbf{u}_2$, and $\mathbf{u}_3$. It starts by normalizing $\mathbf{v}_1$ to obtain $\mathbf{u}_1$. For $\mathbf{v}_2$, the algorithm subtracts the projection of $\mathbf{v}_2$ onto $\mathbf{u}_1$ to produce a new vector that is orthogonal to $\mathbf{u}_1$, then normalizes this vector to get $\mathbf{u}_2$. The third vector, $\mathbf{u}_3$, is computed as the cross product of $\mathbf{u}_1$ and $\mathbf{u}_2$, ensuring that it is orthogonal to both. This method ensures the generation of three orthonormal vectors

suitable for forming a basis in 3D space, essential for applications such as rotations represented by SO(3).

For a general case where the number of input vectors is m, the Gram-Schmidt orthonormalization process can be extended to handle this scenario. The algorithm transforms a set of m linearly independent vectors into an orthonormal set. This generalization is applicable as long as the number of vectors does not exceed the dimensionality of the space, which is n. For instance, in $\mathbb{R}^3$, you can have at most 3 linearly independent vectors to form a complete orthonormal basis.

The process involves normalizing each vector and then subtracting the projections onto all previously computed orthonormal vectors to ensure orthogonality. Each resulting vector is then normalized to ensure it has unit length.

Algorithm 3 Gram-Schmidt Orthonormalization for m Vectors

Require: m linearly independent vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m\}$ in $\mathbb{R}^n$
Ensure: Orthonormal vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$
Initialize $\mathbf{U} \leftarrow \emptyset$
for $k = 1$ to m **do**
    $\mathbf{u}_k \leftarrow \mathbf{v}_k$
    for $j = 1$ to $k - 1$ **do**
        $\mathbf{u}_k \leftarrow \mathbf{u}_k - \langle \mathbf{v}_k, \mathbf{u}_j \rangle \mathbf{u}_j$
    end for
    if $\|\mathbf{u}_k\| \neq 0$ **then**
        $\mathbf{u}_k \leftarrow \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|}$
    else
        Error: Vectors are not linearly independent.
    end if
    Append $\mathbf{u}_k$ to $\mathbf{U}$
end for
return $\mathbf{U}$

To extend this algorithm for generating a new orthonormal vector from m existing orthonormal vectors, follow these steps:

1. **Form a New Vector**: Start with a new vector $\mathbf{v}_{new}$ that is not in the span of the existing $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$.

2. **Apply Gram-Schmidt**: Use the Gram-Schmidt process to orthogonalize $\mathbf{v}_{new}$ with respect to the existing $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$.

3. **Normalize**: Normalize the resulting vector to obtain a new orthonormal vector $\mathbf{u}_{new}$.

However, this method is only valid if the input vectors are linearly independent. To ensure linear independence before applying the Gram-Schmidt algorithm, a regularization term that enforces orthonormality can be introduced. Given two sets of normalized vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, the regularization term can be defined similarly to Equation 3.11:

$$\mathrm{L}(\mathbf{X}_1, \mathbf{X}_2) = \frac{1}{N \cdot M} \|\mathbf{X}_1 \cdot \mathbf{X}_2 - I_{N \times M}\|,$$

where N and M denote the dimensions of these matrices.

After projection, one can apply specific loss functions designed for the SO(3) manifold to compute gradients for the backpropagation process. For instance, the geodesic loss functions given by Equation 3.10 and Equation 3.12 can be used to guide the training of the model, ensuring that the learned rotations are accurate and valid.

As discussed in [38], and as reflected in Table 3.1, this method meets all the criteria for a robust projection technique. Moreover, both [43] and [38] report that this process outperforms previous methods in terms of performance.

3.6.7 Procrutes Orthonormalization

Any 3×3 matrix M can be approximated by the nearest rotation matrix through the *Frobenius* norm minimization problem, known as **the special orthogonal Procrustes problem**. This is a similar process, to the one demonstrated of the rotation finding in Algorithm 1:

$$\text{Procrustes}(M) = \arg \min_{\tilde{R} \in \mathrm{SO}(3)} \|\tilde{R} - M\|_F^2$$

To find the optimal rotation matrix $\tilde{R}$, we solve this minimization problem using the singular value decomposition (SVD) of M. If M is decomposed into $UDV^\top$, where U and V are orthogonal matrices and D is a diagonal matrix with singular values $\alpha_1 \geq \alpha_2 \geq \alpha_3 \geq 0$, the optimal rotation matrix $\tilde{R}$ can be computed as:

$$\tilde{R} = U S V^\top$$

In this expression, the matrix S is defined as:

$$S = \mathrm{diag}(1, 1, \det(U) \det(V))$$

where $\det(U)$ and $\det(V)$ are the determinants of the orthogonal matrices U and V, respectively. This ensures that the final rotation matrix $\tilde{R}$ has a determinant of $+1$, thereby guaranteeing that $\tilde{R}$ is indeed a proper rotation matrix and not a reflection.

The uniqueness of the solution is assured under the conditions that $\det(M) > 0$ or $\alpha_2 \neq \alpha_3$. In practical terms, the Procrustes mapping is well-defined for almost all 3×3 matrices except for those in a set of measure zero, which are usually ignored. This approach provides a reliable method for finding the nearest rotation matrix to any given matrix M, aligning with the requirements and properties discussed Table 3.1.

In their work, [38] demonstrated that using this projection function achieves the best results while adding only a negligible overhead of 3 additional degrees of freedom per learned rotation. To learn the rotation obtained by this method, one can either learn the pre-image or directly learn the rotation itself by employing one of the loss functions presented in Subsection 3.6.2.

4 Deep learning on latent spaces

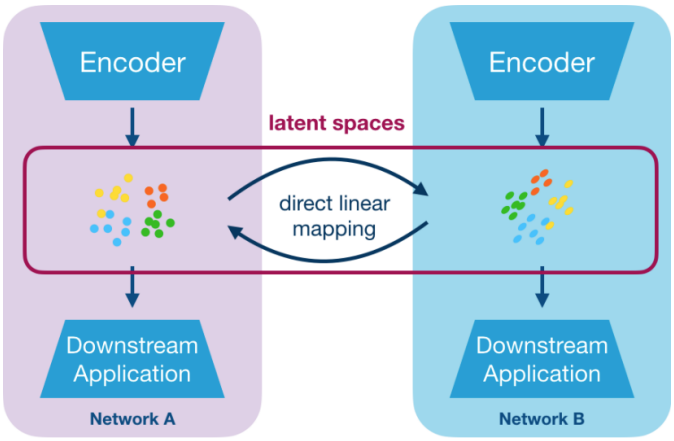


Figure 4.1: Sharing information between modalities

Representation learning [45] is a central approach in artificial intelligence, focused on uncovering the underlying structure of complex data. The goal is to discover robust data representation. However, modeling these manifolds poses challenges due to stochasticities in training and external factors that can lead to inconsistent representations for similar data samples, ultimately hindering knowledge transfer between networks.

To overcome these challenges, the concept of relative representations [46] has been introduced. By projecting the latent spaces of neural networks trained on similar data into a shared relative space based on distances between data points, this approach captures the intrinsic structure of the latent space. Focusing on angles relative to anchor points, it achieves results comparable to those using original encodings and shows that different latent spaces sharing the same data semantics primarily differ by an angle-preserving transformation. This concept becomes even more compelling when aligning latent spaces across different modalities.

Deep learning has revolutionized various fields by allowing models to learn complex patterns directly from data, often across multiple dimensions. This capability has led to new advancements in machine learning, particularly through the use of low-

dimensional latent spaces where data is encoded into compact, abstract forms that are both flexible and dynamic.

Generative models [21] such as Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Diffusion Models are prime examples of how latent spaces are utilized in deep learning. These models map data into a latent space—a lower-dimensional, continuous vector space — where the underlying structure is captured in a compressed form. VAEs explicitly model the latent space as a probabilistic distribution, allowing for the generation of new data samples by sampling from this distribution and decoding them back into the data space. GANs involve a generator that transforms random noise from the latent space into realistic data, while a discriminator learns to distinguish between real and generated data, refining the generative process. Diffusion models, a more recent innovation, reverse a diffusion process to gradually transform noise into data, offering precise control over the generation process and producing high-quality results. These generative models highlight the power of latent spaces in creating new, unseen data, with applications in image synthesis, style transfer, and beyond.

The key idea in these approaches is the ability to interpolate on a smooth manifold derived from the lossy encoders of neural networks. While two modalities may appear vastly different in the original image space, encoding them into lower-dimensional latent spaces can reveal shared properties, bringing them closer together. This enables meaningful transitions between seemingly unrelated data points, facilitating tasks such as interpolation, morphing, and style transfer [47].

Figure 4.2: Style Transfer: given an input and a target, change the input such that it follows the same texture patterns of the target, while maintaining its core structure.

In this thesis, the goal is to search for effective interpolation techniques between distinct modalities, such as RGB images and their corresponding depth maps, to improve the reconstruction of depth information from a monocular RGB image. This is particularly challenging, as depth perception often requires complex, multi-view inputs that are not directly available from a single RGB image alone. By leveraging the

smooth manifold structure within the latent space, it is attempted to minimize the gap between these modalities, enabling more accurate and reliable depth map prediction from single-view RGB images.

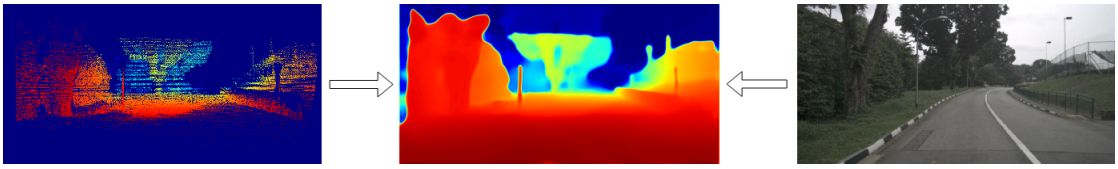


Figure 4.3: The goal. How to guide the RGB cues to produce a more accurate depth map, given the ground truth LiDAR data?

4.1 Autoencoders

Similar to the Principal Component Analysis (PCA) [48], neural networks can be trained to take very high dimensional inputs, such as RGB images, and learn to extract the most meaningful features out of them. This is done by using a lossy-architecture (some features will be discarded), but in contrast to PCA, that also introduces non-linearity while doing so, allowing to capture even more complex features. To that end, let's revise one of the most basic building stones of today's deep learning architectures — the Autoencoder.

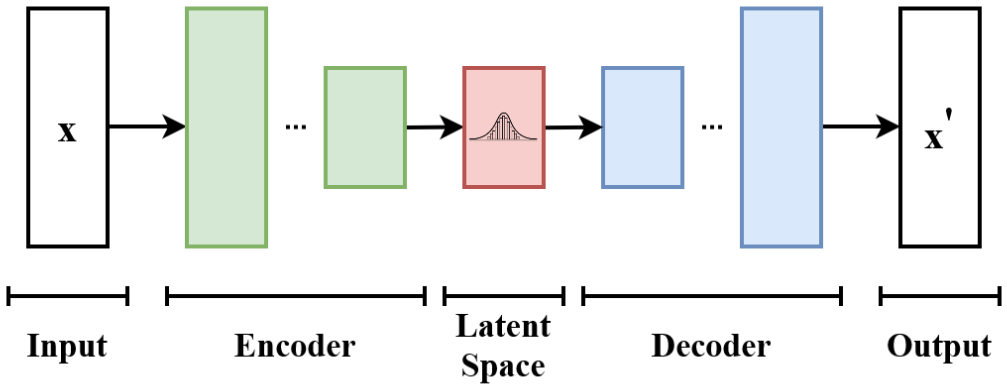


Figure 4.4: An autoencoder: The encoder encodes the high-dimensional input into a low-dimensional latent space with a non-linear lossy function. This latent space can take many forms. Then, it is decoded to solve a downstream task.

At its core, an autoencoder, as shown in Figure 4.4, is a neural network with three main components: an Encoder, a Latent Space, and a Decoder. The encoder takes a high-dimensional input and compresses it into a lower-dimensional representation, called the **latent space**. The decoder then tries to reconstruct the original input from this latent representation. The loss function measures the difference between the input and the decoder's output, guiding the training process so that the latent space captures the most important features of the input. Because of this, the encoder is often referred to as a "feature extractor."

The latent space is a condensed version of the input, usually containing only the most relevant features for the task. This reduction in dimensionality forces the network to filter out less important details. The latent space is also known as the "bottleneck" because it squeezes the input into a smaller, more compact form. The size of the latent space is critical: if it's too small, the decoder might struggle to accurately reconstruct the input. On the other hand, if it's too large, the network may retain unnecessary details, which goes against the encoder's purpose.

The shallower layers of the encoder extract low-dimensional features, such as edges and textures, while the deeper layers capture more abstract concepts, like shapes and objects.

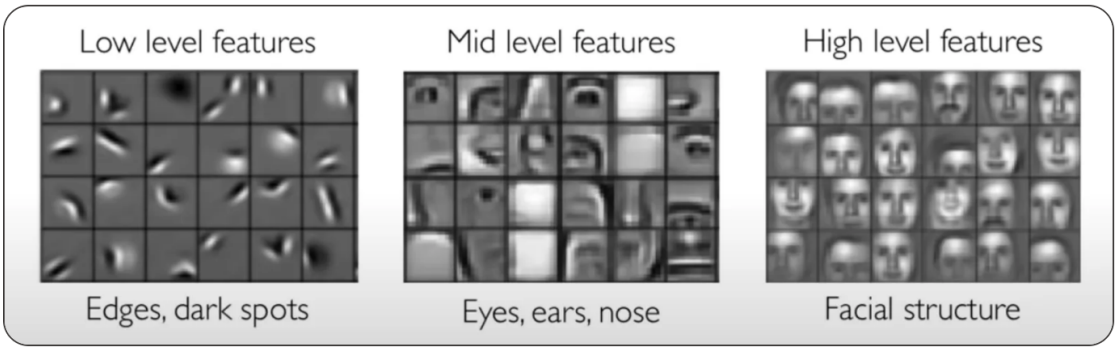


Figure 4.5: Feature level in different hidden layers of a neural network. While the lower level features capture patterns in the input, deeper layers already compute more task-specific features, to solve the task at hand.

The original autoencoder architecture is a fully-connected neural network, using only fully-connected layers. This means the encoder can only extract **global features** from the input, often leading to suboptimal results, especially when compared to architectures that use convolutions as their building blocks. Convolutions are better suited for computer vision tasks because they extract **local features**. These features are built up gradually, depending on the receptive field, which determines how many

pixels of the original input influence a pixel in an intermediate layer:

$$r_l = r_{l-1} + \left(\prod_{i=1}^{l-1} s_i \right) \cdot (k_l - 1) \quad \text{for} \quad l \geq 2 \tag{4.1}$$

Here, r_l is the receptive field size at the l-th layer, s_i is the stride in the i-th dimension, and k_l is the kernel size at the l-th layer. The receptive field of the first layer is simply $r_1 = k_1$.

