Lightweight Deep Learning for Monocular Depth Estimation

by

Tim Heydrich
Lakehead University

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Lightweight Deep Learning for Monocular Depth Estimation

by

Tim Heydrich
Lakehead University

Supervisory Committee

Dr. Shan Du, Supervisor
(Department of Computer Science, Mathematics, Physics and Statistics, The University of British Columbia Okanagan, Canada)

Dr. Yimin Yang, Co-Supervisor
(Department of Computer Science, Lakehead University, ON, Canada)

Amin Safaei, Departmental Member
(Department of Computer Science, Lakehead University, ON, Canada)

Dr. Thangarajah Akilan , External Member
(Department of Software Engineering, , Lakehead University, ON, Canada)

ABSTRACT

Monocular depth estimation is a challenging but significant part of computer vision with many applications in other areas of study. This estimation method aims to provide a relative depth prediction for a single input image. In the past, conventional methods have been able to give rough depth estimations however their accuracies were not sufficient. In recent years, due to the rise of deep convolutional neural networks (DCNNs), the accuracy of the depth estimations has increased. However, DCNNs do so at the expense of compute resources and time. This leads to the need for more lightweight solutions for the task.

In this thesis, we use recent advances made in lightweight network design to reduce complexity. Furthermore, we use conventional methods to increase the performance of lightweight networks. Specifically, we propose a novel lightweight network architecture which has a significantly reduced complexity compared to current methods while still maintaining a competitive accuracy. We propose an encoder-decoder architecture that utilizes DiCE units [47] to reduce the complexity of the encoder. In addition, we utilize a custom designed decoder based on depthwise-separable convolutions. Furthermore, we propose a novel lightweight self-supervised training framework which leverages conventional methods to remove the need for pose estimation that current self-supervised methods have. Similar to current unsupervised and self-supervised methods, out method needs a pair of stereo images during training. However, we take advantage of this need to compute a ground truth approximation. Doing this we are able to eliminate the need for pose estimation that other self-supervised approaches have. Both our lightweight network and our self-supervised framework reduce the size and complexity of current state-of-the-art methods while maintaining competitive results in their respective areas.

ACKNOWLEDGEMENTS

"Knowing where you came from is no less important than knowing where you are going."
[62]

- Neil DeGrasse Tyson

I would like to express my sincerest thanks and gratitude to the following people:

First of all, my supervisor **Dr. Shan Du**, for her knowledge, patience and guidance through my Master's. She provided me with the opportunity to explore my own research interests and always showed great interest in my work. I will always appreciate the constructive suggestions she provided me with and her continued support. Her immense knowledge and insights helped me overcome any difficulties I faced. I am truly privileged to have studied under her excellent supervision.

My co-supervisor, **Dr. Yimin Yang**, for his support and suggestions during my Master's study. He gave me the knowledge and tools to excel and improve my understanding of deep learning. I am thankful to have been his student and to have had him as my co-supervisor.

I would also like to thank my family and friends for their continued love and support. Their encouragement and understanding have helped me to complete my goals.

Last but not least I would like to thank the following founding sources for their financial support of my research:

- **Dr. Shan Du (Research Grants)**
- **Lakehead University Faculty of Graduate Studies**
- **Lakehead University Faculty of Science and Environmental Studies**
- **Toronto Vector Institute**

Publications

Tim Heydrich, Shan Du, Yimin Yang, Xiangyu Ma, Yu Liu. A Novel Lightweight Network for Fast Monocular Depth Estimation, in preparation (towards IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)), 2022

Tim Heydrich, Shan Du, Yimin Yang. A Lightweight Self-Supervised Training Framework for Monocular Depth Estimation, in preparation (towards IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)), 2022

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

Depth estimation has long been an area of interest in computer vision and other areas of study. Depth estimation produces a depth map which gives the relative distance from the camera to objects in the scene. These relative distances are displayed as grey scale images where a lighter tone represents objects closer to the camera and darker tones represent objects that are further away. Monocular Depth Estimation (MDE), depth from a single image, in particular has a wide range of applications, from robotics to scene reconstruction [41] and augmented reality (AR) [23]. In particular, these depth maps can be used for object avoidance, placing objects or characters in an AR environment or simply to create a 3D representation of a scene. Examples for pairs of captured RGB images and resulting depth estimations can be seen in Figure 1.1. In the figure there are two sample scenes taken from the NYUv2 dataset [56]. On the left are the captured RGB images and on the right are the corresponding

depth maps for each of the two scenes. In this example the grey scale of the depth estimation is given as a fake colour transition as it provides better visual contrast. Using this fake colour mapping the darker portions of the image are in the foreground and the light green portions more toward the back of the room.

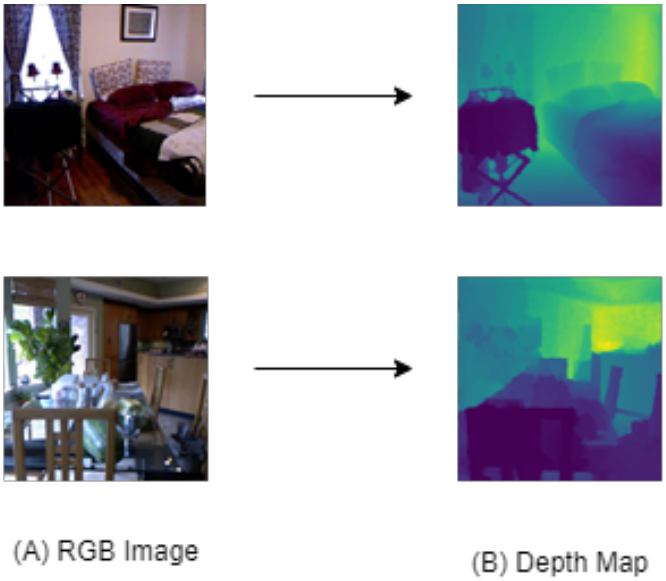

Figure 1.1: Example of depth maps. (A) shows the captured RGB images. (B) represents the corresponding depth maps. Grey scale is given fake colour transition to give better visual contrast.

Depth maps can be challenging to obtain. There are two main ways which can be used to obtain depth maps, the first is the use of sophisticated hardware and the second are depth estimation algorithms. While hardware based solutions are generally able to capture high quality depth maps they tend to be expensive and not feasible for most application. The most effective piece of hardware to acquire depth maps is a Light Detection and Ranging (LiDAR) sensor. LiDAR works by targeting objects with a laser and measuring the time it takes for the reflected light to return to the sensor. These sensors are highly effective at producing depth maps of a given scene. The downside is the cost, size and power consumption of the sensor. Furthermore, LiDAR and other hardware based solutions are often not usable for many applications, for example, when the goal is to create an AR application for mobile devices each device would need a LiDAR sensor to be able to use the app. Only a select few mobile device have integrated LiDAR sensor making the utilization of such hardware unfeasible. If the utilization of hardware based depth estimation

is an option it should be preferred as it tends to give higher resolution and more accurate depth estimations.

However, a much more feasible solution for most real world situations is the utilization of depth estimation algorithms. These algorithms utilize various image cues to provide a depth estimation. These depth estimations tend to be less accurate than their hardware counterparts but the algorithms have a much wider range in terms of target devices and applications. In this thesis, we mainly focus on different methods of algorithmic depth estimation.

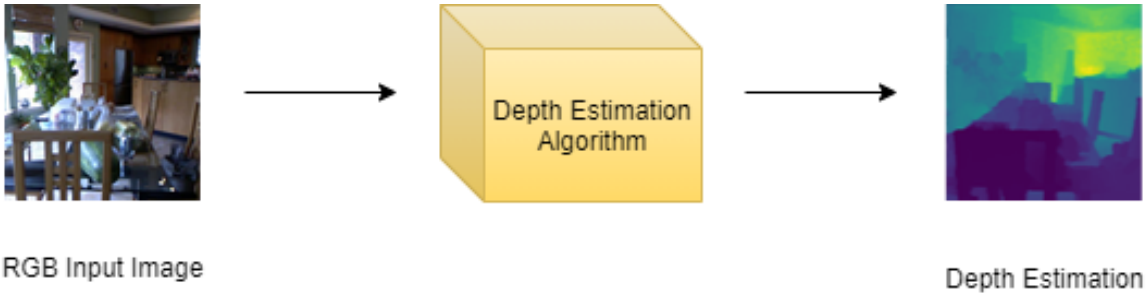

Figure 1.2: Example of a depth estimation algorithm. Left side is the input image for the algorithm. Right side is the resulting depth estimation.

Monocular depth estimation is of particular interest as it provides the ability to infer depth from a single image. A basic idea of the flow for a depth estimation algorithm is given in Figure 1.2. The input (left) for the algorithm being a RGB image and the output (right) being the depth estimation. The algorithm itself is displayed as a black box which receives the input image and utilizes the available cues to infer the depth map. The viability of various different depth estimation algorithms have been researched in recent years. There are two distinct categories for depth estimation algorithms, traditional and deep-learning. Traditional methods have particular difficulty with monocular depth estimation as it is challenging to infer 3D information from a single 2D source. To this end a large number of traditional methods need two or more input images to be able to compute a depth estimation. These approaches utilize the information obtained from moving the camera or having more than one view point to find correspondence between the images which can be used to calculate a depth estimation. A great example of the utilization of stereo image pairs is the work of Kamencay et al. [36]. Their approach relies on disparity calculation between the stereo pair as well as knowledge of camera parameters. The disparity is the pixel distance between the two images, how much each pixel has shifted

from one image to the other. This is an ill-posed problem as there is bound to be occluded pixels between the two images resulting in imperfect disparity maps. These disparity maps can then be used in correspondence with the cameras' parameters to calculate the depth estimation. This is done by using a baseline distance, physical distance between the two viewpoints or cameras, and the focal length. The actual depth calculation is fairly trivial once a disparity has been found it is defined by Equation 1.1. This calculation is done for each pixel of the disparity map.

$$depth = baseline \times focal/disparity. \tag{1.1}$$

This approach can also be extended to utilize more than two images to increase the overall quality as there will be more images available from which to source pixel matches in the disparity calculation step.

Despite the challenges attached to monocular depth estimation there are traditional approaches that are able compute depth estimations. One of the most common approaches [61] exploits the out of focus blurriness of the image. The approach performs a series of blurring and deblurring operations on the image in order to achieve the depth estimation. Each point in the image is increasingly blurred which increases the relative depth as well as deblurred to decrease the relative depth. These operations can be used to determine at which increment of blurriness the point would reach infinite distance. Based on that blurriness increment the depth can be obtained.

