Master Thesis

Analysing Multi-view Depth Estimation in a Common Framework

Author:	Saiprasad Barke
Matriculation Nr:	4559259
Examiner:	Prof. Dr. Thomas Brox
Second Examiner:	Prof. Dr. Matthias Teschner
Advisor:	Philipp Schröppel

Albert-Ludwigs-Universität Freiburg

Faculty of Engineering

Department of Computer Science

Chair for Computer Vision

September 15th, 2023

Writing period

15. 03. 2023 – 15. 09. 2023

Examiner

Prof. Dr. Thomas Brox

Second Examiner

Prof. Dr. Matthias Teschner

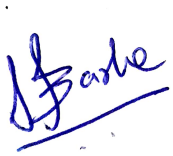
Advisor

Philipp Schröppel

"Dedicated to the memory of my father, Govind Barke, who always believed in my ability to be successful in the academic arena. You are gone but your belief in me has made this journey possible."

Declaration

I hereby declare that I am the sole author and composer of my thesis and that no other sources or learning aids other than those listed have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare that my thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.



Mumbai, India	Saiprasad Barke
Place, Date	Signature

Abstract

Understanding the key factors influencing the performance of depth estimation models is crucial in advancing computer vision applications such as 3D reconstruction, autonomous navigation, and augmented reality. This thesis explores these influencing factors within the context of the Robust Multi-View Depth(RobustMVD)[1] framework. We implement several models under the RobustMVD umbrella, each uniquely different in architectural elements, to decipher their individual and collective impacts on depth estimation performance. We also examine the contributions of different data augmentation techniques and investigate the implications of the choice of training datasets. Further, we scrutinize the effects of manipulating data properties, like the number of source views, source selection approach, and diverse training hyperparameters, on the model's predictive performance and uncertainty estimation, offering insights into the principles that underlie the performance of multi-view depth estimation models to empower future model design and refinement of depth estimation methodologies.

We observe that the depth estimation performance within the RobustMVD framework is substantially influenced by the selection of the correlation layer and the feature extractor's capability to produce representative features. Additionally, the choice of input data and the way it's processed plays an indispensable role in model robustness. We also underscore the importance of normalizing features before correlation, a simple yet effective enhancement. The research also sheds light on potential avenues for future exploration, emphasizing the myriad of methodologies that, when aptly combined, can significantly boost the performance of Multi-View Stereo models. This report outlines the experimental setup, presents the results of the ablation study, and discusses potential directions for future research in the domain of depth estimation in a multi-view setting.

Contents

List of Figures

List of Tables

1. Introduction

1.1. Background of the Problem

Depth estimation is a cornerstone task in the realm of computer vision, with a wide array of applications, including 3D reconstruction, autonomous navigation, augmented reality, 3D printing, computational photography, computer video games, and preservation of cultural heritage. As we advance into an era defined by data and artificial intelligence, the importance of developing robust and efficient depth estimation models is paramount. When materials, viewpoints, and lighting conditions are well-defined, the task becomes more tractable. However, if any of these factors are not known, the problem typically becomes ill-posed. This is because a variety of combinations that involve geometry, materials, viewpoints, and lighting can result in identical photographs. In such cases, the ambiguity introduced by these unknown variables makes it difficult to uniquely determine the exact characteristics of the scene from the given images.

Humans possess an impressive capacity to solve such inverse problems, even when the solutions are uncertain. We can draw on our prior knowledge to make estimations about the size, arrangement, and coarse proximity of objects in the environment in relation to our own position and between the objects themselves. This proficiency can be attributed to previous experiential interactions with diverse objects and contexts, culminating in constructing cognitive priors that facilitate an insightful understanding of the visual attributes and spatial interconnections within the three-dimensional world around us. For decades, computer scientists have worked hard to recreate this amazing ability that we take for granted to enable computers to do the same. Inspired by the human binocular vision system, stereo matching and its extension Multi-View Stereo (MVS)[2] based depth estimation models have garnered immense attention. In particular, Multi-View Stereo models have become extremely successful due to their ability to take advantage of multiple views of a scene to produce accurate depth maps.

While traditional Multi-View Stereo methods have been successful to some extent, they often rely heavily on handcrafted features and similarity measures. This leads them to have limitations in handling poorly textured regions, and the models may struggle with occlusions

or varying lighting conditions. With the rise of deep learning and especially Convolutional Neural Networks (CNN)[3] and in recent years, the Visual Transformer(ViT)[4, 5, 6], depth estimation has seen a paradigm shift, with models now capable of learning features directly from data and achieving superior performance. This, combined with the growing accessibility of extensive training datasets, has opened a new era of Multi-View Stereo methodologies[2]. These approaches address the complex inverse problem of inferring depth information from RGB images.

However, it has been observed in practice that the performance of these models depends on various factors, such as the underlying network architecture, training datasets, data augmentations, and several hyperparameters of the network and training setup. It has also been observed that models trained on a specific data modality do not generalize well to other modalities without significant adaptations and fine-tuning. This robustness characteristic is extremely important in the real world. In practical settings involving multi-camera layouts, we often have access to camera positions but lack depth range and alignment details of the ground truth and the predicted depth maps. Deep learning MVS methodologies aim to accurately determine relevant depth maps and their appropriate scales from this stream of visual data.

In this thesis, we explore which combinations of the factors mentioned above yield superior generalization results through different data modalities, particularly in challenging situations such as areas with intricate structures, sections without texture, reflective surfaces, zones with pronounced gradients, scenes involving minor camera movements, and instances of occlusions.

1.2. A Brief Overview of RobustMVD

In this study, we take a closer look at the Robust Multi-View Depth framework[1], which provides a common ground to evaluate multi-view depth models on their generalization capabilities in various domains. The framework has also given us the DispNet based RobustMVD depth estimation model and a novel input image data augmentation technique called *Scale Augmentation*[1]. This approach has attracted much attention in the community by demonstrating robust cross-domain scale-agnostic multi-view depth estimation.

The framework also introduces the Robust Multi-View Depth benchmark [1] founded on existing datasets with different modalities. This benchmark offers a three-fold evaluation protocol. Firstly, it gauges the performance of estimated depth maps across distinct data domains through a zero-shot assessment paradigm. Second, it quantifies uncertainties using the Area-Under-the-Sparsification-Error-Curve (AUSE)[7] metric. Additionally, the benchmark encompasses an absolute evaluation scenario motivated by practical use cases, where precise

camera poses are provided to the model. The benchmark emphasizes depth estimation's generalization capabilities rather than just performance on specific datasets. It is proposed as an additional evaluation tool alongside traditional benchmarks such as KITTI, ScanNet, DTU, Tanks and Temples, and others. Overall, Robust Multi-View Depth offers a comprehensive platform to evaluate the generalization capabilities of depth estimation models in various domains, addressing both depth prediction and associated uncertainties.

Despite the effectiveness of the RobustMVD framework, the complexity and multifaceted nature of the Multi-View Stereo raise several intriguing questions. How do individual components of the network architecture impact overall performance? What is the role of input data augmentations, and how do they affect the model's ability to generalize? How do the choice of data sets and the properties of the data influence performance? Also, how do the various hyperparameters of the models and the training setup steer the accuracy and efficiency of the models?

1.3. Ablation Study and its Objectives

In this thesis, we aim to address these questions systematically. To this end, we conducted an extensive ablation study with the RobustMVD framework, where we investigated the contribution of each of these components and factors to the overall performance of the model. We implemented different models within the RobustMVD framework, each designed with different network architecture components. The API is written in a manner that makes it easier to conduct experiments in a plug-and-play fashion by making the interfaces of the different architectural components and the data loading for different datasets compatible. We then evaluate the impact of different input augmentations and assess the effect of training these models on various datasets in a known depth range setting. Furthermore, we consider a range of data properties, such as the number and sampling strategy of source views, and perform an in-depth exploration of the impacts of altering different hyperparameters in the network and training setup.

Through our ablation study, we aim to shed light on the intricacies of depth estimation using Multi-View Stereo and contribute to the understanding of the principles driving the performance of multi-view depth estimation models. By identifying the most influential factors, we aim to inform the design of future models and enhance the effectiveness of depth estimation methodologies.

1.4. Report Structure

The rest of this report is organized as follows: In Chapter 2, we revisit the fundamental concepts of multi-view geometry, disparity, and depth estimation in a multi-view setting. We also have a detailed study of a typical Multi-View Stereo depth estimation pipeline to explain the terminology used in this thesis. In Chapter 3, we conduct a thorough literature review of existing research in the domain of depth estimation in an MVS setting. In Chapter 4, we present the details of our experimental setup, datasets, and model performance evaluation metrics. In Chapter 5 we discuss the results of our ablation study, organized by factors under consideration: **1)** network architecture components, **2)** datasets, their properties and augmentations and **3)** hyperparameters of the training setup. Finally, in Chapter 6, we draw conclusions from our findings and discuss potential avenues for future research in this domain. Model implementation details and quantitative analyses are presented in the Appendix.

2. Background

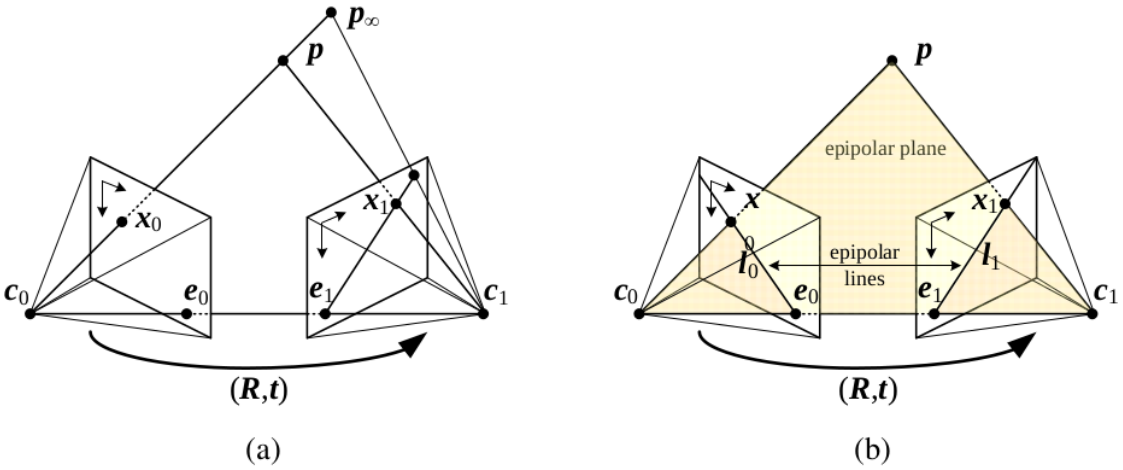


Figure 1.: A typical Epipolar Geometry Setup. $\mathbf{p}$ is the 3D point in the scene. $\mathbf{c_0}$ and $\mathbf{c_1}$ are the centers of the cameras. The triangle defined by the points $\mathbf{p}, \mathbf{c_0}, \mathbf{c_1}$ is the epipolar plane. The line $\mathbf{c_0}$,$\mathbf{c_1}$ that connects the two camera centers is the baseline, and the lines $\mathbf{x_0}, \mathbf{e_0}$, and $\mathbf{x_1}, \mathbf{e_1}$ are the epipolar lines. This image is obtained from [8]

In Multi-View Stereo methodologies, camera matrices, source views, and reference views play crucial roles in establishing correspondences, predicting depth, and, ultimately, the reconstruction process. There is an interesting relationship between the multiple cameras, the 3D point in the scene, and its projections in the image planes of the cameras. These relationships can be described by epipolar geometry. Figure 1 shows a standard epipolar geometry setup. This setup involves 2 cameras at $\mathbf{c_0}$ and $\mathbf{c_1}$ observing the same 3D point $\mathbf{p}$ in the scene. $\mathbf{x_0}$ and $\mathbf{x_1}$ are the projections of $\mathbf{p}$ on the image planes of the two cameras.

First, we will define the terminology associated with epipolar geometry and the estimation of the depth/disparity. Then, in the subsequent sections, we will formalize the definitions and draw connections between the different concepts.

2.1. Relevant Terminology

- **Epipolar Plane:** Epipolar plane is the plane defined by the centers of the cameras and a 3D point in the scene. In Figure 1, the triangle defined by the points $\mathbf{p}, \mathbf{c_0}, \mathbf{c_1}$ is the epipolar plane.
- **Epipolar Line:** The intersections of the epipolar plane with the image planes are known as the epipolar lines. In Figure 1, the two lines $\mathbf{x_0}, \mathbf{e_0}$ and $\mathbf{x_1}, \mathbf{e_1}$ are the epipolar lines.
- **Epipoles:** The points of intersection of the baseline $\mathbf{c_0}, \mathbf{c_1}$ with the image planes are the epipoles. In Figure 1, the points $\mathbf{e_0}$ and $\mathbf{e_1}$ are the epipoles.
- **Epipolar Constraint:** It is the fundamental rule of epipolar geometry that states that any projection $\mathbf{x_0}$ of the 3D point $\mathbf{p}$ in one image must lie on the corresponding epipolar line $\mathbf{x_1}, \mathbf{e_1}$ in the other image. This restriction greatly reduces the search space when attempting to match points between two images, from a 2D search to a 1D search along the epipolar line.
- **Camera Matrices:** A camera matrix represents the intrinsic (lens) and extrinsic (pose) parameters of a camera. It consists of a 3×3 matrix containing the intrinsic parameters such as focal length, principal point, and lens distortion coefficients, along with a 4×4 matrix representing the extrinsic parameters, which describe the camera's position and orientation in 3D space. Camera matrices are used to project 3D points onto the image plane and project 2D points back onto the 3D space when reconstructing the scene.
- **Homography:** A transformation H that maps the points in one image plane to the corresponding points in another image plane.
- **Homography Matrix:** An arbitrary 3×3 transformation matrix in a homogeneous coordinate space that represents the homography transformation.
- **Homography Warping:** The process of applying a homography transformation to an image or part of an image so that the first image aligns with the second image.
- **Rectification:** The process of transforming stereo images so that their corresponding epipolar lines become collinear and parallel to the x-axis. This simplifies the search for the corresponding points from a 2D search to a 1D search. This leads to the two image planes being parallel to each other.

- **Rectified Images:** The images obtained after rectifying. The epipolar lines in these images are parallel to the x-axis.
- **Image Correspondences:** This refers to the process or result of finding common features or points between two or more images.
- **Disparity:** The difference in the position of a 3D point observed in two different images. In rectified images, the disparity corresponds to the horizontal difference.
- **Disparity Map:** An image in which the value of each pixel represents the disparity of that pixel between two stereo-rectified images.
- **Depth Map:** An image in which the value of each pixel represents the estimated depth of that pixel in the scene.
- **Source Views:** In MVS, source views refer to a set of input images captured from different viewpoints around a scene. These views are used to extract visual information and establish correspondences between image points. Source views provide multiple perspectives that are necessary to accurately estimate the depth values in the scene.
- **Reference View:** A reference view is a specific source view selected as a reference frame for the depth estimation process. It serves as an anchor image for calculating the depth map or point cloud of the scene. The other source views are then aligned and matched with the reference view to establish correspondences using the camera's intrinsic and extrinsics matrices to compute the depth information.
- **View Selection Strategy:** Choosing the right images as source and reference views is a crucial step in MVS. Each view has its own extrinsic camera parameters. These choices can significantly impact the accuracy and quality of the final reconstruction. A good strategy is to choose source views that cover the entire scene with adequate overlap and reference views that are centrally located or provide the most informative view of the scene with minimum occlusions and good illumination.

2.2. Epipolar Geometry and Image Correspondences

Now that we have established a foundation of 3D geometry vocabulary, we will thoroughly analyze the formal definitions of the abovementioned concepts. Given a pixel in one image, how can we compute its correspondence in another image efficiently and reliably? For the

scenarios discussed in this thesis, we have the positions and calibration data of the cameras available.

In Figure 1a, we can see how one pixel $\mathbf{x_0}$ projects to an epipolar line segment in the other image. One end of the segment is defined by the projection of the original viewing ray at infinity, denoted as $\mathbf{p}\infty$. The other end is defined by the projection of the original camera center, $\mathbf{c_0}$, into the second camera, known as the epipole $\mathbf{e_1}$. By projecting the epipolar line from the second image back into the first, we can create another line bounded by the corresponding epipole $\mathbf{e_0}$. These line segments can be extended to infinity, creating a pair of corresponding epipolar lines as shown in Figure 1b that are the intersections of the two image planes and the epipolar plane that is bounded by both camera centers $\mathbf{c_0}$ and $\mathbf{c_1}$ and the 3D point of interest, $\mathbf{p}$ [9]. Now, let us examine the different coordinate systems and camera matrices.

The commonly used coordinate systems in Computer Vision are:

1. **World coordinate system (3D):** This is a 3D cartesian coordinate system with an arbitrary origin. It is the base reference system. A point in this coordinate system can be denoted as $\mathbf{P_w} = (X_w, Y_w, Z_w)$.

2. **Camera coordinate system (3D):** This is also a 3D cartesian coordinate system with its origin at the origin of the camera. The coordinates are measured with respect to the camera center and orientation. A point in the camera coordinate system is denoted as $\mathbf{P_c} = (X_c, Y_c, Z_c)$. One can go from the world coordinate system to the camera coordinate system using the camera extrinsics matrix. This 4×4 matrix can be broken down into a rotational $\mathbf{R}_{3\times3}$ and a translational $\mathbf{t}_{3\times1}$ component. The camera coordinates can be obtained from the world coordinates using the extrinsics matrix by Equation1.

$$\begin{pmatrix} \mathbf{P_c}^\intercal \\ 1 \end{pmatrix}_{4\times1} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & \mathbf{1}_{1\times1} \end{bmatrix}_{4\times4} \begin{pmatrix} \mathbf{P_w}^\intercal \\ 1 \end{pmatrix}_{4\times1} \tag{1}$$

3. **Image coordinate system (2D):** This coordinate system is obtained when the 3D camera coordinates are projected onto the 2D image plane. This projection operation results in the loss of depth information or the Z coordinate. A point in the image coordinate system is denoted as $\mathbf{P_i} = (X_i, Y_i)$. For the pinhole camera model, the 2D image plane is located at focal length f away from the camera. This projection involves the camera

intrinsics matrix. The camera intrinsics matrix is given by Equation 2.

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

Here, assuming square pixels, f_x = f_y = f is the focal length of the camera. The skew factor is denoted by s, which is usually 0. c_x and c_y denote the coordinates of the camera's center. We can obtain the 2D image coordinates from the 3D camera coordinates with Equation 3:

$$\mathbf{X}_i = f\frac{\mathbf{X_c}}{\mathbf{Z_c}} \qquad \text{and} \qquad \mathbf{Y}_i = f\frac{\mathbf{Y_c}}{\mathbf{Z_c}} \tag{3}$$

This can also be represented in matrix form by Equation 4:

$$\begin{pmatrix} \mathbf{P_i}^\intercal \\ Z_c \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} \mathbf{P_c}^\intercal \\ 1 \end{pmatrix} \tag{4}$$

4. **Pixel coordinate system (2D):** Pixel coordinate system is obtained when the image coordinate system is discretized. Another pair of parameters called the pixel width ρ_u and the pixel height ρ_v come into play for this transformation. The pixel coordinates can be calculated by dividing the image coordinates by the width and height of the pixels. Since the pixel coordinate system begins at the top-left of the image, we also require the remaining two parameters of the camera intrinsics matrix, c_x and c_y. This is given in Equation 5.

$$\mathbf{X}_i = \frac{f}{\rho_u}\frac{\mathbf{X_c}}{\mathbf{Z_c}} + c_x \qquad \text{and} \qquad \mathbf{Y}_i = \frac{f}{\rho_v}\frac{\mathbf{Y_c}}{\mathbf{Z_c}} + c_y \tag{5}$$

This can be represented in matrix form by Equation 6.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{bmatrix} \frac{1}{\rho_u} & 0 & c_x \\ 0 & \frac{1}{\rho_v} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} \mathbf{X_i}^\intercal \\ \mathbf{Z_c} \end{pmatrix} \tag{6}$$

Using the above equations, we get Equation 7, which is the transform from world coordinates to pixel coordinates using the camera intrinsics and extrinsics matrices. This

product of the intrinsics and the extrinsics matrices is known as the projection matrix $\mathbf{P}$:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \underbrace{\begin{bmatrix} \frac{f}{\rho_u} & 0 & c_x & 0 \\ 0 & \frac{f}{\rho_v} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{Intrinsics}} \underbrace{\begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & \mathbf{1}_{1\times1} \end{bmatrix}_{4\times4}}_{\text{Extrinsics}} \begin{pmatrix} \mathbf{P_w}^\intercal \\ 1 \end{pmatrix}_{4\times1} \quad \text{and} \quad \mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{7}$$

In the stereo configuration depicted in Figure 1, several foundational assumptions are essential. Both cameras exist within a canonical configuration. The term "canonical" in stereo vision refers to a rectified scenario wherein the epipolar lines of the two cameras are horizontally aligned. Such rectification considerably streamlines the process of disparity estimation. Further reinforcing this simplification, both cameras are assumed to possess identical intrinsic matrices, denoted by $\mathbf{K}$. In this setup, we also assume that the origin of the world coordinate system is at the first camera, with the second camera offset first by a rotation $\mathbf{R}$ and then by a translation $\mathbf{t}$. Specifically, in this harmonized environment, determining disparity for a given plane at depth d equates to computing the corresponding homography, which is a projective transformation for that specific depth. Since the image planes in a rectified setup are parallel, the transformation induced by a plane-specific homography invariably results in a spatial shift. Every point on the delineated plane undergoes this shift, which is equivalent in magnitude to the plane's established disparity.

This foundational stereo model resembles the paradigm found in Multi-View Stereo (MVS). Within the MVS framework, source image projections are methodically warped onto fronto-parallel planes. These planes reside within the frustum of a specified reference camera and span diverse depth levels. As these planes undergo spatial adjustments, either approaching or receding from the camera setup, the inherent disparity alters. This change in disparity underscores the principle that every depth-defined plane possesses a unique disparity value and, by extension, a distinctive homography.

Next, we will discuss the connection between disparity and depth, as well as introduce cost volumes and various types of loss functions in the following section.

2.3. Disparity and Depth

As a problem domain, depth estimation aims to predict the distance from the camera to each pixel in an image or a sequence of images. In the context of MVS, where the camera parameters are known, the problems of depth or disparity estimation and that of solving for the 3D geometry

of the scene are the same, and these are, in turn, equivalent to solving the problem of accurately determining the point correspondences across the images [10]. Knowing the likelihood and uncertainty of estimating these point correspondences is also important.

There is a straightforward inverse relationship between depth Z and disparity d for rectified images. This is given by Equation 8. [8, 11]

$$d = f\frac{B}{Z} \tag{8}$$

where f is the focal length in pixels, and B is the baseline length connecting the two camera centers. In such a case, extracting depth from a set of images is equivalent to estimating the disparity map $d(x, y)$ [8].

Simply put, objects closer to the cameras have larger disparities than those farther away. By computing the disparity for every pixel or feature point between two stereo images, a disparity map can be generated. This can then be converted into a depth map using the camera's parameters and geometry. This interchangeability of depth and disparity also holds for the Multi-View Stereo setup. However, it is worth noting that MVS does not typically employ rectified images. Instead, it involves multiple scene images captured from varied viewpoints that might have substantial angular differences. This is where cost volumes come into play. In Multi-View Stereo, the cost volume is a 3D volume computed as the similarity between the warped source features and the reference image feature at multiple fronto-parallel planes of the reference camera frustum. One of the benefits of using cost volumes is that they do not require rectified images, making them a more viable option than obtaining correspondences between a pair of images in a general 3D setup.

The goal of learning-based Multi-View Stereo depth estimation is to learn a predictor f_θ that can predict a depth map $\hat{D}$ that is a close approximation of the ground truth depth map D. This can be written as a minimization of Equation 9 [12]:

$$L(\mathbf{I}) = d(f_\theta(\mathbf{I}), D) \tag{9}$$

Here, $d(\cdot, \cdot)$ is any similarity metric such as $L1$, $L2$ or a classification loss between the predicted depth map $f_\theta(\mathbf{I})$ and the ground truth D. θ is the set of model parameters.

Given the continuous nature of depth values, this task is generally approached in the context of deep learning as a regression problem [13, 14, 15, 16]. Regression involves using a soft-argmin operation to generate a depth map from the 3D cost volume, with each depth hypothesis being weighted differently. This method allows for sub-pixel depth estimation by summing discrete weighted depth hypotheses. However, it can result in various combinations of weights

producing the same depth, making it harder for the model to converge and increasing the risk of overfitting the training data, thereby reducing the generalizability of the model.

This task is also approached as a classification problem by some methodologies [17, 18]. Such an approach predicts the likelihood of each depth hypothesis in the 3D cost volume, selecting the depth map corresponding to the maximum probability. The main advantage of a classification method is that it constrains the cost volume via the cross-entropy loss, which makes it more robust than regression methods. Still, there is no exact depth due to discretization.

In the following section, we will look at depth estimation in a Multi-View Stereo setup where the epipolar constraint does not hold.

2.4. Depth Estimation with Multi-view Stereo

Furukawa *et al.* [10] defined the depth prediction problem as a 1D feature matching problem along the epipolar lines. Multi-View Stereo (MVS) is a class of computer vision methods that aim to reconstruct a 3D model of a scene from multiple images taken from different viewpoints [2]. These techniques take advantage of the parallax effect, which is the apparent displacement of an object caused by a change in the observer's point of view. The greater the distance between the viewpoints, i.e., the length of the baseline, the more pronounced the parallax effect.

Depth estimation in the context of MVS involves determining the distance of each pixel (or a group of pixels) from the camera to construct a depth map, which can generate a 3D scene model. This process involves finding correspondences between different images and using epipolar geometric constraints to triangulate the position of each point. For the sake of simplicity of explanation, we use the words "images" and "features" interchangeably. However, it should be noted that state-of-the-art Multi-View Stereo methodologies employ features extracted from images to find correspondences and matching costs.