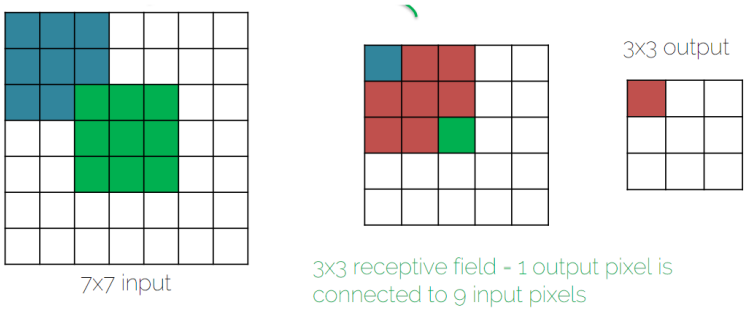


Figure 4.6: Example:
$r_2 = r_1 + (s_1) \cdot (k_2 - 1) = 3 + 1 \times (3 - 1) = 5$
$r_3 = r_2 + (s_2) \cdot (k_3 - 1) = 5 + 1 \times (3 - 1) = 7$

As the encoder goes deeper, pixels in the latent space capture more meaningful information, thanks to an increasing receptive field. Fully-convolutional autoencoders with skip connections between the encoder and decoder scales allow high-level features to flow directly to the decoder. This enhances output reconstruction because the latent space, while lossy, still benefits from some low-level features. The skip connections ensure that important details are retained and passed along, improving the overall quality of the reconstruction.

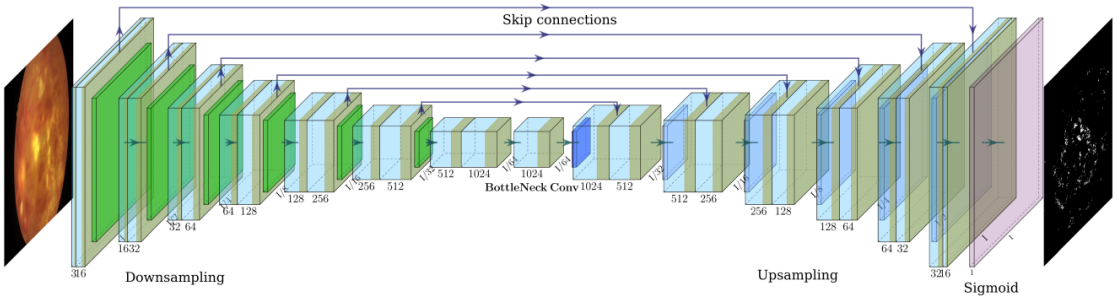


Figure 4.7: A fully convolutional Autoencoder, that incorporates skip-connections, to pass fine-grained features. A possible drawback: relax the reduction of the most useful features into the latent space.

However, while skip connections are beneficial for tasks like segmentation or depth estimation, they are less suitable for image reconstruction. The main goal in image reconstruction is to train the encoder to be an efficient feature extractor, and skip connections can undermine this by allowing the network to bypass the encoder's role, thereby reducing the focus on compressing information into the latent space. For this reason, skip connections should not always be the default, especially when the latent space needs to be as representative as possible.

4.2 Relative Representation of latent spaces

Relative representations are a novel approach in the field of representation learning [45]. Unlike traditional representations, which rely on absolute positions in the Euclidean space, relative representations focus on the relationships or differences between learned latent points.

Formally, let z_1 and z_2 be two points in the latent space, representing different data samples. The relative representation r between these two points is defined as:

$$r = z_2 - z_1$$

Here, z_1 is often referred to as the **anchor**, a known reference point in the latent space. The relative representation r encodes the transformation required to move from the anchor z_1 to the new point z_2.

For example, such a framework is introduced in [46]. Their method enables latent spaces of arbitrary neural models to communicate with each other. This is achieved by projecting the latent spaces into a common one, transitioning from an **absolute coordinate frame** to a *relative space*: each sample is represented as a function of a set of

fixed samples denoted as an *anchor set*. Specifically, the new representation is computed by independently projecting each sample point $\mathbf{x}$ in the latent space $\mathbf{X} \in \mathbb{R}^{n \times d}$ into the anchor set $\mathbf{A}_\mathbf{X} \subset \mathbf{X}$. Formally, this is expressed as

$$\tilde{\mathbf{X}}_{\text{rel}} = \tilde{\mathbf{X}}_{\text{abs}} \cdot \tilde{\mathbf{A}}_\mathbf{X}^T \tag{4.2}$$

where $\mathbf{X}_{\text{rel}} \in \mathbb{R}^{n \times k}$, $\mathbf{X}_{\text{abs}} \in \mathbb{R}^{n \times d}$ and $\mathbf{A}_\mathbf{X} \in \mathbb{R}^{k \times d}$. Samples in $\mathbf{X}$ and in $\mathbf{A}$ are rescaled to unit norm, i.e.,

$$\tilde{\mathbf{X}}_{\text{abs}} = \left\{ \frac{\mathbf{x}}{\|\mathbf{x}\|_2} \mid \mathbf{x} \in \mathbf{X} \right\}$$

$$\tilde{\mathbf{A}}_\mathbf{X} = \left\{ \frac{\mathbf{a}}{\|\mathbf{a}\|_2} \mid \mathbf{a} \in \mathbf{A}_\mathbf{X} \right\}$$

This is important, so the values obtained in Equation 4.2 represent the cosine similarity between the dataset's latent spaces and the fixed-anchors, which a geodesic distance on the unit-sphere.

4.3 Aligning latent spaces

Assume two models that encode different modalities into separate latent spaces. While latent spaces provide flexibility for manipulating encoded data, they also allow each model to use different representations that best suit the task and data. This can make aligning latent spaces across modalities very challenging, and sometimes even impossible.

Using an **absolute** representation in Euclidean space, for example, often leads to difficulties in alignment. To achieve meaningful alignment, one would typically look for a shared manifold where both latent spaces can reside with similar characteristics. However, this may result in some performance loss, as the shared manifold might not fully capture the specific needs of each modality.

Directly aligning two modalities by learning their representations in parallel can be tough. For instance, depth and RGB features, used for estimating depth in an RGB image, often occupy entirely different manifolds in the latent space, even when encoded by the **same** encoder. Finding a shared manifold between these spaces is difficult because the features represent very different information, both semantically and geometrically.

However, it's important to consider how closely the latent spaces need to be aligned. Recent research has explored the relationships between latent spaces for tasks like image classification. A well-known example is CLIP by OpenAI ([3]), which aligns image and text embeddings by clustering them based on their class. This approach

leads to impressive performance, not only on the training data but also in zero-shot experiments on other datasets, making the model general and robust. It's crucial to note that in this case, the alignment doesn't require image and text embeddings to be numerically identical in space (both in magnitude and direction). Instead, they are only clustered together to enhance class distinction. This type of alignment works well for discrete tasks like classification, but it's insufficient for regression tasks, such as depth estimation or image reconstruction, where precise numerical alignment is necessary for meaningful results.

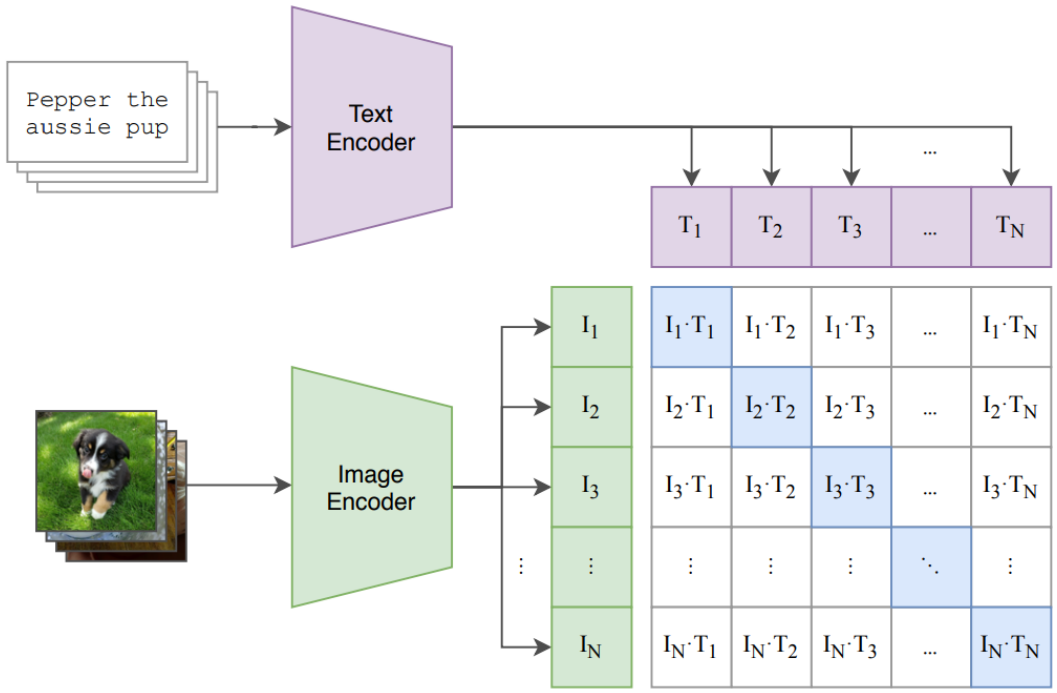


Figure 4.8: CLIP's contrastive learning of latent spaces. Bring semantically similar objects together and push the rest afar. Introduced an incredible leap in performance.

To address this, we can use more intelligent data representations, like projecting the data into a relative representation space Section 4.2, constraining the latent spaces to subspaces defined by a limited set of anchors, or finding projection functions between the spaces, such as linear or rigid transformations.

4.4 Rotations on Latent spaces

As mentioned earlier, aligning two modalities in latent space to achieve similar performance for downstream regression tasks is a very challenging problem. One approach to address these challenges is to rotate the latent spaces, aligning them in a shared relative space. This involves finding the optimal rotation matrix that minimizes the distance between corresponding points in the two spaces. By applying an **angle-preserving** rotation, as described in Chapter 3, we can align the latent spaces while preserving the relationships between data points, enabling direct inference between modalities (e.g., as shown in Figure 4.1).

In the works of [5], [46], [49], the authors discuss aligning latent spaces by finding the optimal rotation matrix that minimizes the distance between points in the two spaces. While [46] suggests applying this method to the relative representations of latent spaces, [5] proposes doing so directly on the absolute representations, avoiding the retraining required by projecting to the relative manifold. Both approaches show impressive performance when aligning models trained on the same datasets or even different modalities, as long as the models generalize well and do not overfit. However, these methods rely on Procrustes Analysis to find the rotation that is the closest to the optimal one, as outlined in Algorithm 1. Additionally, these approaches assume that the target latent space is available during inference and do not attempt to learn the rotations by using a neural network.

In this thesis, one of the goals is to learn these rotations. Specifically, the aim is to embed RGB images and their corresponding depth maps into latent spaces and align them so that the RGB-aligned features will be decoded into more accurate depth maps. Since the depth map is unavailable during inference, the objective is to learn rotations that approximate those obtained through Procrustes alignment. This would allow estimating a rotation for a new, unseen image and will hopefully improve depth map reconstruction compared to learning from RGB cues alone.

However, learning these rotations is not straightforward, especially in high-dimensional spaces. Previous works deal with high-dimensional latent spaces, far beyond 3D. Learning or constructing rotations in such dimensions is challenging, both mathematically and computationally. [46] shows that larger latent spaces improve performance, which is typical for autoencoders. However, larger latent spaces require rotations of much higher dimensionality, where many loss functions do not perform well or fail entirely. For example, geodesic losses, defined in Equation 3.10, do not work well in dimensions higher than 3, based on experiments conducted in this thesis. Additionally, we cannot regress Euler angles or quaternions, which are specific to rotations in $SO(3)$. This leaves us with learning full matrices, whose size grows exponentially with the dimensionality of the latent space.

Consider a latent space vector of length D. For Procrustes analysis or Gram-Schmidt in n dimensions, we must regress a $D \times D$ matrix, meaning D^2 degrees of freedom for the preimage. For example, while a latent space of length 3 requires 9 DoF, a latent space of length 16 requires 256 DoF, and a space of length 200 requires 40,000 DoF. Besides being computationally expensive, projecting and accurately estimating this projection becomes increasingly difficult.

Furthermore, previous approaches focus on lower-dimensional datasets, such as MNIST (handwritten digits of size $1 \times 8 \times 8$) [50] and MNIST-fashion (fashion items of size $1 \times 28 \times 28$) [51]. These datasets allow for reducing the input images to a single latent space that represents global features, using both convolutional and fully connected layers. However, in the case of NuScenes [7] or KITTI [8], the input resolution is much higher (e.g., $3 \times 1280 \times 384$ for KITTI). Reducing such high-resolution images into a single latent space is not trivial and very computationally expensive. To achieve this, one would need to reduce the spatial size to 1×1, requiring thousands of channels to pass meaningful information through the latent space bottleneck, where any mathematical operation on that vector would be impractical.

Therefore, this thesis explored a novel technique to mitigate those problems. Instead of reducing an image to a single latent space, each pixel in the intermediate tensor, which is the output of the encoder, would be considered as an independent latent space.

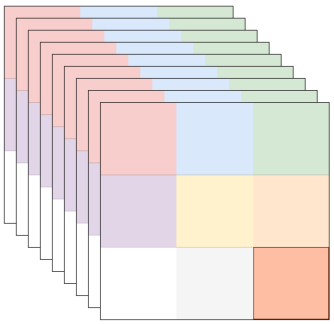


Figure 4.9: A bottleneck layer of a fully convolutional neural network, of shape $C \times H \times W$. Each pixel, that is defined in a different color across the channels of the tensor, is viewed as a separate latent space.

This representation allows us to treat these pixels as if they were independent, even though they are inherently connected due to the nature of CNNs and their receptive fields. In convolutional autoencoders, however, the number of channels typically increases as the spatial size (resolution) decreases, compensating for the loss of information (Figure 4.6). As a result, each pixel at the bottleneck stage usually has

a very high dimensionality. Interestingly, since each entry of the elongated pixel is generated by a different kernel (i.e., a different set of weights), we can regress them in such a way that allows us to break each pixel into smaller components, similar to the "heads" mechanism in attention models [26]. This approach enables us to reduce the dimensionality of the latent space and make rotations more manageable. For instance, if our latent space has a shape of $B \times 192 \times H \times W$, we can decompose each of the $H \cdot W$ pixels into point clouds, where each point cloud consists of 64 points, and each point is represented as a 3D vector. This allows us to estimate a rotation for the point cloud using robust methods and loss functions that are well-defined for SO(3) and are computationally more efficient. However, as discussed later, increasing the number of points makes it more challenging to regress a meaningful point cloud. This difficulty arises because the point clouds are regressed in a latent space that is inherently flexible and lacks structure—unless we impose one. Additionally, the dimensionality of each "head" or point could vary, being 2D, 3D, or even higher, which creates a balancing act between dimensionality and the number of points. Moreover, unlike traditional point clouds, such as those generated by LiDAR (Figure 2.3), where each point is independent, the points in this context are correlated, and their order is meaningful.