Deep neural networks (DNNs) are, thanks to the development of Convolutional Neural Networks (CNNs), able to greatly improve on the traditional methods in various computer vision tasks. DNNs excel at tasks like image classification [30, 32, 58], image segmentation [34, 70] or image super resolution [13, 33]. Due to the success of CNNs with various computer vision tasks it is no surprise that a number of different Deep Learning (DL) based depth estimation approaches were developed. Deep learning in general and therefore also monocular depth estimation methods can be categorized into three groups, the first group being supervised approaches [5, 17, 39]. These approaches require the existence of input image and ground truth depth map pairs for their training strategies. They are capable of creating high accuracy depth estimations, however, the need for labeled ground truth data is a problem as it can be challenging to obtain those depth maps. The second group is unsupervised approaches [22, 25], they do not have a need for ground truth data, however they do need a pair of stereo images for training. Unsupervised approaches are still capable of

generating high accuracy depth maps but they cannot compete with state-of-the-art supervised methods. The third group is based on self-supervised learning [23, 44], self-supervised approaches create the middle ground between supervised and unsupervised in terms of accuracy. Self-supervision does not rely on ground-truth data, instead they use secondary data, such as pose, obtained from the input images to improve the results achieved by unsupervised counterparts. While there are many MDE approaches available, most state-of-the-art approaches focus largely on improving performance and achieving the best possible depth prediction. This drive for increased performance comes at the cost of resources. So called heavy networks are capable of great accuracy but they require large amounts of compute resources and time for training as well as inference. The solution is lightweight network design, with the goal of creating a network architecture that has a significant decrease in size and complexity but that is still capable of achieving competitive results.

In this thesis we propose two distinct MDE methods, one supervised lightweight CNN as well as a self-supervised training framework for lightweight neural networks. In chapter 3 we propose a novel lightweight neural network specifically designed for the task of monocular depth estimation. We utilize recent advancements in neural network design to minimize both size and complexity while maintaining a competitive accuracy. In chapter 4 we then propose a lightweight self-supervised training framework with the purpose of increasing performance for lightweight networks. Our framework uses the stereo image pair required by most unsupervised training methods to compute a disparity approximation which is then compared to the networks predictions to give a self-supervised loss. We further enhance our framework with the addition of adversarial learning by adding a discriminator. In this instance the network making the disparity prediction would be the generator. This gives us a Generative Adversarial Network (GAN) [24] training structure which further improves the generators performance. Each method is explained in greater detail in the corresponding chapters.

1.2 Motivation

The motivation for this thesis is the growing need for lightweight depth estimation algorithms. Both faster and more effective methods are needed to satisfy the demand at hand. Despite the wide range of powerful deep neural networks available there is a demand for more lightweight methods. Heavy neural networks are capable of

producing high accuracy depth estimation. However, they are incredibly resource intensive making them unsuitable for a number of applications. In our work we develop and investigate a number of lightweight neural network based approaches that handle the MDE problem. These lightweight approaches can prove useful for resource weak systems such as mobile devices with low compute capability and limited power.

1.3 Problem Description

As discussed monocular depth estimation is a challenging task as there is only limited 3D information left after the scene has been projected to a 2D image. Inferring the lost 3D information is an ill posed problem since the cues that are available in the 2D images give some idea about the structure of the original 3D scene, however, they are not capable of providing complete scene information. While there is specialized hardware available it is unsuitable for most applications. The current state-of-the-art deep learning based solutions are capable of producing high quality depth estimations at the expense of compute resources and time. The challenge with designing a lightweight approach is not only the size constraint but also the goal of maintaining competitive network performance while working with reduced compute capability. Lastly when designing a self-supervised approach the key goal is to increase a networks performance by providing it with additional information. Obtaining this information from the source images can be challenging because, as mentioned above, the images do not contain all the information needed to create completely accurate depth maps. The added challenge of extracting additional information without significantly increasing complexity and resource drain further constraints the design process.

1.4 Contribution

In this thesis we propose two distinct lightweight methods for monocular depth estimation. The main contributions are the following.

First, we propose a novel lightweight network architecture. Our novel architecture leverages the advances made to CNN design over the recent years. We propose an Encoder-Decoder based architecture where the encoder is tasked with extracting feature information from the input image. The decoder's job is to upsample those extracted features and to perform the final depth prediction. We propose a novel

encoder architecture which utilizes the advances made by Dimension-wise Convolutions for Efficient Networks (DiCENet) [47]. Our encoder is specifically designed to be lightweight as well as optimized for the task of monocular depth estimation. We further propose two novel decoder architectures to create an optimal lightweight network. The first decoder utilizes DiCE units [47] which are also present in our encoder and the second utilizes depthwise-separable convolutions [55]. Both decoders meet our strict criteria to be lightweight while still maintaining competitive accuracy.

Second, we propose a novel self-supervised training framework. Our framework utilizes existing training environments and boosts their performance at minimal cost to complexity. At training time our framework uses a stereo pair of images, common for unsupervised or self-supervised MDE, to infer a disparity map estimation. In addition we employ a GAN based approach to further increase network performance.

1.5 Organization of Thesis

This thesis consists of five chapters. This first chapter was about giving an introduction to the topic. The rest of the thesis is structured as follows.

Chapter II We present the literature review. We provide background information for relevant techniques and methods. Then, we give detailed insight into the current state-of-the-art approaches for monocular depth estimation.

Chapter III We give a detailed description of our proposed lightweight encoder and both decoders. We further provide the training strategy as well as experimental results.

Chapter IV We provide insight into our self-supervised training framework. We give detailed explanations of the various components and the training strategy as a whole. We also provide experimental results for our framework.

Chapter V We give a conclusion as well as a summary of our thesis and the contributions of our methods.

All references used are presented at the end of this thesis.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Disparity Map Calculation

Disparity maps have been used for monocular depth estimation tasks for a long time. They have been employed by traditional methods such as the one proposed by Kamencay et al. [36]. Disparity maps show the pixel difference between a stereo pair of images, the difference in location of objects in the images. Disparity calculation,

much like MDE, is an ill-posed problem. Between any two stereo images there will be pixels that are visible in the first image that are either occluded or missing completely in the second image. The most important step when calculating a disparity map is to find the pixel correlations in the two images, so determine where pixel x in image A is in image B. There are three algorithmic methods that both determine pixel correlation while providing the disparity value. The first is the Sum of Absolute Differences (SAD) [27, 64], the second is the Sum of Squared Differences (SSD) [38] and the third is the Normalized Cross Correlation (NCC) [68]. The simplest and most commonly used ones are SAD and SSD, both utilize a window $W(x, y)$ around a center pixel (i, j). Both compute the intensity difference for pixel (i, j). The equation for SAD is given in Equation 2.1 and the equation for SSD is Equation 2.2.

$$SAD(x, y, d) = \sum_{(i,j)eW(x,y)}^{N} |I_L(i, j) - I_R(i - d, j)|, \tag{2.1}$$

$$SSD(x, y, d) = \sum_{(i,j)eW(x,y)}^{N} |I_L(i, j) - I_R(i - d, j)|^2, \tag{2.2}$$

where I_L and I_R represent the left and right image respectively and $W(x, y)$ is the window around the pixel (x, y). Regardless of whether SAD or SSD is employed, the calculation is performed over the entire row of pixels in the target image. In detail, for every pixel (i, j) in the left image the calculation is performed for each corresponding row j in the right image to determine best pixel correspondence. While this process is effective at determining pixel correspondence it can be very time consuming with larger images. To this end a Block Matching version is widely used. Instead of determining individual pixel correspondence entire blocks are compared. Both SAD and SSD can be used but instead of performing the calculation only for pixel (x, y) in window $W(x, y)$ the value for each pixel in the block is computed simultaneously as matrix calculations. This process is capable of greatly increasing the efficiency of this method with only minor loss in quality. The loss in quality is minimal because in both I_L and I_R the area surrounding each pixel will be similar. With Block Matching the size of each block can greatly determine the quality of the disparity map, smaller block sizes produce more noise and larger block sizes produce overly smooth disparity maps. It needs to be noted that regardless of block matching or pixel wise calculation in order for this method to work effectively the cameras utilized have to be parallel.

2.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have grown in popularity for a variety of machine learning tasks, most noticeably for computer vision tasks as their design and function greatly benefits from the multidimensional data used for those tasks. CNNs are a sub-type of Artificial Neural Networks (ANNs). These networks often consist of a variety of different layer types, including but not limited to, convolutional layers which give the CNNs their name, activation layers, pooling layers and fully connected layers. CNNs when compared to other ANNs have the advantage of being able to extract and refine features from multidimensional data very effectively and efficiently due to the kernels that make up a convolutional layer. While these kernels are effective at multidimensional feature extraction they suffer greatly when used on 1-Dimensional data as they are unable to establish proper feature relations.

Convolution Layers

Convolution Layers (CLs) [40] give the CNNs their name and are what sets them apart from other ANNs. The invention of CLs is generally accredited to LeCun et al. [40] as their work is the main contributor to the CNNs utilized in recent years. However, there is some discussion since the work of Fukushima et al. [18] with their proposed Neocognitron works similarly to the CLs proposed by LeCun et al.. Regardless of accreditation, CLs utilized in today's CNNs consist of a number of convolution kernels. The kernels consist of a number of trainable parameters who perform the function of extracting relevant feature information from the input. The number of kernels for each convolution layer is determined by the desired amount of output channels for that layer. Furthermore, while the size of each kernel within a layer needs to be the same, the size of kernels in different layers can vary. Typical sizes include 3x3, 5x5 or 7x7, this size represents the number of pixels the kernel processes each step. Each kernel traverses the input both height and widthwise and at each step performs the convolution operation. This operation can in basic terms be compared to scalar matrix multiplication the product of which is a single value which replaces the kernel sized window in the output feature map. Therefore, the convolution operation reduces the dimensions of the input image during the extraction of relevant features. During the network's backward pass backpropagation adjusts each kernel's parameters based on the loss of the specific task. This process optimizes the kernels to extract desirable feature maps for the task. For each layer the extracted feature map can be defined

by the following:

$$L^i = H^i \otimes L^{i-1} + b^i, \tag{2.3}$$

where L^i is the feature map output of the $i-th$ CL, L^{i-1} is the previous layer's output, or in case of the first layer the input data, H^i represents the kernels of $i-th$ layer and b^i is the bias for that layer. $\otimes$ represents the convolution operation. The dimensions of the output feature map can be determined for each layer using the following equation:

$$D_O = (D_I + 2P - D_K)/S + 1, \tag{2.4}$$

where D_O is the output dimensions, D_I is the dimensions of the input for that layer, P is the padding, D_K is the kernel dimensions and S is the stride.