Okutomi et al. in [11] discretized the depth space and turned the task of binocular stereo depth estimation into a classification problem. The plane sweep algorithm [19] applies the same discretization principle to a Multi-View Stereo setup. We have already established in the previous sections the relationship between finding matching correspondences between images, depth/disparity estimation, and the 3D geometry of the scene. The zero-mean normalized cross-correlation (NCC) is one of the most common and successful similarity measures used in Multi-View Stereo algorithms. Before computing this similarity metric, the source image is warped to the reference camera frustum and projected across different depth levels in this frustum. This is achieved by using homography warping with the planesweep algorithm. In this section, we will dive into the planesweep algorithm and the cost volumes of the feature

matching costs that form the core of the MVS methodologies studied in this thesis.

Collins stated, "For a multi-image technique to be considered truly effective, it should be able to handle any number of images, have an algorithmic complexity of $O(n)$ in terms of the number of images used (n) and should utilize all the images in an equal manner." [19]. Figure 2 shows the plane sweep operation. This method operates under the assumption that the regions in space where multiple viewing rays of image features converge are highly probable to be the 3D positions of observed features within the scene. After the source image has been projected to the reference image frustum at multiple depth hypotheses and a particular point in space at depth z, captured by different cameras, appears similar, then that point is likely a real 3D point at depth z from the camera.

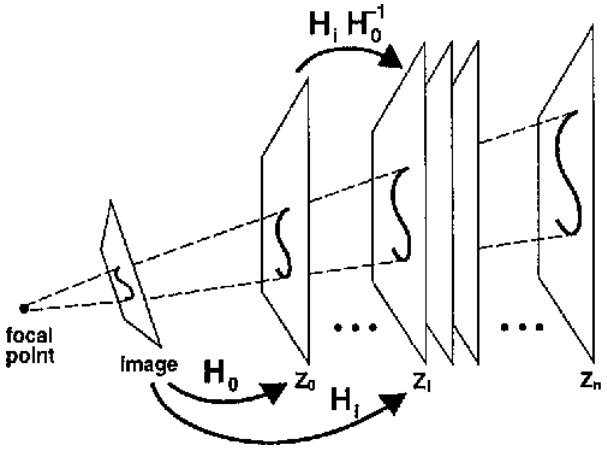


Figure 2.: Plane sweep operation. Pixels from each image are back-projected onto successive positions $Z = z_i$ of a plane that sweeps through the camera's frustum with homography H_i. H_0 is the homography that maps the image points to the plane at $Z = z_0$. By combining the two, we can say that the homography that maps pixels between the planes $Z = z_0$ and $Z = z_i$ directly, by forward projection from the z_0 plane onto the image, then back-projection to the z_i plane, can be written as $H_i {H_0}^{-1}$. This image is taken from [19]

.

The entire depth range in the reference frustum is discretized. This is a common step in many MVS algorithms and is essential for constructing the cost volume. The depth d_i at the i^{th} plane can be calculated as:

$$d_i = \mathbf{D_{min}} + i \times \frac{\mathbf{D_{max}} - \mathbf{D_{min}}}{N-1} \quad \text{for} \quad i = 0, \ldots, N-1 \tag{10}$$

Here, $[\mathbf{D_{min}}, \mathbf{D_{max}}]$ is the depth range of the scene. N is the number of depth planes in the cost volume. In the camera's frustum, each of these depth levels corresponds to a "plane"

parallel to the image plane of the camera. Then, every pixel in the reference image will have N depth hypotheses corresponding to these N planes. The task of the MVS algorithm will be to determine the most likely depth value for each pixel. Increasing N reduces the discretization errors in the results and increases the number of parameters in the network and, thus, the GPU memory. In this thesis, we use $N = 256$ for RobustMVD and $N = 128$ for MVSNet.

Once the depth has been discretized, we must align the pixels between the reference and the source images using a homography $\mathbf{x}' \ \mathbf{H_k}(d_i) \cdot \mathbf{x}$ where $\sim$ indicates equality up to scale [8, 13]. Only then can we measure the similarity between the reference and the source features. In the depth hypothesis d_i, we first project all pixels of a source feature into space with d_i and then back-project these points through the reference camera. Thus using Equation 7 and the projection equation from Figure 2, the homography between the k^{th} source image and the reference image at depth d_i can be obtained as:

$$\mathbf{H_k}(d_i) = d_i \mathbf{K_0} \mathbf{T_0} {\mathbf{T_k}}^{-1} {\mathbf{K_k}}^{-1} \tag{11}$$

where $\mathbf{K_0}$ and $\mathbf{T_0}$ are the camera intrinsics and extrinsics matrices of the reference image. [20]

After the depth discretization and the warping operation, we can now construct a cost volume. This is essentially a 3D volume where one dimension corresponds to the image's width, another to its height, and the third to the depth levels. Each voxel 3D pixel in this volume corresponds to a pixel in the reference image and a specific depth level. The voxel's value or "cost" represents how well that depth matches the observations from the other images based on some metric. Depth estimation involves finding the depth level with the minimum cost for each pixel and producing a dense depth or disparity map. The methods explored in this thesis use normalized cross-correlation as a similarity metric.

One of the baselines MVSNet does not employ any correlation but only warps the source features to the reference features and fuses the cost volumes with a variance-based strategy. RobustMVD uses the Dispnet correlation layer [21] that employs the full correlation and decimates the feature dimension. Schröppel *et al.*in the RobustMVD framework [1] improved the plane sweep homography warping described above for the pairwise cost volume construction. This is done by explicitly enforcing the epipolar constraint on the sampling points in the source image before the warping and correlation operations. The sampling points for a point on the reference image are constrained to its epipolar line in the source image at the different depth levels. This reduces the search space for matching points to a 1D search along the epipolar line. As a result of this explicit constraining, the correlation costs are computed between a more accurately warped source and reference image. Figure 3 shows the cost volume construction

operation in a typical MVS setup. All the models implemented in this thesis use this improved version of the planesweep warping operation.

In the next section, we take a look at the complete MVS pipeline and describe in detail each component of the pipeline.

2.5. Multi-view Depth Estimation Pipeline

When using deep learning for depth estimation in an MVS setting, the network is typically trained to learn these complex mappings from the input images to the feature maps, cost volumes, and a corresponding depth map guided by a suitable loss function. A common approach involves feeding the network with pairs of images and their corresponding disparity (or depth) maps during the training phase. The network learns to extract features from the images and associate the similarity between the images with the depths. This learned model can be used to predict depth maps from new pairs of images.

One significant advantage of using deep learning for this task is that networks can learn to handle a wide range of complex scenes and lighting conditions, which can be challenging for traditional MVS methods. Moreover, using convolutional layers in these networks allows them to consider the local context in the images, which can significantly improve the accuracy of depth estimation.

An example of a deep learning method for MVS is the MVSNet framework, which uses a differentiable homography warping [13] layer to aggregate information from multiple views into a cost volume using a variance-based fusion and a 3D CNN to regress the depth map from this cost volume. This framework allows for end-to-end training and can handle an arbitrary number of input views.

The standard multiview depth estimation pipeline consists of several stages that collectively generate accurate depth maps or point clouds from multiple input images. The main components of this pipeline include a feature encoder, a matching costs computation and aggregation unit, and a component to denoise the aggregated costs known as the regularization unit. The final depth is estimated from the denoised costs. We will dive into each of these components, their different flavors, and the network architectures that use these components as their basis.

1. **Input and Preprocessing:** The input to the pipeline is a set of images of a scene taken from different viewpoints (source views), with one image selected as the reference view. Input images are often preprocessed, including normalization and resizing, to ensure that they are in a suitable format for a neural network. To maintain the integrity of our

experiments, we have fixed the input sizes and the sizes of the evaluation data. If a model is trained with smaller inputs due to memory constraints on the GPU, it is explicitly stated. All sizes are multiples of 32. This addresses the size mismatch in spatial dimensions caused by strided convolutional operations on input tensors and skip connections to a layer.

2. **Feature Extraction:** In this stage, a Convolutional Neural Network (CNN) is used to extract feature maps from input images. The feature extractor scans the images and learns to identify patterns and structures that are important for depth estimation, such as edges, corners, and textures. This process reduces the high-dimensional image data to a more manageable, lower-dimensional form that still contains the information necessary for depth estimation. In this thesis, we examine the feature encoders of MVSNet and DispNet. We also explore the effects of using a multiscale architecture such as a UNet [22] for feature extraction and also look at fusing the learned features with features generated by a pre-trained ViT such as DINO [23].

3. **Cost Volume Construction:** To find correspondences between different views, a cost volume is often constructed. The cost volume effectively represents the likelihood that each pixel in the reference image has a certain depth. We described the construction of a pairwise cost volume using different correlation types and homography warping with the plane sweep algorithm in detail in Section 2.4 and Section 2.3. The individual pairwise cost volumes of the reference and source features are fused using different strategies to obtain the aggregated cost volume. The learned fusion involves learning weights for each pairwise cost volume using a 2D or 3D CNN. Views with more occluded points are given less weight in the aggregated cost volume. Average fusion simply fuses all pairwise volumes equally. Variance-based fusion, which is used in MVSNet [13], compares the variance of the warped source features and the reference feature to fuse the warped volumes. In this thesis, we examine all three fusion strategies.

4. **Cost Volume Processing and Depth Map Estimation:** The cost volume is typically processed using a 3D CNN [13, 17, 15, 16], or RNN [18] that aggregates and regularizes information across different views and depths to robustly estimate the depth map of the reference view. This is an essential step due to cost volumes being inherently noisy due to occlusions, poor texture and lighting, and non-Lambertian surfaces. Some techniques use a coarse-to-fine pattern to better estimate the depth and reduce memory consumption. The first level of the network predicts a coarse depth map. This is used as input for subsequent levels to yield finer results. Multiscale feature extractors are used to obtain

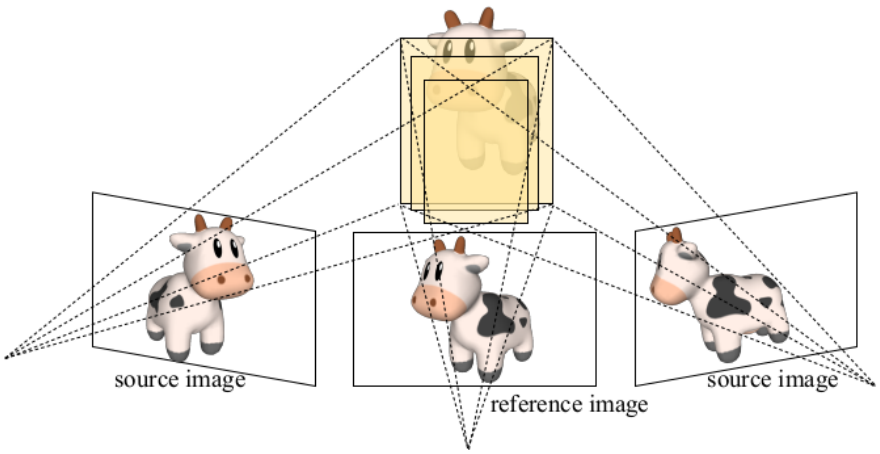


Figure 3.: Cost volume construction. Using homography warping with a planesweep volume along the reference camera frustum and computing the matching costs at the different depth planes. This image is taken from [20].

features at different resolutions. The low-resolution features are used at the coarse level, and the high-resolution features are used as input to the finer levels along with the coarse depth map from the previous level. There are also different implementations of this strategy. Some methods simply use cascade fusion, such as CasMVSNet and CVP-MVSNet[15, 16] while others, such as Vis-MVSNet [17], also use the uncertainty map to weigh depth residuals. In this thesis, we explore regularization using a regular 2D UNet for RobustMVD and 3D UNet for MVSNet. We also look at cascade warping and implement and experiment with a coarse-to-fine pattern for both these baselines. The coarse-to-fine architecture is shown in Figure 4

5. **Refinement:** The initial depth map from the previous stage is often coarse and may contain inaccuracies. Therefore, it is common to refine the depth map using various methods. This could involve an additional CNN that takes the initial depth map and the reference image as input and learns to predict corrections to the depth values. The network can be trained to minimize the difference between its predictions and ground truth depth maps, if available. In this thesis, we only look at the fundamental elements of the MVS pipeline discussed previously and their interaction. We do not look at the effects of additional image-refinement strategies.

6. **Training:** The entire pipeline can be trained from beginning to end using supervised learning, as ground-truth depth maps are available. If not, unsupervised or self-supervised

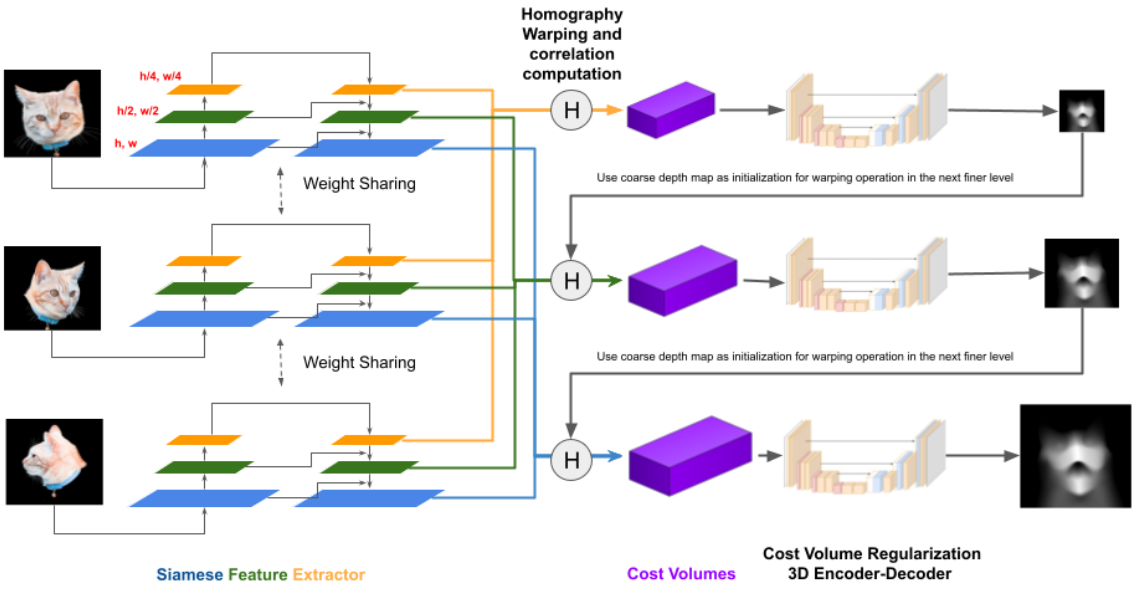


Figure 4.: Coarse-to-fine pattern. This figure shows a typical coarse-to-fine design. Cost volumes at finer levels are constructed using the coarser depth predictions as initialization. In this figure, a single 3D UNet is used as the regularization network. 3D UNets in a stacked hourglass configuration [24] are also used to regularize raw cost volumes (in purple). The cost volumes at finer levels have a smaller depth range centered around the previous coarse depth predictions and are coupled with an adaptive subdivision of depth intervals for finer levels. Multilevel features with different spatial resolutions and channel depth are used to construct the cost volumes and depth maps at different levels. A multilevel feature extractor, such as a feature pyramid network or a 2D UNet, can be used for this purpose.

methods can be used, where the training signal comes from photometric consistency or other cues. All experiments in this thesis are performed in a supervised manner, with the correlation between the source and reference images as a signal to guide the entire process.

7. **Testing and Evaluation:** Once the pipeline is trained, it can be used to estimate depth maps from new multiview image sets. The datasets used to evaluate and train networks are discussed in detail in Section 4.2. The quality of the depth maps can be evaluated using various metrics, such as mean absolute error or mean squared error, if ground truth is available. The metrics are discussed in Section 4.3. Visual inspection and downstream tasks can also be used for evaluation.

There are several versions of this pipeline that are tailored to different tasks, data availability,

and current research advancements. Some of the recent approaches suggest the inclusion of transformer architectures, attention mechanisms, and other advanced techniques such as coarse-to-fine architecture, feature pyramids, and pre-trained vision transformers as feature extractors in the pipeline to enhance the accuracy of depth estimation. In the following chapter, we will conduct a comprehensive evaluation of these methodologies.

3. Related Work

This chapter contains a comprehensive overview of the literature on the application of deep learning-driven Multi-View Stereo methods to the problem of Depth Estimation. A typical setup to tackle this problem involves taking multiple perspectives of a scene from different viewpoints. These images are then used as input to compute a depth map of the scene. Traditionally, the depth estimation process with Multi-View Stereo is decomposed into three important steps: extraction of features from input images, aggregation of matching costs from features, and prediction of depth from aggregated costs. We divide our literature review into sections as per these three steps.

In Section 2.1, we examine the relevant literature for feature extraction. In Section 2.2, we explore the state of the art in the domain of disparity estimation and cost volume construction, which is a fundamental task for any stereo methodology. Finally, in Section 2.3, we present the state of the art in the domain of deep learning-driven Multi-View Stereo methods for depth estimation. For the purpose of this review, we use the terms "disparity" and "depth" interchangeably. Depth is easily obtained as it is inversely proportional to disparity. Both depth and disparity estimation employ similar methodologies.

3.1. Feature Extraction

This is a crucial step in enhancing the fidelity of depth estimation and ensuring the robustness of the correspondences for computing the matching costs between views. Feature descriptors ensure better correspondences compared to RGB color images, particularly in regions that lack definitive texture or have repetitive patterns, and also support photometric and geometric consistency across the different views.

Classical methodologies such as Scale-Invariant Feature Transform (SIFT) [25], Speeded-Up Robust Features (SURF) [26], Oriented FAST and Rotated BRIEF (ORB) [27] and neural network architectures involving convolutional neural networks (CNN) such as UNets [22], VGGs [28], ResNets [29] are employed to discern and characterize salient regions within images. Zbontar and LeCun [30] introduce a deep Siamese network to compute matching costs.

Feature pyramid networks (FPN) [31] are popular feature extractors that produce multiscale features that are used for downstream MVS tasks. Chang *et al.* in PSMNet [32] propose a pyramid stereo matching network to take advantage of global context information using spatial pyramid pooling (SPP) [33] and dilated convolutions [34] to expand receptive fields. Recently, the use of Visual Transformers (ViT) [4, 35, 5, 6] for feature extraction gives superior results compared to the use of only FPNs or UNet. In this thesis, we implement the FPN and UNet feature extractors. We also use a pretrained DINO ViT to augment the learned features.

3.2. Cost Volumes and Disparity Estimation

Disparity estimation is most commonly associated with stereo vision, where two cameras capture images of the same scene from slightly different viewpoints. Disparity represents the horizontal difference in the x coordinates of congruent pixels in rectified images. Image rectification simplifies the problem of calculating the 3D point coordinates in the scene to a 1D search on the same epipolar line via correspondences between two views. After the correspondences are identified between the views, the disparities are calculated methodically and transmuted into depth maps. Instrumental to this computation is the cost volume, which is a 3D or 4D data structure encapsulating matching costs that represent the similarity of a pixel's appearance in one image to its counterpart in another image across potential disparities or depths. These cost volumes, especially in contemporary deep learning-driven MVS algorithms, serve as intermediate representations, enabling neural networks to be trained for disparity or depth refinement. In an MVS setting, rectifying image pairs is not feasible because of the large angles between the images. The advantage of cost volumes is that they do not require rectified images. However, raw cost volumes contain a lot of noise due to the presence of occluded points, smoothness, reflections, noise, etc. Before regressing a depth or disparity map from these cost volumes, it, therefore, becomes necessary to perform a regularization step on them. This regularization step is performed using 3D or 2D CNN-based network architectures. We now look at the state-of-the-art in cost volume construction and regularization.

SGM [36] seeks to find correspondences and estimate the disparity by minimizing an energy function that combines both data and smoothness terms. Dispnet [21] was the first end-to-end deep learning approach to combine feature extraction with disparity estimation. RobustMVD uses the components of the DispNet architecture, mainly the correlation layer described in the article, to obtain a similarity cost between the left and right images. This correlation is based on a dot product that decimates the feature dimension from the cost volume. Pang *et al.* [37] extend DispNet by introducing a two-stage network called cascade residual learning (CRL).

The final disparity map is the sum of the disparity maps and their multiscale residuals from the first and second stages.

GC-Net [38] proposed a *left-right feature concatenation* based 4D cost volume that also incorporates the feature dimension and used 3D convolutions to regularize the same. Similarly, DPSNet [39] also concatenates the features pairwise instead of using a similarity metric and fuses the different cost volumes simply by averaging them. Sun *et al.* in PWC-net [40] was the first method to use pyramidal processing, warping, and cost volumes to estimate optical flow in an end-to-end manner. These principles and network architecture are extensively replicated in Multi-View Stereo. HSMNet [41] uses Volumetric Pyramid Pooling, which is an extension of SPP to decode the 4D cost volume. Guo *et al.* in GWC-Net[42] used Groupwise correlation to form the cost volume instead of a fused representation. They also used stacked UNets, also known as a "Stacked Hourglass Configuration" to regularize the cost volume. This was first used in PSMNet [32] to learn better context features. CFNet by Shen *et al.* [43] used both fused and cascade cost volume representations to address domain differences across different datasets. GANet [44] replaces the 3D CNN regularization with a Semi-Global Guided Aggregation (SGA) and Local Guided Aggregation (LGA) layer to reduce memory requirements. AANet [45] used similar principles to aggregate the cost volumes by using Intra-Scale Aggregation (ISA) and Cross-Scale Aggregation (CSA). UCS-Net [14] used multiple small volumes called Adaptive Thin Volumes (ATV) instead of a large plane sweep volume to regress a depth map in a coarse-to-fine manner. In this thesis, we extend the Groupwise correlation to the Multi-View Stereo scenario and also implement the stacked hourglass cost volume regularization network. We also explore the effects of Groupwise correlation in a multi-correlation layer instead of warp-only correlation and full correlation.

3.3. Multi-View Stereo

Multi-view stereo involves deducing the 3D structure of a viewed scene using multiple images, each with known intrinsic and extrinsic camera parameters[10]. Structure-from-motion (SfM) algorithms[46] are used to obtain these parameters. Conventional MVS algorithms focus on finding correspondences between reference and source images by utilizing plane sweep volumes and optimizing for photometric and geometric consistency. More advanced approaches, such as COLMAP[47], compute visibility information and aggregate pairwise correlations based on a probabilistic framework, where visibility and depth are alternately updated in an iterative manner. However, approaches such as SurfaceNet[48], LSM[49], and RayNet[50] use a volumetric approach to estimate the geometry of the scene rather than computing a plane

sweep volume. DeepMVS[51] was the first method to use deep neural networks for MVS. This approach aligns the reference view with the source views using a correlation layer. This layer selects patches from the source images based on potential depth values and contrasts them with patches from the reference image. Matching attributes from different views are then fused via max-pooling.

In comparison, MVSNet[13] uses a methodology in which the source views and the reference view are compared within a learned feature environment, using the variance of the warped source features with the reference features to create the aggregated cost volume. Subsequent innovations have further refined this methodology. For instance, R-MVSNet[18] enhances computational efficiency at the expense of a longer training time by employing recurrent processing of depth slices in the cost volume. Both CVP-MVSNet[16] and CAS-MVSNet[15] implement correlation sequentially in a cascaded manner using a coarse-to-fine approach to refine the coarse depth map and optimize computational resources and support higher resolution outputs. PVSNet [52] introduced the two-view pixel-wise visibility-aware cost volume construction and aggregation and an anti-noise training strategy of using non-matching views to train the network. Meanwhile, Vis-MVSNet[17] also uses this coarse-to-fine strategy to improve the initial depth map while using the predicted uncertainty map of the scene to weigh the pixels of the predicted depth map, which is used to calculate the cost volume in the finer levels of the pipeline, thus accounting for pixel-wise visibility. All these methods use either a classification or regression approach to obtain the depth. Peng *et al.* in UniMVSNet [53] introduce a novel representation of costs called Unification and the Unified Focal Loss to unify the classification advantage of constraining the cost volume and the regression advantage of obtaining sub-pixel depth prediction.

Most models require both the minimum and maximum depth values of the scene being viewed and deliver depth predictions within this specified range. Scale Augmentation [1] introduced in the RobustMVD framework can be used to train the networks to be scale-agnostic and thus demonstrate better generalization results on a range of data modalities.

It is a significant undertaking to incorporate, train, and evaluate models that utilize all of these methodologies, some of which are beyond the scope of this thesis. This thesis focuses on implementing select methodologies alongside our baseline models as individual components to analyze their impact on network performance. Now that we have discussed the state-of-the-art, the next chapter will cover the experimental setup, baselines, datasets, and evaluation metrics for this thesis.

4. Setup

The main goal of this thesis is to analyze the effects of using different variations of architectural components in existing MVS models. To this end, we developed the different components in the Robust-MVD framework to be interoperable and interchangeable. Each new "combination" of architectural components represents a new model, which is first trained and then evaluated under the Robust-MVD framework. We also experiment with data augmentations and data and training setup parameters such as the number of source views, view selection strategy, number of groups in Groupwise correlation, etc. In this chapter, we first discuss the experimental setup that includes the baseline models, followed by the training and evaluation procedure, the datasets, metrics, the hardware, and the ablation study protocol.

4.1. Baseline Models

Within the Robust-MVD framework, you can find the Robust MVD Baseline model [1] designed for multiview depth estimation. This model takes advantage of existing components from FlowNet and DispNet but distinguishes itself by incorporating a unique Scale Augmentation procedure. We consider the seminal MVSNet[13] as a baseline. We also choose Vis-MVSNet[17] as a baseline.

1. **MVSNet:** As a baseline, we incorporate MVSNet, a seminal deep learning-based model for multiview depth estimation. MVSNet employs a differentiable homography-based plane sweep operation, which allows it to form a cost volume with discrete depth values. This model introduces the concept of variance-based cost volume aggregation, which is crucial for handling inherent noise and redundancy in cost volumes. This cost volume fusion methodology is later used in CasMVSNet, CVP-MVSNet, VisMVSNet, and many others with some variations. All MVSNet derivatives and the baseline are trained with depth information from the scene. Table 2 shows the detailed architecture of MVSNet.