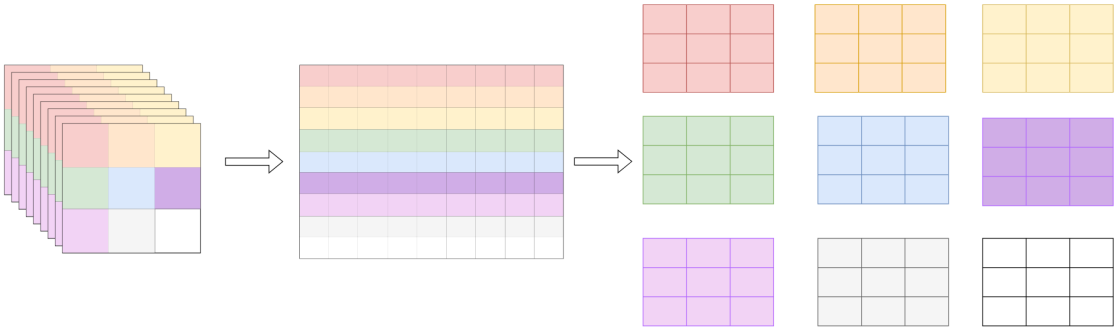


Figure 4.10: Break a bottleneck layers' to separate 3D point clouds. Take each pixel across the channels, and divide it into $n-$dim points.

Given a bottleneck X of shape $B \times C \times H \times W$, perform the following in PyTorch:

$$X_{\text{pc}} = X.permute(0,2,3,1).reshape(B, H \cdot W, C)$$

In this example, each point is represented by a pixel, across the channels.
To break down the bottleneck tensor into point clouds of lower dimensionality, do:

$$X_{\text{pc}} = X.permute(0,2,3,1).reshape(B, H \cdot W, C//d_{\text{point}}, d_{\text{point}})$$

On the other hand, the bottleneck tensor can be broken down into more globally oriented latent spaces by considering each channel individually. However, this approach

presents challenges when attempting to convert the channels into point clouds, as different entries within each channel correspond to different positions in the global structure of the feature map. This makes it unclear how semantically meaningful the resulting points would be. One possible approach is to process the channels with a global fully connected layer, but this contradicts the inherent structure of the feature map, where each entry represents a local neighborhood of the original image. Therefore, it is more feasible to treat each entire channel across the spatial dimensions as a single point, allowing the model to learn a global geometry.

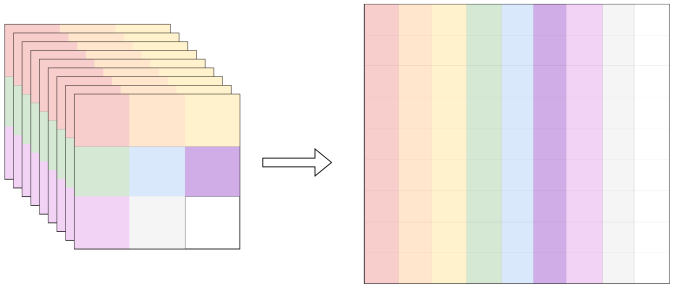


Figure 4.11: Flatten the bottleneck channels across the spatial dimensions. Have a more global context, but would negate the power of convolutions if processed any further.

Given a bottleneck X of shape $B \times C \times H \times W$, perform the following in PyTorch:

$$X_{\text{pc}} = X.flatten(-2)$$

In this example, each point is represented by a channel, across the spatial dimensions.

The next question is what should be compared between these points. As described above, the point clouds lack any inherent structure and will usually take a form that best suits the downstream task. [5] suggests that models trained on data with similar distributions tend to produce semantically and geometrically aligned latent spaces, where the alignment between these spaces is meaningful. However, when dealing with different modalities, the orientation of the latent spaces in space can vary significantly.

One straightforward solution to address this issue is to project each point onto the unit sphere by normalizing it:

$$X'_{\text{pc}} \xrightarrow{\text{normalized}} \frac{X_{\text{pc}}}{\|X_{\text{pc}}\|_2}$$

This projection places the points on a well-defined geometric manifold. However, the points still retain the flexibility to take on infinitely many directions on the sphere, meaning the structure can still vary greatly.

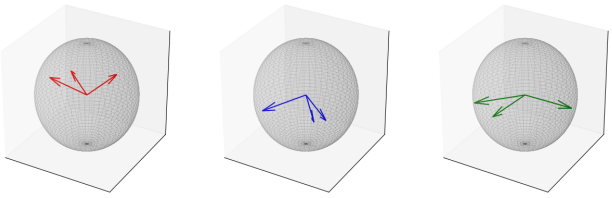


Figure 4.12: Each pixel is broken into a 3D point cloud, where we have a solid framework to manipulate the features with.

5 Loss Functions for depth estimation in monocular images

Loss functions represent the goal of a neural network's training process. The sole purpose of such a network is to minimize the loss value obtained by these functions on the training set. While we, as developers, aim to minimize the loss over unseen data represented by the validation and test sets, this is only a byproduct of the process of minimizing the training loss. The challenge lies in minimizing the loss in such a way that the model doesn't overfit to the training data—by merely memorizing it or learning overly specific characteristics—but instead learns general features that are useful for both training and inference.

5.1 Depth related loss functions

In the context of depth prediction, we deal with a regression problem where the goal is to predict a continuous value (the depth) given an input image. Typically, these values are represented as real numbers in the range of $[0, 1]$. Although, some works approach this problem as a classification task, where depth values are separated into integer bins, ranging from 0 to some maximum value [52]. In the regression scenario, the literature commonly proposes straightforward functions, such as the Mean Squared Error (MSE) or the Mean Absolute Error (MAE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{5.1}$$

where $\hat{y}_i$ is the predicted value, y_i is the actual value, and N is the number of residuals.

However, these functions are not always the best choice, as they might not be ideal for the task at hand. For example, [53] suggests using Huber or BerHu (bounded reverse Huber) loss functions, which are robust combinations of MAE and MSE. These functions remain continuous but are more robust towards outliers.

The Huber loss and BerHu loss both address limitations of the MSE in regression problems, particularly when dealing with outliers. The Huber loss combines the

properties of MSE and MAE, behaving like MSE for small errors within a threshold, and like MAE for larger errors beyond the threshold. Specifically, for a given error $r = y - \hat{y}$, the Huber loss L_δ is defined as:

$$L_\delta(r) = \begin{cases} \frac{1}{2}r^2 & \text{for } |r| \leq \delta \\ \delta(|r| - \frac{1}{2}\delta) & \text{for } |r| > \delta \end{cases} \tag{5.2}$$

This makes it less sensitive to outliers than MSE, while maintaining smooth differentiability, which is beneficial for optimization. On the other hand, the BerHu loss behaves like MAE for small errors but switches to a quadratic form (similar to MSE) for larger errors. The BerHu loss L_δ is given by:

$$L_\delta(r) = \begin{cases} |r| & \text{for } |r| \leq \delta \\ \frac{r^2+\delta^2}{2\delta} & \text{for } |r| > \delta \end{cases} \tag{5.3}$$

This switching mechanism allows BerHu to balance sensitivity to small errors with robustness to large ones. Both Huber and BerHu losses provide better gradient behavior and outlier robustness compared to MSE, making them more adaptable and effective for data with heavy-tailed distributions or significant outliers.

This is particularly relevant, as [53] argues that these loss functions emphasize regions in the image closer to the camera, where the most important information resides, while being less likely to focus on outliers.

In addition to traditional loss functions, structural loss functions like Sobel Kernel loss and Structural Similarity Index Measure (SSIM) are often employed to enhance the structural accuracy of depth maps. These losses focus on preserving edges and finer details in the depth predictions by comparing the structural integrity between the predicted and ground truth depth maps.

The Sobel Kernel loss emphasizes edge information by applying a Sobel filter to extract gradients. This is achieved by convolving the predicted and ground truth depth maps with Sobel operators in both the horizontal and vertical directions. Let G_x and G_y be the horizontal and vertical Sobel operators, respectively. The gradient magnitudes for the predicted depth map $\hat{D}$ and ground truth depth map D can be computed as:

$$\nabla\hat{D} = \sqrt{(G_x * \hat{D})^2 + (G_y * \hat{D})^2} \tag{5.4}$$

$$\nabla D = \sqrt{(G_x * D)^2 + (G_y * D)^2} \tag{5.5}$$

Where the "$*$" operation is a convolution with the sobel kernel of some size k. The Sobel Kernel loss L_{Sobel} is then defined as the Mean Squared Error between the gradient magnitudes of the predicted and ground truth depth maps:

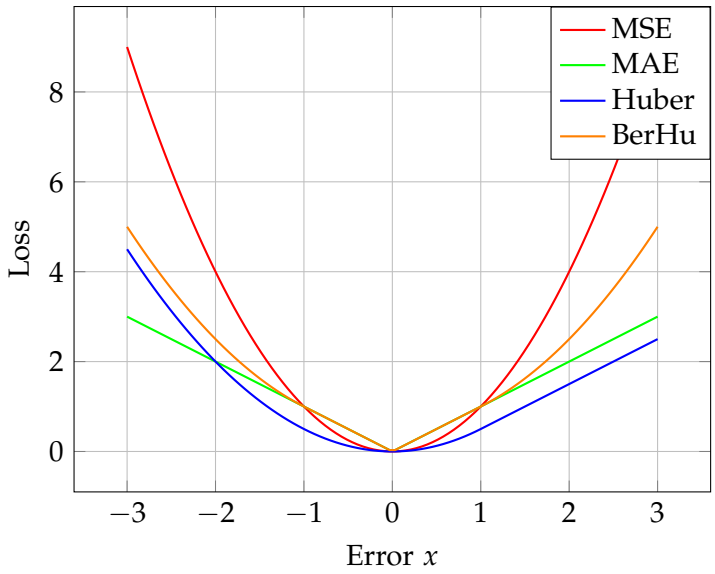


Figure 5.1: A comparison of loss-functions that are commonly used in depth estimation research. While BerHu and Huber have shown better overall performance, they require fine-tuning of a hyperparameter.

$$L_{\text{Sobel}} = \frac{1}{N}\sum_{i=1}^{N}(\nabla\hat{D}_i - \nabla D_i)^2 \tag{5.6}$$

Figure 5.2: The Sobel gradients map, allows us to compare structural features between two images

On the other hand, SSIM evaluates the similarity between the predicted and ground truth depth maps in terms of luminance, contrast, and structure. Given two patches x and y (from the predicted and ground truth depth maps, respectively), the SSIM index is computed as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{5.7}$$

where μ_x and μ_y are the mean values, σ_x^2 and σ_y^2 are the variances, and σ_{xy} is the covariance between patches x and y. The constants C_1 and C_2 stabilize the division.

The SSIM loss L_{SSIM} is then defined as:

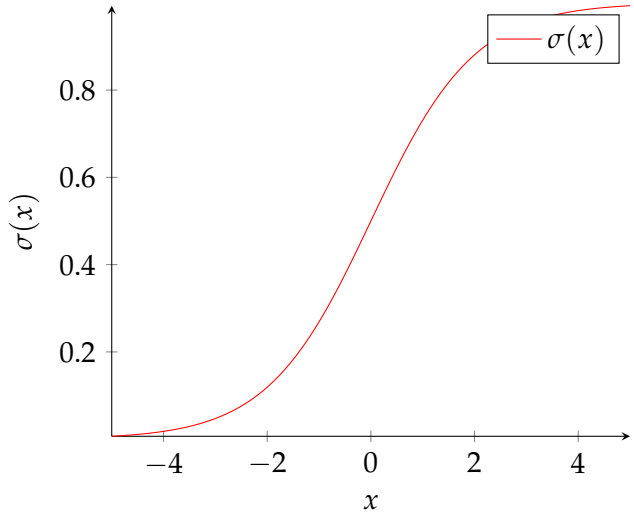
$$L_{\text{SSIM}} = 1 - \text{SSIM}(\hat{D}, D) \tag{5.8}$$

By incorporating these structural loss functions into the training process, models can generate depth maps that are not only numerically accurate but also visually coherent and structurally consistent with the scene.

As previously discussed, a depth map is typically a single-channel tensor of some resolution (i.e., with a shape of $1 \times H \times W$) containing values in the range of $[0, 1]$. Neural networks often regress values in $\mathbb{R}$, so it is common practice to apply a sigmoid function to map the regressed "logits" into the range $[0, 1]$. The sigmoid function is applied element-wise to each entry of the tensor and is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.9}$$

Figure 5.3: Sigmoid function



Given a map of logits D, the depth map $\hat{D}$ can be obtained by applying the sigmoid function:

$$\hat{D} = \sigma(D) \tag{5.10}$$

However, the sigmoid function is not commonly used as a non-linear activation function in deep learning due to several drawbacks. Its maximum gradient value is 0.25 (which occurs at $x = 0$), and despite being smooth, the sigmoid function is not zero-centered (it produces only positive values) and is effective only in a narrow range, roughly $[-2, 2]$. Beyond this range, the function saturates, resulting in very small gradients. This can cause a problem known as the "vanishing gradient" issue, where gradients become too small for deeper layers of a network to learn effectively, leading to slower training in deeper layers compared to shallower ones.

In the context of depth maps, many values are concentrated either in regions close to the camera, where significant information is present, or at the far edges with fine details. The use of sigmoid might force the model to learn logits that lead to very small gradients through the sigmoid layer. This could cause the network to focus disproportionately on certain values, that reside closer to the mean of all ranges.

Fortunately, since the sigmoid function is continuous and bijective, it has an inverse function:

$$\frac{\partial \sigma(x)}{\partial x} = \ln\left(\frac{y}{1-y}\right) \tag{5.11}$$

This allows us to project the ground truth depth values onto the logits' manifold, ensuring that each residual retains a significant gradient magnitude. However, it doesn't seem to make much of a difference.