Activation Layers

Activation Layers (ALs) are of great importance for effectively training and utilizing CNNs. Each AL utilizes a corresponding activation function to perform their task. ALs receive the output of a CL as input and determine whether the input value can be activated or not. In other words, they determine which neurons are to be turned on and which turned off. Furthermore, the activation functions give the CNNs the ability to learn more complex data by normalizing the output into the range of -1 to 1 or 0 to 1. While there are a number of possible activation functions to choose from the following are the ones most relevant for our work. Figure 2.1 gives the graphs for the relevant activation functions.

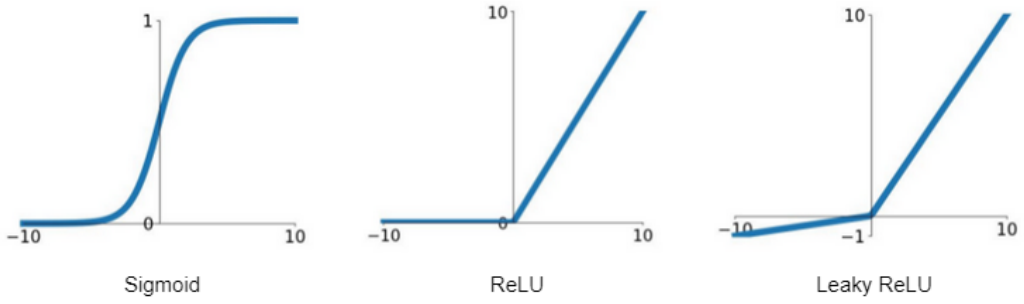

Figure 2.1: Plots for relevant activation functions [63].

1) Sigmoid: The sigmoid function takes the input and constraints it into the range of 0 to 1. It can be expressed as follows:

$$sigmoid(x) = \frac{1}{1 + e^{-x}}. \tag{2.5}$$

This brings large positive number to 1 and large negative number 0. While this activation function produces smooth gradients as output it suffers greatly from the vanishing gradient problem as there is little to no variation between extremely high or extremely low input values.

2) Rectified Linear Unit (ReLU): The ReLU function does not limit the upper range of input values but limits the bottom range to 0, setting all negative values to 0. ReLU can be expressed as follows:

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{2.6}$$

ReLU is able to avoid the vanishing gradient problem as well as give computational simplicity. Furthermore, it is able to speed up the networks convergence due to non-saturating linearity. ReLU's drawback is that all negative input values are discarded and set to 0. This leads to weights, that produced negative values, not being updated during backpropagation.

3) Leaky-ReLU / Parametric ReLU: In order to rectify the issues of ReLU, Leaky ReLU or it's derivative Parametric ReLU utilize a small negative slope instead of discarding all negative values. Parametric ReLU is defined as follows:

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases}. \tag{2.7}$$

For standard Leaky-ReLU α is fixed value producing a fixed slope for the negative values. Parametric ReLU improves this by making α a trainable parameter. The optimal value for each activation layer is determined during the iterative training of the network.

Pooling Layers

Pooling layers are also frequently used in CNN design as they are helpful in overcoming overfitting issues. The high dimensionality of the data commonly used causes the network to get extremely good at predicting the training set but unable to make any prediction on the test set or any other input. This means that instead of learning the relevant information to make accurate generalized predictions the network instead learns the training set as a whole and is only able to make accurate decisions for

said training set. Pooling layers are able to mitigate this by reducing the spacial dimensionality, width and height, of their input. There are two types of pooling layers commonly used in modern CNNs, the first is Max Pooling and the second is Average Pooling. The basic of both pooling approaches works the same, a block size is chosen and the passed input is spatially split into equal blocks of the chosen size. Each block is then used to determine a single value for the output of the pooling layer. Each block is reduced to a single value, reducing the overall spacial dimensions of the input. In the case of max pooling, the maximum value of the block is chosen. Average pooling averages all the values in the block and that average is then the value for the output. A good example of max and average pooling is given in Figure 2.2.

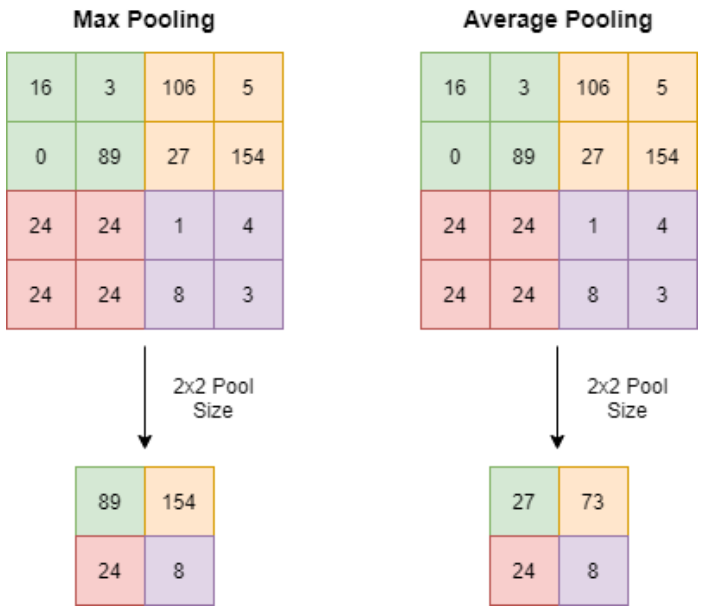

Figure 2.2: Visual comparison of Max Pooling and Average Pooling.

While average pooling is used in many CNNs, max pooling is often the preferred choice. It only passes values that were present in the input onto the next layer, unlike average pooling which computes new values. Furthermore, max pooling is computationally less expensive and can also lead to faster network convergence.

Fully Connected Layers

While Fully Connected Layers (FCLs) are of great importance for a variety of applications such as image classification they are only of limited interest for our work which is why we will only be covering them briefly. FCLs are the basic neural network

structure that has been used before CLs were the standard for multidimensional data. They consist of a fixed number of neurons where each neuron in each layer is connected to each neuron in the previous and following layer. In today's application they usually receive the feature map output from a series of convolution layers. The CLs multidimensional feature map is converted into a 1D feature vector which is passed to the FCLs. The FCLs then utilize that feature vector to make the final prediction for the labels or classes of the given input.

2.1.3 Generative Adversarial Networks

Generative Adversarial Networks have grown in popularity in recent years. They have a variety of applications from image generation [28, 60] to 3D view synthesis [19, 20]. Furthermore, the adversarial training stratagem has been utilized by various computer vision tasks to improve overall network performance. At it's core a GAN consists of a generator and a discriminator. The generator is usually the network performing the desired task by generating the desired output. The discriminator's job is to distinguish between the generator's output and a real sample. Generator and discriminator are competing against one another, the generator is trying to generate more realistic images and the discriminator is trying to better differentiate between fake and real. However, while it is generally the goal to achieve a peak performance generator both generator and discriminator are trained simultaneously. The GAN's goal is to train both generator and discriminator until the generator is able to generate realistic outputs that are indistinguishable from real world sample data and the discriminator can no longer differentiate the two.

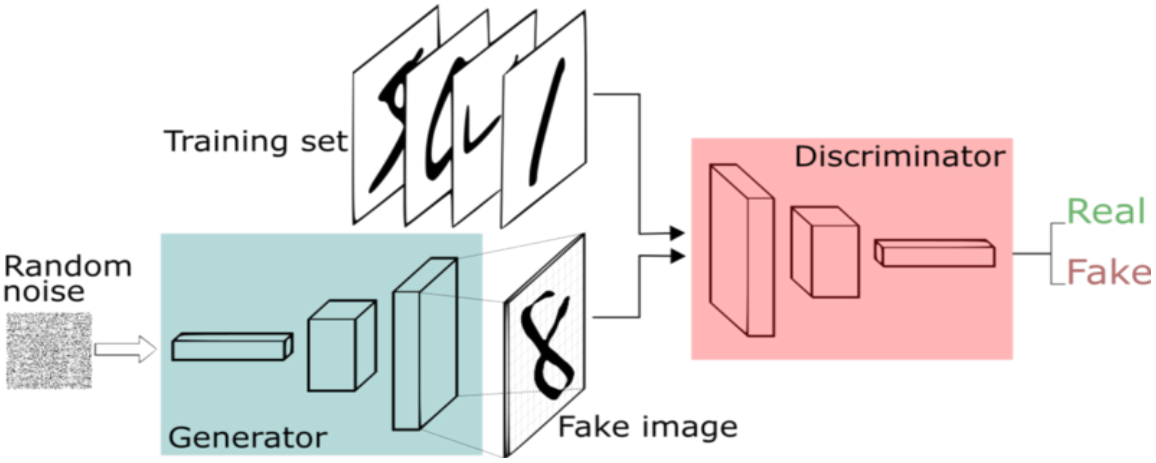

Figure 2.3: Basic flow of a GAN pipeline [57].

A general example for this is given in Figure 2.3. In the example the generator takes a random noise input, sourced from a Gaussian distribution, to generate an image of a handwritten number. The discriminator is passed either a fake image or an image from the sample training set. It then tries to determine whether the passed image is fake or real. The optimization, or loss, function for a GAN can be defined as shown in Equation 2.8.

$$L_{GAN} = \min_{G} \max_{D} V(D, G) = E_x[log(D(x)] + E_z[log(1 - D(G(z)))], \tag{2.8}$$

where $D(X)$ is the discriminator's estimated probability that the real data instance x is real, $G(z)$ is the generator's output given noise z and $D(G(z))$ represents the discriminator's estimate of the probability that the generator's fake output is real. E_x and E_z are the expected values over all real instances and all fake instances respectively. The generator tries to minimize this function while the discriminator tries to maximize it.

GANs have been very successful at various tasks including boosting the performance of existing methods. However, GANs suffer from vanishing gradient problems because the discriminator is unable to pass enough information back to the generator. Furthermore, GANs suffer from mode collapse meaning the generator is only able to produce a single or a small set of outputs. In order to resolve this issue a number of improved GAN architectures were developed such as WGAN [6], LSGAN [46] or RaGAN [35].