2. **RobustMVD:** Building on the principles of MVSNet, the implementation and training routine of RobustMVD extends it by adding scale-consistent depth prediction and uncer-

tainty estimation capabilities. RobustMVD uses components from DispNet, which is an end-to-end convolutional neural network designed to learn disparity values from stereo images. It is important to note that the parent network of RobustMVD, i.e. DispNet, is used to estimate the disparity between a pair of Stereo Images. Consequently, RobustMVD also builds the reference image frustum for the planesweep warping operation in the linear inverse depth space, unlike MVSNet and our other baseline Vis-MVSNet, which sample the depth planes in the linear depth space. All RobustMVD based derivatives and the baseline are trained in a fixed depth setting of $[0.4, 1000]$ meters and with scale augmentation. Table 3 shows the detailed architecture of RobustMVD.

3. **Vis-MVSNet:** We also choose Vis-MVSNet as a baseline model, as it implements features of previous networks such as CasMVSNet [15] and CVP-MVSNet [16] mainly the coarse-to-fine pattern, while also incorporating pixel-wise visibility in its estimations.

Table 1 is a taxonomy of some selected MVS methodologies. We have included the baseline models and other methods that laid the foundation for the current state-of-the-art. This chart illustrates the primary elements of a typical MVS pipeline. The rows feature various models and the distinct types of components incorporated within them.

Architecture	**Feature Extraction**	**Correlation Layer**	**Cost Volume Fusion**	**Cost Volume Regularization**	**Coarse-to-fine Refinement**	**Loss**
MVSNet	Siamese Encoder	Warp Only	Variance	3D UNet	No	Regression
RobustMVD	Siamese Encoder	Full	Learned Weights Average	2D UNet	No	Classification
Vis-MVSNet	Siamese UNet	Groupwise	Visibility Weighted Average	Stacked Hourglass	Yes	Classification
CVP-MVSNet	Siamese FPN	Warp Only	Variance	3D UNet	Yes	Regression
CasMVSNet	Siamese FPN	Warp Only	Variance	3D UNet	Yes	Regression
GWC-Net	Siamese UNet	Groupwise	Concatenation	Stacked Hourglass	No	Regression
MVSFormer	Siamese FPN + Pretrained ViT	Groupwise	Visibility weighted variance	3D UNet	Yes	Regression + Classification
R-MVSNet	Siamese Encoder	Warp Only	Variance	RNN	No	Classification

Table 1.: Comparison of MVS architectures. This table highlights the diversity of components and modularization that is possible due to the clear-cut separation of concerns of the different architectural components in an MVS pipeline. It must be noted that while GWC-Net is a stereo method, it can be implemented as an Multi-View Stereo method similar to how RobustMVD is implemented as the MVS extension of DispNet.

Operation	Kernel	Stride	Ch I/O	InpRes	OutRes	Input	Output
(1) Siamese encoder							
for each view $i = 0, .., k$:							
2D convolution + BatchNorm + ReLU	3×3	1	3/8	768×576	768×576	Image I_i	$conv1a_i$
2D convolution + BatchNorm + ReLU	3×3	1	8/8	768×576	768×576	$conv1a_i$	$conv1b_i$
2D convolution + BatchNorm + ReLU	5×5	2	8/16	768×576	384×288	$conv1b_i$	$conv2a_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	384×288	384×288	$conv2a_i$	$conv2b_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	384×288	384×288	$conv2b_i$	$conv2c_i$
2D convolution + BatchNorm + ReLU	5×5	2	16/32	384×288	192×144	$conv2c_i$	$conv3a_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	192×144	192×144	$conv3a_i$	$conv3b_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	192×144	192×144	$conv3b_i$	$conv3c_i$
2D convolution	1×1	1	32/32	192×144	192×144	$conv3c_i$	$conv3r_i$
(2) Plane sweep warping							
for each source view $i = 1, .., k$:							
warp view 0 and i	-	-	32/32	192×144	D×192×144	$conv3r_0$, $conv3r_i$	cost-volume C_i
3) Cost-volume fusion							
for each source view $i = 1, .., k$:							
fuse to cost-volume **C**							
$\mathbf{C} = \frac{\sum_{i=1}^{k} (\mathbf{C}_i - \overline{\mathbf{C}}_i)^2}{k}$ [13]	-	-	32/32	D×192×144	D×192×144	C_i , $conv3r_0$	**C**
(4) Cost-volume Regularization							
3D convolution + BatchNorm + ReLU	3×3×3	1	32/8	D×192×144	D×192×144	**C**	$3Dconv0$
3D convolution + BatchNorm + ReLU	3×3×3	2	8/16	D×192×144	$D/2$×96×72	$3Dconv0$	$3Dconv1$
3D convolution + BatchNorm + ReLU	3×3×3	1	16/16	$D/2$×96×72	$D/2$×96×72	$3Dconv1$	$3Dconv2$
3D convolution + BatchNorm + ReLU	3×3×3	2	16/32	$D/2$×96×72	$D/4$×48×36	$3Dconv2$	$3Dconv3$
3D convolution + BatchNorm + ReLU	3×3×3	1	32/32	$D/4$×48×36	$D/4$×48×36	$3Dconv3$	$3Dconv4$
3D convolution + BatchNorm + ReLU	3×3×3	2	32/64	$D/4$×48×36	$D/8$×24×18	$3Dconv4$	$3Dconv5$
3D convolution + BatchNorm + ReLU	3×3×3	1	64/64	$D/8$×24×18	$D/8$×24×18	$3Dconv5$	$3Dconv6$
Transposed 3D convolution + ReLU	3×3×3	2	64/32	$D/8$×24×18	$D/4$×48×36	$3Dconv6$	$3Dconv7$
Transposed 3D convolution + ReLU	3×3×3	2	32/16	$D/4$×48×36	$D/2$×96×72	$3Dconv7 + 3Dconv4$	$3Dconv9$
Transposed 3D convolution + ReLU	3×3×3	2	16/8	$D/2$×96×72	D×192×144	$3Dconv9 + 3Dconv2$	$3Dconv11$
3D convolution	3×3×3	1	8/1	D×192×144	D×192×144	$3Dconv11 + 3Dconv0$	$prob$

Table 2.: Architecture of the MVSNet Baseline model. The notation is the same as in the original MVSNet implementation [13]. The model takes views $V = (V_0, \ldots, V_k)$ as input where V_0 is the keyview with an image I_0 and $V_{1,\ldots,k}$ are source views with images $I_{i,\ldots,k}$. D is the number of candidate depth values in the plane sweep correlation layer ($D = 256$ for the original paper and $D = 128$ for our comparisons). $\overline{\mathbf{C}}_i$ is the average volume among all feature volumes, and all of the operations above are element-wise[13]. The resolutions are indicated for the training settings and differ for test inputs.

Operation	Kernel	Stride	Ch I/O	InpRes	OutRes	Input	Output
(1) Siamese encoder							
for each view $i = 0, .., k$:							
convolution + LeakyReLU	7×7	2	3/64	768×384	384×192	Image I_i	$conv1_i$
convolution + LeakyReLU	5×5	2	64/128	384×192	192×96	$conv1_i$	$conv2_i$
convolution + LeakyReLU	3×3	2	128/256	192×96	96×48	$conv2_i$	$conv3a_i$
(2) Context encoder							
convolution	1×1	1	256/32	96×48	96×48	$conv3a_0$	ctx
(3) Plane sweep correlation							
for each source view $i = 1, .., k$:							
correlate view 0 and i	-	-	256/D = 256	96×48	96×48	$conv3a_0$, $conv3a_i$	cost-volume C_i
4) Cost-volume fusion							
4.1 for each source view $i = 1, .., k$:							
convolution	3×3	1	256/128	96×48	96×48	C_i	$convf1_i$
convolution	3×3	1	128/1	96×48	96×48	$convf1_i$	weight w_i
4.2 fuse to cost-volume **C**							
$\mathbf{C} = (\Sigma exp(w_i).C_i)/(\Sigma exp(w_i))$	-	-	256/256	96×48	96×48	C_i , w_i	**C**
(5) Cost-volume Regularization							
convolution + LeakyReLU	3×3	1	288/256	96×48	96×48	$\mathbf{C} + ctx$	$conv3b$
convolution + LeakyReLU	3×3	2	256/512	96×48	48×24	$conv3b$	$conv4a$
convolution + LeakyReLU	3×3	1	512/512	48×24	48×24	$conv4a$	$conv4b$
convolution + LeakyReLU	3×3	2	512/512	48×24	24×12	$conv4b$	$conv5a$
convolution + LeakyReLU	3×3	1	512/512	24×12	24×12	$conv5a$	$conv5b$
convolution + LeakyReLU	3×3	2	512/1024	24×12	12×6	$conv5b$	$conv6a$
convolution + LeakyReLU	3×3	1	1024/1024	12×6	12×6	$conv6a$	$conv6b$
convolution + ReLUAndSigmoid	3×3	1	1024/2	12×6	12×6	$conv6b$	$pred6$
transposed convolution + LeakyReLU	4×4	2	1024/512	12×6	24×12	$conv6b$	$upconv5$
convolution + LeakyReLU	3×3	1	1025/512	24×12	24×12	$upconv5 + pr6 + conv5b$	$iconv5$
convolution + ReLUAndSigmoid	3×3	1	512/2	24×12	24×12	$iconv5$	$pred5$
transposed convolution	4×4	2	512/256	24×12	48×24	$iconv5$	$upconv4$
convolution + LeakyReLU	3×3	1	769/256	48×24	48×24	$upconv4 + pr5 + conv4b$	$iconv4$
convolution + ReLUAndSigmoid	3×3	1	256/2	48×24	48×24	$iconv4$	$pred4$
transposed convolution + LeakyReLU	4×4	2	256/128	48×24	96×48	$iconv4$	$upconv3$
convolution + LeakyReLU	3×3	1	385/128	96×48	96×48	$upconv3 + pr4 + conv3b$	$iconv3$
convolution + ReLUAndSigmoid	3×3	1	128/2	96×48	96×48	$iconv3$	$pred3$
transposed convolution	4×4	2	128/64	96×48	192×96	$iconv3$	$upconv2$
convolution + LeakyReLU	3×3	1	193/64	192×96	192×96	$upconv2 + pr3 + conv2_0$	$iconv2$
convolution + ReLUAndSigmoid	3×3	1	64/2	192×96	192×96	$iconv2$	$pred2$
transposed convolution	4×4	2	64/32	192×96	384×192	$iconv2$	$upconv1$
convolution + LeakyReLU	3×3	1	97/32	384×192	384×192	$upconv1 + pr2 + conv1_0$	$iconv1$
convolution + ReLUAndSigmoid	3×3	1	32/2	384×192	384×192	$iconv1$	$pred1$

Table 3.: Architecture of the Robust MVD Baseline model. The notation is the same as in the DispNet paper [21]. The model takes views $V = (V_0, \ldots, V_k)$ as inputs where V_0 is the key view with an image I_0 and $V_{1,\ldots,k}$ are source views with images $I_{i,\ldots,k}$. D is the number of candidate inverse depth values in the plane sweep correlation layer (here: $D = 256$). The resolutions are indicated for training settings and differ for test inputs. This table is adapted from [1].

4.2. MVS Datasets

For our experiments, we use BlendedMVS for training the models. For evaluating the models we use KITTI, DTU, ScanNet, Tanks and Temples and ETH3D.

1. **BlendedMVS**[54]: This is a large-scale synthetic Multi-View Stereo dataset tailored explicitly for deep learning applications. This dataset addresses the lack of extensive training data for learning-based MVS. Provides high-quality textured meshes from images of various scenes. These meshes are then rendered to color images and depth maps and further blended with input images to generate the training input. The data set covers more than 17k high-resolution images, covering various scenarios such as cities, architecture, sculptures, and small objects, depicting 106 training and seven validation scenes. It comprises synthetic images blended with real background textures, providing a broad range of photorealistic scenes. Being synthetic, the camera calibration is accurate enough for the purpose of MVS training.

2. **DTU**[55, 56]: The DTU Robot Image Data set is a comprehensive dataset that contains 124 scenes of various objects. The data was collected using an industrial robot-mounted structured light scanner. The scenes include a wide range of objects that were captured under different lighting conditions, enhancing the variability and complexity of the dataset. The dataset comprises 128 scans with 49 views captured in seven distinct lighting conditions. These scans are divided into 79 for training, 18 for validation, and 22 for evaluation. The large variety of scene types makes this dataset a challenging but rewarding choice for training and evaluating depth estimation models. It is a perfect dataset for the use case of 3D reconstruction

3. **KITTI**[57, 58]: Developed on an autonomous driving platform, KITTI presents real-world computer vision benchmarks. It comprises tasks like stereo imaging, optical flow, visual odometry, 3D object detection, and 3D tracking. The dataset was captured using high-resolution color and grayscale video cameras in the city of Karlsruhe, Germany, in adjoining rural areas and on the highways. It includes LIDAR point clouds, GPS, and IMU data, making it one of the richest datasets for vehicle-based scene understanding in real-world settings with varying illumination conditions, large dynamic ranges, and various occlusions.

4. **ETH3D**[59]: This dataset addresses the limitations of existing multiview stereo benchmarks. ETH3D contains a mix of indoor and outdoor scenes captured using a high-precision laser scanner, high-resolution DSLR images, and synchronized low-resolution

stereo videos. It contains 25 high-resolution scenes and ten low-resolution scenes. It covers various viewpoints and scene types, from natural scenes to man-made indoor and outdoor environments. The high temporal and spatial resolution of the data set makes it particularly valuable for real-world applications. Considering MVS with deep learning, ETH3D is widely acknowledged as the most difficult MVS task, containing many low-textured regions such as white walls and reflective floors. This is also reflected in our evaluations, where the models perform relatively poorly on ETH3D compared to their evaluation performance on the other test datasets.

5. **Tanks and Temples**[60]: This benchmark dataset for image-based 3D reconstruction was curated under realistic conditions with various indoor and outdoor scenes. The ground truth data for this dataset were generated using an industrial laser scanner, capturing diverse viewpoints and scene types. The challenge lies in the intricate structures it presents, making it a crucial dataset for training and testing depth estimation models.

6. **ScanNet**[61]: This RGB-D video dataset contains 2.5 million views in over 1500 scans. It includes annotations with 3D camera poses, surface reconstructions, and instance-level semantic segmentations. ScanNet's breadth of views and annotation depth support progress in various 3D scene understanding tasks, making it a versatile choice for model evaluation.

4.2.1. Training Dataset Variants

For training the different models in our experiments, we define two variants of the BlendedMVS dataset. These variants differ in the number of views and the view selection strategy.

1. **BlendedMVS-MVSNet variant**: This variant of the BlendedMVS dataset contains samples that have two source views and one key view. Each view in a scene becomes the key view and the two source views are chosen that have the highest matching scores to the key view. These are selected using matching scores obtained from COLMAP[47] This variant is as per the setting of the original MVSNet paper [13]. This variant contains 16094 training samples in total.

2. **BlendedMVS-RMVD variant**: This variant of the BlendedMVS dataset contains samples that have four source views and one key view. Each view in a scene becomes the key view and the four source views are chosen randomly from the remaining views. This variant is as per the setting of the original RobustMVD paper [1]. This variant contains 1774920 training samples in total.

4.2.2. Test Dataset Parameters

A concise comparison of the different attributes of the evaluation datasets is provided in Table 4.

For evaluating the trained models, we use the following settings for the number of views, the view selection strategy, and the spatial resolutions of the images in the test datasets:

- **Number of source views** V: This is fixed to 2. We do this to limit the computational requirements during inference.
- **View selection strategy**: We selected the two best-matching source images from the images available in the scene after the reference image was chosen.
- **Image resolutions for test datasets**: We used the following image resolutions to evaluate the trained models. We have two sets of input resolutions. The first is as follows.
  1. **KITTI**: $H = 384$ and $W = 1280$
  2. **DTU**: $H = 896$ and $W = 1216$
  3. **ScanNet**: $H = 448$ and $W = 640$
  4. **Tanks and Temples**: $H = 704$ and $W = 1280$
  5. **ETH3D**: $H = 768$ and $W = 1152$

  We use these resolutions for comparatively smaller models that leave sufficient memory on the P100 for the input data and the intermediary tensors and their operations. The second set of resolutions is used to evaluate models with a significantly larger footprint that do not fit on the P100 along with the data. This is as follows.
  1. **KITTI**: $H = 384$ and $W = 1280$
  2. **DTU**: $H = 672$ and $W = 960$
  3. **ScanNet**: $H = 448$ and $W = 640$
  4. **Tanks and Temples**: $H = 576$ and $W = 960$
  5. **ETH3D**: $H = 576$ and $W = 864$

  In the results tables presented in the next chapter, we mark the models evaluated using these smaller resolution settings with a $\star$. We can see here that the spatial resolutions for DTU, ETH3D, and Tanks and Temples are 0.75 times the resolutions in the normal setting. The resolutions for KITTI and ScanNet are retained.

Test set	**KITTI [57, 58]**	**ScanNet [61]**	**ETH3D [59]**	**DTU [55, 56]**	**T&T [60]**
Domain	driving	indoor	in- & outdoor	tabletop	in- & outdoor
Setting	DFV	DFV	MVS	MVS	MVS
Cam Motion	small	small	large	small	small
Scene Scale	$2-85\,\mathrm{m}$	$0.2-9\,\mathrm{m}$	$0.3-60\,\mathrm{m}$	$0.4-1.2\,\mathrm{m}$	$1.1-42\,\mathrm{m}$
Split based on	test split from Eigen *et al.* [62]	test split from Tang and Tan [63]	orig. train split	val. split from Yao *et al.* [13]	orig. train split
Full Res	1226x370	640x480	6048x4032	1600x1200	1962x1092
# Samples	93	200	104	110	69

Table 4.: Test sets of the Robust MVD Benchmark are based on KITTI, ScanNet, ETH3D, DTU, and Tanks and Temples (T&T). These datasets are common for depth-from-video (DFV) or Multi-View Stereo (MVS) and cover different domains and scene scales. This table is adapted from [1]

4.3. Metrics

- **Absolute Relative Error (AbsRel)**[62, 58, 1]: AbsRel is a widely used metric for depth estimation tasks. It computes the average of the absolute difference between the predicted and the ground truth depth values, normalized by the ground truth. By using absolute values, we ensure that the error is always positive, disregarding whether the estimated depth was too shallow or too deep compared to the ground truth. By taking the relative error, we account for the fact that depth perception is usually logarithmic in nature, meaning that small inaccuracies at close range are often as perceptually significant as large inaccuracies at long range. This metric, therefore, offers a balanced measure of the model's depth estimation performance, irrespective of the depth range. A lower AbsRel implies a more accurate depth estimation model. In the results section of this report, AbsRel is indicated by *rel*. Absrel is calculated as:

$$rel = 100 \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} \frac{|z_{i,j} - z^*_{i,j}|}{z^*_{i,j}} \tag{12}$$

  For each of the m pixels with a valid depth for the ground truth, we use the index j. The index i is used for the n samples in the test set.

- **Inlier Ratio** [62, 58, 1]: The Inlier ratio refers to the proportion of data points (or pixels, in the case of images) that fit well with the estimated model, often determined using a

predefined threshold. For depth estimation, it is particularly useful for understanding the proportion of the depth map that the model accurately predicts. A high Inlier ratio suggests that the model is well-fitted to the data and performs satisfactorily. It can also indicate how robust the model is against outliers and how well it generalizes to different environments. For the purpose of this thesis, we use a threshold of 1.03. In the results section of this report, the inlier ratio is indicated by τ. The inlier ratio is given by:

$$\tau = 100 \cdot \frac{1}{n} \sum_{i=1}^{n} \frac{1}{m} \sum_{j=1}^{m} [max\left(\frac{z_{i,j}}{z^*_{i,j}}, \frac{z^*_{i,j}}{z_{i,j}}\right) < 1.03] \tag{13}$$

The notation is the same as in the Absrel equation. $[\cdot]$ denotes the Iverson bracket. The Inlier Ratio represents the percentage of pixels that have accurate predictions. A prediction is deemed correct if it has an error below 3%.

- **Area Under the Sparsification Error Curve (AUSE)** [7]: Uncertainty estimation is a crucial aspect of any predictive model, providing insight into the model's confidence in its predictions. For depth estimation, uncertainty can indicate regions in the scene where depth is difficult to estimate because of factors such as textureless regions, occlusions, or specular reflections. The AUSE metric measures the calibration of predicted uncertainties, i.e., whether high uncertainty predictions correspond to high errors and low uncertainty predictions to low errors.

  Sparsification involves retaining only a portion of the predicted pixels, based on the model's estimated uncertainty. As fewer predictions are retained (i.e., greater sparsification), ideally only keeping the most accurate predictions, the error of the retained predictions is the sparsification error. The plot of the incorrect predictions against the fraction of predictions retained is known as a sparsification plot. The area under this curve (AUSE) informs us about how well the model's confidence aligns with its accuracy.

  A well-calibrated model, scoring low on the AUSE, can accurately predict the depth and its own performance on each prediction. Such a model would be highly valuable in real-world applications, where understanding the reliability of predictions is critical for safe and effective decision making. A value of 0 for AUSE is considered ideal as it signifies a precise match between estimated uncertainties and real errors.

- **Space Complexity**: This refers to the memory required to store the model and its parameters during training. The space complexity of depth estimation models is largely determined by the complexity of the model's architecture, the resolution of the input

images, and the number of views. Deep learning models, in particular, often have millions of parameters, which can take up a lot of memory. Therefore, models with high space complexity may not be feasible on devices with limited memory resources. MVS methods typically have high space complexity due to the necessity to store the cost volume. The dimensions of this 4D tensor can be immense depending on the number of views, the resolution of the images, and the number of possible depth levels considered. In addition, storing intermediate results for each pair of images also contributes to the overall space complexity. This makes it crucial to explore more memory-efficient representations or processing mechanisms, such as sparse cost volumes or cost volume compression.

- **Inference Time**: This measures how long it takes the model to generate output given a new input after it has been trained. For multiview depth estimation, this could depend on factors such as the number of views, the resolution of the images, the generation of cost volume, its aggregation, and finally, the depth map estimation that contributes to the overall inference time. Inference efficiency is especially important for applications that require near-real-time depth estimation, such as robotics or autonomous vehicles. Fast inference time is critical for applications that require real-time depth estimation, such as autonomous vehicles or augmented reality.

In our experiments, the Absolute Relative Error and Inlier Ratios are computed for each of the individual test sets described in the previous section and in Table 4. Their averages, along with AUSE, model runtimes, and memory consumption, are also reported.

4.4. Hardware and Software

The P100 GPUs are chosen for their computing prowess. The P100 has a substantial memory of nearly 16GB and high-performance CUDA cores that work with CUDA 11.8, enabling them to process the complex and resource-intensive deep learning models used in our experiments. We train and evaluate the models on a single P100 GPU. This is to eliminate complexity arising out of multi-GPU training. The Robust Multi-View Depth framework is implemented in PyTorch. The Anaconda environment runs Python 3.9 and PyTorch 2.0.1.

4.5. Ablation Study Protocol: Constants and Variables

Our ablation study protocol systematically explores the variations and influences of different factors within the RobustMVD framework. We carefully design our experiments to ensure a

thorough investigation of each element and its collective contribution to the overall performance of the Multi-View Depth (MVD) estimation model. The order of importance of these factors and their interactions is a critical part of our investigation. The most important are changes in the architecture of the network, followed by augmentations and properties of the input, and finally the training setup hyperparameters. During the variation of the architecture components, the other factors are kept constant. To explore the effects of input data of different modalities and the augmentations performed thereupon, we keep the architecture constant. By systematically modifying each element in our model and observing its effect on the model's performance, we can understand the impact of each factor on depth estimation, which can subsequently inform and inspire the design of more efficient and effective models. This robust approach allows us to explore the broad and intricate landscape of depth estimation.

4.5.1. Constants

To ensure the integrity and fairness of our experiment, we maintain a constant experimental setting. We control hardware specifications by ensuring all experiments run on the same graphics card. We also maintain a consistent random seed across all of our experiments to ensure that any variations in performance can be attributed solely to the variable under investigation and not to the inherent randomness in the initialization or training process. Important hyperparameters such as the number of depth bins, batch size, initial learning rates, decay schedules of learning rates, and the resolution of the input data are also kept constant. The effect of the number of depth bins is well understood, as increasing the number of depth bins will give a more accurate output. The resolution of the input images is kept constant since the evaluation occurs on a host of different resolutions, regardless of the resolution of the input data during training. This approach also enables us to train the networks on a single GPU instead of deploying them to a multi-GPU setup, which would introduce additional complexity to the training process that needs to be taken into account.

4.5.2. Variables

1. **Specific Architectural Components:** Several key architectural components form the heart of any depth estimation model. We delve into various parts, including choosing a coarse-to-fine architecture, plane sweep correlation versus Groupwise correlation, feature extraction network selection, and cost volume fusion method. Each component is individually altered while keeping others constant to analyze their individual and combined effects.

2. **Dataset Properties:** This involves varying the dataset or combinations of datasets, the number of source views, and the sampling strategy for the source views. Each of these variations is explored to understand how the properties of the data influence the performance and generalizability of the model.

3. **Preprocessing Techniques and Image Augmentations:** We consider different types of augmentations, such as scale augmentation, light augmentations, and spatial augmentations. We modify each augmentation independently and evaluate the impact on model performance.

4. **Training Setup Hyperparameters:** We investigate the effects of training hyperparameters, such as the number of groups in Groupwise correlation and normalization of features before correlation. Each hyperparameter is carefully varied while others are kept constant, and its impact on the model's performance is evaluated.

In this way, the extensive setup of the experiment, which entails carefully chosen models, diverse datasets, high-end hardware, and comprehensive performance metrics, allows for a rigorous ablation study on the RobustMVD framework. We hope the experimental findings guide the development of more accurate, reliable, and efficient depth estimation models, fostering advancements in this field.

In the next chapter, we take a look at the experiments performed and their results. Based on these results, we derive some inferences into the workings of the models and their respective components as well.