5.2 Domain adaptation: CORAL

When attempting to align two modalities, it is crucial to minimize the gap between the two domains. One interesting approach is proposed by [54], which aligns the second-order statistics of the source and target distributions. The authors describe it as "frustratingly easy" to implement and demonstrate how straightforward it is to incorporate into existing models, between any layers. To achieve this, they use the same model (i.e., shared weights) on both domains and apply the loss on a tensor that is used directly for some other downstream task, to avoid mode collapse. Without this precaution, the model could trivially map any input to zero. The loss function is defined as:

$$\ell_{\text{CORAL}} = \frac{1}{4d^2}\|C_S - C_T\|_F^2 \tag{5.12}$$

where $\|\cdot\|_F^2$ denotes the squared matrix Frobenius norm. The covariance matrices of the source and target data are given by:

$$C_S = \frac{1}{n_S - 1}\left(D_S^\top D_S - \frac{1}{n_S}(1^\top D_S)^\top (1^\top D_S)\right)$$

$$C_T = \frac{1}{n_T - 1}\left(D_T^\top D_T - \frac{1}{n_T}(1^\top D_T)^\top (1^\top D_T)\right)$$

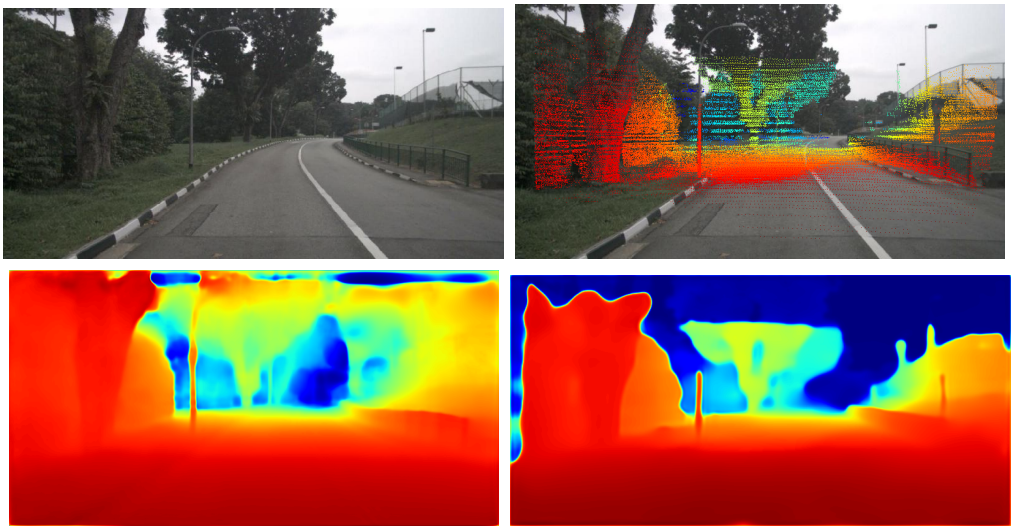
1. D_S and D_T are the feature matrices for the source and target domains, respectively, where rows are samples and columns are features.

2. n_S and n_T are the number of samples in the source and target datasets.

3. The term $D_S^\top D_S$ (or $D_T^\top D_T$) computes the sum of the pairwise outer products of the feature vectors, which is related to the covariance of the dataset.

4. The subtraction $\frac{1}{n_S}(1^\top D_S)^\top (1^\top D_S)$ (or the analogous term for the target) removes the contribution of the mean from the covariance, making it a *centered covariance matrix*, hence focusing solely on the geometrical structure between the source and the target.

5.3 Sky-Loss: Reducing Uncertainty



RGB | LiDAR point cloud
Without infinity loss | With infinity loss

Figure 5.4: Comparison of depth prediction visual results given the **infinity loss**. ***Source:*** *Scene **00453** from the NuScenes dataset* [7].

In the context of autonomous driving datasets, such as NuScenes [7] the depth maps provided as ground truth are often partial and do not represent the entire image comprehensively (Section 2.9). For example, in NuScenes, these depth maps are obtained by projecting a LiDAR point cloud onto a 2D plane. Since LiDAR captures data by projecting light, infinitely far objects, such as the sky, are excluded from representation. Consequently, when directly estimating a depth map for an image with such partial ground truth, significant uncertainty can arise in the sky regions, where no supervision is available.

To address this issue, this thesis proposes a new loss function, named an infinitz-Loss, which leverages available semantic information to identify sky regions in the image and creates a quasi-mask. This mask is then used to penalize these regions as much as possible. The loss function is defined as:

$$L_{\text{infinity}}(y, \hat{y}, t) = (1 - y) \cdot \hat{y} + y \cdot (t - \hat{y}) \tag{5.13}$$

Here, y represents the binary mask, where $y = 0$ for pixels segmented as sky and $y = 1$ otherwise. t represents a threshold, chosen to be the inverse of the maximum

depth value (here, 100 meters). The function takes as input values for pixels segmented as sky and those with values below the threshold. This approach pushes true sky values towards zero (representing infinity, given an inverse-depth scheme), while penalizing falsely assigned low values to increase.

Thus, the model learns to minimize the uncertainty associated with infinitely distant objects, such as the sky.

6 Method

This chapter provides the technical details of the deep learning approach, covering aspects such as loss functions, hardware setup, model architecture, and selected hyper-parameters.

6.1 Loss functions and dynamic weighting

The primary goal of this work is depth estimation. While previous research has explored combining BerHu and edge losses, adding extra regularization terms increases the complexity of the optimization process. This can make training more challenging, as it requires tuning additional hyperparameters. Since the main objective of this work is to explore different ways of using the given ground truth depth information to guide the RGB branch, it was found sufficient to use the MSE loss function (Equation 5.1) for simplicity and to ensure fair comparison between experiments. Future work could improve upon my results with better loss functions and architectures.

The model uses a sigmoid function on the regressed logits to obtain estimated depth values in the range of (0, 1), where the ground truth are in the range of (0.01, 1), due to the maximum distance of $100m$ and the inverse depth scheme.

In addition, balancing the combined loss terms is also crucial, especially when they conflict with each other, like two groups pulling a rope in different directions. Therefore, many works ([55]) use a weighted sum for all loss terms, typically giving more weight to the main loss term. However, finding the right weighting is challenging, as it can easily lead to suboptimal performance on the main task while not significantly improving the secondary ones. On top of that, different loss functions produce values of different magnitudes. For example, the difference between MAE and MSE losses is substantial: given two tensors, T_1 and T_2, if the MAE loss is around $1e^{-4}$, the MSE loss between the same tensors might be around $1e^{-8}$. Clearly, if both losses were used together, the MAE would dominate the gradient during backpropagation, leaving the MSE with little to no effect.

To address this, I've developed a dynamic approach for weighting the regularization terms. While the main loss (depth loss) is weighted fully (magnitude of 1), each secondary loss is weighted relative to the main one. Specifically, the secondary loss is

adjusted to match the magnitude of the main loss by multiplying it by the inverse of their ratio. Given a main loss $\mathcal{L}_M$ and a secondary loss $\mathcal{L}_S$, the first step is:

$$\mathcal{L}'_S = \mathcal{L}_S \cdot \frac{\mathcal{L}_M}{\mathcal{L}_S}$$

This adjustment ensures all losses are comparable, regardless of whether the secondary losses are larger or smaller than the main loss.

Now that the secondary loss has the same magnitude as the main loss, we can assign it a more precise weight independent of its original magnitude. Each secondary loss term is then weighted by a dynamic factor that is adjusted by a scheduler, similar to how learning rates are scheduled (e.g., with a one-cycle-lr scheduler [56]).

The idea behind this is that at the beginning of training, when features are raw and more flexible, the secondary losses should have a significant influence to guide the main loss in a direction that satisfies all objectives. However, as training progresses, the model should rely more on the main loss, as it is the most important. A high secondary loss at this stage could lead to overshooting the minima and suboptimal results.

Therefore, given the following function and the two losses $\mathcal{L}_M$ and $\mathcal{L}_S$, the final loss is calculated as follows:

```
def adjust_loss(x, y, nominator=1):
    return (x * nominator) / (y + 1e-5)
```

$$\mathcal{L}_{\text{final}} = \mathcal{L}_M + \text{adjust\_loss}(\mathcal{L}_M, \mathcal{L}_S, \text{s\_scheduler.curr\_lr}) \cdot \mathcal{L}_S \tag{6.1}$$

6.2 Architecture and details

All experiments were conducted on a V100 GPU with 16 GB of V-RAM. This setup facilitated a careful selection of architecture, enabling swift execution while ensuring reasonable performance. However, state-of-the-art architectures, such as visual transformers (Section 2.5), proved too computationally intensive to allow for effective experimentation with architecture and hyperparameters.

The NuScenes dataset consists of 22,376 images, each paired with 23-beam LiDAR point clouds projected onto 2D depth maps. These depth maps are sparse, lacking critical information, which introduces significant uncertainty. Additionally, the shape and volume of the point clouds vary across samples, and the alignment between points and pixels is accurate only up to a certain error margin. These factors add to the already challenging task of depth estimation. The dataset is split into training, validation, and test sets with 17,902, 2,237, and 2,237 images, respectively (an 80/10/10 split). The

original NuScenes image resolution is 1200×1600. However, both RGB images and their corresponding 2D depth maps are downsampled to 384×786 to allow for a downsampling factor of up to 2^8 within an autoencoder. This downsampling also reduces the computational load, enabling faster training and experimentation.

Each experiment was run for 40 epochs with a batch size of 8, totaling approximately 90,000 training steps.

Learning rate scheduling is a key factor in deep learning training. While [56] introduced a method for achieving superior convergence (reaching a better minimum) in less time, it cannot be guaranteed that using their method would yield a significant difference compared to a standard learning rate decay scheme in this case. Therefore, the built-in PyTorch implementation is used, primarily for learning rate warm up. The training is divided into two phases: in the first phase, the learning rate warms up rapidly from 6.5e-5 to 1.5e-4 over 6 epochs, while the remainder of the training uses a more gradual decay down to a minimum value of 9e-6 for fine-tuning. The warm-up phase is crucial, as during this early stage, the weights are primarily influenced by the main depth loss function, establishing a strong feature extraction foundation. After this phase, secondary losses, such as infinity loss or latent space alignment loss, are given more weight. These secondary losses start with small coefficients (e.g., infinity loss starts at 5e-2 and is adjusted relative to the main loss for controlled magnitude) but gradually increase after the warmup phase to guide the weights in the desired direction.

Additionally, the RAdam optimizer is used, chosen based on the extensive library found in [57]. Among the optimizers tested, RAdam demonstrated the best balance between performance and speed. Variants of the Adam optimizer [58] often include weight decay [59]; here, a weight decay coefficient of 2e-1 was used. Also, the PyTorch implementation of the RAdam is super efficient, and allows both fast training and converging to a good minima.

In summary:

- OneCycleLR scheduler (PyTorch implementation: [60]), with initial LR: 6.5e-5, mid LR: 1.5e-4, end LR: 9e-6, warmup epochs: 6.
- Infinity loss with OneCycleLR scheduler, with initial LR: 9e-2, mid LR: 7e-2, end LR: 5e-2, warmup epochs: 6.
- Secondary loss with OneCycleLR scheduler, with initial LR: 5e-2, mid LR: 1.4e-1, end LR: 1.8e-1, warmup epochs: 6.
- RAdam[61] with a weight decay of 2e-1.

Since the primary goal of this project is to compare different approaches to latent space alignment, it was opted for a simple yet efficient backbone as the model, that will be used throughout all the experiments.

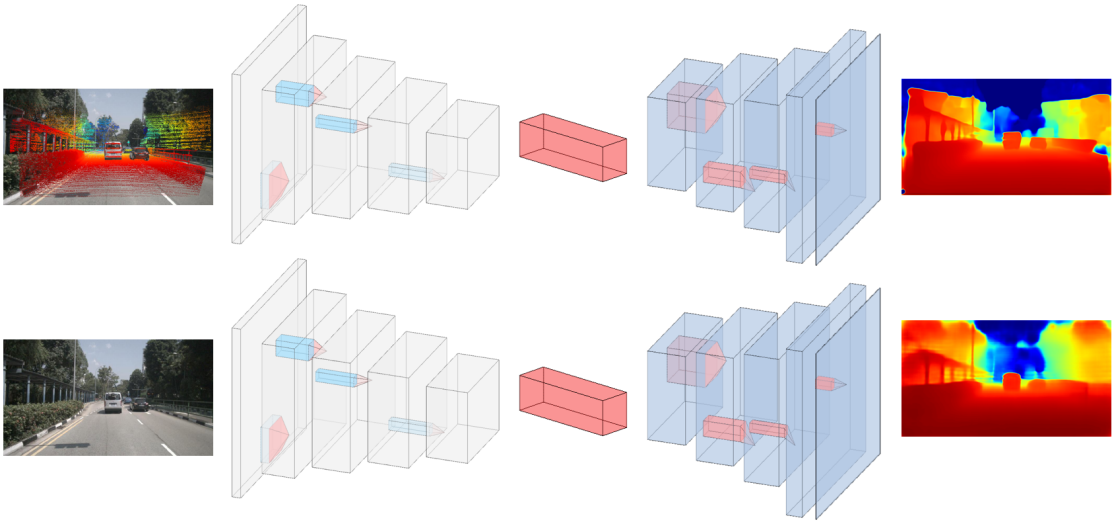


Figure 6.1: Model architecture that is unchanged for all the experiments. A simple autoencoder, with CBAM [62] layer at the end of each scale, besides at the highest resolutions, to avoid negligible performance at high cost of runtime.

The architecture used in this work is a simple convolutions-based autoencoder. Unlike U-Net[63], it doesn't utilize skip connections between the encoder and decoder at different scales. This choice is intentional because the goal isn't to achieve state-of-the-art results, but to compare different methods. Skip connections would allow the network to bypass the need for the encoder to extract meaningful features in the latent space by passing useful information directly to the decoder, which isn't beneficial for this work's objective.

The resolution is reduced from $(384, 768)$ to $(12, 24)$, a reduction by a factor of 2^5. All inputs to the network, including both RGB and depth values, are scaled to the range of $(0, 1)$. This is because the depth values also reside in this range. Therefore, no dataset-specific normalization is applied.

The core computational units are Dense-Blocks [31], chosen for their ability to capture dense and meaningful features. For the depth branch, a Convolutional Block Attention Module (CBAM - Section 2.6) [62] have been integrated after each computational block to aggregate more global information while keeping the inference time relatively low. The RGB-D branch does not require such modifications, as its performance is already strong. All experiments are conducted on the latent spaces, so the encoder and decoder

architectures remain unchanged.

As shown in Figure 6.1, during training, the model uses two Autoencoders: one for each branch. The interaction between the two branches occurs at the latent space level during the "projection" stage, where various manipulations can be performed. Although both branches could share weights, separating them allows each to focus on extracting features relevant to their specific task. This results in more parameters during training, but it doesn't affect inference since the depth branch isn't needed.

The RGB branch is deeper and contains more layers: At each scale of both encoder and decoder, besides the first one, there are two blocks of DenseBlocks [64] and a Spatial-Attention layer in the end [62]. The RGB-D however, requires much less computational power for achieving amazing reconstruction results, and therefore uses a single DenseBlock at each scale, alongside the very efficient CBAM.

Given the hardware constraints, it's important to minimize calculations at higher resolutions. To address this, the decoder begins with a modern approach—using a convolution with a kernel size of 7, stride of 4, and padding of 3, reducing the spatial dimensions by a factor of 4. Similarly, the final upsampling layer uses a transpose convolution with a kernel size and stride of 4. This ensures that the most computationally expensive operations happen at lower resolutions, making the network more efficient. Although max-pooling could be more efficient for downsampling, convolutions were chosen to avoid issues like dead gradients and underperformance.