2.1.4 Supervision During Network Training

Training of a neural network is a key step as this teaches the network how to process the given data to achieve the desired output. There are a number of different training stratagems currently being used. The most relevant for us are supervised, unsupervised and self-supervised training. Each of these methods has their own set of advantages and disadvantages. The details are given below.

Supervised Learning

Supervised learning is the most common and most straight forward approach. It requires the existence of labeled data for training. The term labeled data means that for any given input in the training set a corresponding expected output is given. The

network is then given the training input and makes a prediction about the output. The network's output is then compared to the expected output and an appropriate loss function is used to determine how well the network performed. This loss is then used during backpropagation to adjust the network's parameters with the goal of improving the network's performance for the next pass. While this method is simple and extremely effective at training network it has the drawback of requiring labeled data for training. Deep convolutional neural networks require very large datasets for training as they need to be exposed to several different samples in order to generalize and not overfit. The need for large datasets can be problematic for several different areas of computer vision as the acquisition of labeled data is challenging due to a number of different reasons.

Unsupervised Learning

Unsupervised learning removes the need for labeled data. Only the input data is needed during training. The network's performance is evaluated differently depending on the task at hand. In the case of unsupervised classification, or clustering, this is usually done through a combination of cohesion and separation of the clusters. For MDE it is usually done by utilizing the networks prediction to warp or reproject the input image and then compare the warped image to the original input to determine the network performance and adjust it accordingly. While unsupervised learning is able to train a network without labeled data it usually does this at the cost of performance. In most cases networks trained using an unsupervised training method do not achieve the same accuracy that the same networks trained on supervised training stratagems do.

Self-Supervised Learning

Self-Supervised learning forms the middle ground between supervised and unsupervised learning. Self-supervised training methods do not require labeled data during training but are able to achieve accuracies closer to that of supervised learning. This is done by utilizing secondary information to boost the performance of the target network i.e. the network to be trained. This secondary information is either obtained through additional sensor information available at training time or by extracting additional information from the input data and passing that to the target network or using it to modify the networks prediction. Despite improving network performance

compared to unsupervised methods self-supervised approaches still fall short of supervised methods. Furthermore, the extraction and utilization of additional information increases the overall complexity of the training process and therefore also increases resource consumption and training time.

2.1.5 Network Size and Complexity

When designing lightweight networks, both the size and the computational complexity are of great importance. The network size describes how much storage and random-access memory (RAM) a neural network needs to operate. One of the main attributes that can be used to gauge the size of a neural network is the number of parameters. It needs to be noted that there are other factors that determine the RAM requirements of a neural network such as the number of activations as well as what precision is used to store each of the values. One of the goals when creating a lightweight neural network is a low number of parameters to reduce the size. Additionally, the number of parameters is also related to the complexity of a neural network. Most often this complexity is measured in terms of Multiply-Accumulate (MAC) operations [9]. These MAC operations describe the computational complexity of a neural network as it determines how many operations are performed during each forward pass of a neural network. Reducing the number of MACs is the main goal of lightweight network design as reduced complexity results in faster performance on systems with low computational resources. A lower MAC count also results in lower power consumption, this is key for systems where power consumption is a major concern. An increased number of operations results in higher power consumption as pointed out in [65].

2.1.6 Heavy vs Lightweight Networks

In recent years the goal of network design, regardless of application whether it be monocular depth estimation, super resolution or image classification, has mostly been to achieve the maximum possible performance. To this end, a number of new network architectures were developed, including but not limited to ResNet [30], DenseNet [32] or VGG [58]. These architectures excel at image classification performed on the ImageNet [12] dataset. Furthermore, these base network architectures were utilized in other disciplines as a base to enhance existing models. While these architectures are incredibly powerful and achieve great results they do so at the cost of resources. Some basic details about the architectures design will be given in the following. The

basic architecture or design idea for each network is given in Figure 2.4

VGG is a very simple yet effective network design. It is based on the successful AlexNet and achieved remarkable results for it's time. As with both of the other two architectures VGG offers a set of different versions, each version referring to it's size. Here we will focus on the most commonly used version of VGG, VGG-16, the 16 refers to the number of layers containing weights. It consists of 13 convolution layers and 3 fully connected layers. The convolution layers are grouped into 2 groups of 2 followed by 3 groups of 3 whose output is fed to the FCLs. While achieving good results and improving upon it's predecessor VGG like many deep neural networks can suffer from vanishing gradients. Furthermore, it is incredibly large at 138M parameters and therefore also computationally expensive.

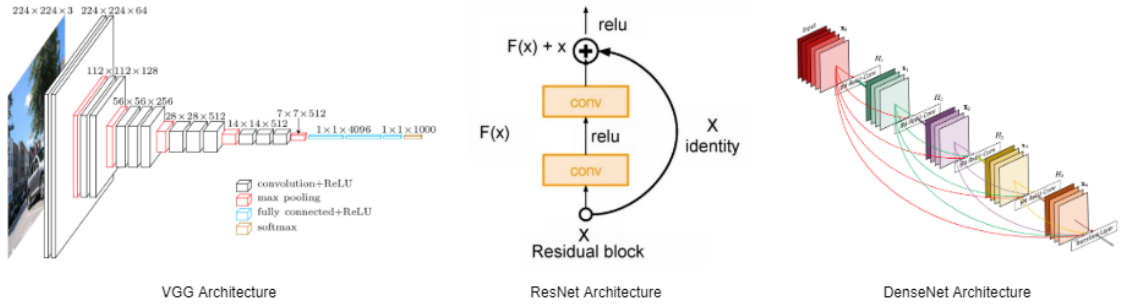

Figure 2.4: Diagrams [32, 43, 53] for the three heavy networks VGG, ResNet and DenseNet. For VGG the whole architecture is shown, for ResNet and Densenet only the basic concept is shown due to the size of the overall architecture.

ResNet mitigates the effect of vanishing gradients and is able to go deeper because of it. The standard version of ResNet, ResNet50 consists of 50 weight layers. ResNet mitigates the vanishing gradients through the use of shortcut connections as seen in Figure 2.4. These shortcut connections enable what is called residual learning. The input layer connects with the output layer directly this forms the identity mapping. Due to residual block's identity mapping ResNet is able to go deeper and wider without being concerned about vanishing gradients. Despite consisting of more layers than VGG-16, ResNet50 is able to have a lower parameter count and computational complexity. This is largely due to the use of pooling layers as well as using only one FCL. ResNet50 has a parameter count of 25.6M, significantly lower than that of VGG.

DenseNet further develops the use of shortcuts. Unlike the ones used by ResNet they not only link each layers input to its output but rather they connect it to

each following layer's output. Furthermore, while ResNet uses additive connections DenseNet uses concatenative connections. Additive meaning the input is added to the output and concatenative meaning it is stacked on. This yields maximum internal supervision. The commonly used version of DenseNet is DenseNet-121 the 121 again represents the number of layers with trainable parameters. DenseNet-121's parameter count is 7.2M.

While all three of these networks improve upon each other and achieve remarkable results their size and complexity is of concern for a variety of different tasks. When the target application is using a resource poor system then utilizing so called heavy networks is not an option. Furthermore, even with powerful hardware available, training these networks is not only resource intensive but also time consuming. This is amplified when attempting to train said networks on weaker systems. In order to minimize the computational cost of using deep neural networks a number of lightweight networks were developed. Lightweight referring to both minimized size and computational complexity. The most important ones for us are MobileNet [31] and DiCENet [47]. Both networks achieve a significant reduction in complexity while still achieving competitive results. The computational complexity of a network is usually evaluated based on the MAC operations performed by the network. A higher value means a more complex network requiring more resources, computational, power or otherwise. A detailed comparison between these two lightweight networks and previously mentioned heavy networks is given in Table 2.1.

MobileNet utilizes so called depthwise-separable convolutions [55] these convolutions take advantage of matrix multiplication rules where a single matrix can be split into a product of two other matrices. With regards to convolutions each regular convolution is split into two different convolutions a depthwise and a pointwise convolution. The depthwise convolution performs the spatial convolution while the pointwise performs the channelwise convolution, increasing or decreasing the channel count. In terms of kernels this means that each kernel is split into two sub kernels which if matrix multiplied would produce the original kernel. For example a kernel which would receive a 3 channel RGB image as input, has the spatial dimensions of 5x5 and the desired feature map channel count is 256, so 256x5x5x3. This kernel would be split into 3 5x5x1 as well as 256 1x1x3 kernels. The following equations can be used to determine the number of operations needed for conventional convolutions (2.9) and depthwise-separable convolutions (2.10). In addition, the equation to determine the reduction in computational cost (2.11) for using depthwise-separable

convolutions is also given.

$$Ops = D_K \cdot D_K \cdot C_I \cdot C_O \cdot D_F \cdot D_F, \tag{2.9}$$

$$Ops = D_K \cdot D_K \cdot C_I \cdot D_F \cdot D_F + C_I \cdot C_O \cdot D_F \cdot D_F, \tag{2.10}$$

$$Red = \frac{eq.(2.10)}{eq.(2.9)} = \frac{1}{C_O} + \frac{1}{D_K^2}, \tag{2.11}$$

where D_K is the dimensions of the kernel, C_I is the input channel number, C_O is the output channel count and D_F is the dimensions of the output feature map. Using Equation 2.11 and a 3x3 kernel the computational cost is on average 8 to 9 times lower for depthwise-separable convolutions compared to standard convolutions. MobileNet does not exclusively use depthwise-separable convolutions rather a mix of conventional and depthwise-separable convolutions. This is because the reduction in operations and parameters also decreases the networks performance so a balance between lightweight design and performance had to be found. Due to the use of depthwise-separable convolutions MobileNet has a parameter count of 4.2M, however, as evident from the comparison in Table 2.1 the computational cost is significantly less than that of other heavier networks.

DiCENet [47] addresses some of the issues related to depthwise-separable convolutions. While they are able to achieve a significant reduction in complexity, the amount of pointwise convolutions, right side in Equation 2.10, make up a major part of the convolutions overall complexity. This creates a bottleneck for the computational cost. Furthermore, depthwise-separable convolutions cannot establish channelwise relations which play a major part in the performance reduction mentioned. Mehta et al. propose a solution to both of these problems, the DiCE unit, Figure 2.5.