5. Experiments and Discussions

In this chapter, we present the experiments and their results. We dive into the implementation details of each experiment and make inferences from the results. Section 1 introduces the models and their training configurations that serve as baselines for the modified models. In Section 2, we look at the effects of changing specific architectural components of the existing baseline model and using design choices, such as the coarse-to-fine methodology. In Section 3, we look at different dataset properties, such as the number of source views and the view selection strategy. We also look at the effects of scale augmentation on the model performance. In Section 4, we look at model hyperparameters such as the number of groups in Groupwise correlation and the normalization of feature maps before correlation.

5.1. Baseline

1. **MVSNet**: We trained the baseline MVSNet model using the *BlendedMVS-MVSNet variant* (refer to Chapter 4,Subsection 4.2.1), which consists of 16904 samples. Each data point has a key view and two best-matching source views. We applied photometric and spatial augmentations, such as color jitter and normalization, uniformly to all views. The training input size for all implementations is $\mathbf{H} = \mathbf{576}$ and $\mathbf{W} = \mathbf{768}$ unless explicitly specified. This is also true for all models based on the MVSNet model. The batch size was set to 1, and the learning rate was 0.0001. We used the original paper's scheduler and Adam optimizer settings and trained the model for 160000 iterations. To facilitate comparison with our implemented models, we present three baselines with different variables: **1.)** $\mathbf{D} = \mathbf{256}$, which is the number of depth planes in the planesweep operation and is the same as in the original paper. This setting allows us to compare our implementations with the original paper's implementation. **2.)** $\mathbf{D} = \mathbf{128}$, which is the default setting that we use for most of our implementations. **3.)** $\mathbf{D} = \mathbf{128}$ with a training input resolution of $\mathbf{H} = \mathbf{480}$ and $\mathbf{W} = \mathbf{640}$, which we used for our implementations with high memory requirements, such as MVSNet augmented with the DINO feature extractor [6]. All the baselines **1**, **2**, and **3** are also evaluated on smaller resolutions mentioned in

the previous chapter. In the baseline and subsequent results tables presented, we mark the models evaluated using the smaller resolution settings mentioned in the previous chapter in Section 4.2.2 with a $\star$. Table 5a contains the results of the MVSNet baselines evaluated on our chosen test data sets.

2. **RobustMVD**: The RobustMVD baseline is also trained on the BlendedMVS dataset. There are two different settings for the number of views in the data samples and the batch size. **1.)** The first setting uses the *BlendedMVS-RMVD variant* with a batchsize of 4. This is the setting of the original RobustMVD baseline [1]. **2.)** The second setting is the same as the one for the MVSNet baseline explained in the previous point, i.e., the *BlendedMVS-MVSNet variant*. The batch size in this setting is 1. This setting is used for implementations that are memory intensive. In addition to these variables, we maintain the following parameters constant: The size of the training input is fixed to $\mathbf{H} = \mathbf{384}$ and $\mathbf{W} = \mathbf{768}$ for all RobustMVD derivative models. We use the RobustMVD loss[1], Adam optimizer, and Flownet scheduler [64] used in the original paper. Each RobustMVD derivative model is trained for 600000 iterations unless explicitly specified. The input data are uniformly augmented with spatial, photometric, eraser, and scale augmentations, the same as in the original paper. All baselines and derivative implementations are trained without depth range as a training input to the model. All RobustMVD based derivatives are also trained with data augmented with Scale Augmentation. Table 5b contains the results of the RobustMVD baselines evaluated on our chosen test data sets.

3. **Wrapped Original Implementations**: For these baselines, we import the original implementations from their respective repositories. We implement wrapper classes for these original implementations that follow the Robust Multi-View Depth framework schema for input and output and evaluate the pretrained models therein under the Robust Multi-View Depth benchmark. The pretrained MVSNet model is trained by the original authors on DTU[13], while the pretrained Vis-MVSNet model is trained on BlendedMVS [17]. Table 5c contains the results of the wrapped original implementations evaluated on our chosen test data sets.

The chosen settings and models provide a comprehensive baseline for comparing the derivative models implemented and evaluated hereafter as a part of this thesis. From Table 5, we see that our implementation of MVSNet outperforms the original wrapped implementation by a significant margin. This is because our implementation does not simply warp the source image to the reference image frustum to obtain the cost volume. Rather, it performs the planesweep operation by sampling points at the different depth levels along the epipolar line in the source

image. This explicit enforcement of the epipolar constraint to compute the sampling points in the source image leads to a more accurate warped representation and consequently leads to more accurate matching costs in the computed pairwise cost volume.

Approach	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet [1]															
$V = 2, D = 256$ ⋆	9.20	45.09	2.97	81.41	8.79	34.13	5.81	78.47	18.45	38.67	9.05	55.55	0.28	67.24	7276
$V = 2, D = 128$	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	0.26	65.2	5302
$V = 2, D = 128$ ⋆	11.44	40.50	2.84	82.24	9.80	32.31	8.92	78.97	25.28	39.87	11.65	54.78	0.27	50.78	3654
$V = 2, D = 128,$ $H = 480, W = 640$	8.42	46.64	2.88	83.12	9.47	33.94	5.97	82.65	24.61	40.46	10.27	57.36	0.27	37.56	5338
$V = 2, D = 128,$ $H = 480, W = 640$ ⋆	8.42	46.64	2.63	84.68	9.47	33.94	5.53	81.88	19.70	41.30	9.15	57.69	0.27	51.07	3663
b) RobustMVD [1]															
BlendedMVS-RMVD variant $(V = 4, B = 4)$ (Baseline from [1])	7.42	39.81	3.23	79.07	9.59	30.72	7.49	69.31	9.62	42.67	7.47	52.32	0.26	31.21	2159
BlendedMVS-MVSNet variant $(V = 2, B = 1)$	8.22	35.51	4.74	68.00	9.06	31.07	12.58	62.87	11.58	36.60	9.24	46.81	0.28	30.24	2125
BlendedMVS-MVSNet variant $V = 2, B = 1$ ⋆	8.21	35.51	4.24	71.04	9.06	31.07	11.34	60.78	11.13	36.58	8.80	47.00	0.28	25.23	1136
c) Original Wrapped Models															
MVSNet-pl wrapped (Baseline from [13])	21.91	18.21	2.56	82.19	22.12	20.54	9.29	69.49	38.92	30.25	18.96	44.14	0.33	105.73	7827
MVSNet-pl wrapped ⋆	21.91	18.21	2.07	85.28	22.12	20.54	9.30	65.68	36.16	30.13	18.31	43.97	0.35	73.55	5307
Vis-MVSNet wrapped (Baseline from [17])	14.97	46.94	2.44	84.10	9.97	31.65	5.22	82.62	14.71	43.19	9.47	57.70	0.33	104.73	7827

Table 5.: Baseline Results. **a)** MVSNet is trained and evaluated with 128 and 256 depth levels (D). We also train MVSNet with smaller inputs. This baseline is used to compare results with models having a high memory footprint. Each of these configurations is also evaluated with smaller inputs. These evaluation results are indicated by ⋆. **b)** RobustMVD is trained on two different settings: The first setting uses the *BlendedMVS-RMVD variant* with $B = 4$, and the second setting uses *BlendedMVS-MVSNet variant* with $B = 1$. The third evaluation has the same training configuration as the second one but is evaluated on smaller inputs. This has been indicated by a ⋆. All RobustMVD baselines and derivative models use $D = 256$. In all RobustMVD baselines, Scale Augmentation is applied to the training data. **c)** Wrapped original implementations of the baselines implemented in the RobustMVD framework. The MVSNet-pl model is trained on DTU with two source views, and Vis-MVSNet is trained with four source views on BlendedMVS. We also evaluate the MVSNet-pl model with the smaller inputs indicated by a ⋆.

5.2. Effects of Specific Architecture Components

In this section, we examine the effects of exchanging specific components of the architectures of the Multi-View Stereo models from our baselines. We have subdivided this section according to the Multi-View Stereo pipeline explained in Section 2.5. Only one component is exchanged at a time, leading to several possible derived models. In Section 5.2.1, we look at the effects of exchanging the feature encoder module in the baselines Multi-View Stereo and RobustMVD. In Section 5.2.2, we change the method to construct the cost volume from the extracted features. In Section 5.2.3, we look at the different fusion methods for the individual cost volumes computed from the reference features and the source features into a fused cost volume. In 5.2.4, we examine the effects of changing the cost volume regularization as a whole and parts of the cost volume regularization network. Specifically, we replace the 3D decoder of the cost volume regularization unit of MVSNet with a 2D decoder. Finally, in 5.2.5, we look at the effects of configuring the models in a coarse-to-fine design.

5.2.1. Choice of Feature Extraction Network Component

In this section, we will examine every combination of the feature extractor and base model, highlighting its notable characteristics, discussing the obstacles we faced when integrating the new feature extractor network with our base models, and clarifying the results. Table 6 offers a breakdown of the evaluation results for each configuration. All models are trained with the same parameters as our baselines unless otherwise specified.

1. **MVSNet Base: Switching out the MVSNet feature encoder with a DispNet encoder.**: The MVSNet and DispNet feature extraction encoders, while similar in the sense that they both involve a sequence of convolutional layers to form an encoder, have several differences in their specific structures and design choices that influence their behavior.

   In the MVSNet encoder, each sequential block $(conv1, conv2, conv3)$ comprises multiple convolution layers. More specifically, the $conv1$ block contains two convolutional layers, and the $conv2$ and $conv3$ sequential blocks contain three convolution layers each. This can be seen in Table 2.1. This architecture promotes a deeper model that theoretically can learn more complex and abstract representations. The DispNet encoder, on the other hand, uses one convolution layer per sequential block, making it shallower compared to the original MVSNet encoder. However, the DispNet encoder has more channels in the output features. Table 3.1 shows the architecture of the DispNet encoder in detail. The MVSNet encoder uses batch normalization after each convolutional layer and a

Feature Extractor (FE)	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet Base															
a1) Default Training and Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$)	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	0.26	65.2	5302
DispNet FE ($V = 2, D = 128$)	11.31	34.21	3.12	81.39	11.61	25.82	**6.45**	71.91	**21.64**	36.78	**10.83**	50.02	**0.26**	128.84	9601
FPN FE ($V = 2, D = 128$)	**9.16**	**50.32**	**2.66**	**82.18**	**9.77**	**36.34**	6.56	**84.63**	26.74	**37.27**	10.98	**58.15**	0.28	**48.97**	6716
a2) Default Training and reduced Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$) ⋆	11.44	40.50	2.84	82.24	**9.80**	32.31	8.92	78.97	25.28	39.87	11.65	54.78	0.27	**50.78**	**3654**
UNet FE ($V = 2, D = 128$) ⋆	**6.98**	**52.44**	**2.76**	**82.77**	11.21	**32.42**	**5.68**	**84.38**	**19.31**	**41.62**	**9.19**	**58.72**	**0.25**	71.19	7518
a3) Reduced Training and reduced Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$) $H = 480, W = 640$ ⋆	8.42	46.64	2.63	**84.68**	9.47	33.94	5.53	81.88	19.70	**41.30**	**9.15**	57.69	**0.27**	**51.07**	**3663**
DINO ViT FE ($\frac{H}{2}, \frac{W}{2}$) $H = 480, W = 640$ ⋆	**7.64**	50.28	4.86	75.27	48.67	10.31	7.71	79.05	**19.11**	40.31	17.60	51.04	0.37	726.56	9970
DINO ViT FE (H, W) $H = 480, W = 640$ ⋆	9.07	**51.31**	**2.59**	84.43	**8.01**	**37.12**	**5.52**	**83.12**	23.35	39.84	9.71	**59.16**	0.29	2986.24	10174
b) RobustMVD base															
b1) Train on *BlendedMVS-RMVD variant*															
RobustMVD Base ($V = 4, B = 4$)	**7.42**	**39.81**	**3.23**	79.07	9.59	30.72	7.49	69.31	9.62	**42.67**	7.47	**52.32**	0.26	**31.21**	2159
MVSNet FE ($V = 4, B = 4$)	7.46	37.10	3.30	**79.54**	**9.53**	**31.40**	**6.10**	**70.96**	**9.00**	41.69	**7.08**	52.14	0.26	48.44	**1797**
b2) Train on *BlendedMVS-MVSNet variant*															
RobustMVD Base ($V = 2, B = 1$)	**8.22**	**35.51**	**4.74**	68.00	**9.06**	**31.07**	12.58	**62.87**	**11.58**	**36.60**	**9.24**	**46.81**	0.28	**30.24**	2125
MVSNet FE ($V = 2, B = 1$)	8.96	32.23	4.86	**68.40**	9.69	30.15	**10.85**	59.73	14.78	32.45	9.83	44.59	**0.27**	46.10	**1786**

Table 6.: Changing the feature extractor. All MVSNet derivatives are trained on *BlendedMVS-MVSNet variant*. RobustMVD with MVSNet feature extractor are trained on both *BlendedMVS-RMVD variant* and *BlendedMVS-MVSNet variant*. We have three baselines for MVSNet, each with their respective configurations, and two baselines for RobustMVD. It is important to note that each model in subtable **a** and **b** has to be compared with its respective baseline (given in Grey below the dotted line in each subtable). In **a.1**, we see that both *DispNet FE* ($V = 2, D = 128$) and *FPN FE* ($V = 2, D = 128$) perform better than their baseline. In **a.2**, *UNet FE* ($V = 2, D = 128$)⋆ also performs better than its baseline. However, in **a.3**, both variants of DINO ViT, while performing better in some test datasets, have an overall lower average performance than their baseline. They also have a significantly higher inference time and memory requirement. In **b.1**, we can see that while RobustMVD with MVSNet feature extractor trained on *BlendedMVS-RMVD variant* performs slightly better than its baseline, in **b.2** RobustMVD with MVSNet feature extractor trained on *BlendedMVS-MVSNet variant* performs slightly worse.

ReLU activation function. The DispNet encoder does not have batch normalization and uses Leaky ReLU activation, which might help the shallower architecture learn features better. The MVSNet encoder architecture includes an additional reduction/projection layer $conv3r$ at the end, using a 1×1 convolution [65, 66]. This layer increases the depth of the network while maintaining a minimal increment in the parameter count.

In the process of substituting the MVSNet encoder with the DispNet encoder within the overarching MVSNet architecture, we alter the number of input channels feeding into both the MVSNet cost volume regularization encoder and decoder. Due to the inherent architectural similarities and componential parallels between the two encoders, this substitution is notably seamless and does not require any modification or manipulation of input data that are destined for downstream network elements. The DispNet encoder is more aggressive in increasing the number of channels and has a shallower architecture than the original MVSNet encoder. The result of this change can be seen in Table 6a.1 in the *DispNet FE* model. We see that this model performs better than the vanilla MVSNet model *MVSNet Base* $(V = 2, D = 128)$ due to the steep increase in the number of channels of the extracted features from 32 for the original MVSNet feature encoder to 256 for the DispNet encoder. This trumps the effects of the depth increase offered by the MVSNet encoder[1].

2. **MVSNet Base: Switching out the MVSNet feature encoder with a 2D FPN.**: In this experiment, we replace the Siamese Encoder of MVSNet with a Feature Pyramid Network.[2] The architecture of this Feature Extractor is shown in detail in Table 13. We use a stride of two pixels in the first layer. This is to reduce the spatial resolution of the output feature map of the FPN to half the size of the input image. We made this change due to memory constraints. The resolutions for this are indicated in Table 13 in black. We use the final output of the FPN, i.e., $feature0$, as an input for the correlation computation. $feature0$ has eight channels and is half the size of the input image. In Table 6a.1 in the *FPN FE* model, we can see that it outperforms its respective baseline *MVSNet Base* $(V = 2, D = 128)$. The deeper architecture of the FPN facilitates better learning of the features compared to the original encoder of MVSNet.

3. **MVSNet Base: Switching out the MVSNet feature encoder with a 2D UNet.**: In this experiment, we replace the Siamese Encoder of MVSNet with a UNet.[3] The architecture

[1]The evaluation on DTU uses the resolution $(H = 768, W = 1152)$
[2]Code taken from https://github.com/kwea123/CasMVSNet_pl
[3]Code taken from https://github.com/milesial/Pytorch-UNet.git

of the UNet is shown in Table14. Unlike the FPN, here, we use the penultimate upsampled feature $feat1$ as the input to the correlation layer. This feature has 16 channels and has half the spatial resolution as the input image. We perform this modification due to memory constraints. It is also for this reason that we evaluate this model on the smaller inputs as indicated by the $\star$. From Table 6a.2 in the *UNet FE* model, we can see that it outperforms its respective baseline *MVSNet Base* $(V = 2, D = 128)\star$. It has been observed that when the UNet and FPN feature extractors are compared on the KITTI and ScanNet datasets at the same resolution, UNet performs better than FPN. However, it should be noted that this does not necessarily mean that UNet will outperform FPN on the remaining three datasets.

4. **MVSNet Base: Augmenting the features extracted by the MVSNet feature encoder with features from a pretrained DINO ViT.**: In this experiment, we explore the effects of augmenting the features extracted by the siamese encoder of MVSNet with features extracted from a pretrained DINO ViT [6][4]. Due to memory constraints, we use the smaller DINO architecture *dino-vits8* [23] with a patch size of 8. It should be noted that this is the only model in all our experiments trained on two P100 GPUs with a smaller input size of $H = 480$ and $W = 640$. In our implementation, we placed the pretrained ViT backbone on one GPU and froze its weights, and placed the rest of the model on the second GPU. During training, we use the features from the pretrained frozen DINO ViT in two configurations. In the first one, we reduce the input size of the images by a factor of 0.5 before passing it on to the pretrained ViT, while the rest of the model uses the actual input size. This is to replicate the methodology from [35]. In a separate experiment, we pass the full-sized images to the pretrained frozen ViT for feature extraction, and the downsampling operation is performed afterward.

   As per the methodology of Cao *et al.*[35], we fuse the output of the features extracted using DINO with the saliency maps extracted from the [*CLS*] token from the last layer of DINO. This step is performed in both of our experiments with DINO ViT. This [*CLS*] token contains the global context of the entire image and is very useful as a saliency map to distinguish between the foreground and the background, allowing the downstream network to focus only on the foreground objects in the scene and ignore the masked background. The saliency maps are extracted by averaging the attention heads from the last layer of the [*CLS*] token. All values are then normalized to range between 0 and 1. For this fusion of features and the saliency maps, we use the *VITDecoder*[5] module from

[4]Code taken from https://github.com/ShirAmir/dino-vit-features.git
[5]Code taken from https://github.com/ewrfcas/MVSFormer.git

MVSFormer [35], which consists of a Gated Linear Unit and transpose convolutions to fuse attention with the features and increases the spatial resolution of the ViT feature maps. For the experiment using half-sized images, these ViT attention-augmented features are then concatenated directly with the learned features of the MVSNet Siamese encoder along the channel dimension and are given to the warping layer. For the experiment using the full-sized images, we downsample the ViT attention-augmented features by a factor of $\frac{1}{2}$ and then concatenate these downsampled features with the learned features from the MVSNet Siamese encoder along the channel dimension. It should be noted that the ViT features are also used during inference. Figure 5 shows the feature augmentation process in detail, along with the dimensions of the feature maps and the input images. Table 15 shows the different layers in the ViT decoder.

During inference, for the model trained with half-sized images to the ViT, we observe an improvement compared to our baseline model for KITTI and ETH3D and a decrease in performance for DTU and Tanks and Temples. This can be seen in Table 6a.3.For ScanNet, the performance is significantly worse. This leads to an overall reduction in the average metrics for this model. We attribute this to the comparatively smaller inputs of the ScanNet dataset, which are further halved before being fed to the ViT, leading to the loss of finer details in the input images. For the model trained with the full-sized images passed to the ViT, we see an improvement in performance for DTU, ScanNet, and Tanks and Temples. We observe a slight drop in performance for KITTI and a significant drop in performance for ETH3D. This affects its overall average performance, which is lower than its respective baseline. Both models have a higher inference time and memory requirement than their baseline.

5. **RobustMVD Base: Switching out the DispNet feature encoder in RobustMVD and replacing it with the MVSNet encoder.:** Consequently, based on the findings of previous experiments, it is prudent to investigate the impact of replacing the feature encoder in DispNet with the MVSNet feature encoder. Our goal was to evaluate whether the distinctive features extracted by MVSNet's encoder would enhance or degrade the performance of DispNet. Initially, the MVSNet encoder was incompatible with the DispNet architecture. Therefore, we had to adapt the MVSNet encoder for effective integration. Our adaptation involved several significant changes.

   Firstly, we increased the initial number of feature channels from 8 to 64. This leads to an increase in the number of channels in the extracted features of the MVSNet encoder from 32 to 256. This decision, although initially based on the need to make the MVSNet

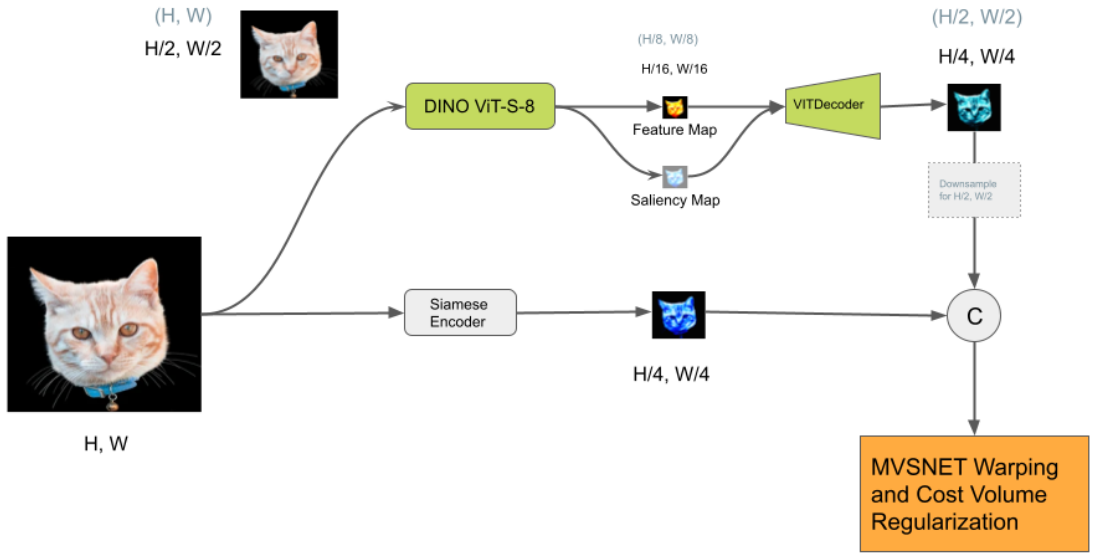


Figure 5.: Using ViT features to augment learned features. Here, we see how features and saliency maps obtained from the pretrained DINO-ViT-S-8 backbone are used to augment the learned features from MVSNet siamese encoder. The final size of the features and saliency maps obtained from the ViT is $\frac{H}{16}, \frac{W}{16}$. These features and saliency maps are then fused and upscaled to $\frac{H}{4}, \frac{W}{4}$ by the VITDecoder module[35]. For the setting where the ViT feature extractor is given the full-sized images as input (in Grey), we downsample the fused output of the ViTDecoder by a factor of $\frac{1}{2}$ (shown by the dotted grey rectangle) before concatenating with the learned features from the Siamese encoder.

encoder compatible with the rest of the DispNet architecture, also has the added benefit of potentially extracting more complex features from the input images and maintaining the deeper architecture of the MVSNet encoder, acknowledging the trade-off of increased computational complexity.

Secondly, we adjusted the stride of the first convolutional layer from 1 to 2. This allowed the convolutional filters to skip every other pixel, reducing the output size and computational cost while constructing the correlations and the cost volume. We perform this change to maintain the spatial dimensions of the skip connections from the MVSNet encoder to the DispNet decoder. Note that this operation might lead to the loss of some fine-grained detail due to the reduction in input size.

In the original MVSNet encoder, every convolution layer uses batch normalization. We changed this in our modified encoder. Specifically, the last convolution layer of

each block ($conv1b$, $conv2c$, and $conv3c$ in Table2.1) was implemented without batch normalization. $conv1b$ and $conv2c$ are fed as skip connections to the decoder part of the cost volume regularization network (refer to Table2.5 for details of the skip connections). Although batch normalization generally improves the speed, performance, and stability of a neural network, we found that, in our case, it led to training divergence. This phenomenon might have been due to the model's increased sensitivity to changes in the distribution of each layer's input, leading to amplified noise or exploding gradients. Therefore, to stabilize our training, we removed it from the last layers. It should be noted that the GC-Net[38] architecture also removes BatchNorm and ReLU for the layers whose outputs are used as skip connections to the cost volume regularization network.

Another significant modification was removing the $conv3r$ layer from the MVSNet encoder. The $conv3r$ layer is initially used to increase the depth of the encoder without changing the spatial dimensions of the output through the use of a 1x1 kernel, potentially facilitating the learning of more complex representations. It should be noted from Table 3.2 that the DispNet architecture has an additional component, the *Context Encoder*, which comes after the feature encoder. It contains a layer, $conv3a_0$, that performs a 1x1 convolution to reduce the depth of the feature map from 256 to 32 while maintaining the spatial dimensions in a similar fashion to the function of the $conv3r$ layer in the original MVSNet encoder (refer to Table 2.1). The encoded context obtained from this layer is concatenated to the output of the cost volume fusion layer further down into the network before passing the concatenated tensor to the cost volume regularization layer. By removing the $conv3r$ layer from the original MVSNet encoder, we bridged the gap between the two different encoder architectures. We initially tried incorporating the $conv3r$ and the *Context Encoder* in the network. However, including both components led to a divergence in training. Therefore, we removed the final $conv3r$ layer from the MVSNet encoder and directly used the output of $conv3c$.

During the evaluation, the model trained on *BlendedMVS-MVSNet variant* showed a slight degradation in performance, as seen in Table 6b.2. This is counter-intuitive to our expectation of a performance increase given the increase in depth offered by the MVSNet encoder and our modification to increase the feature channels from 32 to 256.