While the RGB branch is tasked with **depth estimation**, the depth branch handles **depth reconstruction**. However, this often leads to overfitting, as it can simply memorize depth values rather than learning meaningful features—something the RGB branch struggles with. That can be observed by the fact that there is no generalization gap between the training and validation losses of this branch, i.e. the training and validation losses converge to the same values. To counter this, regularization techniques like *Dropout*[65] and *weight-decay*[59] (An aggressive factor of 0.2 is used) are essential to force the depth branch to generalize. Another option is to add an extra downsampling layer to further compress the latent space, making it harder for the network to simply pass through the depth values.

For activation, GELU [66] is used because it has been shown to outperform ReLU in many cases, despite being more computationally intensive. GroupNorm is preferred for normalization, as it's more robust than BatchNorm when dealing with smaller batch sizes (fewer than 16), which can introduce too much noise into the statistics ([67]).

7 Experiments

This section presents the various experiments conducted in this work, all focused on different methods for aligning **only** the latent spaces derived from the RGB and RGB-D branches. The objective is to ensure that the RGB features are decoded into a more accurate depth map. For each experiment, the motivation and key details are outlined.

The inputs used are either the original NuScenes RGB images, the ground-truth depth maps, or RGB-D images. The RGB-D images are created by repeating the single-channel depth map three times and adding it to each channel of the corresponding RGB image. No preprocessing is applied, meaning that there are no learnable weights to adjust the inputs before combining them. Additionally, unless specifically stated otherwise, in all experiments, the RGB branch is decoded and compared to the ground truth. It appears that any attempt to simply align the latent spaces and use a decoder trained only on the depth latent space distribution results in suboptimal estimation.

Unless stated otherwise, weight decay coefficient is set to 0.2, to counter the overfitting in the RGB branch, and produce more general features, that **might** be easier to align. In addition, secondary losses are assigned with **dynamic-coefficients** λ, as described in Section 6.1. The adjustment to the main loss is marked by AL (Adjust Loss):

$$\mathcal{L}'_y = \mathrm{AL}(\mathcal{L}_x, \mathcal{L}_y) = \frac{\mathcal{L}_x}{\mathcal{L}_y} \cdot \mathcal{L}_y$$

Where $\mathcal{L}_x$ is the leading loss, and is always the RGB depth error, unless stated otherwise.

Dictionary for $\mathcal{L}_x$, where x is a placeholder for:

- rgb: RGB depth branch
- rgbd: RGB-D depth branch
- r-inf: Infinity loss for the RGB branch
- d-inf: Infinity loss for the RGB-D branch

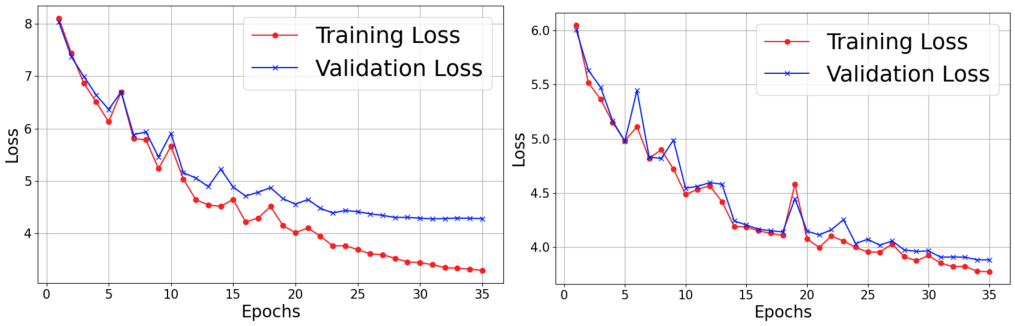
For all experiments, unless stated otherwise, the base loss function is:

$$\mathcal{L}_{\text{main}} = \mathcal{L}_{\text{rgb}} + \lambda_{\text{r-inf}} \cdot \mathcal{L}'_{\text{inf}} + \mathcal{L}_{\text{rgbd}} + \lambda_{\text{d-inf}} \cdot \mathcal{L}'_{\text{d-inf}} \tag{7.1}$$

7.1 Reduced LiDAR as guidance

As mentioned earlier, during training, one faces two distinct tasks: the RGB branch handles *depth prediction*, while the RGB-D branch focuses on *depth reconstruction*. The RGB features in the RGB-D branch aim to create more realistic depth maps that account for the structures in the image and help minimize the domain gap between RGB and Depth.

However, one could notice a significant issue with the RGB-D branch: despite its ability to reconstruct the depth map with high accuracy (even when only a sparse point cloud is provided), it maintains a very small generalization gap between the training and validation losses, as seen in Figure 8.1.



RGB branch RGB-D Branch

Figure 7.1: Differences in training between the RGB and RGB-D branches for the MSE experiment. While the RGB branch develops a *generalization gap*, the RGB-D branch overfits excessively. Interestingly, even when the RGB branch overfits better to the training data, the RGB-D branch still manages to maintain a very low generalization gap.

Therefore, it is crucial that before attempting any alignment in the latent space, one must ensure that the RGB-D branch generalizes and produces more "general" features, rather than just relying on depth values passed through the network. This dependency would make alignment very challenging. To that end, there are a few possible solutions:

- Reduce the capacity of the RGB-D encoder.
- Combine the RGB and RGB-D encoder.
- Dilute the LiDAR point cloud (e.g. 10% of the original points).
- Add Gaussian noise to the depth values.

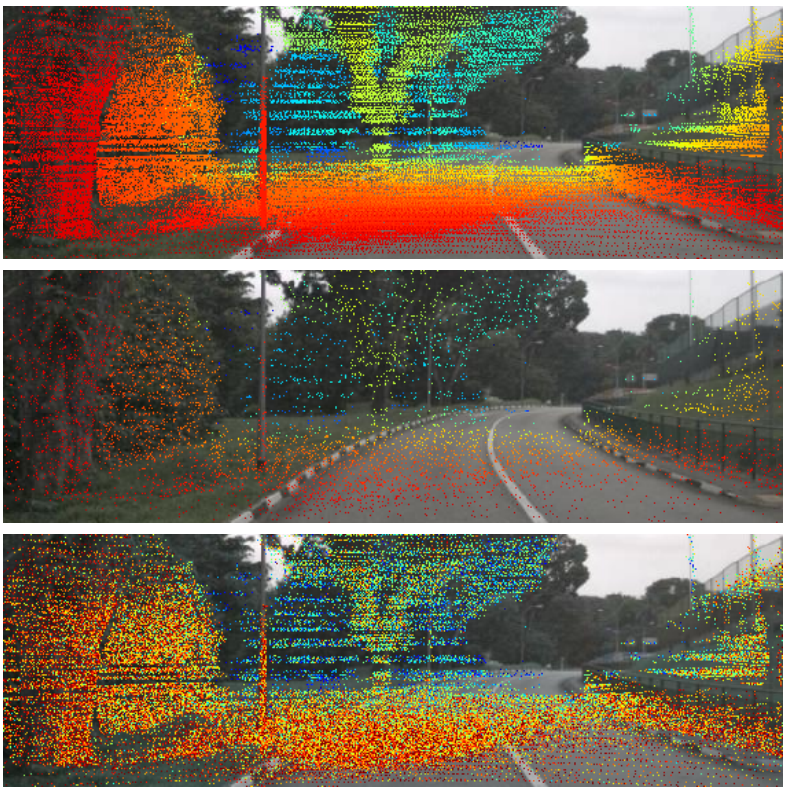


Figure 7.2: Different possible LiDAR depth values as inputs. TOP: Full LiDAR point cloud. MID: Diluted point cloud of $\frac{1}{10}$ of the original size. DOWN: With added Gaussian noise with $std = 0.2$

.

7.2 Baseline - Only RGB

- **Input**: RGB image.
- **Motivation**: To compare performance against a model that incorporates the infinity loss.
- **Details**: A single loss function is used.
- **Latent space manipulation**: None.
- **Expectation**: The results should improve, as the higher uncertainty makes the problem more relaxed.
- **Loss function**: $\mathcal{L} = \mathcal{L}_{\text{rgb}}$
- **Observations**: The model produced better numerical results, but the visual quality of the depth maps was poor. The model struggled significantly in areas with high uncertainty.

7.3 Only RGB + Infinity Loss

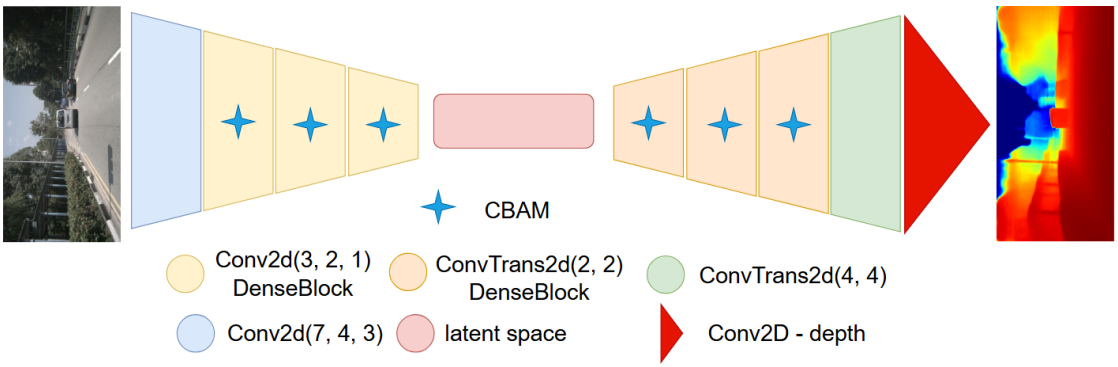


Figure 7.3: Inference: predict a depth map from a single RGB image

- **Input**: Only RGB images.
- **Motivation**: This is the most basic experiment, serving as a baseline for the final goal: estimating a depth map from a single RGB image.
- **Details**: Applied with a high weight decay (0.3) to counteract the aggressive overfitting of the RGB branch.

- **Latent space manipulation**: None.
- **Expectation**: Act as a baseline for comparison and help determine which methods truly leverage latent space alignment.
- **Loss function**: $\mathcal{L} = \mathcal{L}_{\text{rgb}} + \lambda_{\text{r-inf}} \cdot \mathcal{L}'_{\text{r-inf}}$
- **Observations**: Set a baseline of an RMSE loss of 4.206m, which is, given the small model and lack of skip connections - a very interesting score.

7.4 Direct alignment: MSE between the latent spaces

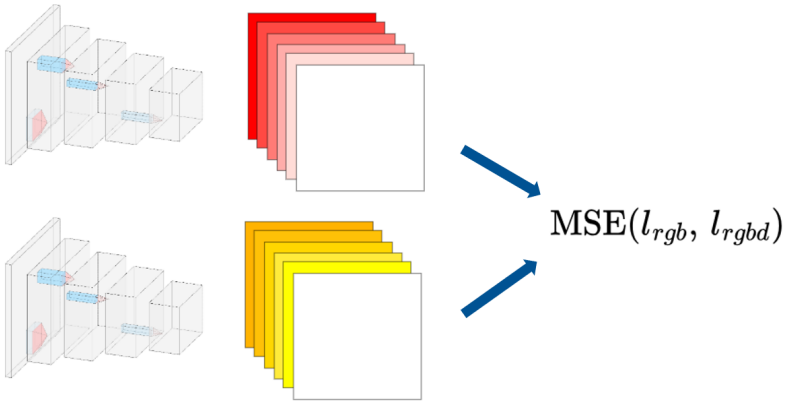


Figure 7.4: The most obvious trick in the book. Compare the two latent spaces with the MSE, with no regard to structure of representation.

- **Input**: Two branches, RGB and RGB-D.
- **Motivation**: The basic idea is to minimize the distance between the entries of the latent spaces, which are values in the n-dimensional Euclidean space, without considering differences in distribution or geometric structure.
- **Details**: The latent spaces are created by a linear convolution layer, without any activation or normalization layers. This section is divided into two experiments:
  1. End-to-End: Train both encoders and decoder in one time, while trying to minimize the gap between the latent spaces with the MSE loss (Equation 5.1).
  2. Fresh Decoder: After the End-to-end stage and aligning the latent spaces, reinitialize the decoder, and train it on the aligned latent spaces as input. Could be done with freezing the decoders and without.

- **Loss function**:

$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_{\text{mse}} \cdot \text{MSE}(l_{\text{rgb}}, l_{\text{rgbd}}))$$

- **Observations**:
  1. End-to-end: In both detached and undetached scenarios (where the depth latent space is detached to prevent it from learning to align with the RGB and behaves as a fixed ground truth), the model performed slightly better than the baseline only-RGB experiment.
  2. Surprisingly enough, the new decoder managed easily enough to distinguish between the latent spaces and reconstruct with the exact same results.

7.5 Second order alignment: CORAL

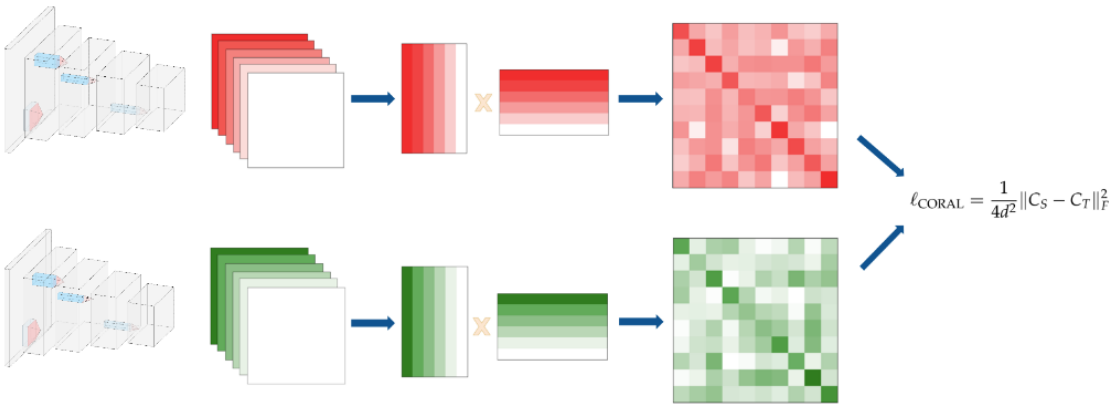


Figure 7.5: Instead of comparing the regressed logits directly, compare the statistics of a down stream hidden layer, using **shared** weights (the same encoder for both modalities) Described in [54] as *frustratingly easy*.

- **Input**: RGB and RGB-D images.
- **Motivation**: Direct alignment between two different distributions is a very challenging task. The CORAL (Section 5.2) loss function aligns two distributions by matching their mean and covariance, ensuring that the latent spaces have similar statistical properties. This is useful in aligning latent spaces from different modalities (e.g., RGB and depth) because it not only aligns individual elements but also their relationships and variance patterns. Compared to a direct loss function like MSE, which only minimizes element-wise differences, CORAL can be more

effective for latent space alignment as it preserves the overall structure of the data and accounts for correlations between features, leading to better alignment across distributions.