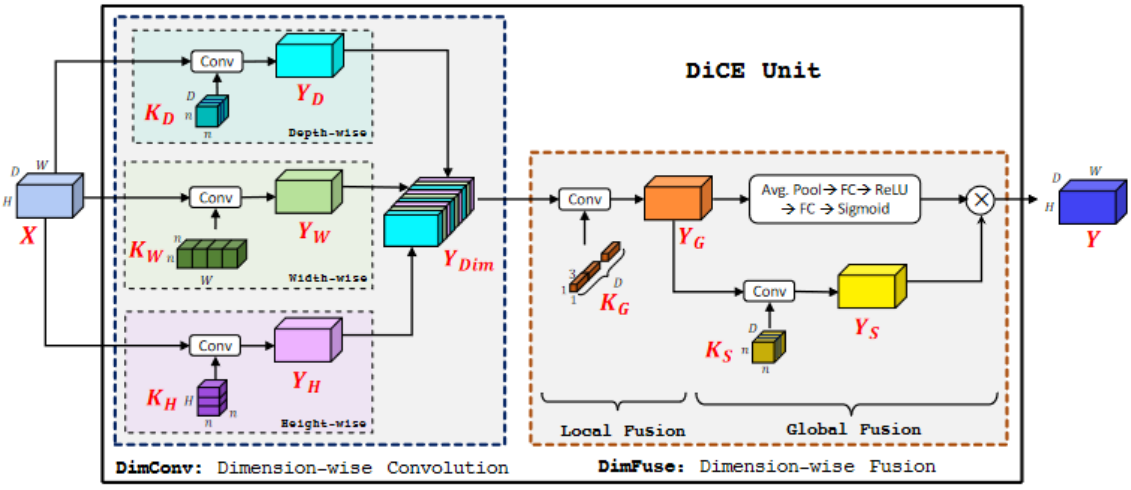

Figure 2.5: Overview for the architecture and flow of a DiCE unit [47].

Unlike depthwise-separable convolutions the DiCE unit splits a regular convolution into three sub-convolutions, depthwise, widthwise and heightwise. These three convolutions are able to perform the spatial reduction as well as modify the channel count. The output of the three sub-convolutions is combined and then fed through what is called Dimension-wise Fusion. This fusion creates the channelwise relations needed to achieve better performance. The DiCE unit does not eliminate the need for pointwise convolutions completely but it significantly reduces the computational bottleneck they pose. While each individual DiCE unit is computationally more expensive than a depthwise-separable convolution they are usually able to achieve better accuracy. The DiCENet proposed by Mehta et al. utilizes a series of StridedDiCE units, regular DiCE units and ShuffelDiCE units. The overall network architecture has s slight increase in parameters, 5.1M, but achieves significantly better performance.

Table 2.1: Comparison of both heavy and lightweight networks covered in this section. MACs count and Top 1 Acc. are for the ImageNet dataset. For Param (M) and MACs (B) lower is better for Top 1 Acc. higher is better.

Network	Param (M) ↓	MACs (B) ↓	Top 1 Acc. ↑
VGG-16	138	15.5	73.4
ResNet-50	25	4.1	76.0
ResNet-152	60.2	11.3	**77.8**
DenseNet-121	7.9	2.9	75.0
DenseNet-201	20	4.4	77.4
MobileNet	**4.2**	**0.57**	70.6
DiCENet	5.1	-	75.7

2.2 Related Works

2.2.1 Networks for Monocular Depth Estimation

The generalized network overview in Section 2.1.6 just covered the architecture design of the base networks used primarily for image classification. MDE requires different network design to be effective as the existing network design is not suitable to output 2D images. An Encoder-Decoder Architecture (EDA) is therefore usually employed for this task. As the name suggest the EDA consists of an encoder whose task it is to extract the features and a decoder which upsamples these features into the final depth prediction. The EDA is also frequently utilized in other areas of computer vision such as image segmentation [52] or optical flow estimation [14]. Since it is a sequence-to-sequence model it is the appropriate network architecture for any tasks that take an image as an input and also produce an image as output. UNet [52] is one of the most commonly known EDAs. For the purpose of MDE the encoder is usually a well established network from image classification such as the ones discussed in Section 2.1.6. These networks offer great and proven feature extraction capabilities. Furthermore, these architectures offer the ability to use transfer learning. Transfer learning refers to the use of pre-trained weights from the ImageNet classification. These ImageNet weights have great feature extraction generalization and offer improved network performance across disciplines. The decoder is usually custom designed to utilize the features of the encoder. The decoder often consists of a number of convolutions as well as interpolation layers or deconvolution layers to in-

crease the spatial dimensions while decreasing the channel dimension. In recent years the EDA has further been advanced through the use of skip-connections [5, 22, 65]. These skip-connections connect the encoder with the decoder. Encoder layer outputs with the same spatial dimensions as decoder layer outputs are concatenated or added. These skip-connections are like the shortcut connections used by ResNet or DenseNet they allow for different gradient passing during backpropagation. This gives the earlier layers in the network better odds of being properly updated during this step. Which in turn leads to better results and faster convergence time.

2.2.2 Supervised vs. Unsupervised vs. Self-Supervised MDE

There are key differences between supervised, unsupervised and self-supervised training as previously discussed in Section 2.1.4. In the following the different current approaches are laid out and explained for each of the three approaches. Furthermore, in the following sections (2.2.3, 2.2.4) the differences in network design, specifically lightweight design, are laid out as there are key differences.

Supervised Approaches

Supervised approaches have greatly benefited from several large scale datasets such as NYUv2 [56], KITTI [21] and Cityscapes [11]. NYUv2 offers a large range of interior scenes with ground truth depth captured using a Kinect. KITTI and Cityscapes offer exterior scenes captured using a calibrated pair of stereo cameras. These datasets enabled a number of CNN based approaches [15, 17, 29, 39, 65] that were able to excel at the task using supervised learning methods. Most of these approaches utilize an existing network architecture such as ResNet [30], DenseNet [32] or MobileNet [31] as feature encoders to which a specifically designed decoder is attached whose output is the depth prediction. These networks are able to achieve great results. However, as mentioned above, supervised learning comes with the disadvantage of requiring large amounts of labeled data for training. The datasets available cover a number of scenarios where depth estimation could be needed but there are still various scenarios that are not present in the datasets.

Unsupervised Approaches

Unsupervised approaches are able to overcome the need for labeled data. The most notable approach in recent years is Monodepth [22]. It utilizes a pair of stereo images

during training but not at inference time. Monodepth employs a network structure similar to that of supervised models, however, instead of predicting a single depth image a number of disparity maps are predicted at different resolutions. These disparities are then used in combination with the input images to produce 'fake' images which can be compared with the original input images to evaluate network performance and calculate network loss. Monodepth is given a single input image and predicts the disparities for both the left and the right image. This approach is very effective in training a number of different network architectures. It is also the basis of several new and improved approaches published in recent years [10, 25, 44, 50, 66]. Poggi et al. [50] improved upon Monodepth by using the binocular training images to simulate a trinocular setup which led to more accurate results. Most relevant for this thesis is the work of Groenendijk et al. [25], they proposed a number of different GANs to be used in conjunction with the training structure proposed by [22]. Their approach utilizes the full resolution, same size as input, 'fake' image calculated from the disparity predictions to calculate not only the reconstruction image loss but also for adversarial training. The discriminator is passed the original input image as well as the calculated image.

The biggest challenge for unsupervised approaches is that they are unable to exceed the performance of supervised models. While they are able to achieve adequate results without the need of labeled data they do not achieve the same performance in most cases.

Self-Supervised Approaches

Self-Supervised approaches are able to offset the reduced performance of unsupervised methods. They are able to create a middle ground between unsupervised and supervised approaches. The most notable self-supervised method is MonodepthV2 [23]. There are similarities between Monodepth and MonodepthV2, both try to minimize the photometric reprojection loss. However, unlike Monodepth, which uses a stereo pair of images for training, MonodepthV2 utilizes a temporal monocular video feed for training. In order to utilize said temporal features MonodepthV2 uses a secondary network for pose estimation of the input images. MonodepthV2 has the option to be either trained on stereo inputs or on monocular temporal inputs. Another approach which also utilizes temporal inputs is MiniNet [44], it self classifies as unsupervised but it has several key similarities with MonodepthV2, both utilize secondary net-

works for pose estimation between the frames as well as try to minimize photometric reconstruction loss. MiniNet utilizes two shared weight networks for pose estimation, one for the frame before the input frame and one for the frame after the input frame. Both [23] and [44] utilize ResNet18 as their pose network which drastically increases complexity and training time. Even with its shared weight approach MiniNet still requires two passes through the pose network, one for each of the reference frames. While both MonoDepthV2 and MiniNet are capable of increasing the generator's performance they do so at the cost of increased training time and resource consumption.

2.2.3 Lightweight Supervised Network Design

As previously mentioned lightweight networks aim to lower computational complexity and size while maintaining a competitive performance. The general network design covered in Section 2.2.1 suffers similarly as other tasks from the goal to achieve the best possible performance. Furthermore, in most cases the encoder is based on an existing network architecture which results in a similarly heavy network driven design. The current-state-of-the-art lightweight approach is FastDepth [65] proposed by Wofk et al., Figure 2.6. FastDepth also utilizes an existing network architecture as encoder, unlike previous methods it utilizes the lightweight MobileNet as base for it's encoder. Using MobileNet as encoder is already a significant reduction in complexity. The authors further proposed a novel depthwise-separable convolution based decoder to further reduce the networks complexity. In addition, skip-connections are utilized to achieve optimal network performance. The skip-connections relay information from the encoder to the decoder. Overall, FastDepth achieves a major reduction in complexity with 0.5% to 1.5% the complexity of current state-of-the-art heavy networks. Despite this significant reduction in complexity it still maintains competitive results with a reduction in accuracy of 2% to 7%.

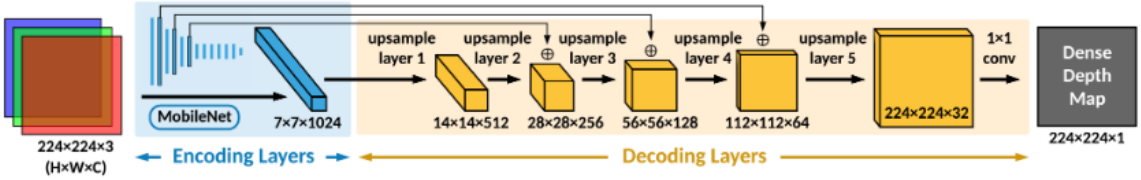

Figure 2.6: FastDepth architecture overview, including encoder, decoder design and skip-connections. [65].