We also trained this configuration with *BlendedMVS-RMVD variant*, and there was a slight performance increase compared to its respective baseline of RobustMVD trained with *BlendedMVS-RMVD variant* as seen in Table 6b.1. This highlights the sensitivity of the models to the variants of the datasets and batch sizes for training and its consequent

effects on the evaluation. We will explore this further in the Section 5.3 on the properties of the data and datasets.

5.2.2. Choice of Correlation Layer

In this section, we first take a look at the different types of correlation layers employed in our baseline models as well as in our experiments. We then discuss the results of the experiments in detail.

Correlation Layer w/o	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
Normalized Features	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet Base															
a1) Default Training and Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$)	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	**38.51**	12.99	**55.56**	**0.26**	65.2	5302
Groupwise ($G = 4$) with Learned Fusion	9.90	40.89	3.17	**81.40**	8.86	33.22	**6.59**	**80.80**	21.43	38.06	9.99	54.87	0.27	**34.23**	**3153**
Groupwise ($G = 4$) + WarpOnly with Learned Fusion	**8.17**	**43.59**	4.01	78.52	**8.58**	**34.98**	6.71	80.18	**19.21**	38.09	**9.34**	55.07	0.28	84.28	8527
b) RobustMVD Base															
b1) Train on *BlendedMVS-MVSNet variant* and reduced Evaluation Input sizes															
RobustMVD Base ($V = 2, B = 1$) ⋆	**8.21**	**35.51**	**4.24**	**71.04**	**9.06**	**31.07**	11.34	**60.78**	**11.13**	**36.58**	**8.80**	**47.00**	0.28	**25.23**	**1136**
Groupwise ($G = 4$) with Learned Fusion (600k iterations) and 2D cost regularization ⋆	10.21	30.50	4.92	69.04	38.84	30.21	9.59	55.03	15.29	31.05	15.77	43.16	**0.26**	64.1	7374
Groupwise ($G = 4$) with Learned Fusion (1000k iterations) and 2D cost regularization ⋆	9.49	30.09	4.38	71.04	39.83	30.14	**7.37**	56.49	13.89	32.43	14.99	44.04	0.27	61.38	7370
Groupwise ($G = 4$) + Full Corr with Learned fusion and 3D cost regularization ⋆	32.21	12.91	5868.07	0	2061.13	0.63	96.27	14.69	795.89	5.01	1770.71	6.65	0.35	127.32	7504

Table 7.: Changing the correlation layer. **a)** All MVSNet and RobustMVD derivatives are trained on *BlendedMVS-MVSNet variant*. In **a.1**, we can see that both implementations with Groupwise correlation (GWC) and the GWC combined with differential homography outperform the baseline. The model using only GWC as a correlation layer has a significantly lesser inference time and memory requirement. In **b.1**, all our implemented models for RobustMVD using GWC perform very poorly compared to the baseline. Due to the high memory requirements of GWC for RobustMVD due to 256 channels in the feature maps, we use the smaller evaluation inputs. This is indicated by a ⋆. The baselines are shaded in grey.

1. **Plane sweep based Warping**[13]: As the name suggests, this is not really a correlation operation. This operation was used first in MVSNet and later in some of its extensions

[15, 41, 18] to warp the source features to the reference image frustum at different depth levels and build a cost volume. Consequently, these methods also use a variance-based cost volume fusion methodology, which looks at the variance between the warped source features and the reference features. We saw the working of this in Section 4 of the Background chapter.

2. **Full Correlation** [21]: The Full Correlation, was introduced for the first time in the DispNet paper. The Full correlation is computed as per the following equation.

$$c(\mathcal{F}_{ref}, \mathcal{F}_{source}) = \langle \mathcal{F}_{ref}, \mathcal{F}^{T}_{source} \rangle \tag{14}$$

where $\langle :, : \rangle$ denotes matrix multiplication. This operation contains $C \times W^2 \times H^2$ multiplications, is computationally intensive, and is performed for each pair of source and reference images. Here C is the number of channels in the feature maps of the reference and source images. We can see that this operation decimates the feature dimension of the two feature maps. The correlation map has dimensions $(B \times (W \times H) \times (W \times H))$. It contains matching similarity costs between the key view pixels and points on their respective epipolar lines in the source view for a set of sampled inverse depth values. The individual correlation maps are then fused together with a learned weights fusion or simply averaging them together. RobustMVD uses this correlation method to obtain a 2D cost map.

3. **Groupwise correlation** [42]: Groupwise correlation (GWC) was first used by Guo *et al.* to calculate the similarity between the left and right images in stereo matching by disparity estimation. This thesis extends it to the depth estimation with the MVS scenario. After the initial warping step, we the source features warped at each depth level with a projection of the reference features along the depth of the frustum. Unlike traditional correlation, which calculates the similarity between pixels along all feature channels, Groupwise correlation considers a group of channels simultaneously. The Groupwise correlation volume is computed as follows:

   a) Given the warped and projected source and projected reference features of dimensions (B, C, D, H, W), we reshape them into tensors of shape $(B, G, \frac{C}{G}, D, H, W)$. Here, G is the number of groups.

   b) We then take an element-wise inner product between the reshaped tensors and sum over the $\frac{C}{G}$ or the group-channels dimension.

   c) The cost volume obtained V_{gwc} has a shape (B, G, D, H, W) and is given as:

$$V_{gwc} = \Sigma_{group-channels}(A \odot B)$$

where A is the reference feature and B is the warped source feature projected along the depth of the frustum. And $\odot$ represents the element-wise inner product. The summation operation is performed along the $\frac{C}{G}$ dimension after taking the product. We used learned fusion to fuse the individual cost volumes.

It should be noted that since the correlation layer and the subsequent fusion operation are the most defining parts of an Multi-View Stereo pipeline, we do not switch the full correlation and the differential homography between RobustMVD and MVSNet. Instead, we perform two different types of experiments. In the first type, we replace the existing correlation layer with Groupwise correlation and use learned fusion to fuse the individual pairwise cost volumes while retaining the remaining architecture as it is. In the second type, we concatenate the output of the original correlation layer of the model with the output of the Groupwise correlation and use learned fusion to fuse the concatenated volumes. The fused volume is then given as input to either a 3D or 2D cost volume regularization network.

For our experiments, we set the value of G to 4. From Table 7a.1, we can see that Groupwise correlation outperforms our MVSNet baseline by a significant margin. We also implemented a MVSNet multi-correlation model that computes both plane sweep warping volume and Groupwise correlation volume for each reference and source pair. We then concatenate both volumes and fuse the individual concatenated volumes using learned fusion by computing the weights for each concatenated volume. This model outperforms both the baseline and the model with only Groupwise correlation.

For RobustMVD, it should be noted that the models are trained on the *BlendedMVS-MVSNet variant*, which contains two best views and a batch size of 1. In Table 7b.1, we observe a significant reduction in performance compared to the respective baseline. We include an additional layer to the architecture called the *CostVolume3DContextEncoder*. This contains three 3D convolutional layers, each with a kernel size of one, to reduce the number of channels in V_{gwc} from G to 1. We then squeezed the channel dimension and used the 2D DispNet cost volume regularization network to regress the depth and the uncertainty maps from this reduced cost volume. We conjectured that the model needed more time to learn the additional weights, so we trained it for 1000000 iterations instead of the usual 600000. We observed a slight improvement in performance for the model that was trained for a longer period; however, it was not enough to justify the continuation of training.

The third experiment for RobustMVD consists of the concatenation of the output of the

Groupwise correlation operation and the Full correlation operation. Since the full correlation output does not have an explicit channel dimension, we unsqueeze the tensor at the first dimension and concatenate it to the output of the GWC layer. We further concatenate the output of the context encoder unit of the RobustMVD model (refer to Table 3.2) to this by expanding it along the depth dimension. This concatenated tensor is then passed on to the 3D cost volume regularization network. In Table 7b.1 in the third entry, we can see that this model performs very poorly compared to the baseline and the other implemented models.

5.2.3. Choice of Cost Volume Fusion Method

Fusion Method	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet Base															
a1) Default Training and Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$)	**11.44**	**40.50**	**2.95**	**81.26**	**9.80**	**32.31**	**9.31**	**80.24**	**31.45**	**38.51**	**12.99**	**55.56**	**0.26**	65.2	5302
MVSNet with Learned Fusion	27.53	10.48	9.71	49.54	15.52	17.25	12.73	50.88	38.42	16.50	20.78	28.93	0.39	37.21	5732
MVSNet with Average Fusion	37.42	10.44	9.29	48.40	16.91	18.91	14.28	47.44	40.21	14.36	22.06	27.87	0.37	**35.82**	**4322**
b)RobustMVD Base															
b1) Train on *BlendedMVS-MVSNet variant* and default Evaluation Input sizes															
RobustMVD Base ($V = 2, B = 1$)	**8.22**	**35.51**	4.74	68.00	9.06	31.07	12.58	62.87	**11.58**	36.60	9.24	46.81	0.28	**30.24**	2125
RobustMVD with Average Fusion ($V = 2$)	8.99	34.82	**4.06**	**73.66**	**8.83**	**31.68**	**9.91**	**65.65**	13.34	**39.16**	**9.02**	**48.99**	**0.26**	45.53	**1783**

Table 8.: Changing the cost volume fusion layer. **a)** For experiments with the fusion layer, both MVSNet derivatives and RobustMVD derivatives are trained on *BlendedMVS-MVSNet variant*. A variance-based fusion only makes sense if there is only warping and no correlation computation between the reference and source features, as in MVSNet. In **a.1**, we can see that both learned fusion and average fusion don't work well with MVSNet for this reason. Consequently, we omit the combination RobustMVD with Variance Fusion because RobustMVD computes the full correlation between the reference and the source features. In **b.1**, we can see that average fusion gives slightly better results for average fusion compared to learned fusion for the given number of views. It should be noted that this might not necessarily be the case if the same experiment is repeated with a larger number of source views.

1. **Variance Fusion** [13]: This is used in the baseline MVSNet model. Variance-based fusion of individual cost volumes operates on the principle that all views contribute equally to the final fused cost volume. A cost metric based on variance is only applicable in models that utilize a differential homography warping approach, as correlation

computation between reference and source features is not explicitly performed during the homography warping process.Table 2.3 gives the equation to obtain the fused cost volume from the individual cost volumes. We only use variance-based fusion for some MVSNet derivatives that only warp the source features to the reference features, as the cost volumes generated by any model that computes correlations cannot be fused with a variance metric. Hence, we cannot fuse the full correlations of RobustMVD with variance fusion.

2. **Learned Fusion**[1, 67]: In learned fusion, we pass the individual cost volumes through a small 2D or 3D CNN network to learn the respective weights for each volume. First, we utilize the per-pixel (for 2D cost maps) or voxel weights (for 3D cost volumes) that we acquired. We applied these weights to each individual volume and then calculated a weighted average to produce the final fused cost volume. Unlike the variance operation, learned fusion can be used for derivatives of MVSNet that employ a different correlation layer other than the differential warping and RobustMVD. Therefore, as part of our ablation study, we implemented MVSNet with learned fusion. In Table 8a.1, we can see that the learned fusion for MVSNet performed much worse than the variance-based fusion baseline. This is because variance-based fusion implicitly computes the similarity between reference and individual warped source volumes instead of explicitly computing the correlation between the two. Hence, learned fusion does not work well in such a scenario, with no actual matching costs in the cost volume. We used learned fusion extensively in our models, where the correlation is computed explicitly in the correlation layer, such as the MVSNet models using Groupwise correlation as a correlation layer in the previous section.

3. **Average Fusion**[39]: Average fusion, like learned fusion, can be used for both MVSNet and RobustMVD models. In Table 8a.1, we can see the performance is slightly worse than the learned fusion for MVSNet and significantly worse than the baseline based on variance fusion. In Table 8b.1 RobustMVD with average fusion performs slightly better than our baseline model, which uses a learned fusion strategy. Schröppel *et al.*demonstrates this by training RobustMVD with average fusion using four views instead of two. This model exhibits a slightly lower performance than the baseline RobustMVD model trained with learned fusion. [1]

Cost Vol Regularization	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet Base															
a1) Default Training and Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$)	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	0.26	65.2	**5302**
DispNet 2D Decoder	**7.79**	**48.92**	2.83	83.22	**8.20**	**36.84**	**5.75**	**83.76**	**19.26**	**40.63**	**8.77**	**58.67**	0.29	627.45	5491
3D UNet Stacked Hourglass	10.63	39.58	**2.64**	**85.33**	9.96	32.90	6.13	81.82	25.73	38.31	11.02	55.59	**0.25**	**45.54**	5373

Table 9.: Changing the cost volume regularization layer. **a)** All MVSNet derivatives are trained on *BlendedMVS-MVSNet variant*. It is evident that both of our models outperform the baseline. Particularly, the implementation of MVSNet using a 2D DispNet decoder shows significant improvement over the baseline. However, we can see that it takes significantly longer to make inferences compared to the baseline. In the second entry, we see that applying the Stacked Hourglass regularization yields better results than the baseline without causing a significant increase in inference time and memory requirements.

5.2.4. Choice of Cost Volume Regularization Network Component

1. **Switching out the MVSNet fused cost-volume decoder with the DispNet fused cost-volume decoder.:** In the original MVSNet, the cost volume is a 3D volume with dimensions (Batch size, Channels, Depth levels, Height, Width). Here, each pixel across all depth planes is associated with a probability of that pixel at that depth being part of the object surface in the scene. It is a structured cost volume that is then passed to a 3D CNN Encoder Decoder cost volume regularization component. Now, since we wanted to switch to the DispNet cost volume decoder, which is a 2D decoder, we deal with this problem by slicing the fused cost volume along the depth dimension in a similar fashion to [18].

   The simplest way to integrate the 3D fused cost volume into a 2D decoder is to slice it at each depth level and pass each layer individually to the 2D decoder. The output of each layer could then be interpreted as a probability map for that depth. The depth map can then be obtained by choosing the depth with the maximum probability for each pixel. However, this increases the time complexity, as the 2D decoder needs to process the image once for each depth layer.

   The skip connections from the fused cost volume encoder are also three-dimensional. However, since they are intermediary representations, they have a larger number of depth levels compared to the final output of the cost volume encoder. To reconcile this

difference, we use trilinear interpolation to match the slices from intermediate volumes with fewer depth dimensions, including the final output, to those of the output with the largest depth dimension, which in our case is $3DConv0$. For instance, given that we have volumes of sizes (1, 64, 32, 4, 5) and (1, 32, 64, 8, 10), we interpolate along the depth dimension of the first volume to match the depth dimension of the second volume, which gives the new volume of shape (1, 64, 64, 4, 5).

To preserve the spatial correlation between the slices, we maintain the order of the slices along the depth dimension, as it represents the spatial relationship between different depths. In this manner, we use the 3D skip connections in the 2D decoder by transforming the 3D volumes into a series of individual 2D depth planes, which we then feed into the 2D decoder at the appropriate layers. This process is visualized in Figure 6.

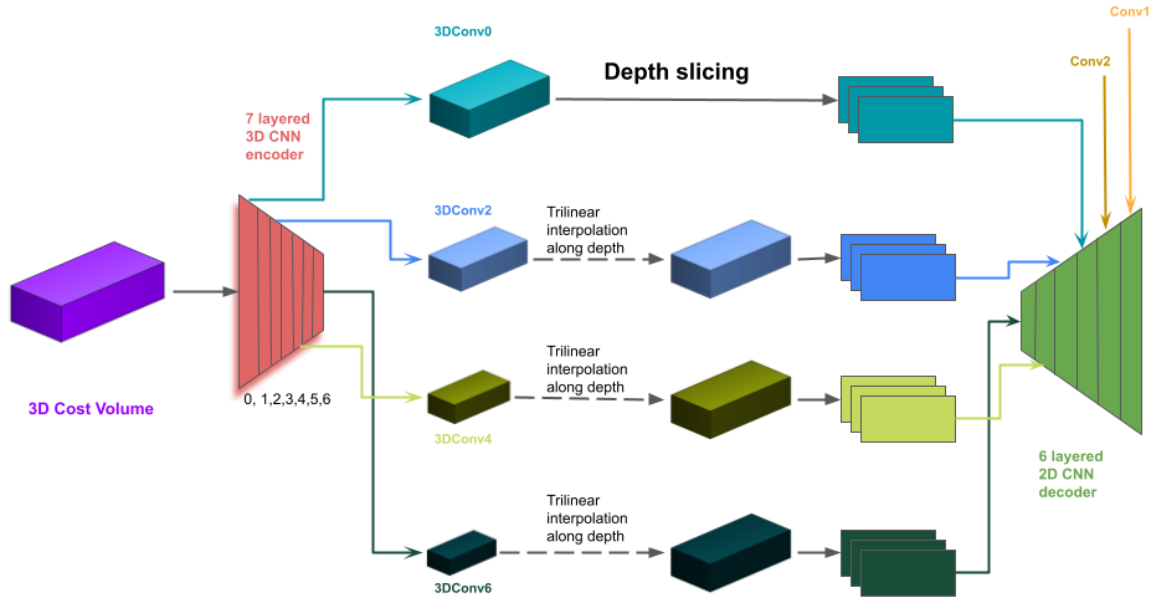


Figure 6.: Using a 2D Decoder for 3D Cost Volume Regularization. This figure shows the method of *depth slicing* used to regularize a 3D cost volume using a 3D-2D hybrid Encoder-Decoder. The final and intermediate skip connection outputs of the 3D encoder are interpolated along the depth dimension to match the number of depth levels in each of them. Then, each *depth slice* of the final encoder output, along with its corresponding depth slices from the intermediate representations, are sent individually to the 2D decoder for processing. The decoder outputs from each tuple of *depth slices* are then stacked along the depth dimension, and the depth map is regressed from this cost volume. This final step is not shown in the figure.

This method takes longer to train than a normal 3D convolution decoder, as the decoder

is called for each tuple of depth slices. These tuples look like ($3Dconv0[:,:,i,:,:]$, $3Dconv2[:,:,i,:,:]$, $3Dconv4[:,:,i,:,:]$, $3Dconv6[:,:,i,:,:]$). The maximum value of i depends on the number of depth sampling points chosen at the beginning of the training, which in our case is $D = 128$. In Table 9a.1, we can see that this model gives much better results compared to the baseline of training MVSNet on *BlendedMVS-MVSNet variant* with 128 depth samples. It also performs better than training MVSNet with 256 depth values. The drawback of this model is that it takes a long time to train because each depth slice is processed by the cost volume decoder individually. The output of each iteration is aggregated into a new regularized cost volume along the depth dimension. The final depth map is regressed from this cost volume.

2. **MVSNet with Stacked Hourglass Cost Volume Regularization Network**: In this experiment, we implemented a stacked hourglass network, which is used for regularizing the raw fused cost volume. The architecture for this network is given in detail in Table 16. Each *hourglass* in the architecture is an encoder-decoder network with skip connections going from the encoder to the decoder. Our architecture incorporates three distinct hourglass modules. After every level, a depth map is generated from the probability volume. To calculate the loss, we use a weighted summation of the smooth L1 loss between the depth maps and ground truth at each output level of the stacked hourglass. From Table 9a.1, we can see that this model also performs better than the baseline without significantly increasing inference time and memory requirement.

5.2.5. Usage of Coarse to Fine Architecture

For the experiments in this section, we re-engineered our baseline models in a coarse-to-fine design [15]. We consider a three-stage design. The depth range in the first stage, R_1, encompasses the entire depth range of the input scene. Subsequent stages use the previously predicted output to constrict this initial hypothesized range. Mathematically, this is given by $R_{k+1} = R_k \times w_k$, where R_k is the hypothesis range at stage k and $w_k < 1$ is the reduction factor. Similarly, the depth interval at the initial stage is symbolized as I_1. This interval is relatively larger than that in conventional singular cost volume formulations used in our baselines, making the primary depth estimation coarser. For subsequent stages, we use narrower intervals to derive finer-grained outputs. This can be written as $I_{k+1} = I_k \times p_k$, where I_k is the hypothesis plane interval during the k^{th} stage and $p_k < 1$ is the decrement factor. For stage k, the total number of hypothesis planes, D_k, is computed as $D_k = \frac{R_k}{I_k}$. We use a value of $p_k = \frac{1}{2}$. We use [32, 16, 8] depth hypotheses from the coarsest to the finest level. The interval ratios are set to [4, 2, 1].

This is such that the product of the interval ratio and the number of depth hypotheses at the coarsest level $32 \times 4 = 128$, which is equal to the number of depth hypotheses that we use for a single stage MVSNet model[15]. The cascade formulation reduces the overall count of hypothesis planes as we refine the initial depth range in subsequent stages. Similar to Equation 11, the homography warping function for the $k + 1^{th}$ stage can be reformulated as:

$$H_i(d_k^m + \Delta_{k+1}^m) = K_i \cdot R_i \cdot (I - \frac{(t_1 - t_i) \cdot n_1^T}{d_k^m + \Delta_{k+1}^m}) \cdot R_1^T \cdot K_1^{-1} \cdot d_k^m \tag{15}$$

Here, d_k^m stands for the predicted depth of pixel m during the k^{th} stage, while Δ_{k+1}^m represents the m^{th} pixel's residual depth for the $k + 1^{th}$ stage[20].

We used the same smooth $l1$ loss here as we did for our baselines with a slight modification. The predicted depth maps of each stage k are weighted by a factor λ_k, and the total loss is the sum of all weighted losses.

In this section, we examine the impact of changing the feature extractors on the performance of models for coarse-to-fine design. Generating multilevel features is the most crucial component, so we focus on this. We trained the models using two different feature extractors: FPN and UNet. Unlike the FPN in Section 5.2.1, we use a stride of one in the first $conv1a$ layer. Thus, the feature maps at each level have twice the resolution of the feature maps from the other implementation of FPN. These resolutions are given in Table 13 in red. Similarly, for the UNet, we use all the features starting from $feat2$ at the coarsest level and $feat0$ at the finest level. This is possible due to the reduced number of depth hypotheses used in a coarse-to-fine architecture. Our experiments yielded results that are outlined in Table 10. We observed that the MVSNet coarse-to-fine model with FPN feature extractor outperformed the baseline and had a much lower memory requirement, albeit requiring slightly more time for training and inference. This might be attributed to the adaptive number of depth levels and the cascading design of depth refinement. However, the UNet feature extractor-based coarse-to-fine design performed poorly compared to the baseline. Therefore, we can conclude that the FPN feature extractor is more suitable for a coarse-to-fine scenario than the UNet.

Feature Extractor for	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
Coarse-to-Fine	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) MVSNet Base															
a1) Default Training and Evaluation Input sizes															
MVSNet Base ($V = 2, D = 128$)	**11.44**	**40.50**	**2.95**	**81.26**	9.80	32.31	9.31	80.24	31.45	**38.51**	12.99	**55.56**	**0.26**	**65.2**	5302
Cascade MVSNet with FPN FE	16.11	26.53	3.31	79.77	**9.66**	33.42	**5.83**	**81.55**	**19.78**	37.34	**10.94**	51.72	0.88	168.31	2101
Cascade MVSNet with UNet FE	24.15	23.94	3.21	79.90	10.20	**33.69**	7.66	80.14	25.85	36.54	14.22	50.84	0.86	154.31	**2044**

Table 10.: Changing the feature extractor for Coarse-to-Fine model design. **a)**All MVSNet derivatives are trained on *BlendedMVS-MVSNet variant*. From the results, it is evident that the MVSNet coarse-to-fine model with FPN feature extractor outperforms the baseline. On the other hand, the model that has the UNet feature extractor performs poorly. Both models have a higher degree of uncertainty than the baseline and take longer to make inferences. However, the memory requirements for these models are significantly lower.

5.3. Effects of Changing Training Dataset Properties

In this section, we analyze how modifying the properties of the input data impacts the model's performance. We do not make any changes to the test input data. All other settings are kept constant, and only a single attribute is changed in each experiment. All MVSNet models are given the camera intrinsics, poses, and depth range as input. All RobustMVD models are given only the camera intrinsics and the poses as inputs. We do this to maintain similarity with the respective baselines. All RobustMVD models are trained with Scale Augmentation unless explicitly specified.

1. **Choice of training dataset/combination of datasets**: The choice of dataset used for training the model greatly affects the model's performance. Most models for depth estimation with Multi-View Stereo are trained on DTU or BlendedMVS directly, or BlendedMVS is used for fine-tuning after training on DTU. Following the recommendations of Schröppel *et al.* in their paper [1], we train all our models on BlendedMVS. As part of our ablation study, we train our MVSNet and RobustMVD baselines on DTU. For both runs, we used only two best-matching source views, as the memory requirements of DTU are twice that of BlendedMVS.