- **Details**: Different encoders for each branch, but a shared decoder.Both latent spaces are decoded. When taking the loss, the RGB-D tensor is **not** detached, as this allows closer alignment without collapsing into the RGB manifold.
- **Latent space manipulation**: Minimize the distance between the covariance matrices of the channels of each branch. This could also be attempted between the pixels instead.
- **Expectation**: Unifying both branches in structure should provide some guidance for alignment.
- **Loss function**:
$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_{\text{coral}} \cdot \text{coral}(l_{\text{rgb}}, l_{\text{rgbd}}))$$
- **Observations**: This simple structural alignment has achieved promising results. It highlights that a loss function which considers a more global context, rather than just pixel-wise differences, better suits latent space alignment. The resulting depth maps between the branches are visually almost distinguishable.

7.6 Spatial Attention: weighted maps

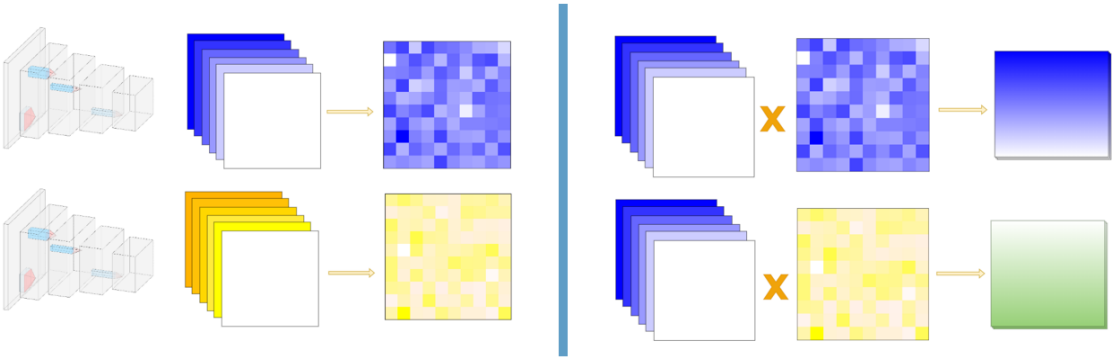


Figure 7.6: Create two spatial attention maps from the RGB and RGB-D branches, and apply them to the RGB latent space. Minimize the distance between the two weights maps by some loss, e.g. MSE, CORAL, etc.

- **input**: Two branches, RGB and Depth.

- **Motivation**: The idea behind this approach, is to have an RGB latent space as a baseline, and then only manipulate it using learned weight maps, created as described in CBAM (Section 2.6).This will force the model to learn more general features, as the number of Degrees of Freedom (DoF) is much smaller.
- **Details**: The following pipeline is proposed:
  1. Regress an RGB latent space.
  2. Create two independent spatial maps [62] from both RGB and RGBD branches.
  3. Create two new latent spaces by applying the regressed weights to the RGB latent space, and decode.
  4. Use the MSE loss to minimize the distance between the two weights maps.
- **Expectation**: Both branches now rely on the RGB latent space, modified by some secondary values. Restricting of the guidance features to the range of (0, 1) with the sigmoid function and to a single feature map of shape 12×24 might relax the aligning process.
- **Observations**: Slightly improved performance over the baseline. Also, the latent spaces managed to align to some degree, which could be observed later on in **??**.

7.7 Relative Representation

- **Input**: Two branches, RGB and RGB-D.
- **Motivation**: As shown in Section 4.2, it is a powerful technique to have an offline pool of vectors as reference, from which one can create a joined manifold to latent spaces, regardless of their original modality and distribution. This creates a memory that is learned, besides the weights, that is influenced by the RGB branch, but more importantly - the depth cues that are now integrated and saved, to some extent, within the network, and could be a reference to any new unseen scene.
- **Details**: The candidate latent spaces are the pixels of the tensor-shaped latent space, across the channels. As described in [46], the distance scheme between a candidate latent space c and an anchor a is chosen to be the cosine angle between them, achieved by:

$$d(\mathbf{c}, \mathbf{a}) = \frac{\mathbf{c} \cdot \mathbf{a}}{\|\mathbf{c}\| \|\mathbf{a}\|} = \cos\theta.$$

This distance scheme is invariant to global linear transformation, and drops the dependency on the scale of the *absolute* representations. If the number of anchors is equal to the dimension of the latent space, then no further processing is required. Otherwise, an MLP is applied to bring them back to the shape that the decoder expects. This section divides into a few different experiments:

1. Random initialized anchors:

   Create a pool of size (latent dim $\times$ number of reference vectors) as learnable parameters that initialized randomly, to be tweaked by both RGB and RGB-D values during training, to create the desired reference space. Now, reshape the latent space from $B \times C \times H \times W$ to $B \times (H \cdot W) \times C$. Secondly, normalize it and the reference vectors to have a length of 1. Then, use matrix multiplication between the permuted tensor and the reference pool of shape $C \times N$, where N is the number of reference vectors, and normalize across the first dimension. This will result with the cosine similarity between the tensors, and would treat the reference pool as vectors on the unit-sphere. However, there is no aligning process by using some kind of loss function, like MSE.

   - How to create the anchors space? Creating a separate space for each pixel proved to perform poorly, while creating as single global pool performed much better.
   - No manipulation: Given the pool of anchors, represent both RGB and RGB-D candidates with the anchors, and decode them, with no further manipulation. Number of anchors - 192, which is the dimension of the latent space.
   - With manipulation: The same, but now with MSE loss between the representations. The RGB latent spaces go through and additional MLP before, as in the new representations, reducing the gap between them might be easier. Number of anchors - 192, which is the dimension of the latent space.

2. Pretrained latent vectors as anchors:

   In the training set there are 17092 images, divided with $70\% - 30\%$ between daytime and harder conditions. After pretraining a model on both branches (could be from any experiment), therefore, this ratio is picked out of the number of randomly picked anchors, which was a 100. According to [46], a higher number would only improve the performance slightly, so that is already good for the needed validation whether this idea works. Also, here, a few ideas have been tested:

  - Assume that the anchor tensor is of shape $100 \times C \times H \times W$, then it is first transposd to $H \times W \times C \times 100$ and then reshape it to $(H \cdot W) \times C \times 100$. Now, with an non-linear MLP, process each entry of each latent space across the 100 picked anchor to a single value, and result with tensor of shape $(H \cdot W) \times C$. Now, it is transposed to $C \times (H \cdot W)$, to be used as anchors for the candidates.
  - Process each pixel candidate with the saved corresponding vectors from the pretrained vectors, using Conv1D vectors.

- **Loss function**:

$$\mathcal{L}_{\text{main}} + \lambda_{\text{mse}} \cdot \text{MSE}(l_{\text{rgb}}, l_{\text{rgbd}})$$

- **Observations**:
  1. Random initialized anchors:
     - Creating a shared manifold already showed a slight improvement.
     - The combination of a shared representation and minimization of the distance with the MSE loss function have shown the best performance so far.
  2. Pretrained latent vectors as anchors: Worse performance on the RGB branch - simply didn't converge to a meaningful score.

7.8 Linear Transformations (Rotations)

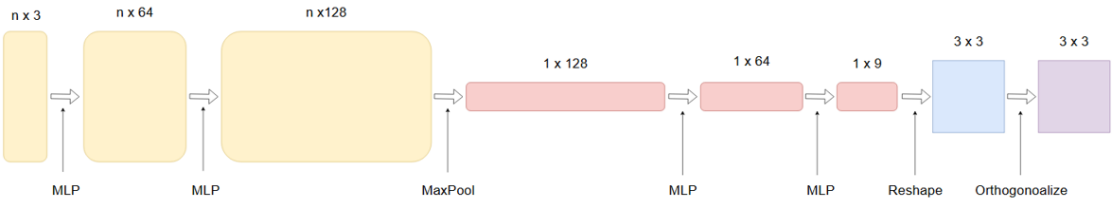


Figure 7.7: PoinNet: Extract individual features for each point, and then aggregate global features by a MaxPool layer, for downstream tasks.

Given a tensor-shaped latent space, performing a rotation on it is not straightforward. Consequently, various methods can be employed to reshape and permute the latent space to apply such a transformation. However, not all options will result in rotations that can be effectively learned by a neural network. An interesting approach to estimating a rotation from a point cloud was introduced by *PointNet* [68] (Figure 7.7). Since

point clouds are assumed to lack structure and are therefore invariant to permutations of points in their array in memory, this work proposed extracting features from each point individually and then aggregating global features using a MaxPool layer across the points for downstream tasks. The orthogonalization step is conducted using the *Procrustes Orthonormalization* process, as implemented in *RoMa* [38], which is the most robust among all possibilities and can be applied to any n and matrices in $SO(n)$.

- **Motivation**: As shown in [5], it is possible to find a rigid transformation, or even just a rotation, between two latent spaces for alignment, resulting in a seamless "stitching" that enables latent space A to solve the task of latent space B. Given that this transformation is recoverable by the Procrustes alignment algorithm, the primary challenge is to estimate it using a neural network. Additionally, the RGB-D point cloud is not reconstructed into a depth map; instead, only the rotated RGB latent is used, once with the RGB-D rotation and once with the RGB rotation. This ensures that both point clouds have the exact same structure and magnitude, as a rotation matrix preserves both, thereby focusing the problem solely on the difference between the angles of the estimated rotations. In addition, the RGB latent space is detached (no gradients are allowed to propagate through) for the RGB reconstruction, to avoid gradient duplication, as the structure of the point cloud should be guided by the RGB-D branch.

- **Details**: This section discusses experiments applying different techniques to manipulate the tensor-shaped latent space of shape $B \times C \times H \times W$. The accompanying equations are written in PyTorch syntax, as they are used in the code:
  - **Local point clouds**: Each pixel is broken into an independent point cloud with varying point dimensions: e.g. 2, 3, 16. Assuming a point dimension d and a latent space tensor A, the tensor is permuted and reshaped into point clouds:
    $$A = A.permute(0,2,3,1).reshape(B, H \cdot W, \frac{C}{d}, d) \tag{7.2}$$
  - **Global point clouds**: All pixels are broken into points of length d across the channels. The entire tensor A is reshaped into a **single** point cloud:
    $$A = A.permute(0,2,3,1).reshape(B, H \cdot W \cdot \frac{C}{d}, d) \tag{7.3}$$

- **Loss function**: While there are well-defined *geodesic* loss functions between rotations of $SO(3)$, for better comparison between experiments with varying point dimension d, the more simple, yet affective, approach has been chosen - the

MSE between the rotation matrices themselves or the MSE between the latent spaces after the rotations are applied.

$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_{\text{mse}} \cdot \text{MSE}(R_{\text{rgb}}, R_{\text{rgbd}}) \tag{7.4}$$

or

$$\mathcal{L} = \mathcal{L}_{\text{main}} + \lambda_{\text{mse}} \cdot \text{MSE}(l_{\text{rgb}}, l_{\text{rgbd}}) \tag{7.5}$$

- **Observations**:
  - **Local point clouds**: Breaking each pixel into a point cloud allowed each pixel to have 9 DoF. Consequently, the network transferred depth values from the input, complicating the alignment process.
  - **Global point clouds**: This experiment heavily depends on the point dimension d. The smaller d is, the fewer DoFs the RGB-D branch can manipulate. The latent spaces were aligned with $d = 3$, but with $d = 16$, the same old erroneous behavior emerged. Therefore, the most promising trial so far has been a global point cloud of 3D points.

8 Results and Discussion

This chapter dives into the results of the experiments. The analysis focuses on how well the proposed methods align the latent spaces derived from both RGB and RGB-D branches and how effective these methods are in enhancing depth map prediction. Root-Mean-Squared-Error (RMSE) is used as a key metric, and visual comparisons are included to showcase the strengths and weaknesses of the approach. Key observations, insights, and limitations from the experiments are discussed, and the broader implications of these findings for the field of depth estimation are considered.

8.1 Results

This section analyzes the numerical and visual results of the experiments, focusing on the key contributions of this thesis: a small survey of methods of aligning latent spaces from two different modalities, with and without rotations, where the latter is the main contribution of this thesis.

All experiments converged to a similar validation error range (approximately RMSE (in meters) of $4.2m$), despite the training loss dropping to $3.1m$. Real alignment was achieved only when the RGB latent space was manipulated using RGB-D values.

Incorporating the CBAM [62] layers resulted in a simple yet effective architecture, outperforming visual transformers in small models while maintaining a relatively small number of parameters. The final model comprises only 8M parameters and processes a single RGB image in 0.02 seconds during inference. Despite its modest size, the model exhibited significant overfitting, indicating sufficient capacity.

Additionally, although the inverse-sigmoid technique (Equation 5.11) for depth prediction was an intriguing concept, it did not result in any numerical improvement in the final depth prediction and was therefore discarded.

An interesting point is that the baseline *"Only RGB"* experiments already achieved very promising results, given the low number of parameters. However, it became evident that training the model without the infinity loss (Equation 5.13) yielded slightly better outcomes. This is because the loss function synthetically added more points of reference to penalize with a loss, where the segmentation of the skies was done using another pretrained model, leading to errors and misalignment. Consequently, the model had to contend with more inconsistencies between the RGB cues and the

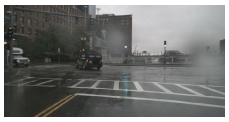
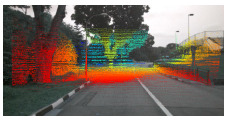
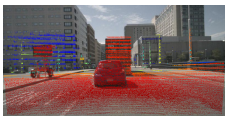
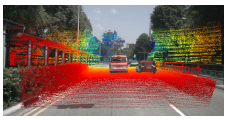
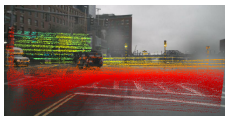
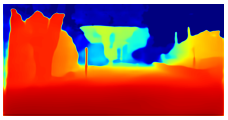
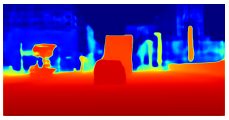
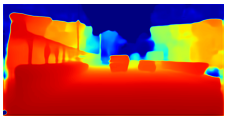
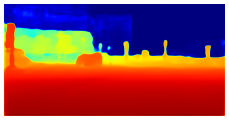
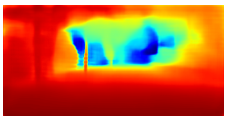
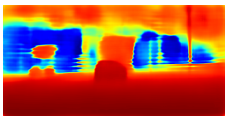
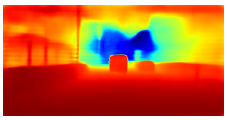
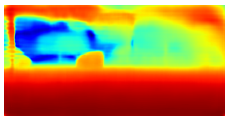
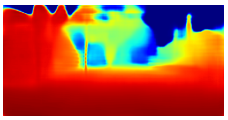
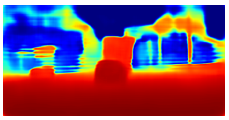
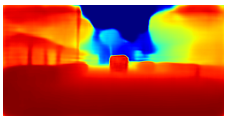
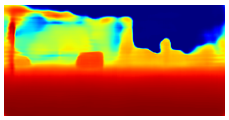
Experiment name	00453	12092	24805	15409-rain
RGB Input				
GT				
Depth Reconstruction				
Only RGB				
Only RGB Infinity Loss				

Table 8.1: Depth prediction and *reconstruction* for 4 different NuScenes images. The depth reconstruction illustrates the goal of this work—what is aimed to be achieved. The last two lines display the baseline, without any attempted alignment.

desired ground truth, resulting in slightly worse baseline performance but producing a much better visual depth map. Ultimately, it was decided to use the loss function in this thesis, for the sake of better visuals for comparisons, as seen in Table 8.1.