2.2.4 Lightweight Unsupervised Network Design

The lightweight network architecture PyDNet [49] utilizes the training framework proposed by Monodepth and is able to achieve competitive results. PyDNet's architecture differs from that of traditional supervised networks as it does not utilize a distinct encoder and decoder, its complete architecture can be seen in Figure 2.7. From the figure it is apparent that PyDNet does not use a dedicated encoder and decoder but rather a compact architecture that combines feature extraction and upsampling into one. The main feature extraction is performed through a series of paired convolutions, strided and non-strided. The output of each pair is used as origin for skip-connections. The output of these convolution pairs is then fed through groups of four convolutions after each a sigmoid activation is used to make the disparity prediction. The output of each of those groups is then upsampled and combined with a dimensionaly matching skip-connection from the initial feature extraction pairs. This architecture where the entire network performs both feature extraction and upsampling is distinctly different in design from that of current lightweight methods used for supervised learning. In addition, PyDNet does not utilize depthwise-separable convolutions but uses conventional convolution layers. There is no clear performance comparison between supervised and unsupervised designs. When compared to state-of-the-art heavy networks, using unsupervised training, PyDNet requires only 6% the size and has a reduction in accuracy of 4%.

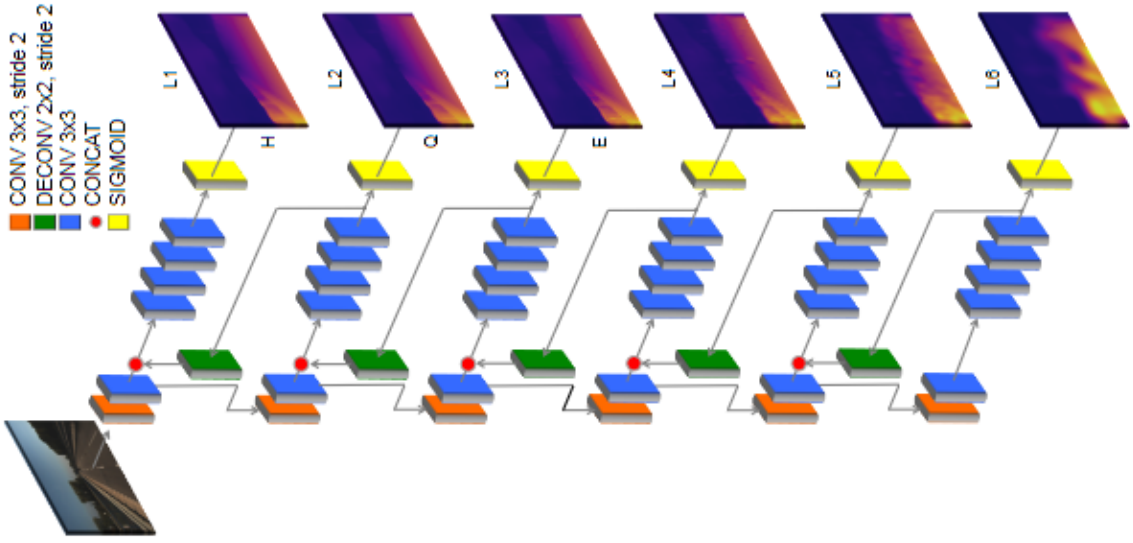

Figure 2.7: PyDNet architecture overview, shows combination of encoder and decoder into one compact network architecture [49].

Chapter 3

A Novel Lightweight Network for Fast Monocular Depth Estimation

3.1 Overview

Depth estimation is of growing interest in many sectors, from robotics to wearable augmented reality gears. Monocular depth estimation attracts more attention due to its cost efficiency and low complexity. Most recent research has developed very

large and resource intensive networks which are not suitable for small systems with limited resources. In this chapter, we propose a lightweight network which leverages the advantages of dimension-wise convolutions and depthwise-separable convolutions to reduce complexity in the architecture. In particular, the proposed depth estimation architecture utilizes a novel DiCE unit-based encoder, optimized for a lightweight encoder-decoder structure. Furthermore, we propose a DiCE unit-based decoder structure as well as an optimized depthwise-separable convolution-based decoder. Both decoders follow a similar five-layer architecture. In the experiments, we have demonstrated the effectiveness of the proposed architecture as well as the comparison between the two proposed decoders. Our novel lightweight network has a significant decrease in both size and complexity at a marginal cost to accuracy when compared to other state-of-the-art lightweight networks.

3.2 Introduction

As previously mentioned, depth estimation is a crucial task for many areas from scene reconstruction [41], to augmented reality and robotics. Monocular depth estimation is preferred for a number of robotic platforms and AR gear due to its cost efficiency and ease of use when compared to more complex solutions such as LiDAR or stereo depth. In recent years research has largely focused on heavy deep neural networks and on increasing the accuracy and quality of the depth prediction [15, 29, 39]. While these networks are capable of producing good results they do so at the expense of computational resources and time. The networks need large amounts of RAM to be stored and have a slow training and inference time making their use in critical systems questionable. This shows the need for lightweight approaches. FastDepth [65] is the most notable research in this area, their network architecture relies on depthwise-separable convolutions to reduce size and complexity. FastDepth utilizes the widely adopted auto-encoder or encoder-decoder structure. Their encoder is the MobileNet [31] architecture and they propose a custom decoder based on depthwise-separable convolutions. Our novel architecture that we propose in this chapter allows us to further reduce the size and complexity through the use of DiCE units [47]. Using the DiCE units we are able to create a novel architecture that reduces FastDepth's size by two thirds and the computational complexity by half with only a minor reduction in accuracy. In particular, we propose a DiCE unit-based encoder and two custom designed decoders. The first decoder utilizes DiCE units and the second one utilizes

depthwise-separable convolutions. As mentioned above, this proposed architecture is capable of greatly reducing both size and complexity of current state-of-the-art lightweight networks. Our architecture is able to reduce the network size by almost 70% and the complexity by 50% while still achieving competitive accuracy with a reduction of less than 6% when compared to other state-of-the-art lightweight networks.

Our contributions are summarized as follows: 1) A DiCE unit-based architecture is proposed to optimize the performance and size of the encoder. 2) A new decoder is proposed which is composed of DiCE units and is capable of achieving competitive results while creating a fully DiCE-based network. 3) A second decoder architecture is proposed based on depthwise-separable convolutions. When compared with FastDepth decoder, it provides modified layer dimensions and skip-connections to achieve optimal information transfer from the encoder.

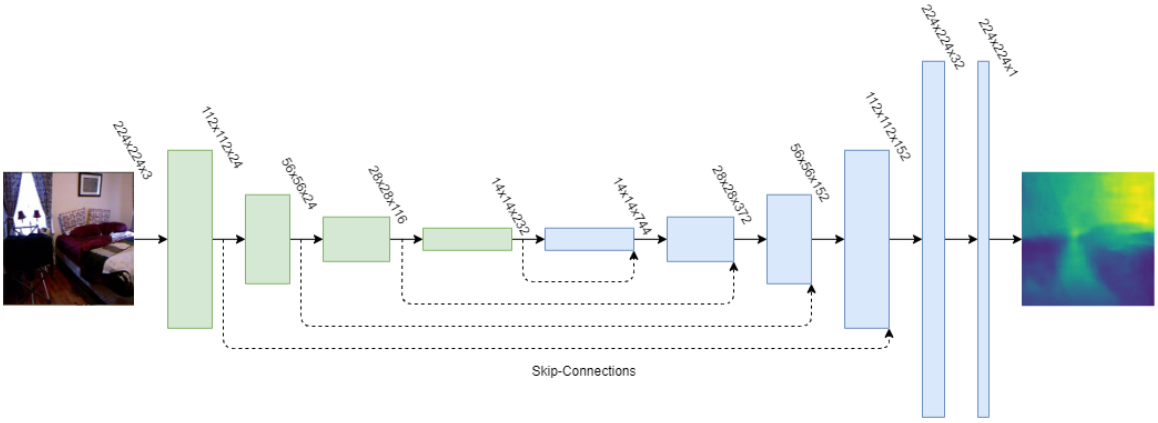

Figure 3.1: Overall architecture overview. Here the encoder is in green and the decoder in blue. Both proposed decoders utilize the same structural idea. The key difference between the decoders is the convolutional structure used - depthwise-separable or DiCE units.

3.3 The Proposed Network Architecture

Our proposed network is a fully convolutional encoder-decoder structure that relies on the encoder to extract features which are then upsampled and combined by the decoder to give the final depth estimation. A general overview of our novel architecture can be seen in Figure 3.1. The figure shows the flow through the network as well as the skip-connections. In addition, the output dimensions for each layer are also

shown in the figure. As further described in Section 3.3.1, the second layer that is depicted in the diagram is a max-pooling layer, which was included in both Figure 3.1 and Figure 3.2 to highlight the size reduction in that step as well as to show it's use for an additional skip-connection.

3.3.1 Encoder

The encoder used in our network is originated from the architecture proposed in [47] but specifically designed for depth estimation. This choice was made due to the fact that DiCENet can compete with MobileNet but offers reduced size and computational complexity in many applications such as object detection and segmentation. DiCENet was originally designed for image classification. In order to preserve the general DiCENet structure we utilized DiCEBlocks that have the same internal structure as the ones used by DiCENet. We propose the use of a four-block structure , Figure 3.2. These blocks consist of various amounts of DiCE units each, the first block consists of a single conventional convolution which performs the initial feature extraction as well as channel modification. The second block is a single max pooling layer which is used to further reduce the feature map dimensions. While this max pooling layer does not perform any feature extraction it is needed to provide a reduced feature map skip-connection for the decoder. The last two blocks consist of a StridedDiCE unit followed by three ShuffleDiCE units in the third block and seven ShuffleDiCE units in the fourth and final block. StridedDiCE units work the same as regular DiCE units, the only difference is that they traverse the input with bigger steps. ShuffleDiCE units consist of one conventional convolution, a batch normalization, a DiCE unit and they perform the channel shuffle operation [67]. Due to the four block network design our proposed structure is able to take advantage of the pre-trained weights that exist for DiCENet while being more lightweight and streamlined for feature extraction instead of image classification. The original DiCENet consisted of several more blocks, similar in structure to block three and four. The choice to utilize four blocks comes from the fact that utilizing more blocks lead to similar overall accuracy for depth estimation. In order to reduce size and complexity, we propose to use the minimum viable number of blocks which in this case is four. Our novel encoder architecture has a significant size and complexity reduction when compared to the encoder of other state-of-the-art lightweight methods such as FastDepth. Furthermore, each of the encoder blocks also provides a good feature output for skip-connections to the decoder. Skip-connections

are commonly used for depth-estimation as they provide feature transfer between the encoder and the decoder. We utilize the output of each encoder block as the features passed to the decoder. The ability to use pre-trained weights and achieve transfer learning is quite important as it provides an overall increase to performance.