   From Table 11a, we can see that the models trained on DTU have lower performance than those trained on BlendedMVS. In the case of MVSNet, which is trained with the depth range from the scene provided to the model, the model overfits to this depth range. This is reflected in the results, where the model performs better on DTU and worse on the

Approach	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) Change Dataset															
a1) MVSNet Baseline ($V = 2, D = 128$)	**11.44**	**40.50**	2.95	81.26	**9.80**	**32.31**	9.31	**80.24**	31.45	**38.51**	**12.99**	**55.56**	0.26	**65.2**	**5302**
MVSNet trained on DTU ($D = 128, V = 2$)	18.67	38.80	**2.32**	**85.84**	12.17	26.55	**8.28**	75.35	**26.44**	31.98	13.57	51.70	-	112.73	8716
a2) RobustMVD Baseline with *BlendedMVS-MVSNet variant* ($V = 2$)	**8.22**	**35.51**	4.74	**68.00**	**9.06**	**31.07**	**12.58**	**62.87**	**11.58**	**36.60**	**9.24**	**46.81**	0.28	30.24	2125
RobustMVD trained on DTU ($D = 256, V = 2$)	48.16	4.07	**3.82**	62.39	33.70	5.98	50.80	10.71	51.14	5.35	37.53	17.70	-	**25.07**	**1932**
b) Scale Augmentation															
b1) MVSNet Baseline ($V = 2, D = 128$)	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	**38.51**	12.99	55.56	0.26	65.2	**5302**
MVSNet ($V = 2, D = 128$) with Scale Aug	**8.06**	**48.01**	**2.63**	**83.27**	**9.24**	**34.91**	**5.78**	**83.15**	**24.46**	37.58	**10.03**	**57.38**	**0.26**	**38.58**	5445
b2) RobustMVD Baseline with *BlendedMVS-RMVD variant* ($V = 4$)	**7.42**	**39.81**	**3.23**	**79.07**	**9.59**	**30.72**	**7.49**	**69.31**	**9.62**	**42.67**	**7.47**	**52.32**	**0.26**	31.21	**2159**
RobustMVD without Scale Aug ($V = 4$)	10.90	26.63	4.00	72.27	10.99	27.32	15.08	58.51	12.13	35.34	10.62	44.01	0.28	**29.35**	2168
c) Number of Source Views															
c1) MVSNet Baseline ($V = 2, D = 128$)	11.44	40.50	**2.95**	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	0.26	65.2	**5302**
MVSNet Base ($D = 128$, **V=4**)	**7.93**	**47.09**	2.99	**82.15**	**9.54**	**33.43**	**7.54**	**82.39**	**23.81**	**39.92**	**10.76**	**57.00**	**0.26**	**38.58**	5445
d) Sampling Strategy for Views															
d1) MVSNet Baseline ($V = 2, D = 128$)	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	**0.26**	**65.2**	**5302**
MVSNet ($D = 128, V = 2$) All Combinations + Random Selection	**7.55**	**46.00**	**2.68**	**83.17**	**8.87**	**34.30**	**6.54**	**81.63**	**17.33**	**41.41**	**8.60**	**57.30**	0.27	76.52	5427
d2) RobustMVD Baseline with *BlendedMVS-RMVD variant* ($V = 4$)	7.42	39.81	**3.23**	**79.07**	9.59	30.72	**7.49**	**69.31**	**9.62**	**42.67**	**7.47**	**52.32**	**0.26**	**31.21**	2159
RobustMVD Base ($V = 4$) *Best matching views*	**7.40**	**40.85**	4.13	74.11	**8.86**	**31.82**	9.13	69.12	10.19	41.00	**7.94**	51.38	0.27	31.61	**2139**

Table 11.: Changing the training data properties. **a)**In this ablation study, we only alter one aspect of the training data at a time. The baselines for each experiment are provided above it in grey. **a)** We can see that both MVSNet and RobustMVD trained on DTU perform poorly compared to their baselines. **b)** Models trained with scale augmentation perform better, particularly on ETH3D, where they often produce poorer results, compared to models trained without scale augmentation. **c)** Increasing the number of source views increases the performance of the model. **d)** An all combinations - random selection strategy for training input data gives better evaluation results than choosing the best-matching source views.

remaining test datasets, which have a different depth range. For RobustMVD, the results are significantly worse than its baseline. The RobustMVD model is trained on a fixed depth range of 0.4 to 1000 meters. This depth range sets the maximum and minimum depth of the reference camera frustum in the planesweep correlation operation. Most scenes in DTU fall in the range of 0.4 to 1.2 meters. Consequently, a RobustMVD model trained on DTU exhibits a significantly worse performance on all datasets except DTU,

which is comparable to the baseline. The BlendedMVS dataset contains scenes with a large variation in the depth range from indoor scenes of objects to outdoor scenes. It is an ideal dataset for training robust Multi-View Stereo models.

2. **Scale Augmentation**: During training, the scale augmentation technique[1] adjusts the ground truth translation vectors by rescaling them before sending them to the model. The ground truth inverse depth map is also scaled using the inverse scaling factor. Any inverse depth values outside the range of $[0.009m^{-1}, 2.75m^{-1}]$ are masked to ensure accurate results. The depth values of the previous training iterations are stored as a histogram. This is then used to determine the scaling factor. The size of the histogram bins is increased logarithmically to ensure optimal performance throughout the full depth range. The scaling factor for the current sample is the ratio between the depth label of the histogram bin with the lowest count and the median ground truth depth value.

   In this experiment, we augment the training data from the *BlendedMVS-MVSNet variant* split for training MVSNet with scale augmentation. We also provide the depth values to the model as is the case in the baseline MVSNet model. In Table 11b, we can see that the results were much better than our baseline model. We also train a RobustMVD model without scale augmentation. We do not provide depth values in this case, similar to the baseline model. Here, we observe that the performance is lowered. Therefore, we can conclude that the scale augmentation technique introduced in [1] increases the overall robustness of the trained model.

3. **Number of Source Views**: This is an important data hyperparameter for training Multi-View Stereo networks. For most MVS methods, there is a limitation on the number of source views due to memory constraints. We noticed this when training the RobustMVD model on DTU, where we reduced the number of source views from four to two. Insufficient views may cause gaps or inaccuracies in 3D reconstruction. Using too many views becomes computationally expensive. It is not always ensured that a higher number of source views will give a higher performance. Therefore, it is important to strike a balance. We investigated this important hyperparameter as part of our ablation study. In Table 11c, After conducting experiments, we found that training the baseline MVSNet model with four views instead of the usual two views produced significantly better results. We utilized the same view selection strategy as the baseline training. However, it is important to note that even though we trained the model on four views, we only used two source views during inference, as with all the other models. This indicates that training a model with more views increases its robustness. Unfortunately, due to memory limitations, we

could not test the upper limit of this hyperparameter. Most Multi-View Stereo research uses four source views as a maximum [35, 17, 68]. Despite requiring more memory and longer training time, this model's improved performance suggests that it is worth considering.

4. **Sampling strategy for views**: When we say that N images are sent to the network, this means that in each scene, each image is taken as a reference image, and its neighboring $N-1$ images serve as source images. Our training data splits, *BlendedMVS-MVSNet variant* and *BlendedMVS-RMVD variant*, use different source view picking strategies. These are as follows.

   - **Best Matching Top N views**: In this strategy, the $N-1$ source views are selected based on how well they match the reference image. This matching can be done using various criteria, such as feature correspondences, texture similarity, or other image similarity measures. The $N-1$ views that best match the reference image are then used to estimate the depth and confidence maps. This approach might lead to more accurate results in cases with high matching quality but could struggle in scenes with occlusions or repetitive patterns. Our baseline MVSNet is trained with this strategy following the original paper [13]. We use this sampling strategy for *BlendedMVS-MVSNet variant*.
   - **All combinations - Random selection**: This strategy involves considering all possible combinations of randomly selected $N-1$ source views from the available images in a scene. This approach aims to capture a more comprehensive view of the scene geometry by considering various combinations of source views. In our experiments, *BlendedMVS-RMVD variant* uses this strategy. This approach is more robust to occlusions and ambiguities. To allow the model to "see" all samples, we can either increase the number of training iterations or increase the batch size per iteration. This can be computationally intensive due to the large number of training sample combinations compared to the previous approach. We used this strategy to train our RobustMVD baseline as per the original paper [1].

   When we trained MVSNet with the All combinations - Random selection strategy, we observed a significant jump in performance during inference. This can be seen from the results in Table 11d. This can be explained by the view selection strategy, which naturally introduces some data noise and occlusions and enhances the model's robustness. Additionally, we modified the training of RobustMVD by using the four best-matching source views instead of the usual sampling technique of *BlendedMVS-RMVD variant*.

As a result, we noticed a minor decrease in the evaluation performance. This could point to a potential factor in the training of RobustMVD that makes it better suited to handle the change of source view sampling strategy. Therefore, we can infer that training on data sampled with a random selection strategy that covers all possible combinations is more effective than using the nearest match source views for the key view.

5.4. Effects of Model Specific Hyperparameters on the Performance

Training of deep neural networks involves many different hyperparameters. Different combinations of hyperparameters give different results. Although looking at all of them is beyond the scope of this thesis, we look at two model-specific hyperparameters that are relevant to the current scenario. These are.

1. **Number of Groups in Groupwise correlation**: This is an important hyperparameter. However, as with all hyperparameters, there should be a balance between computation requirements and accuracy. The upper bound of the number of groups G is the total number of channels in the features, where each channel is treated as its own group. The lower bound is 1, which is equivalent to the full correlation. For MVSNet, we take this upper bound and train the model with $G = 32$ since the MVSNet feature extractor outputs feature maps with 32 channels. We had previously trained MVSNet with $G = 4$. For the current setting of $G = 32$, we can see from Table 12a that there is a slight improvement in the performance of the model compared to $G = 4$, which is as expected. Consequently, there is also an increase in the time complexity and memory requirement as each channel is now processed separately.

2. **Normalization of feature maps before correlation**: Another important choice that we make is whether to normalize the features before correlating them. Normalized features are important for training stability. From Table 12b, we can see that normalized features perform slightly better than unnormalized features for Groupwise correlation and significantly better for the differential homography warping without incurring any additional computational cost. RobustMVD trained without normalized features shows a reduction in performance. Therefore, we conclude that normalized features are better than unnormalized features for computing the correlation volumes between the reference and source features.

Hyperparameter	**KITTI**		**DTU**		**ScanNet**		**T&T**		**ETH3D**		**Average**				
	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	rel ↓	τ ↑	AUSE↓	time ↓ (mSec)	memory ↓ (MB)
a) Number of groups in GWC															
a1) MVSNet Base + GWC ($G = 4$)	9.90	**40.89**	**3.17**	81.40	8.86	33.22	**6.59**	80.80	21.43	38.06	9.99	54.87	0.27	**34.23**	**3153**
MVSNet Base + GWC ($G = 32$)	**9.22**	40.57	3.37	**81.53**	**8.68**	**33.87**	7.19	**80.92**	**18.42**	**39.04**	**9.38**	**55.19**	0.27	46.72	7505
b) Correlation Layer change normalization parameter															
b1) MVSNet Baseline	11.44	40.50	2.95	81.26	9.80	32.31	9.31	80.24	31.45	38.51	12.99	55.56	**0.26**	**65.2**	**5302**
MVSNet Base (Warp Only) Norm True	**7.56**	**47.26**	**2.65**	**82.96**	**8.83**	**34.95**	**6.54**	**82.55**	**24.21**	**40.15**	**9.96**	**57.57**	0.29	68.58	5466
b2) MVSNet Base + GWC ($G = 4$)	9.90	40.89	**3.17**	**81.40**	8.86	33.22	6.59	**80.80**	21.43	38.06	9.99	54.87	0.27	**34.23**	**3153**
MVSNet Base + GWC ($G = 4$) Norm True	**9.56**	**42.51**	3.66	80.23	**8.86**	**33.42**	**5.94**	79.76	**20.01**	**38.93**	**9.61**	**54.97**	**0.27**	35.30	3254
b3) RobustMVD Baseline with *BlendedMVS-RMVD variant* ($V = 4$)	**7.42**	**39.81**	3.23	79.07	9.59	**30.72**	**7.49**	**69.31**	**9.62**	**42.67**	**7.47**	**52.32**	**0.26**	**31.21**	2159
RobustMVD Base (Full Correlation) Norm False	7.82	38.01	**2.82**	**81.38**	**9.43**	30.63	9.90	68.12	10.03	41.80	8.00	51.99	0.27	31.47	**2140**

Table 12.: Changing model specific hyperparameters. **a)** The MVSNet derivatives are trained using *BlendedMVS-MVSNet variant*, while the RobustMVD derivatives are trained using *BlendedMVS-RMVD variant*. The baselines are presented in grey above the results. Upon reviewing the results, we notice that increasing the number of groups in the Groupwise correlation correlation from 4 to 32 slightly improves performance. However, this results in a significant increase in memory consumption. **b)** The second part of the table indicates that normalizing features before correlating them in the case of MVSNet with differential homography warping and with Groupwise correlation leads to improved performance compared to baseline models that don't use feature normalization. In the case of RobustMVD, not using normalized correlation leads to decreased performance compared to the baseline, which normalizes features along the channel dimension. This highlights the importance of normalizing the features before the correlation operation.

6. Discussion, Future Work and Conclusion

6.1. Discussion

The main research goal of this thesis was to implement components of the pipelines of existing Multi-View Stereo methodologies in a modular fashion and perform an ablation study of their individual and collective effects on the performance of the model in the context of depth estimation. To achieve our goal, we developed various modules for different stages of the Multi-View Stereo pipeline.

Changing the Feature Extraction

To extract features, we experimented with different versions of our baseline models, MVSNet and RobustMVD, using both the Feature Pyramid Network and UNet feature extractors. We also incorporated a pretrained frozen DINO ViT into the existing MVSNet model. Additionally, we trained and evaluated the two baseline models with swapped feature extractors. Our findings showed that models with deeper feature extractors, such as the FPN and UNet, outperformed the baselines. However, augmenting the learned features with features from the pretrained DINO ViT produced mixed results.

Changing the Correlation Layer

For the correlation layer, we extended the Groupwise correlation by Guo *et al.*[42] and used it as a standalone correlation layer as well as augmenting the existing correlation volumes with the Groupwise correlation output. We saw that this worked quite well for MVSNet implementations. However, for RobustMVD, which uses the Full correlation from DispNet [21], the models performed poorly.

Changing the Cost Volume Fusion Layer

We also experimented with different fusion methods for the individual cost volumes obtained from the correlation operation. We implemented 3D learned fusion and 3D average fusion to use with the MVSNet models instead of its variance fusion. For RobustMVD, which uses a 2D learned fusion, we implemented a 3D average fusion module and trained and evaluated a model with this layer instead of the learned fusion. We observed a drop in performance for the MVSNet models using learned and average fusion. As its name suggests, the variance fusion layer implicitly computes the variance, i.e., the dissimilarity between the warped source views and the reference view. This is logical since MVSNet employs a differential homography warping as its correlation layer. Learned fusion and average fusion work well for models that compute the correlations between the reference and source features explicitly. We used this learned fusion successfully for models that use Groupwise correlation and the full correlation. For RobustMVD with average fusion, we observed a slight increase in the average performance.

Changing the Cost Volume Regularization Layer

We have incorporated two distinct cost volume regularization strategies for MVSNet. The first one employs the 2D DispNet decoder instead of the 3D MVSNet decoder. This model outperformed the baseline MVSNet for 128 and 256 depth levels, even though it was trained with 128 depth levels. However, this model takes a long time to train and evaluate. Additionally, we have implemented a stacked hourglass cost volume regularization unit, which performed better than its baseline.

Implementing models with a Coarse to Fine Architecture

We also implemented the MVSNet model in a coarse-to-fine pattern where the depth range is updated for each stage using the coarse depth map of the previous stage. We used multilevel feature extractors, i.e., the FPN and the UNet, as feature extractors in these coarse-to-fine models. We saw a significant improvement in the case of the FPN feature extractor. However, the cascade model with the UNet feature extractor performed poorly compared to the baseline. We attribute this to the FPN generating better features than the UNet at the earlier stages of the layered network.

Changing the Training Data Properties

We also experimented with modifying different properties of the input data and training the models on a different dataset, i.e., DTU. We saw that models trained on BlendedMVS performed significantly better than those trained on DTU. We used the Scale Augmentation technique introduced by Schröppel *et al.*[1] with MVSNet, and the model performed significantly better than the baseline. We also removed the Scale Augmentation from RobustMVD, where it is the default, and observed a drop in performance. We conclude that Scale Augmentation makes the trained model more robust. We next increased the number of source views for training MVSNet and observed an increase in performance. Next, we changed the sampling strategy for the source views for MVSNet and RobustMVD instead of their default sampling strategies and observed that the All combinations - Random Selection strategy performed better than the best-matching views strategy.

Changing Model Specific Hyperparameters

In our final study, we altered certain hyperparameters of the training setup. We changed the number of groups in Groupwise correlation from 4 to 32 for MVSNet and observed a slight increase in performance. We also changed the status of the normalization parameter for the features before computing the correlations and observed that models with normalized features being fed to the correlation module performed better than non-normalized features for all three types of correlations explored.

6.2. Future Work

All the above points could be further explored as future directions of research for improving the robustness of the models. Existing correlation layers can be augmented with Rank correlation and census transform to make the model more robust to illumination changes. Incorporating visibility consistency and uncertainty as feedback to the network can help it learn to deal with occlusions or merge information from multiple views to reduce uncertainty. One can also explore the direction of using GANs for adapting features generated for one distribution to another distribution to provide more robust features for the rest of the Multi-View Stereo pipeline. This can be done using adversarial training, where the discriminator tries to distinguish between features of the source and target distributions while the generator (or an adaptation module) tries to make them indistinguishable. Using spatial or depth attention or both in the cost volume regularization unit can help the network distinguish between valid and invalid

pixels or depth hypotheses by enforcing constraints on them based on the surrounding context. One could also look into different strategies for source view selection such that the occlusions are minimized.

6.3. Conclusion

Based on the experiments conducted, we have concluded that selecting the appropriate correlation layer to compute the similarities between the reference and source features is crucial. Choosing a feature extractor that can produce more representative features significantly enhances the model's performance. Additionally, we have found that the input data used to train the model plays a vital role in ensuring its robustness during evaluation. Simple changes, such as altering the sampling strategy for the source views, have resulted in a significant increase in performance. We also emphasize the importance of normalizing the features before correlation, as it is a straightforward approach to improve the models' performance. A coarse-to-fine architecture using multilevel features has equivalent or better performance than the single stage model with reduced memory requirement.

From a qualitative perspective, we have observed that RobustMVD depth maps are comparatively smoother. We attribute this to the skip connections between the RobustMVD cost volume regularization decoder and the feature extractor. However, both MVSNet and RobustMVD models face challenges in distinguishing finer structures located further away from the camera.

To summarize, the research discussed in this thesis is just a small part of the many methodologies that exist for Multi-View Stereo. Although we have assessed the impact of changing individual elements of a deep learning Multi-View Stereo pipeline for depth estimation in isolation, there is still a lot of research required to assess the impacts of the interplay of two or more elements. A strong performance across various domains can be achieved by using the correct combination of input data, feature extractor, correlation, and cost volume regularization within the network.

7. Acknowledgments

Embarking on this thesis journey would have been a considerably more challenging task without the support and guidance of numerous individuals to whom I owe my deepest gratitude.

First, I express my heartfelt appreciation to Philipp Schröeppel, my advisor. His constant encouragement, insightful feedback, and unwavering faith in my abilities were pivotal in shaping this work. Your mentorship, Philipp, has not only benefitted this thesis but also played a profound role in my personal and academic growth.

I am equally grateful to Prof. Thomas Brox for introducing me to this topic and entrusting me with its exploration. His flexibility and consistent availability, even during the busiest times, provided a reassuring anchor throughout the course of this project. Your dedication to students and passion for the subject have been truly inspiring.

To my sister Shraddha Barke and my mother Alka Barke, your enduring love, patience, and belief in me have been my strength during the highs and lows of this journey. The countless sacrifices you have made and your unwavering support have been the wind under my wings. You reminded me of the larger picture when I lost sight of it and celebrated with me during the milestones. This achievement is as much yours as mine. I would also like to thank my late father Govind Barke, who stood by my decision to quit my old job and pursue higher education in Germany. You are the rock that I stand on.

Lastly, to all those who have supported me directly or indirectly, shared their invaluable insights, or believed in me, I extend my sincere thanks. The culmination of this thesis is a testament to the collective effort of all those mentioned and many others who stood by me during this phase of my life.

A. Appendix

Architectures of Implemented Deep Learning Modules

Operation	Kernel	Stride	Ch I/O	InpRes, (if s=1)	OutRes, (if s=1)	Input	Output
for each view $i = 0, .., k$:							
(1) Pyramid Encoder							
2D convolution + BatchNorm + ReLU	3×3	1	3/8	768×576	384×288 , (768×576)	Image I_i	$conv1a_i$
2D convolution + BatchNorm + ReLU	3×3	1	8/8	384×288, (768×576)	384×288, (768×576)	$conv1a_i$	$conv1b_i$
2D convolution + BatchNorm + ReLU	5×5	2	8/16	384×288, (768×576)	192×144, (384×288)	$conv1b_i$	$conv2a_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	192×144, (384×288)	192×144, (384×288)	$conv2a_i$	$conv2b_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	192×144, (384×288)	192×144, (384×288)	$conv2b_i$	$conv2c_i$
2D convolution + BatchNorm + ReLU	5×5	2	16/32	192×144, (384×288)	96×72, (192×144)	$conv2c_i$	$conv3a_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	96×72, (192×144)	96×72, (192×144)	$conv3a_i$	$conv3b_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	96×72, (192×144)	96×72, (192×144)	$conv3b_i$	$conv3c_i$
(2) Multilevel Features							
2D convolution	1×1	1	32/32	96×72, (192×144)	96×72, (192×144)	$conv3c_i$	$feature2_i$
2D convolution	1×1	1	16/32	192×144, (384×288)	192×144, (384×288)	$conv2c_i$	$lat2$
2D convolution	1×1	1	8/32	384×288, (768×576)	384×288, (768×576)	$conv1b_i$	$lat1$
Upsampling and Addition	-	-	32/32	96×72, 192×144, (192×144, 384×288)	192×144, (384×288)	$upsample(feature2_i) + lat2$	$feature1a_i$
Upsampling and Addition	-	-	32/32	192×144, 384×288, (384×288,768×576)	384×288, (768×576)	$upsample(feature1a_i) + lat1$	$feature0a_i$
2D convolution	3×3	1	32/16	192×144, (384×288)	192×144, (384×288)	$feature1a_i$	$feature1_i$
2D convolution	3×3	1	32/8	384×288, (768×576)	384×288, (768×576)	$feature0a_i$	$feature0_i$

Table 13.: Architecture of the Feature Pyramid Network Feature Extractor. The resolutions given in red are used when the FPN is used to generate multi-level features in the coarse-to-fine architecture. It is possible to use the higher resolutions in the correlation layer as the number of depth hypotheses are [32, 16, 8] from the coarsest to the finest level. This is different from the single stage MVSNet which uses 128 depth hypotheses to construct the correlation volume. This increase in resolution is achieved by changing the stride of the first layer $conv1a$ from 2 to 1.

Operation	Kernel	Stride	Ch I/O	InpRes	OutRes	Input	Output
(1) Downsampling							
for each view $i = 0, .., k$:							
2D convolution + BatchNorm + ReLU	3×3	1	3/8	768×576	768×576	Image I_i	$down1a_i$
2D convolution + BatchNorm + ReLU	3×3	1	8/8	768×576	768×576	$down1a_i$	$down1b_i$
2D Maxpooling + 2D convolution + BatchNorm + ReLU	3×3	1	8/16	384×288	384×288	$down1b_i$	$down2a_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	384×288	384×288	$down2a_i$	$down2b_i$
2D Maxpooling + 2D convolution + BatchNorm + ReLU	3×3	1	16/32	192×144	192×144	$down2b_i$	$3a_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	192×144	192×144	$down3a_i$	$down3b_i$
2D Maxpooling + 2D convolution + BatchNorm + ReLU	3×3	1	32/64	96×72	96×72	$down3b_i$	$down4a_i$
2D convolution + BatchNorm + ReLU	3×3	1	64/64	96×72	96×72	$down4a_i$	$down4b_i$
(2) Upsampling							
Transposed 2D convolution	2×2	2	64/32	96×72	192×144	$down4b_i$	$up1a_i$
2D convolution + BatchNorm + ReLU	3×3	1	64/32	192×144, 192×144	192×144	$concat[up1a_i, down3b_i]$	$up1b_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/32	192×144	192×144	$up1b_i$	$feat2_i$
Transposed 2D convolution	2×2	2	32/16	192×144	384×288	$feat2_i$	$up2a_i$
2D convolution + BatchNorm + ReLU	3×3	1	32/16	384×288, 384×288	384×288	$concat[up2a_i, down2b_i]$	$up2b_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/16	384×288	384×288	$up2b_i$	$feat1_i$
Transposed 2D convolution	2×2	2	16/8	384×288	768×576	$feat1_i$	$up3a_i$
2D convolution + BatchNorm + ReLU	3×3	1	16/8	768×576, 768×576	768×576	$concat[up3a_i, down1b_i]$	$up2b_i$
2D convolution + BatchNorm + ReLU	3×3	1	8/8	768×576	768×576	$up2b_i$	$feat0_i$

Table 14.: Architecture of the UNet Feature Extractor. When using the UNet as a feature extractor for the single stage MVSNet model we use $feat1_i$ as the input to the correlation layer. For the multistage coarse-to-fine architecture we use $feat2_i$ for the coarsest level (level 2), $feat1_i$ for the intermediate level (level 1) and $feat0_i$ for the finest level (level 0).