One of the most important observations from this work was the choice of tasks for both the RGB and RGB-D branches. Depth prediction is the main task of this work. However, depth reconstruction has highlighted a critical issue. It was observed that, when attempting to align the RGB and RGB-D features, even when the distance (Mean-Absolute-Error) between them was as small as $1e{-}4$, the model merely passed depth values from the input, failing to learn any general and meaningful features. This can be clearly seen in Figure 8.1, where the RGB branch develops a *generalization gap* between the training and validation losses, while the RGB-D branch does not. This issue persisted even with the introduction of aggressive regularization techniques, such as a high weight-decay coefficient ($\lambda = 0.2$) and Gaussian noise on the depth LiDAR points (with $std = 0.2$). The only effect was that both training and validation losses converged to a higher, yet similar, error value.

Another interesting observation emerged when allowing gradients to flow from

Experiment Name	RGB - RMSE (m)	Depth - RMSE (m)
Only-RGB w/o Inf-Loss	4.156	-
Only-RGB	4.206	-
MSE - End-To-End	4.176	2.447
MSE - 2nd Step	4.185	-
CORAL	4.133	2.570
Weights map	4.153	4.137
Relative Rep	4.159	2.461
Relative Rep - MSE	4.131	2.430

Table 8.2: A numerical comparison of the different experiments, evaluated by the chosen metric of Root-mean-squared-error (RMSE) in meters. Each experiment demonstrates the performance on the RGB and RGB-D ("depth") branches following the attempted alignment.

the RGB decoder. Deep learning models inherently seek the easiest solution to the task at hand. Consequently, while attempting to align the latent spaces to enhance depth prediction, the RGB branches persistently relied on suboptimal RGB cues, which is precisely the challenge this thesis aims to overcome. Conversely, if gradients are restricted, the model struggles to outperform the current methods listed in Table 8.2.

Furthermore, aligning the two absolute latent spaces using the MSE loss function did not lead to significant improvement. This is not surprising, as MSE is a widely known method for aligning tensors and its effectiveness would be well-documented if it were indeed effective. This experiment demonstrated that RGB-D data did not yield meaningful features. The alignment was poor, even with a low MAE error between the latent spaces. The decoder detected small differences between them, reconstructing depth from RGB-D and using RGB cues for the RGB branch without focusing on similarity. The minor performance differences observed in Table 8.2 may be due to random initialization and training steps, or the small model reaching its performance's lower bound.

On top of that, the CORAL [54] loss function offered a unique approach. While it did not align the latent space directly in Euclidean space, this technique forced the two latent spaces to have the same structure across the dataset, as the loss function is applied between all points across the batch. While groundbreaking results were not

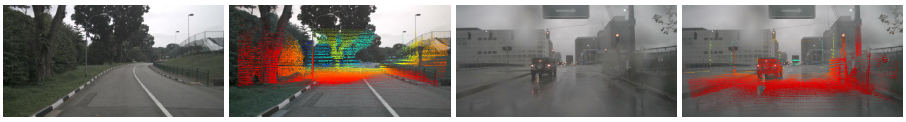
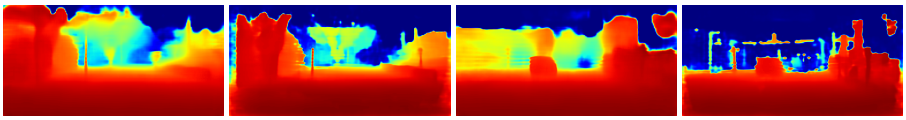
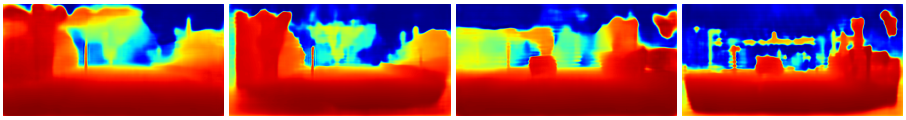
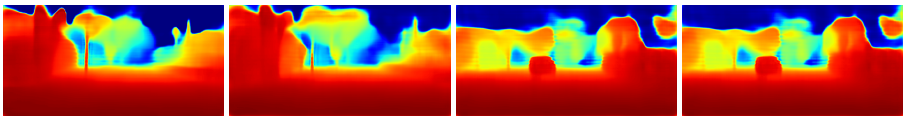
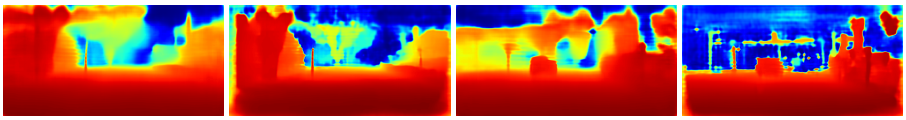
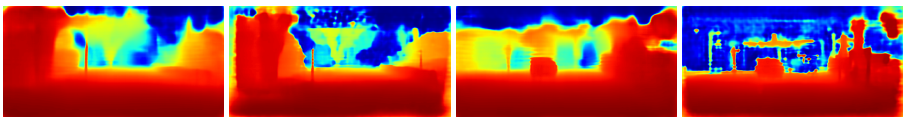
Experiment name	00453 - RGB	00453 - RGB-D	19732 - RGB	19732 - RGB-D
Inputs				
MSE				
CORAL				
Weights maps				
Relative Rep				
Relative Rep (MSE)				

Table 8.3: Depth prediction and *reconstruction* for the 00453 and 19732 scenes from the test-set. This is a visual comparison for the alignment of latent spaces.

expected due to the discussed RGB-D lack of meaningful features' problem, it exhibited some behavior that deserves further investigation once the said problem is overcome.

Furthermore, the *Relative Representations* experiments did not yield the desired results. Again, it might be due to the non-meaningful features, where even with the anchors, the model could decode specific depth values from the input. This could also explain why taking pretrained latent spaces as anchors did not improve performance. The anchors were simply not meaningful, supporting the observations by the authors[46] that the choice of anchors is crucial to the success of the representation.

However, the learnable anchors eventually adopted a structure that the RGB could use on unseen data, leading to better performance than the baseline (RMSE of 4.131m over 4.206m). This improvement was only achieved with **global** anchors, meaning the same set of anchors were used for all candidates, chosen to be the pixels across the channels. Conversely, having a "local" set of anchors for each pixel did not yield

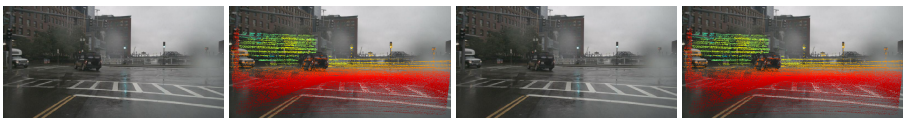
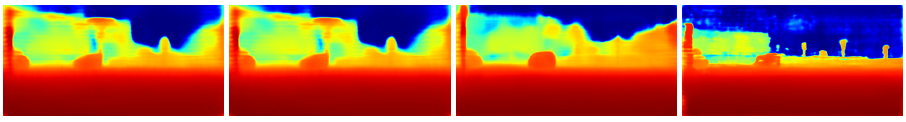
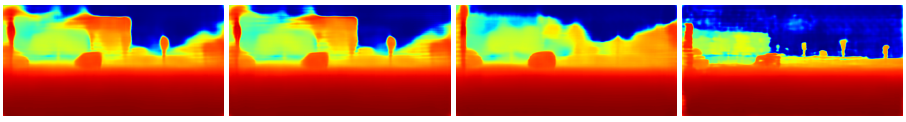
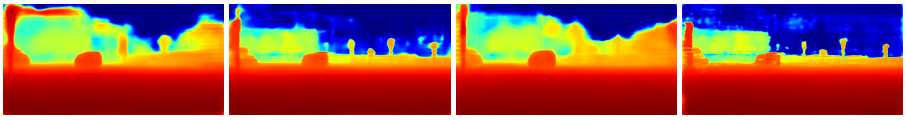
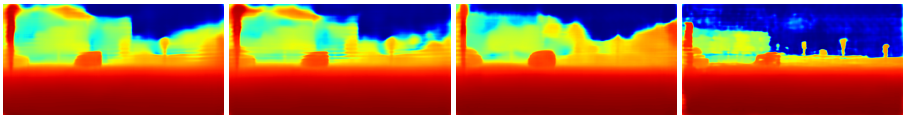
Experiment name	rotations		Relative Rep	
	15402 - RGB	**15402 - RGB-D**	**15402 - RGB**	**15402 - RGB-D**
Inputs				
Global ($dim = 2$)				
Global ($dim = 3$)				
Local ($dim = 3$)				
Global ($dim = 16$)				

Table 8.4: This figure shows the visual reconstruction of models using rotations for latent space alignment. Local point clouds failed, while global ones succeeded. Each experiment is compared to *"Relative Representation with MSE"* — the best method without rotations.

meaningful results, and the model struggled to converge.

All these experiments used the direct RGB-D latent space, and therefore suffered from the RGB-D branch transferring depth values, and not learning meaningful features. This could be visually observed in Table 8.3.

The next two experiments manipulate the RGB latent space rather than aligning RGB and RGB-D spaces directly. In the *weights-maps* experiment, a spatial weights map from the RGB-D branch is applied to the RGB latent space. This reduces the number of DoFs from $C \cdot H \cdot W$ to $H \cdot W$, making the RGB features easier to align and forcing the model to extract more meaningful features (Figure 7.6).

The final set of experiments is the most interesting one. Parallel to this thesis, [5] has shown that using the *Procrustes Alignment* algorithm between two latent spaces, from A to B, can allow for reconstruction in the domain of B. This significant discovery sets the stage to find a method of aligning two latent spaces, so the performance of the weaker

Linear Transformations Comparison					
PC context	**Point dim**	**Loss**	**RMSE (m)**		
			RGB	**Depth**	**Zero-shot**
Global	2	$\text{MSE}(R_{\text{rgb}}, R_{\text{depth}})$	4.419	4.351	5.091
Global	3	$\text{MSE}(R_{\text{rgb}}, R_{\text{depth}})$	4.146	4.117	**4.147**
Local	3	$\text{GEO}(R_{\text{rgb}}, R_{\text{depth}})$	4.177	2.778	5.486
Global	16	$\text{MSE}(R_{\text{rgb}}, R_{\text{depth}})$	4.252	4.252	4.252
Global	16	$\text{MSE}(l_{\text{rgb}}, l_{\text{depth}})$	**4.1084**	3.428	4.380

Table 8.5: Comparison between linear transformations applied to different contexts (global or local point clouds) and varying point dimensions. Different methods utilize different loss functions. Also, the "zero-shot" column describes the *stitching* (RGB encoder, RGB-D decoder) performance with no further training.

branch can leverage the strength of the stronger one. However, since depth values are not available during inference, one must estimate the identified linear transformation (rotation) instead.

The initial trials followed similar procedures as the other experiments. Both RGB and RGB-D latent spaces were broken into **local** (pixel-wise) point clouds, as defined in Section 4.3. The rotation between them was found, a new rotation was estimated with a smaller network (PointNet [68]), and the rotated RGB and RGB-D latent spaces were reconstructed. The ground truth for the estimated rotation was obtained by Procrustes, in a self-supervised manner. This approach faced the same problem as the others, where the RGB-D branch failed to learn meaningful features, as seen in Table 8.5. Therefore, similar to the *weights-maps* experiments, a more robust technique involved manipulating the RGB point clouds with only rotations learned from the RGB-D branch using Procrustes Alignment and the estimated rotation. This ensured that both rotated point clouds retained the same structure and magnitude, differing only in angle due to the properties of rotation matrices (Chapter 3).

Additionally, the latent spaces were reshaped into global point clouds, meaning only a single rotation was needed per image, significantly reducing the DoFs and forcing the model to learn more general features. As the point dimension d increased, alignment became more challenging due to exponential growth in DoFs. This method didn't achieve the goal of creating a shared manifold for enhancing RGB performance with the RGB-D branch substantially, but it still performed best overall. Additionally, as seen in Table 8.6, using the global method for point clouds allowed effective reconstruction

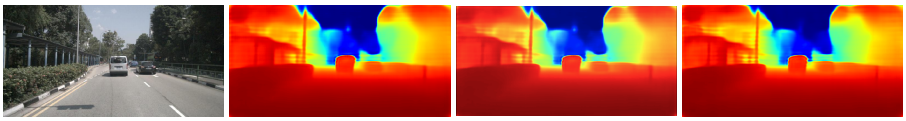
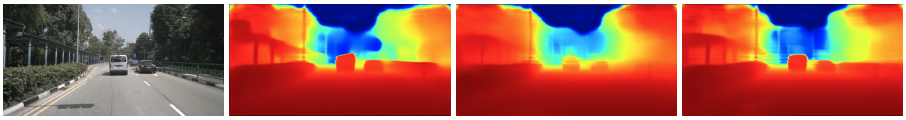
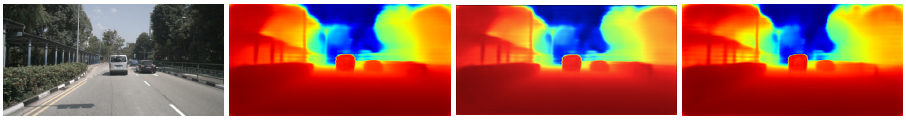
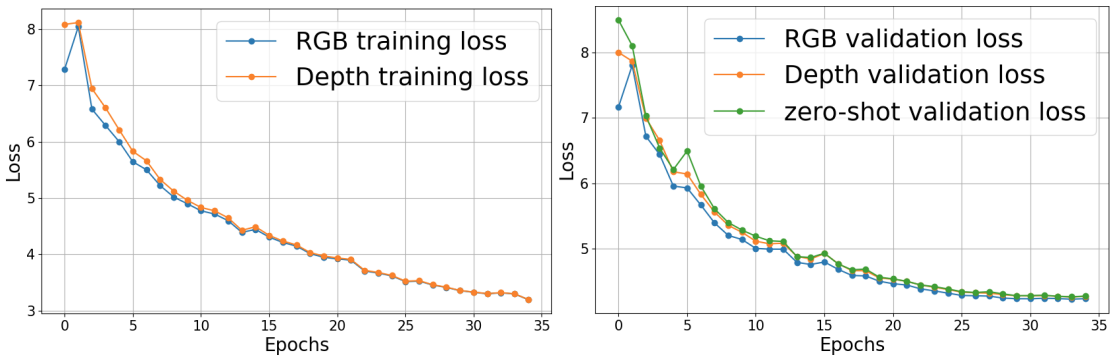
Experiment name	Original	Rotation Zero-Shot RGB-D	Zero-Shot	RGB
Global (3)				
Local (3)				
Global (16)				

Table 8.6: Visual depth maps were generated for the different branches of the rotation experiments: RGB-D, RGB, and the *zero-shot* scenario, which utilizes the RGB encoder and RGB-D decoder without any further training.

of RGB latent spaces with the RGB-D decoder in a zero-shot manner. The RGB reconstruction using the RGB-D decoder closely estimated the guided RGB-D branch, suggesting that fine-tuning them together might achieve the desired goal.