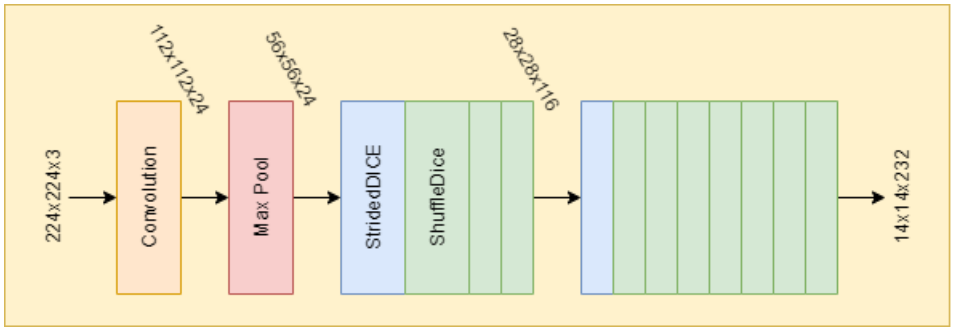

Figure 3.2: Encoder architecture overview. Our novel architecture follows a similar structure as the one proposed in [47] to utilize pre-trained weights, however we propose a modified 4-level version to optimize it for lightweight depth estimation. Color coding in the diagram is used to avoid repetition of layer labels.

3.3.2 Decoder

The decoder's objective is to gradually upsample and combine the feature maps together to form a dense depth map as output. We developed and tested two different decoders for this work named DepthDecode and DiceDecode. A detailed architecture overview of both proposed decoders can be seen in Figure 3.3. The first decoder we propose, DiceDecode, consists of a series of DiCE units, interpolation and concatenative skip-connections. The decoder is grouped into five similarly structured blocks, the first four blocks consist of a DiCE unit followed by a nearest neighbor interpolation and a skip-connection. The fifth block does not utilize skip-connections. The output layer of the decoder is a single pointwise convolution which transforms the channel dimension into the single output dimension of the depth estimation. DiceDecode is capable of producing adequate results while preserving the overall low complexity of the network. After several experiments and network analysis, we believe however that the DiCE units used in our decoder can be replaced with depthwise-separable convolutions which could further reduce the complexity and maintain the accuracy. We came to this conclusion since in the original work [47], the DiCE units were commonly paired with regular convolutions to form the more complex structures such as ShiffleDiCE. Furthermore, while DiCENet is capable of out performing MobileNet as well

as have reduced complexity, we believe this is largely due to the synergy that exists within DiCENet. In our experiments we found that individual DiCE units have increased complexity when compared with depthwise-separable convolutions while also having reduced performance, with regards to MDE. The utilization of more complex DiCE unit structures like ShuffleDiCE is not feasible for the concise design of our decoders. Using basic DiCE units the complexity is already increased when compared to FastDepth's decoder. To this end we propose a second decoder called DepthDecode to overcome the problems faced by DiceDecode.

This second decoder we propose follows a similar overall architecture but the DiCE units are replaced by depthwise-separable convolutions. DepthDecode still follows the five-block architecture where the first four blocks consist of a depthwise convolution followed by a pointwise convolution, an interpolation and a skip-connection. Similar to DiceDecode, the fifth layer does not have a skip-connection. The output layer of our DepthDecode is also a single pointwise convolution. It needs to be noted that our novel DepthDecode differs from FastDepth decoder in both layer dimension and skip-connection utilization. The layer dimensions in our proposed decoder differ significantly from those of FastDepth's decoder.

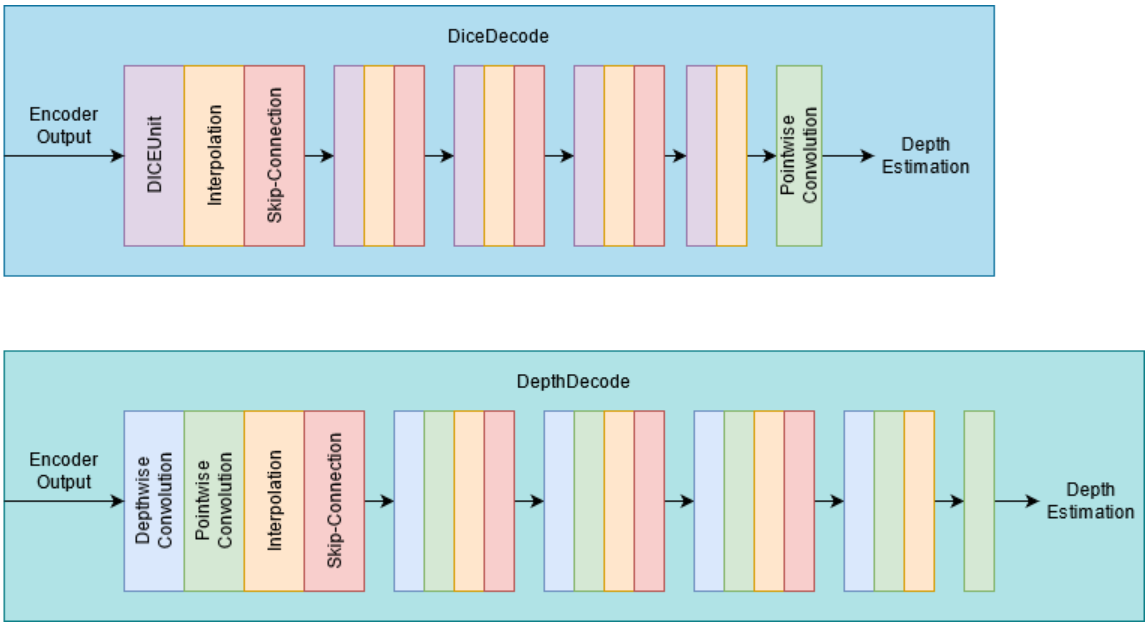

Figure 3.3: Decoder architecture overview. The DiceDecode utilizes DiCE units whereas the DepthDecode utilizes depthwise-separable convolutions. The depthwise-separable convolutions are split into the two sub convolutions, depthwise and pointwise. Color coding in the diagram is used to avoid repetition of layer labels.

As mentioned, both of our novel decoders utilize skip-connections. These skip-connections are important as they provide alternate paths for the gradient which increases overall performance and provides faster convergence. In addition they pass feature map information from different stages of the encoder to the decoder. Since our encoder consists only of four blocks but both of our decoders consist of five, means that the last block in the decoder does not receive any skip-connections. While this is not ideal we made the decision to maintain a five block decoder structure to maintain a 1:1 relation for the spatial dimensions of the input and the output. This means that our networks output has the same dimensions as the input. If the decoders were of a four block design the output would only be half the size of the input image which is not desirable in most cases. The skip-connections employed are concatenating skip-connections as it has been proven in various previous works [39, 65] that they achieve better results when compared to additive skip-connections.

3.3.3 Loss Function

After extensive testing, we utilize the L1 loss function for our final model due to its simplicity. The training for [65] was performed similarly to [45], both used L1 as well. In [45] their experiments showed that it produced better results when compared with the L2 loss, MSE, and the Reversed Huber loss. It is formulated as:

$$L_1 = \sum_{i=1}^{n} |y_{true} - y_{pred}|, \tag{3.1}$$

where n is the number of samples, y_{true} denotes the ground truth target, and y_{pred} is the value predicted by the network.

3.4 Experiments

3.4.1 Experimental Setup

The network is trained and evaluated on the NYU Depth v2 dataset [56] using the official train/test split, example images from the dataset can be seen in Figure 3.4. The training set up is equivalent to the one used by [65] to provide a fair comparison with previous works. This means the use of SVG as optimizer with a learning rate of 0.01, a momentum of 0.9 and a weight decay of 0.0001. The encoder utilizes pre-

trained image classification weights from the ImageNet dataset. The training and evaluation of this model is performed on a Nvidia GTX 1080Ti.

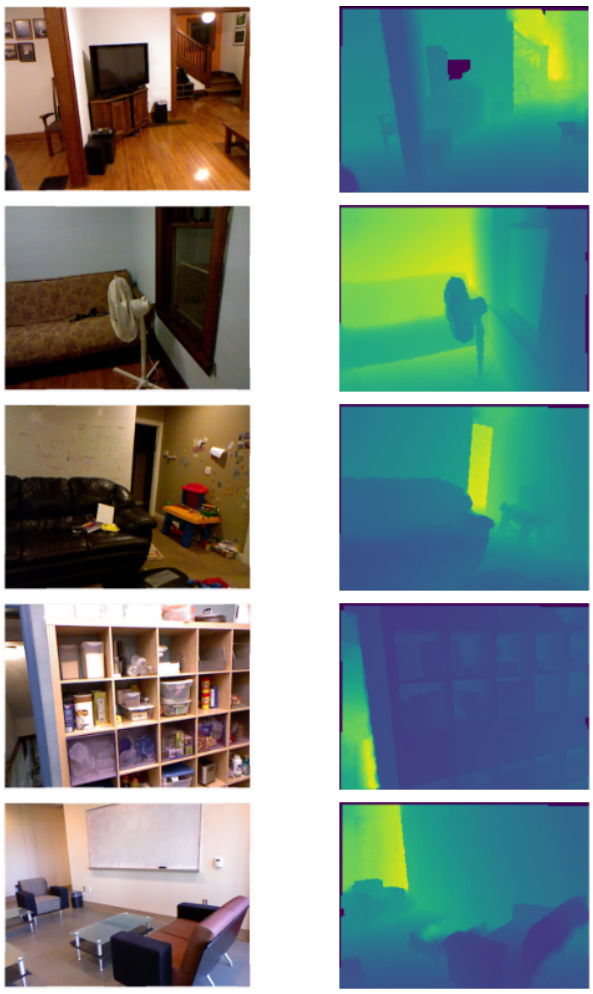

Figure 3.4: Sample images from the NYUv2 dataset. Left are the RGB input images and Right are the corresponding ground truth depth maps.