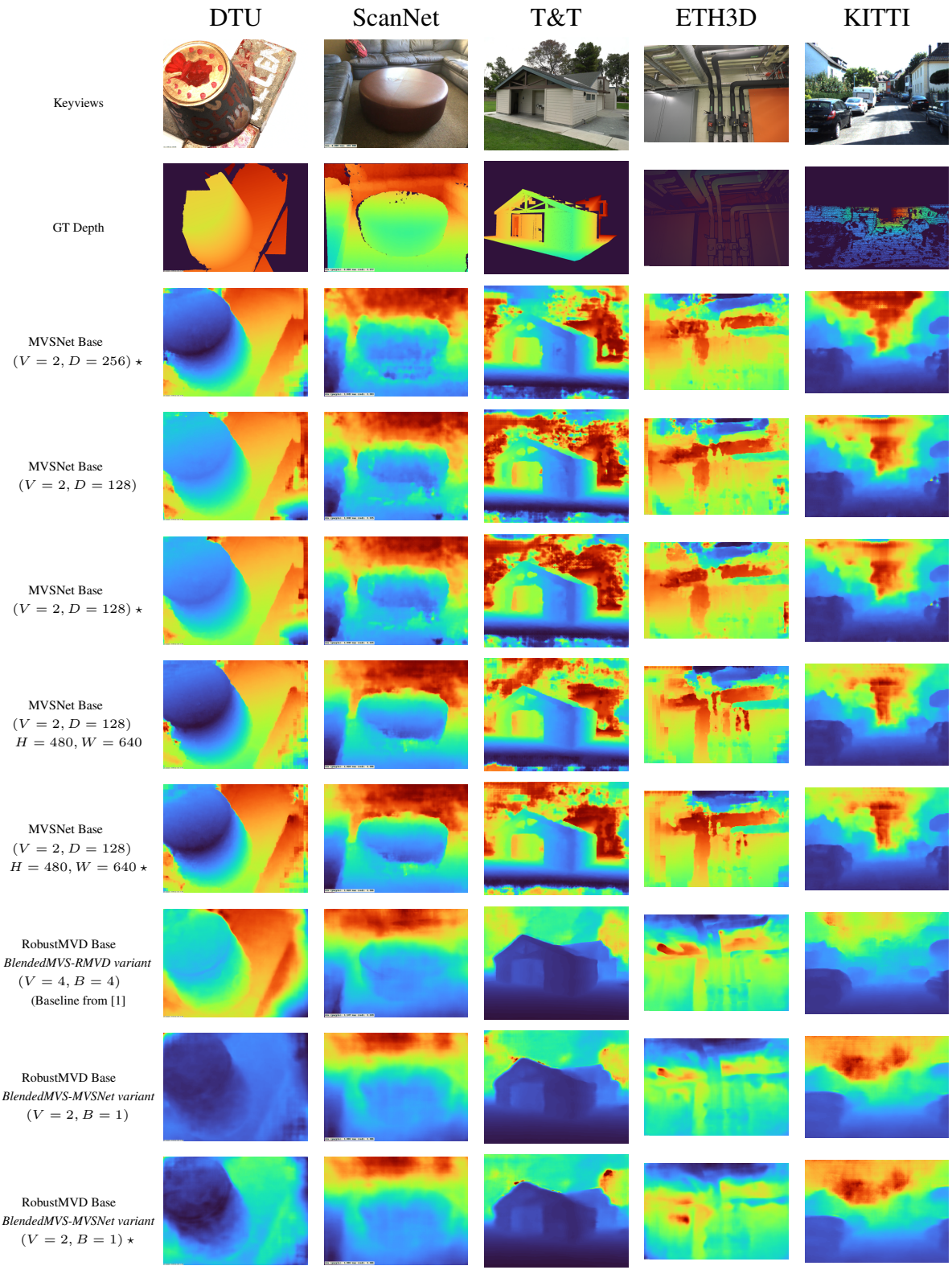
Operation	Kernel	Stride	Ch I/O	InpRes	OutRes	Input	Output
(1) Attention Fusion							
for each feature $f = 0, .., k$:							
2D convolution + BatchNorm + Sigmoid	3×3	1	385/384	$\frac{H}{16}, \frac{W}{16}$	$\frac{H}{16}, \frac{W}{16}$	$concat[features, saliency]$	x_1
2D convolution + BatchNorm + Sigmoid	3×3	1	384/384	$\frac{H}{16}, \frac{W}{16}$	$\frac{H}{16}, \frac{W}{16}$	$[features \times saliency]$	x_2
2D convolution	1×1	1	384/256	$\frac{H}{16}, \frac{W}{16}$	$\frac{H}{16}, \frac{W}{16}$	$[x_1 \times x_2]$	$proj$
(2) Decoder							
Transposed 2D Convolution + Batchnorm + GELU	4×4	2	256/128	$\frac{H}{16}, \frac{W}{16}$	$\frac{H}{8}, \frac{W}{8}$	$proj$	$conv_1^T$
Transposed 2D Convolution + Batchnorm + GELU	4×4	2	128/64	$\frac{H}{8}, \frac{W}{8}$	$\frac{H}{4}, \frac{W}{4}$	$conv_1^T$	$conv_2^T$

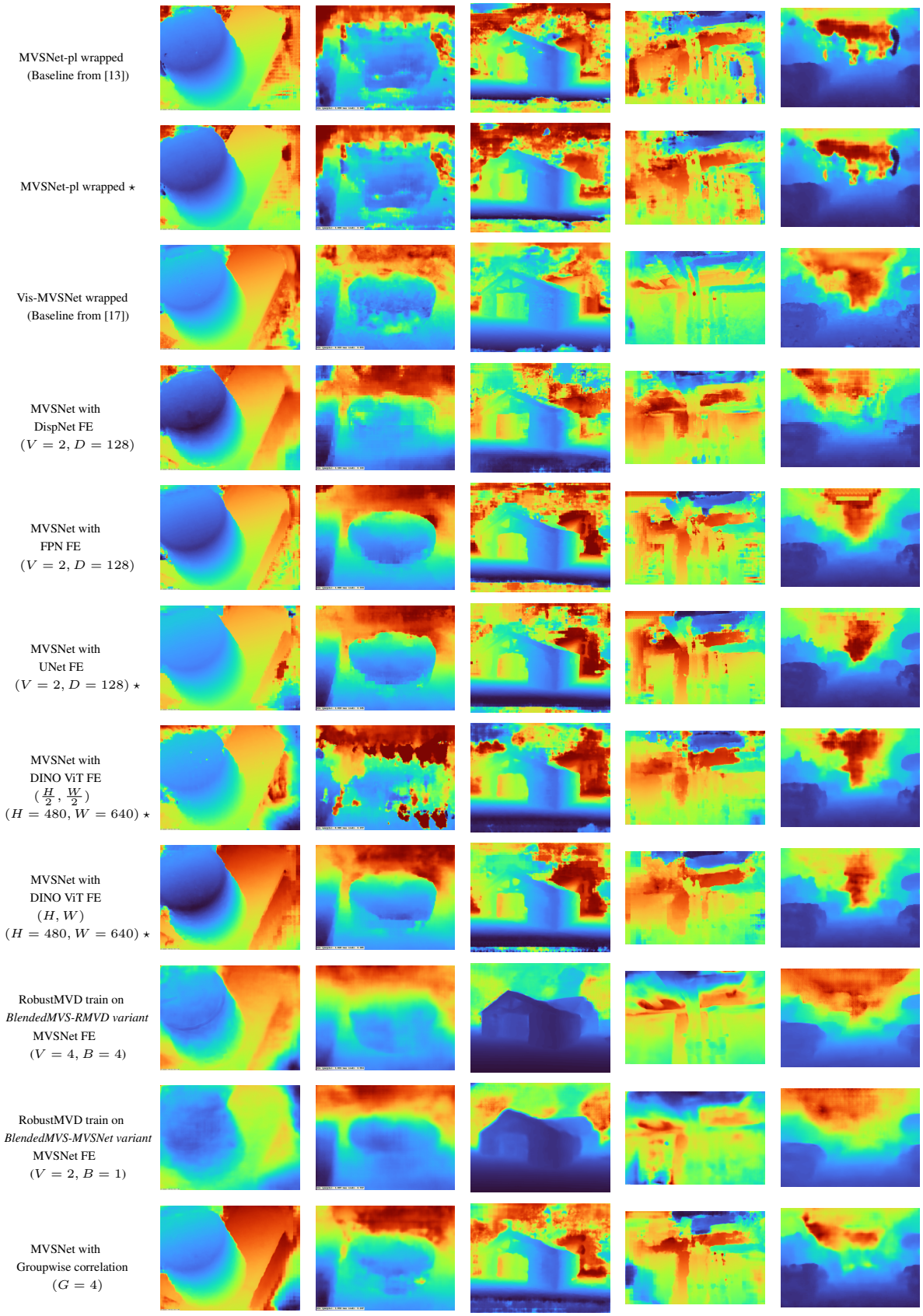
Table 15.: Architecture of the ViT Decoder. The output $conv_2^T$ is concatenated with the learned features of the MVSNet siamese encoder, which also have the dimensions $\frac{H}{4}, \frac{W}{4}$. The concatenated feature map passed as input to the warping operation has 96 channels in total. The pretrained ViT has an output channel dimension d of 64, and the number of attention heads h is 6, making the number of channels in the ViT features 384. The additional single channel is from the saliency map, bringing the total number of channels of input to the Attention Fusion submodule to 385.

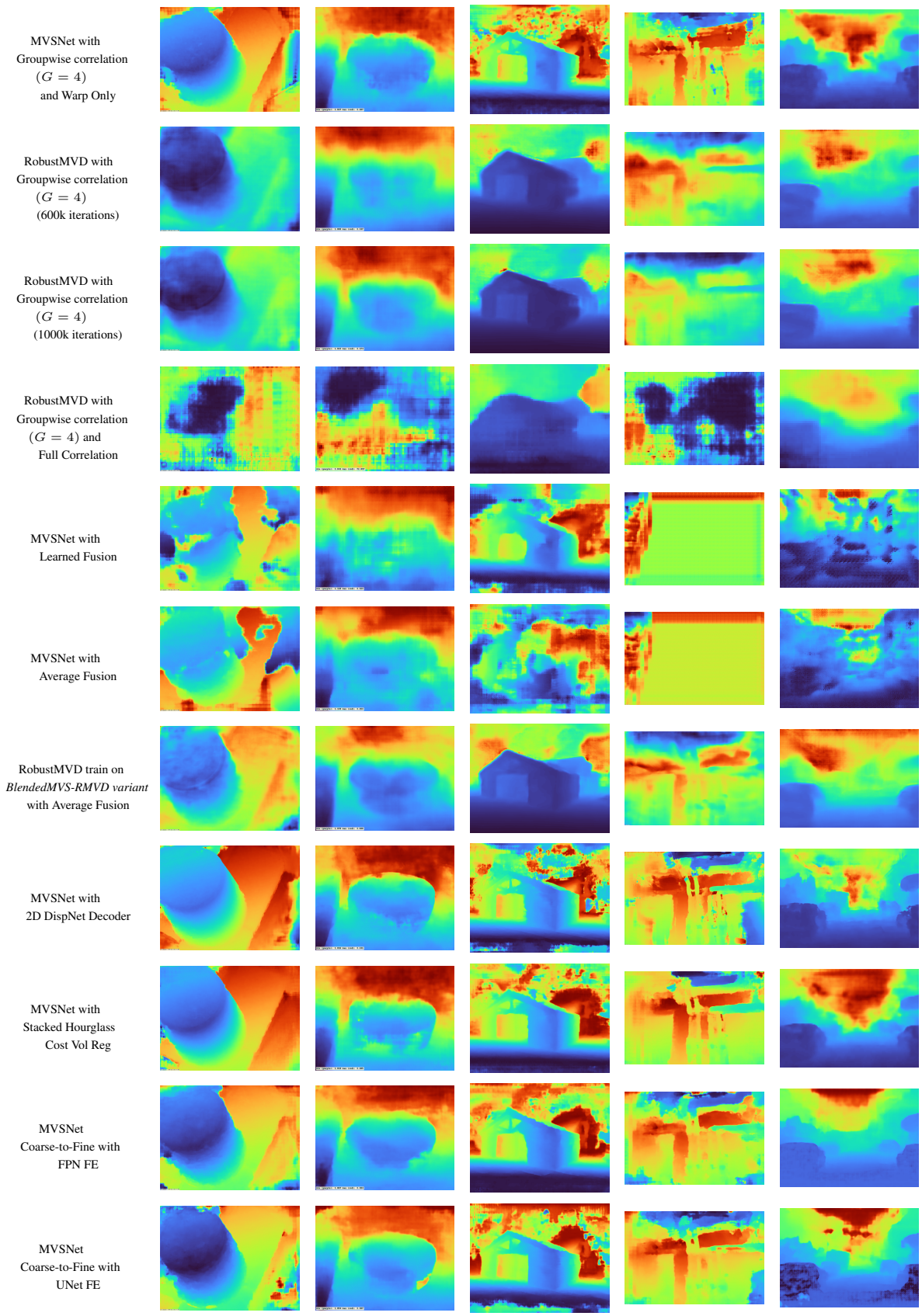
Operation	Kernel	Stride	Ch I/O	InpRes	OutRes	Input	Output
(a) Hourglass 1							
3D convolution + BatchNorm + ReLU	3×3×3	1	32/8	D×192×144	D×192×144	Fused cost volume **C**	$3Dconv_0^a$
3D convolution + BatchNorm + ReLU	3×3×3	2	8/16	D×192×144	$D/2$×96×72	$3Dconv_0^a$	$3Dconv_1^a$
3D convolution + BatchNorm + ReLU	3×3×3	1	16/16	$D/2$×96×72	$D/2$×96×72	$3Dconv_1^a$	$3Dconv_2^a$
3D convolution + BatchNorm + ReLU	3×3×3	2	16/32	$D/2$×96×72	$D/4$×48×36	$3Dconv_2^a$	$3Dconv_3^a$
3D convolution + BatchNorm + ReLU	3×3×3	1	32/32	$D/4$×48×36	$D/4$×48×36	$3Dconv_3^a$	$3Dconv_4^a$
3D convolution + BatchNorm + ReLU	3×3×3	2	32/64	$D/4$×48×36	$D/8$×24×18	$3Dconv_4^a$	$3Dconv_5^a$
3D convolution + BatchNorm + ReLU	3×3×3	1	64/64	$D/8$×24×18	$D/8$×24×18	$3Dconv_5^a$	$3Dconv_6^a$
Transposed 3D convolution + ReLU	3×3×3	2	64/32	$D/8$×24×18	$D/4$×48×36	$3Dconv_6^a$	$3Dconv_7^a$
Transposed 3D convolution + ReLU	3×3×3	2	32/16	$D/4$×48×36	$D/2$×96×72	$3Dconv_7^a + 3Dconv_4^a$	$3Dconv_9^a$
Transposed 3D convolution + ReLU	3×3×3	2	16/8	$D/2$×96×72	D×192×144	$3Dconv_9^a + 3Dconv_2^a$	$3Dconv_1^a1$
3D convolution	3×3×3	1	8/1	D×192×144	D×192×144	$3Dconv_1^a1 + 3Dconv_0^a$	$prob_a$
(b) Hourglass 2							
3D convolution + BatchNorm + ReLU	3×3×3	1	32/8	D×192×144	D×192×144	$3Dconv_1^a1 + 3Dconv_0^a$	$3Dconv_0^b$
3D convolution + BatchNorm + ReLU	3×3×3	2	8/16	D×192×144	$D/2$×96×72	$3Dconv_0^b$	$3Dconv_1^b$
3D convolution + BatchNorm + ReLU	3×3×3	1	16/16	$D/2$×96×72	$D/2$×96×72	$3Dconv_1^b$	$3Dconv_2^b$
3D convolution + BatchNorm + ReLU	3×3×3	2	16/32	$D/2$×96×72	$D/4$×48×36	$3Dconv_2^b$	$3Dconv_3^b$
3D convolution + BatchNorm + ReLU	3×3×3	1	32/32	$D/4$×48×36	$D/4$×48×36	$3Dconv_3^b$	$3Dconv_4^b$
3D convolution + BatchNorm + ReLU	3×3×3	2	32/64	$D/4$×48×36	$D/8$×24×18	$3Dconv_4^b$	$3Dconv_5^b$
3D convolution + BatchNorm + ReLU	3×3×3	1	64/64	$D/8$×24×18	$D/8$×24×18	$3Dconv_5^b$	$3Dconv_6^b$
Transposed 3D convolution + ReLU	3×3×3	2	64/32	$D/8$×24×18	$D/4$×48×36	$3Dconv_6^b$	$3Dconv_7^b$
Transposed 3D convolution + ReLU	3×3×3	2	32/16	$D/4$×48×36	$D/2$×96×72	$3Dconv_7^b + 3Dconv_4^b$	$3Dconv_9^b$
Transposed 3D convolution + ReLU	3×3×3	2	16/8	$D/2$×96×72	D×192×144	$3Dconv_9^b + 3Dconv_2^b$	$3Dconv_1^b1$
3D convolution	3×3×3	1	8/1	D×192×144	D×192×144	$3Dconv_1^b1 + 3Dconv_0^b$	$prob_b$
(c) Hourglass 3							
3D convolution + BatchNorm + ReLU	3×3×3	1	32/8	D×192×144	D×192×144	$3Dconv_1^b1 + 3Dconv_0^b$	$3Dconv_0^c$
3D convolution + BatchNorm + ReLU	3×3×3	2	8/16	D×192×144	$D/2$×96×72	$3Dconv_0^c$	$3Dconv_1^c$
3D convolution + BatchNorm + ReLU	3×3×3	1	16/16	$D/2$×96×72	$D/2$×96×72	$3Dconv_1^c$	$3Dconv_2^c$
3D convolution + BatchNorm + ReLU	3×3×3	2	16/32	$D/2$×96×72	$D/4$×48×36	$3Dconv_2^c$	$3Dconv_3^c$
3D convolution + BatchNorm + ReLU	3×3×3	1	32/32	$D/4$×48×36	$D/4$×48×36	$3Dconv_3^c$	$3Dconv_4^c$
3D convolution + BatchNorm + ReLU	3×3×3	2	32/64	$D/4$×48×36	$D/8$×24×18	$3Dconv_4^c$	$3Dconv_5^c$
3D convolution + BatchNorm + ReLU	3×3×3	1	64/64	$D/8$×24×18	$D/8$×24×18	$3Dconv_5^c$	$3Dconv_6^c$
Transposed 3D convolution + ReLU	3×3×3	2	64/32	$D/8$×24×18	$D/4$×48×36	$3Dconv_6^c$	$3Dconv_7^c$
Transposed 3D convolution + ReLU	3×3×3	2	32/16	$D/4$×48×36	$D/2$×96×72	$3Dconv_7^c + 3Dconv_4^c$	$3Dconv_9^c$
Transposed 3D convolution + ReLU	3×3×3	2	16/8	$D/2$×96×72	D×192×144	$3Dconv_9^c + 3Dconv_2^c$	$3Dconv_1^c1$
3D convolution	3×3×3	1	8/1	D×192×144	D×192×144	$3Dconv_1^c1 + 3Dconv_0^c$	$prob_c$

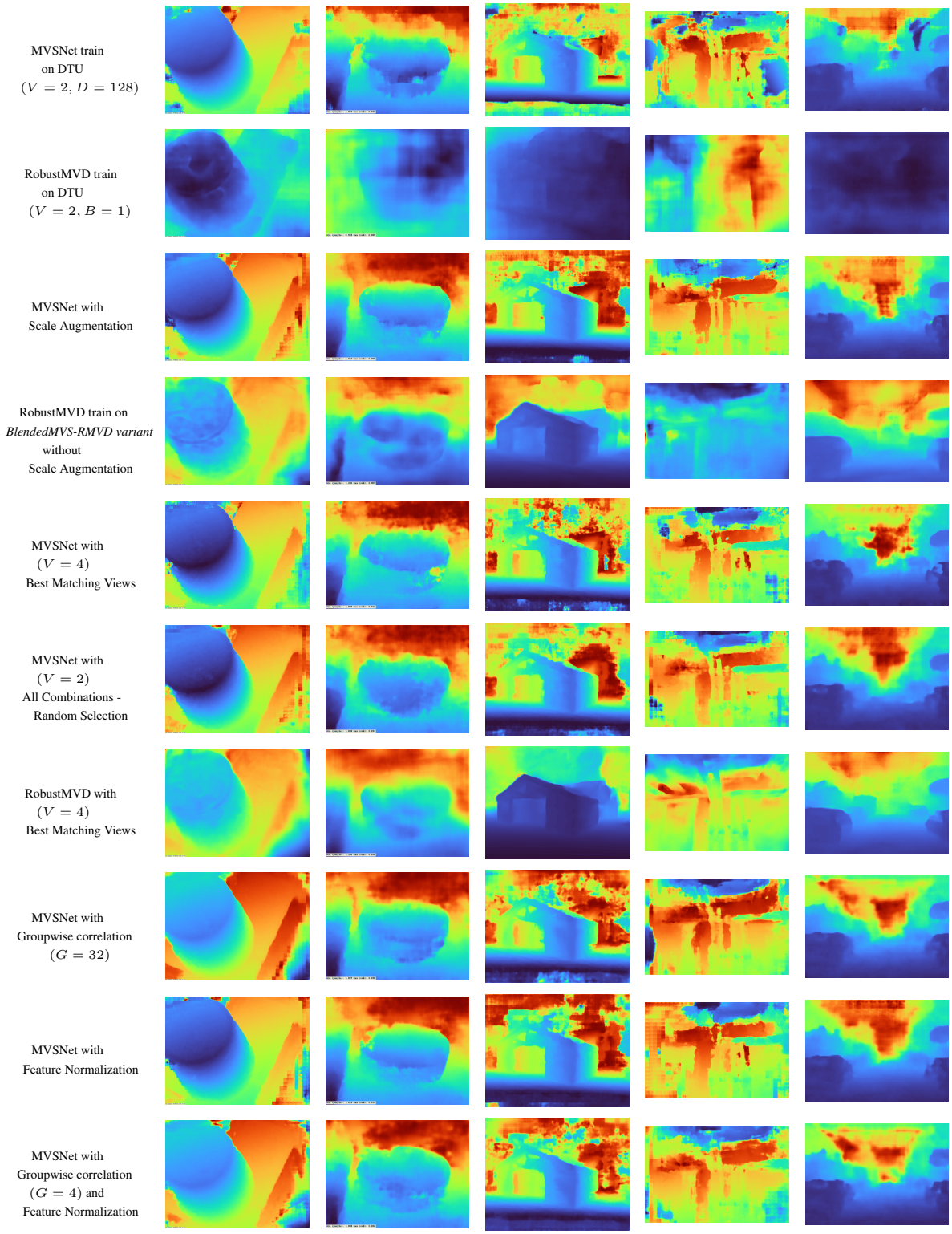
Table 16.: Architecture of the 3D stacked hourglass cost volume regularization module. A depth map is regressed at every level of the stacked hourglass from the probability volumes $prob_a, prob_b, prob_c$. These depth maps are then used to compute the combined loss with weights $\lambda_i = [\frac{1}{2}, 1, 2]$.

Qualitatives: Depth




MVSNet-pl wrapped
(Baseline from [13])
MVSNet-pl wrapped ⋆
Vis-MVSNet wrapped
(Baseline from [17])
MVSNet with
DispNet FE
$(V = 2, D = 128)$
MVSNet with
FPN FE
$(V = 2, D = 128)$
MVSNet with
UNet FE
$(V = 2, D = 128)$ ⋆
MVSNet with
DINO ViT FE
$(\frac{H}{2}, \frac{W}{2})$
$(H = 480, W = 640)$ ⋆
MVSNet with
DINO ViT FE
(H, W)
$(H = 480, W = 640)$ ⋆
RobustMVD train on
BlendedMVS-RMVD variant
MVSNet FE
$(V = 4, B = 4)$
RobustMVD train on
BlendedMVS-MVSNet variant
MVSNet FE
$(V = 2, B = 1)$
MVSNet with
Groupwise correlation
$(G = 4)$


MVSNet with Groupwise correlation ($G = 4$) and Warp Only
RobustMVD with Groupwise correlation ($G = 4$) (600k iterations)
RobustMVD with Groupwise correlation ($G = 4$) (1000k iterations)
RobustMVD with Groupwise correlation ($G = 4$) and Full Correlation
MVSNet with Learned Fusion
MVSNet with Average Fusion
RobustMVD train on *BlendedMVS-RMVD variant* with Average Fusion
MVSNet with 2D DispNet Decoder
MVSNet with Stacked Hourglass Cost Vol Reg
MVSNet Coarse-to-Fine with FPN FE
MVSNet Coarse-to-Fine with UNet FE


MVSNet train on DTU ($V = 2, D = 128$)
RobustMVD train on DTU ($V = 2, B = 1$)
MVSNet with Scale Augmentation
RobustMVD train on *BlendedMVS-RMVD variant* without Scale Augmentation
MVSNet with ($V = 4$) Best Matching Views
MVSNet with ($V = 2$) All Combinations - Random Selection
RobustMVD with ($V = 4$) Best Matching Views
MVSNet with Groupwise correlation ($G = 32$)
MVSNet with Feature Normalization
MVSNet with Groupwise correlation ($G = 4$) and Feature Normalization