Training Validation

Figure 8.1: Training and validation losses of the GLOBAL rotation experiment with point dimension 3. The fact that the RGB-D branch had to regress more general features, led almost immediately to alignment in latent space and the creation of a shared manifold.

A crucial observation is that the most effective loss functions for aligning the point clouds weren't directly applied to the rotation matrices but on the distance metric of the rotated point clouds. This means there was no need for complex geodesic loss functions; the simple MSE loss sufficed.

Additionally, an interesting observation is the influence of geodesic loss functions compared to more conventional ones. In 3D, the geodesic loss functions (Equation 3.12) consider the angles between objects on the SO(3) manifold, which differs from the regular Euclidean space. However, as observed from the above experiments, a more effective method for aligning the point clouds was the use of a simple regression loss function, such as the MSE.

8.2 Conclusion and Future work

In conclusion, this thesis explored various approaches for aligning RGB and RGB-D latent spaces to enhance depth prediction. While traditional methods like MSE did not yield significant improvements, more innovative techniques such as the CORAL loss function and Relative Representations showed more promise, although they all suffered from the limitations of the non-meaningful features in the RGB-D branch.

In addition, incorporating the CBAM layers resulted in a compact and efficient architecture, demonstrating the potential of small models with a minimal number of parameters. The baseline "Only RGB" experiments achieved notable results, but the addition of the infinity loss highlighted the challenges of misalignment between an input and its corresponding ground truth.

However, the proposed mechanism for weighting (Section 6.1) different loss functions proved essential. It gave control back to the developer, addressing the magnitude issue between various loss functions. Without this solution, most experiments wouldn't have been feasible, as it would have been challenging to progress.

The experiments with spatial weights maps and global anchors demonstrated the importance of reducing degrees of freedom to extract meaningful features and achieve better alignment.

However, the main focus is on the rotation mechanism, which provided valuable insights into aligning RGB and RGB-D latent spaces. The Procrustes Alignment algorithm proved to be a significant discovery, showing the potential for creating a shared manifold that allows the performance of the weaker branch to benefit from the stronger one. In the end, it has proved that such transformations exist - and yet the challenge remains to estimate them correctly.

Initial trials demonstrated the challenges of using local point clouds for alignment, due to the RGB-D branch's inability to learn meaningful features. By manipulating the

RGB point clouds with rotations learned from the RGB-D branch and reshaping them into global point clouds, the number of degrees of freedom was reduced, forcing the model to learn more general features.

Despite not achieving the ultimate goal of significantly enhancing the RGB performance to the range of $3m$ RMSE, the rotations experiments yielded the best performance among all tests. The alignment allowed the RGB latent spaces to be effectively reconstructed using the RGB-D decoder in a zero-shot manner, indicating a promising direction for future research.

Future research could explore better tasks for depth reconstruction to help the RGB-D branch learn more meaningful features. Semantic segmentation might be a good alternative, though it would make zero-shot trials impossible. Combining methods like rotations and Relative Representation could also be an interesting path forward.

The model's capacity has been a major bottleneck. Given the success of rotation techniques, experimenting with larger models to capture more detailed features is essential. Revisiting visual transformers as a backbone is also worthwhile since they extract richer, more context-aware features. Studies like [69] show that visual transformers outperform traditional baselines by working in a semantic token space and focusing on different image parts based on context. This leads to richer latent space representations, potentially improving alignment and depth prediction.

List of Figures

List of Tables

Bibliography

[1] L. Piccinelli, Y.-H. Yang, C. Sakaridis, M. Segu, S. Li, L. V. Gool, and F. Yu, *Unidepth: Universal monocular metric depth estimation*, 2024. arXiv: 2403.18913 [cs.CV].

[2] D. Kim and S. Lee, *Clip can understand depth*, 2024. arXiv: 2402.03251 [cs.CV].

[3] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV].

[4] F. Sauerbeck, D. Halperin, L. Connert, and J. Betz, "Camradepth: Semantic guided depth estimation using monocular camera and sparse radar for automotive perception," *IEEE Sensors Journal*, 2023. DOI: 10.1109/JSEN.2023.3321886.

[5] Z. Lähner and M. Moeller, "On the direct alignment of latent spaces," in *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2023. [Online]. Available: https://openreview.net/forum?id=nro8tEfIfw.

[6] S. Jain, A. Radhakrishnan, and C. Uhler, *A mechanism for producing aligned latent spaces with autoencoders*, 2021. arXiv: 2106.15456 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2106.15456.

[7] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2020.

[8] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[9] dmckinnon, *Stereoscopy*, https://github.com/dmckinnon/stereo, 2019.

[10] V. Tankovich, C. Hane, Y. Zhang, A. Kowdle, S. Fanello, and S. Bouaziz, "Hitnet: Hierarchical iterative tile refinement network for real-time stereo matching," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 14 362–14 372.

[11] *Papers with code - stereo depth estimation on kitti 2015*, https://paperswithcode.com/sota/stereo-depth-estimation-on-kitti2015, Accessed: Date.

[12] *Driving sterero dataset*, https://drivingstereo-dataset.github.io/, Accessed: Date.

[13] K. Qian, S. Zhu, X. Zhang, and L. E. Li, "Robust multimodal vehicle detection in foggy weather using complementary lidar and radar signals," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2021, pp. 444–453.

[14] P. Wang, "Research on comparison of lidar and camera in autonomous driving," in *Journal of Physics: Conference Series*, IOP Publishing, vol. 2093, 2021, p. 012 032.

[15] J. Tang, F.-P. Tian, B. An, J. Li, and P. Tan, *Bilateral propagation network for depth completion*, 2024. arXiv: 2403.11270 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2403.11270.

[16] *Kitti: Depth completion benchmark*, https://www.cvlibs.net/datasets/kitti/eval_depth.php, Accessed: Date.

[17] Y. Long, D. Morris, X. Liu, M. Castro, P. Chakravarty, and P. Narayanan, "Radar-camera pixel depth association for depth completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 12 507–12 516.

[18] J.-T. Lin, D. Dai, and L. Van Gool, "Depth estimation from monocular images and sparse radar data," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 10 233–10 240.

[19] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012.

[20] Y. Zheng, C. Zhong, P. Li, H.-a. Gao, Y. Zheng, B. Jin, L. Wang, H. Zhao, G. Zhou, Q. Zhang, and D. Zhao, *Steps: Joint self-supervised nighttime image enhancement and depth estimation*, 2023. arXiv: 2302.01334 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2302.01334.

[21] *Diffusion models vs. gans vs. vaes: Comparison of deep generative models*, https://towardsai.net/p/generative-ai/diffusion-models-vs-gans-vs-vaes-comparison-of-deep-generative-models, Accessed: Date.

[22] L. He, J. Lu, G. Wang, S. Song, and J. Zhou, *Sosd-net: Joint semantic object segmentation and depth estimation from monocular images*, 2021. arXiv: 2101.07422 [cs.CV].

[23] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, *Depth anything: Unleashing the power of large-scale unlabeled data*, 2024. arXiv: 2401.10891 [cs.CV].

[24] S. Patni, A. Agarwal, and C. Arora, *Ecodepth: Effective conditioning of diffusion models for monocular depth estimation*, 2024. arXiv: 2403.18807 [cs.CV].

[25] A. Bochkovskii, A. Delaunoy, H. Germain, M. Santos, Y. Zhou, S. R. Richter, and V. Koltun, *Depth pro: Sharp monocular metric depth in less than a second*, 2024. [Online]. Available: https://arxiv.org/abs/2410.02073.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[27] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132 306, 2020.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, *An image is worth 16x16 words: Transformers for image recognition at scale*, 2021. arXiv: 2010.11929 [cs.CV].

[29] "Vit-pytorch." (2021), [Online]. Available: https://github.com/lucidrains/vit-pytorch.

[30] C. Yang, Y. Wang, J. Zhang, H. Zhang, Z. Wei, Z. Lin, and A. Yuille, *Lite vision transformer with enhanced self-attention*, 2021. arXiv: 2112.10809 [cs.CV].

[31] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2018. arXiv: 1608.06993 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1608.06993.

[32] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[33] M. A. U. Khan, D. Nazir, A. Pagani, H. Mokayed, M. Liwicki, D. Stricker, and M. Z. Afzal, "A comprehensive survey of depth completion approaches," *Sensors*, vol. 22, no. 18, 2022, ISSN: 1424-8220. [Online]. Available: https://www.mdpi.com/1424-8220/22/18/6969.

[34] W. Boettcher, L. Hoyer, O. Unal, K. Li, and D. Dai, *Lidar meta depth completion*, 2023. arXiv: 2307.12761 [cs.CV].

[35] B. Fei, W. Yang, L. Liu, T. Luo, R. Zhang, Y. Li, and Y. He, *Self-supervised learning for pre-training 3d point clouds: A survey*, 2023. arXiv: 2305.04691 [cs.CV].

[36] L. Xia, J. Liu, and T. Wu, *Depth estimation algorithm based on transformer-encoder and feature fusion*, 2024. arXiv: 2403.01370 [cs.CV].

[37] A. Kendall and Y. Gal, *What uncertainties do we need in bayesian deep learning for computer vision?* 2017. arXiv: 1703.04977 [cs.CV].

[38] R. Brégier, *Deep regression on manifolds: A 3d rotation case study*, 2021. arXiv: 2103.16317 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2103.16317.

[39] A. R. Geist, J. Frey, M. Zobro, A. Levina, and G. Martius, *Learning with 3d rotations, a hitchhiker's guide to so(3)*, 2024. arXiv: 2404.11735 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2404.11735.

[40] L. Moschella, V. Maiorca, M. Fumero, A. Norelli, F. Locatello, and E. Rodolà, *Relative representations enable zero-shot latent space communication*, 2023. arXiv: 2209.15430 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2209.15430.

[41] W. Deng, K. Zhang, Z. Wei, L. Wang, C. He, S. Liu, and H. Wei, "Hb-pls: An algorithm for identifying biological process or pathway regulators by integrating huber loss and berhu penalty with partial least square," May 2020. DOI: 10.1101/2020.05.16.089623.

[42] O. Álvarez-Tuñón, Y. Brodskiy, and E. Kayacan, *Loss it right: Euclidean and riemannian metrics in learning-based visual odometry*, 2023. arXiv: 2401.05396 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2401.05396.

[43] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, *On the continuity of rotation representations in neural networks*, 2020. arXiv: 1812.07035 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1812.07035.

[44] L. M. Haines, *Stereographic projections for designs on the sphere*, 2024. arXiv: 2401.05931 [stat.AP]. [Online]. Available: https://arxiv.org/abs/2401.05931.

[45] Y. Bengio, A. Courville, and P. Vincent, *Representation learning: A review and new perspectives*, 2014. arXiv: 1206.5538 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1206.5538.

[46] L. Moschella, V. Maiorca, M. Fumero, A. Norelli, F. Locatello, and E. Rodolà, "Relative representations enable zero-shot latent space communication," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=SrC-nwieGJ.

[47] J. Gao, Y. Liu, Y. Sun, Y. Tang, Y. Zeng, K. Chen, and C. Zhao, *Styleshot: A snapshot on any style*, 2024. arXiv: 2407.01414 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2407.01414.

[48] J. Shlens, *A tutorial on principal component analysis*, 2014. arXiv: 1404.1100 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1404.1100.

[49] V. Maiorca, L. Moschella, A. Norelli, M. Fumero, F. Locatello, and E. Rodolà, *Latent space translation via semantic alignment*, 2024. arXiv: 2311.00664 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2311.00664.

[50] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[51] H. Xiao, K. Rasul, and R. Vollgraf, *Fashion-mnist: A novel image dataset for benchmarking machine learning algorithms*, 2017. arXiv: 1708.07747 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1708.07747.

[52] E. Son and S. J. Lee, "Cabins: Clip-based adaptive bins for monocular depth estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2024, pp. 4557–4567.

[53] S. Paul, B. Jhamb, D. Mishra, and M. S. Kumar, "Edge loss functions for deep-learning depth-map," *Machine Learning with Applications*, vol. 7, p. 100 218, 2022, ISSN: 2666-8270. DOI: https://doi.org/10.1016/j.mlwa.2021.100218. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666827021001092.

[54] B. Sun and K. Saenko, *Deep coral: Correlation alignment for deep domain adaptation*, 2016. arXiv: 1607.01719 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1607.01719.

[55] A. A. Heydari, C. A. Thompson, and A. Mehmood, *Softadapt: Techniques for adaptive loss weighting of neural networks with multi-part loss functions*, 2019. arXiv: 1912.12355 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1912.12355.

[56] L. N. Smith and N. Topin, *Super-convergence: Very fast training of neural networks using large learning rates*, 2018. arXiv: 1708.07120 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1708.07120.

[57] M. Novik, *torch-optimizer – collection of optimization algorithms for PyTorch.* Version 1.0.1, Jan. 2020. [Online]. Available: https://github.com/jettify/pytorch-optimizer.

[58] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1412.6980.

[59] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, 2019. arXiv: 1711.05101 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1711.05101.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Torch.optim.lr$_s$cheduler.onecyclelr*, https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.OneCycleLR.html, Accessed: 2024-09-12, 2019.

[61] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, *On the variance of the adaptive learning rate and beyond*, 2021. arXiv: 1908.03265 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1908.03265.

[62] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, *Cbam: Convolutional block attention module*, 2018. arXiv: 1807.06521 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1807.06521.

[63] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[64] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, 2018. arXiv: 1608.06993 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1608.06993.

[65] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[66] D. Hendrycks and K. Gimpel, *Gaussian error linear units (gelus)*, 2023. arXiv: 1606.08415 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1606.08415.

[67] Y. Wu and K. He, *Group normalization*, 2018. arXiv: 1803.08494 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1803.08494.

[68] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *Pointnet: Deep learning on point sets for 3d classification and segmentation*, 2017. arXiv: 1612.00593 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1612.00593.

[69] B. Wu, C. Xu, X. Dai, A. Wan, P. Zhang, Z. Yan, M. Tomizuka, J. Gonzalez, K. Keutzer, and P. Vajda, "Visual transformers: Where do transformers really belong in vision models?" In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 579–589. DOI: 10.1109/ICCV48922.2021.00064.