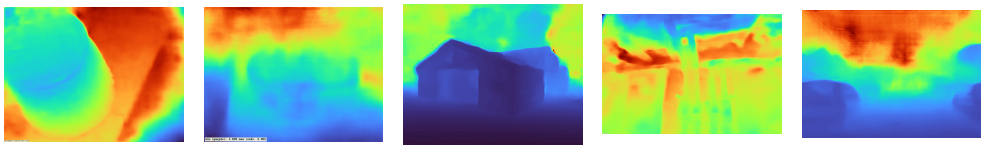
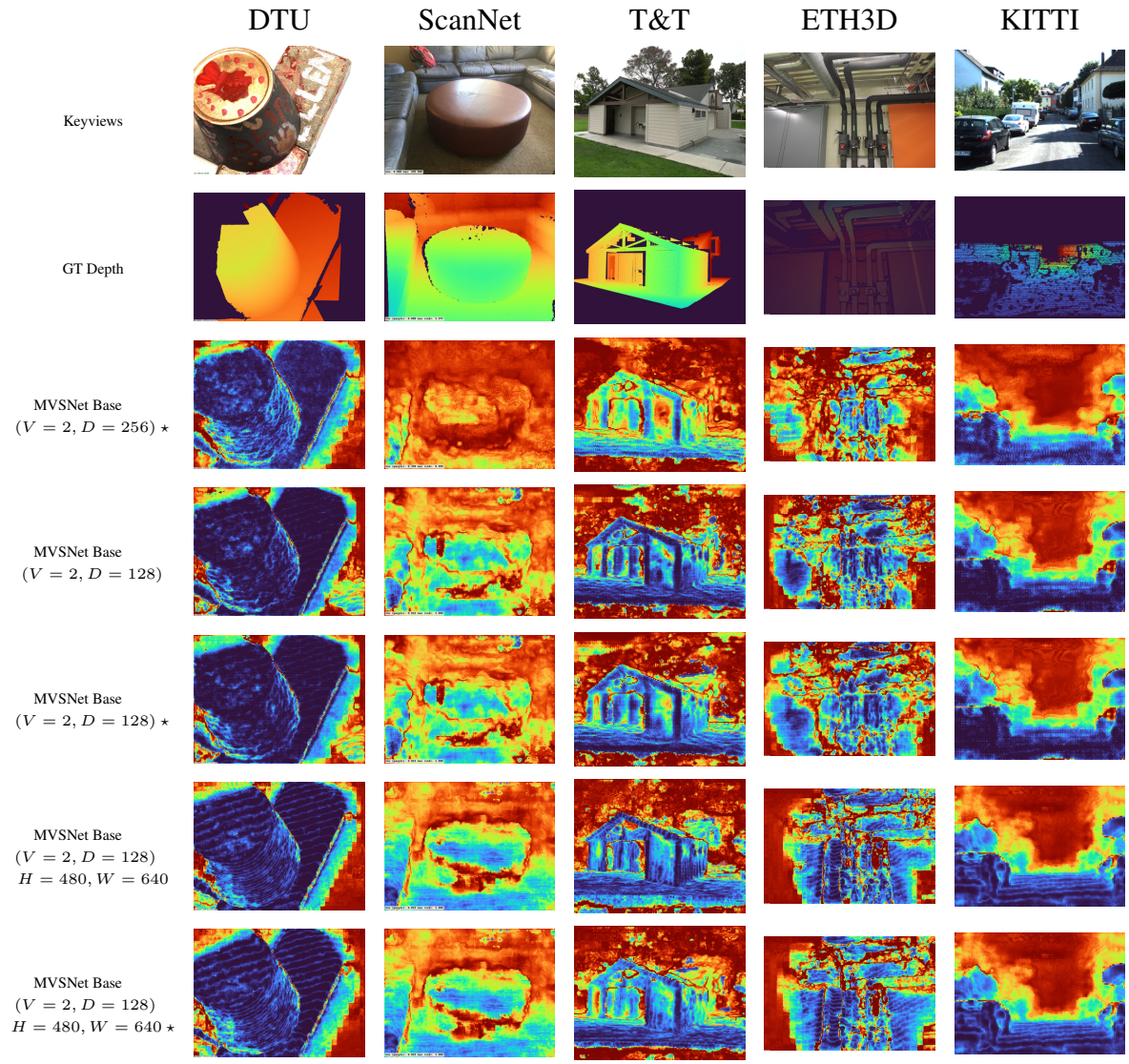
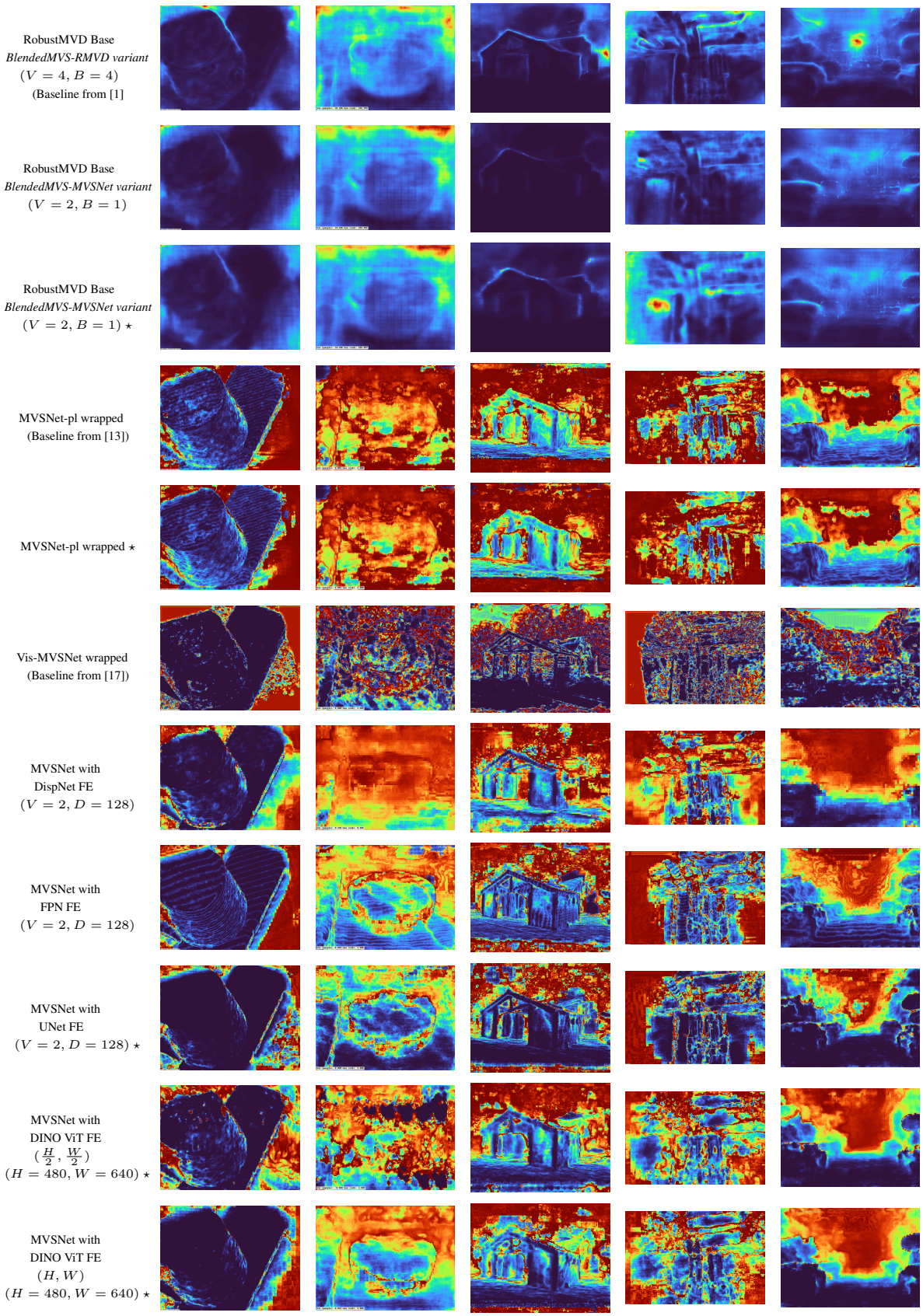
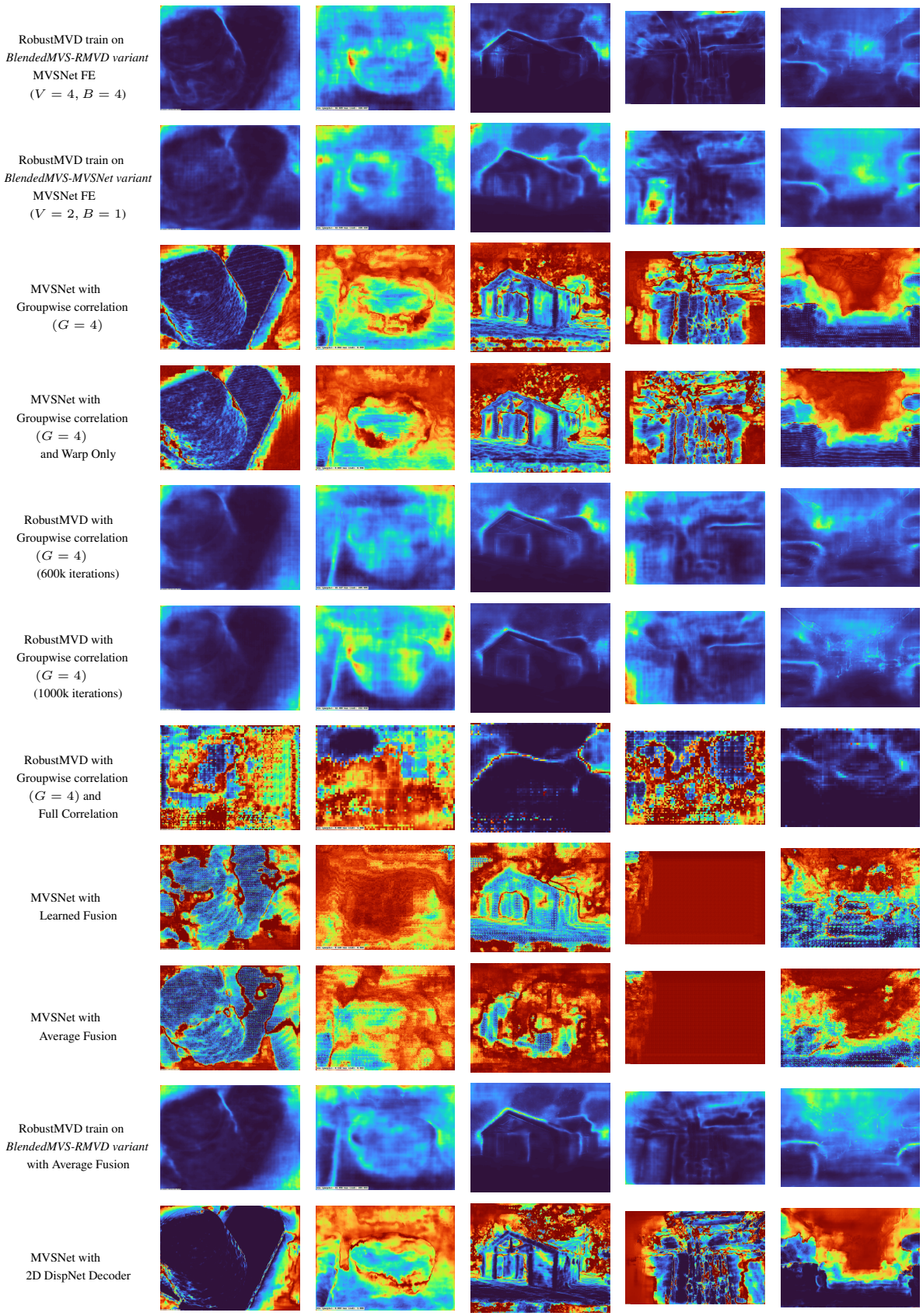


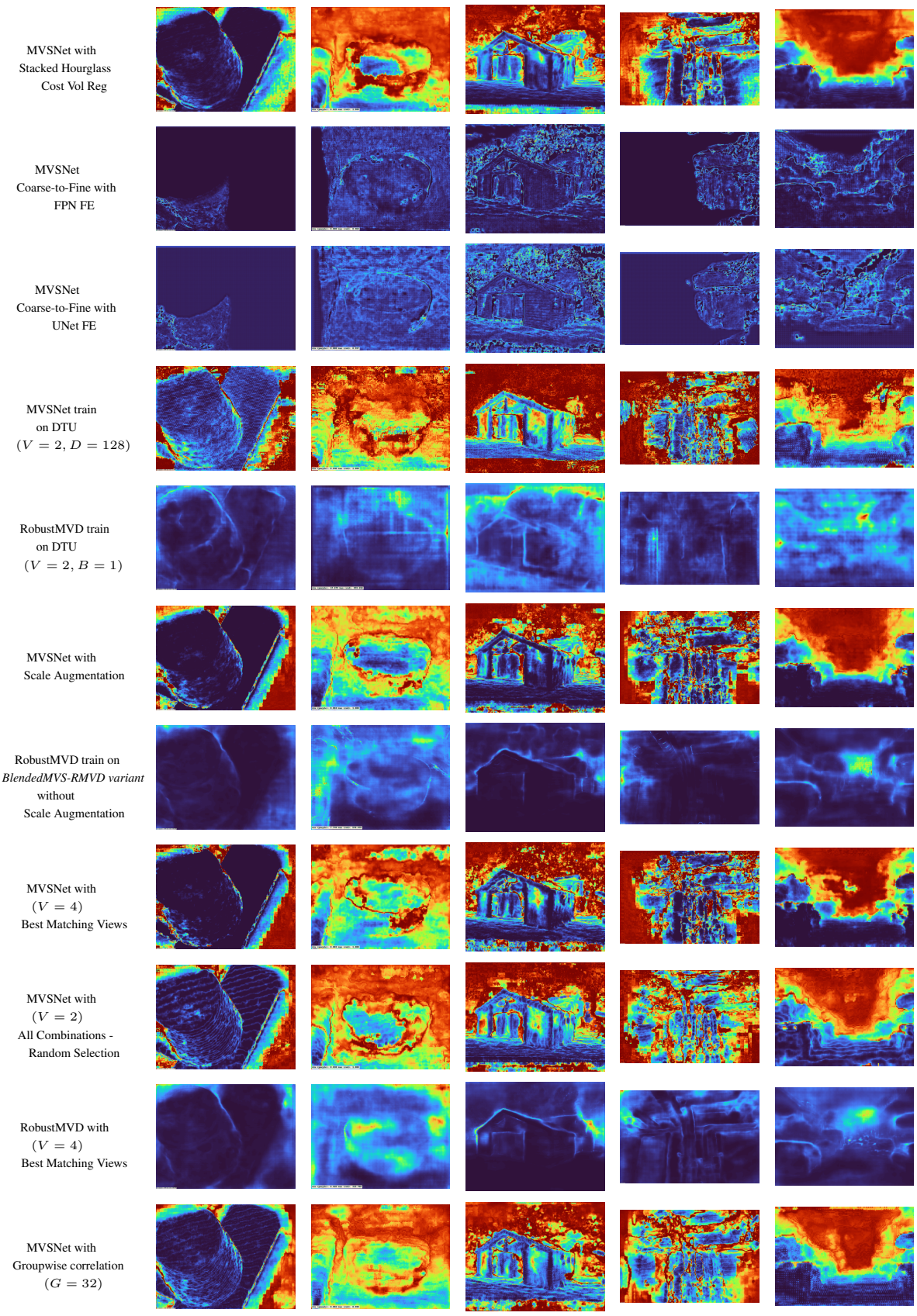
Table 17.: Qualitative Analysis: Depth Predictions These depth maps correspond to the quantitative results in the Experiments chapter. We have maintained the same order as that of the experiments to facilitate easier comparison. The pixels shaded in red are farther away compared to the nearby pixels, which are shaded in blue.

Qualitatives: Uncertainty




RobustMVD Base
BlendedMVS-RMVD variant
$(V = 4, B = 4)$
(Baseline from [1]
RobustMVD Base
BlendedMVS-MVSNet variant
$(V = 2, B = 1)$
RobustMVD Base
BlendedMVS-MVSNet variant
$(V = 2, B = 1)$ ⋆
MVSNet-pl wrapped
(Baseline from [13])
MVSNet-pl wrapped ⋆
Vis-MVSNet wrapped
(Baseline from [17])
MVSNet with
DispNet FE
$(V = 2, D = 128)$
MVSNet with
FPN FE
$(V = 2, D = 128)$
MVSNet with
UNet FE
$(V = 2, D = 128)$ ⋆
MVSNet with
DINO ViT FE
$(\frac{H}{2}, \frac{W}{2})$
$(H = 480, W = 640)$ ⋆
MVSNet with
DINO ViT FE
(H, W)
$(H = 480, W = 640)$ ⋆


RobustMVD train on *BlendedMVS-RMVD variant* MVSNet FE ($V = 4$, $B = 4$)
RobustMVD train on *BlendedMVS-MVSNet variant* MVSNet FE ($V = 2$, $B = 1$)
MVSNet with Groupwise correlation ($G = 4$)
MVSNet with Groupwise correlation ($G = 4$) and Warp Only
RobustMVD with Groupwise correlation ($G = 4$) (600k iterations)
RobustMVD with Groupwise correlation ($G = 4$) (1000k iterations)
RobustMVD with Groupwise correlation ($G = 4$) and Full Correlation
MVSNet with Learned Fusion
MVSNet with Average Fusion
RobustMVD train on *BlendedMVS-RMVD variant* with Average Fusion
MVSNet with 2D DispNet Decoder


MVSNet with Stacked Hourglass Cost Vol Reg
MVSNet Coarse-to-Fine with FPN FE
MVSNet Coarse-to-Fine with UNet FE
MVSNet train on DTU ($V = 2, D = 128$)
RobustMVD train on DTU ($V = 2, B = 1$)
MVSNet with Scale Augmentation
RobustMVD train on *BlendedMVS-RMVD variant* without Scale Augmentation
MVSNet with ($V = 4$) Best Matching Views
MVSNet with ($V = 2$) All Combinations - Random Selection
RobustMVD with ($V = 4$) Best Matching Views
MVSNet with Groupwise correlation ($G = 32$)

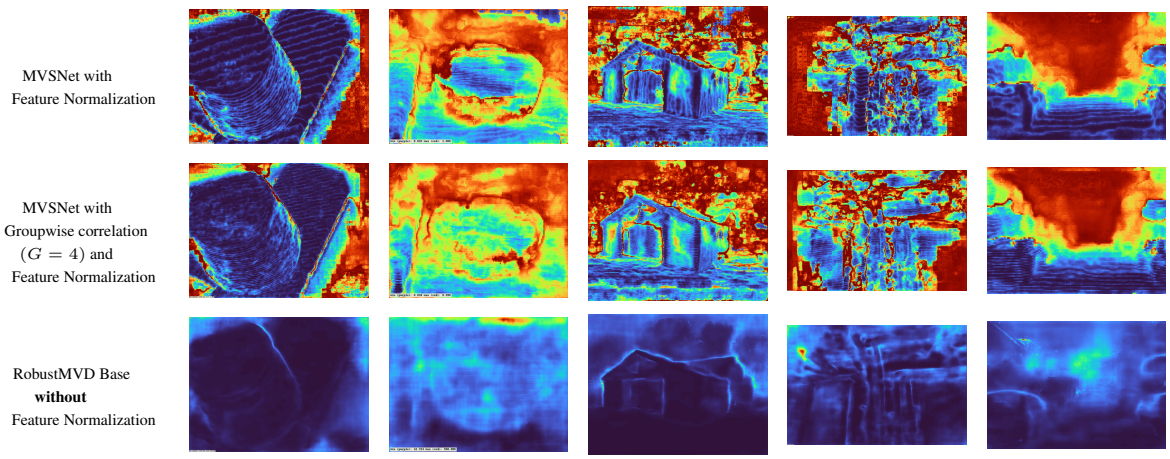


Table 18.: Qualitative Analysis: Uncertainty Estimates. These uncertainty maps correspond to the quantitative results in the Experiments chapter. We have maintained the same order as that of the experiments to facilitate easier comparison. The pixels shaded in red indicate a high level of uncertainty, and blue indicates a low uncertainty.

Bibliography

[1] P. Schröppel, J. Bechtold, A. Amiranashvili, and T. Brox, "A benchmark and a baseline for robust multi-view depth estimation," 2022.

[2] S. M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski, "A comparison and evaluation of multi-view stereo reconstruction algorithms," in *CVPR*, 2006.

[3] Y. Bengio and Y. Lecun, "Convolutional networks for images, speech, and time-series," 11 1997.

[4] 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.

[5] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," 2021.

[6] S. Amir, Y. Gandelsman, S. Bagon, and T. Dekel, "Deep vit features as dense visual descriptors," *arXiv preprint arXiv:2112.05814*, 2021.

[7] E. Ilg, Ö. Çiçek, S. Galesso, A. Klein, O. Makansi, F. Hutter, and T. Brox, "Uncertainty estimates and multi-hypotheses networks for optical flow," in *ECCV*, 2018.

[8] R. Szeliski, *Computer Vision: Algorithms and Applications*. Berlin, Heidelberg: Springer-Verlag, 1st ed., 2010.

[9] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[10] Y. Furukawa and C. Hernández, "Multi-view stereo: A tutorial," *Found. Trends. Comput. Graph. Vis.*, vol. 9, p. 1–148, June 2015.

[11] M. Okutomi and T. Kanade, "A multiple-baseline stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 353–363, 1993.

[12] H. Laga, L. V. Jospin, F. Boussaid, and M. Bennamoun, "A survey on deep learning techniques for stereo-based depth estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, pp. 1738–1764, apr 2022.

[13] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "MVSNet: Depth inference for unstructured multi-view stereo," *ECCV*, September 2018.

[14] S. Cheng, Z. Xu, S. Zhu, Z. Li, L. E. Li, R. Ramamoorthi, and H. Su, "Deep stereo using adaptive thin volume representation with uncertainty awareness," 2020.

[15] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, "Cascade cost volume for high-resolution multi-view stereo and stereo matching," in *CVPR*, 2020.

[16] J. Yang, W. Mao, J. M. Alvarez, and M. Liu, "Cost volume pyramid based depth inference for multi-view stereo," in *CVPR*, June 2020.

[17] J. Zhang, Y. Yao, S. Li, Z. Luo, and T. Fang, "Visibility-aware multi-view stereo network," *BMVC*, 2020.

[18] Y. Yao, Z. Luo, S. Li, T. Shen, T. Fang, and L. Quan, "Recurrent mvsnet for high-resolution multi-view stereo depth inference," *CVPR*, 2019.

[19] R. T. Collins, "A space-sweep approach to true multi-image matching," *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 358–363, 1996.

[20] Q. Zhu, C. Min, Z. Wei, Y. Chen, and G. Wang, "Deep learning for multi-view stereo via plane sweep: A survey," 2021.

[21] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *CVPR*, June 2016.

[22] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[23] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," 2021.

[24] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," 2016.

[25] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2, pp. 1150–1157, Ieee, 1999.

[26] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006* (A. Leonardis, H. Bischof, and A. Pinz, eds.), (Berlin, Heidelberg), pp. 404–417, Springer Berlin Heidelberg, 2006.

[27] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," pp. 2564–2571, 11 2011.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[30] J. Zbontar and Y. LeCun, "Stereo matching by training a convolutional neural network to compare image patches," *Journal of Machine Learning Research*, vol. 17, 2016.

[31] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2017.

[32] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *CVPR*, 2018.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Computer Vision – ECCV 2014*, pp. 346–361, Springer International Publishing, 2014.

[34] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016.

[35] C. Cao, X. Ren, and Y. Fu, "Mvsformer: Multi-view stereo by learning robust image features and temperature-based depth," 2022.

[36] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 807–814 vol. 2, 2005.

[37] J. Pang, W. Sun, J. S. Ren, C. Yang, and Q. Yan, "Cascade residual learning: A two-stage convolutional neural network for stereo matching," in *ICCV*, 2017.

[38] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry, "End-to-end learning of geometry and context for deep stereo regression," in *ICCV*, 2017.

[39] S. Im, H.-G. Jeon, S. Lin, and I. S. Kweon, "Dpsnet: End-to-end deep plane sweep stereo," 2019.

[40] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume," in *CVPR*, 2018.

[41] G. Yang, J. Manela, M. Happold, and D. Ramanan, "Hierarchical deep stereo matching on high-resolution images," 2019.

[42] X. Guo, K. Yang, W. Yang, X. Wang, and H. Li, "Group-wise correlation stereo network," in *CVPR*, 2019.

[43] Z. Shen, Y. Dai, and Z. Rao, "Cfnet: Cascade and fused cost volume for robust stereo matching," 2021.

[44] F. Zhang, V. A. Prisacariu, R. Yang, and P. H. S. Torr, "Ga-net: Guided aggregation net for end-to-end stereo matching," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 185–194, 2019.

[45] H. Xu and J. Zhang, "Aanet: Adaptive aggregation network for efficient stereo matching," 2020.

[46] J. L. Schönberger and J.-M. Frahm, "Structure-from-Motion Revisited," in *CVPR*, 2016.

[47] J. L. Schönberger, E. Zheng, M. Pollefeys, and J.-M. Frahm, "Pixelwise View Selection for Unstructured Multi-View Stereo," in *ECCV*, 2016.

[48] M. Ji, J. Gall, H. Zheng, Y. Liu, and L. Fang, "SurfaceNet: An end-to-end 3d neural network for multiview stereopsis," in *2017 IEEE International Conference on Computer Vision (ICCV)*, IEEE, oct 2017.

[49] A. Kar, C. Häne, and J. Malik, "Learning a multi-view stereo machine," 2017.

[50] D. Paschalidou, A. O. Ulusoy, C. Schmitt, L. van Gool, and A. Geiger, "Raynet: Learning volumetric 3d reconstruction with ray potentials," 2019.

[51] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "DeepMVS: Learning multi-view stereopsis," in *CVPR*, June 2018.

[52] Q. Xu and W. Tao, "PVSNet: Pixelwise visibility-aware multi-view stereo network," *arXiv preprint arXiv:2007.07714*, 2020.

[53] R. Peng, R. Wang, Z. Wang, Y. Lai, and R. Wang, "Rethinking depth estimation for multi-view stereo: A unified representation," 2022.

[54] Y. Yao, Z. Luo, S. Li, J. Zhang, Y. Ren, L. Zhou, T. Fang, and L. Quan, "BlendedMVS: A large-scale dataset for generalized multi-view stereo networks," *CVPR*, 2020.

[55] R. Jensen, A. Dahl, G. Vogiatzis, E. Tola, and H. Aanæs, "Large scale multi-view stereopsis evaluation," in *CVPR*, 2014.

[56] H. Aanæs, R. R. Jensen, G. Vogiatzis, E. Tola, and A. B. Dahl, "Large-scale data for multiple-view stereopsis," *International Journal of Computer Vision*, 2016.

[57] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, 2013.

[58] J. Uhrig, N. Schneider, L. Schneider, U. Franke, T. Brox, and A. Geiger, "Sparsity invariant CNNs," in *3DV*, October 2017.

[59] T. Schöps, J. L. Schönberger, S. Galliani, T. Sattler, K. Schindler, M. Pollefeys, and A. Geiger, "A multi-view stereo benchmark with high-resolution images and multi-camera videos," in *CVPR*, July 2017.

[60] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and Temples: Benchmarking large-scale scene reconstruction," *ACM Transactions on Graphics*, vol. 36, no. 4, 2017.

[61] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-annotated 3d reconstructions of indoor scenes," in *CVPR*, July 2017.

[62] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *NeurIPS*, January 2014.

[63] C. Tang and P. Tan, "BA-Net: Dense bundle adjustment networks," in *ICLR*, May 2019.

[64] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *ICCV*, 2015.

[65] M. Lin, Q. Chen, and S. Yan, "Network in network," 2014.

[66] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

[67] W. Hartmann, S. Galliani, M. Havlena, L. Van Gool, and K. Schindler, "Learned multi-patch similarity," in *CVPR*, 2017.

[68] Z. Zhang, R. Peng, Y. Hu, and R. Wang, "Geomvsnet: Learning multi-view stereo with geometry perception," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21508–21518, June 2023.