3.4.2 Pruning

Pruning is a common technique which can be used to reduce the size and complexity of a neural network by removing neurons that have little or no impact on the network's prediction. Some authors utilized pruning as part of their proposed model architecture to achieve a lower parameter and MAC count. We chose not to apply any pruning as we believe that pruning can be applies to most networks and is not an inherent property or advantage of any specific network architecture. Due to this, we will compare our proposed network to unpruned networks or the unpruned version of

such that included pruning as part of their proposed architecture.

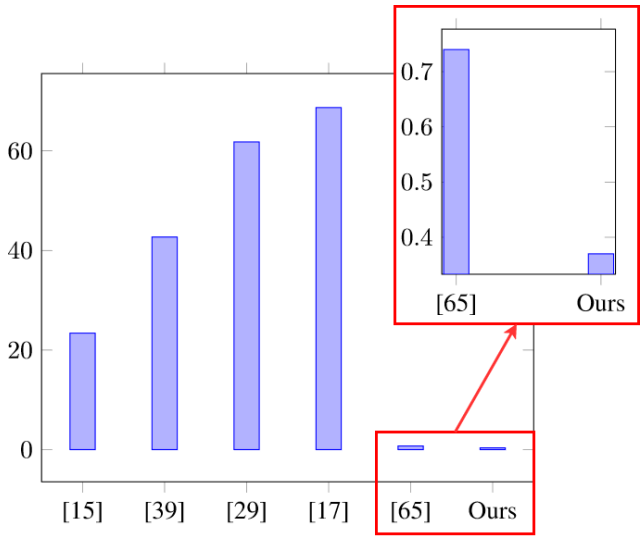

Figure 3.5: Complexity (MACs) comparison between our proposed network and current state-of-the-art networks. The model chosen to represent our proposed network is the one utilizing the DepthDecode decoder.

3.4.3 Comparison to State-of-the-Art

The comparison to state-of-the-art compares our proposed architecture, encoder and both decoders, not only to state-of-the-art heavy networks such as [15, 29, 39] but also to the current state-of-the-art lightweight network [65]. The models are evaluated based on their Absolute Relative Error (Abs Rel), Root Mean Squared Error (RMSE), Absolute Relative Distance (Abs Rel), Mean Squared Logarithmic Error (Log_{10}) as well as the $\delta 1$ metric. In addition, the complexity of each method is also provided, measured by MACs in billion. The number of MACs is computed by counting the number of multiply-accumulate operations each network has to perform during a single forward pass. A visual comparison of the network complexity in MACs is given in Figure 3.5. The detailed comparison with all relevant state-of-the-art networks can be seen in Table 3.1.

In addition, we provide a quick comparison with our immediate competitor FastDepth which is given in Table 3.2. From these comparison, the achievements are quite obvious. Our novel architecture is able to significantly reduce the size achieved by FastDepth without a significant loss to accuracy. From the comparison, it also becomes apparent that a pure use of DiCE units in the decoder is sub-optimal when

Table 3.1: Comparison with current state-of-the-art heavy networks. For $\delta 1$ higher is better and for every other criteria lower is better.

On NYU Depth V2	MACs (G) ↓	Abs Rel ↓	Log_{10} ↓	RMSE ↓	δ 1 ↑
Eigen et al. [15]	23.4	0.158	-	0.641	0.769
Xian et al. [29]	61.8	0.155	0.066	0.660	0.781
Liana et al. [39]	42.7	0.127	0.055	0.573	0.811
DORN (Fu et al 2018) [17]	68.17	**0.115**	**0.051**	**0.509**	**0.828**
Wofk et al. [65]	0.74	-	-	0.599	0.775
Ours (DiceDecode)	0.45	0.181	0.079	0.663	0.704
Ours (DepthDecode)	**0.37**	0.188	0.077	0.654	0.718

compared to depthwise-separable convolutions. As we suspected during our initial experiments, while DiCENet is able to outperform MobileNet and have a lower complexity this does not hold true when comparing individual DiCE units to depthwise-separable convolutions. For our decoders use-case it is not feasible to employ DiCE units. A visual comparison between our proposed work and our immediate competitor FastDepth is given in Figure 3.6. From the comparison, we can see that there is some minor loss in detail. However, the difference in both qualitative and quantitative evaluation is marginal when compared to the reduction in complexity.

Table 3.2: Quick in depth comparison of our proposed architectures with the current state-of-the-art lightweight network FastDepth.

On NYU Depth V2	MACs (G) ↓	Parameters (M) ↓	RMSE ↓	$\delta 1$ ↑
Wofk et al. [65]	0.74	3.93	**0.599**	**0.775**
Ours (DiceDecode)	0.45	1.56	0.663	0.704
Ours (DepthDecode)	**0.37**	**1.21**	0.654	0.718

Overall, our network can still provide adequate depth estimation at a significant size and complexity reduction. We believe that the size reduction achieved here is worth the slight loss in accuracy, less than 6%, as it is able to reduce the model size by almost 70% with respect to the size of FastDepth and the complexity by 50%.

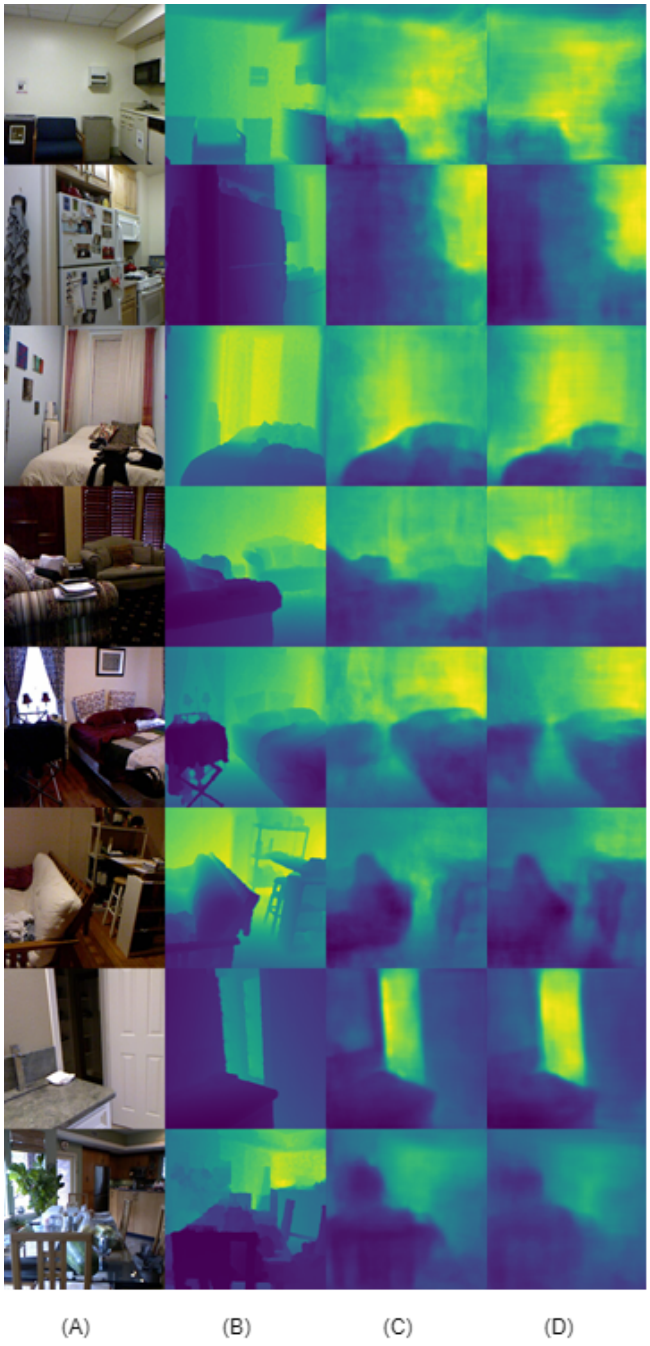

Figure 3.6: Visual comparison between our proposed work and FastDepth. (A) is the RGB input image to the network, (B) is the ground truth target, (C) is FastDepth's result and (D) is our result.

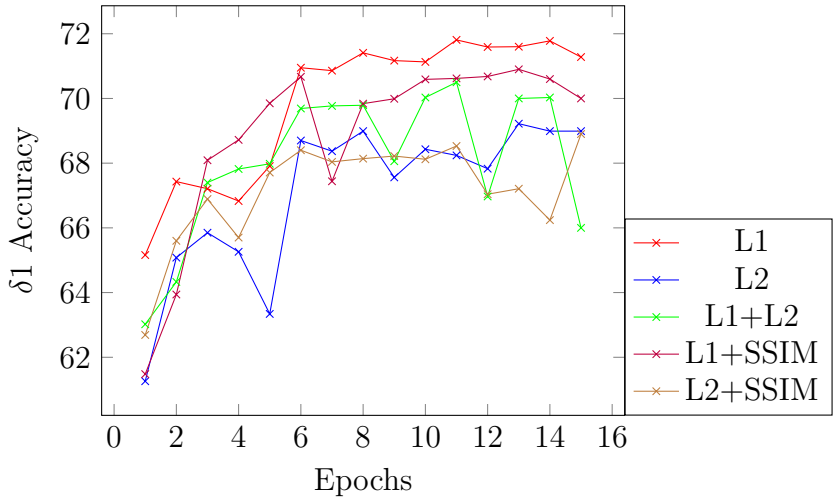

Figure 3.7: Loss function comparison per epoch based on the $\delta 1$ testing accuracy. Results from training our proposed encoder with our DepthDecode decoder.

3.4.4 Ablation Study: Loss Function

As mentioned in 3.3.3 we performed extensive testing on various loss functions. We were able to eliminate certain ones due to the work provided in [45]. They already established that the L1 loss is able to produce better results than the L2 loss, MSE and the Reversed Huber. We wanted to further explore different loss functions. In an attempt to more exhaustively determine the effectiveness of the L1 (eq. 3.1) loss we explored the following loss functions: L2 (eq. 3.2), L1 + SSIM, L1 + L2, L2 + SSIM. Where SSIM refers to the Structural Similarity. The following is the notation for the above mentioned L2 loss, the L1 notation is given in Equation 3.1.

$$L_2 = \sum_{i=1}^{n} (y_{true} - y_{pred})^2, \tag{3.2}$$

where n is the number of samples, y_{true} denotes the ground truth target, and y_{pred} is the value predicted by the network. The SSIM is a metric that reflects the image quality perceptually. It consists of three components which are all independent from one another, Luminance, Contrast and Structure. Using those three components SSIM can determine how two images relate perceptually. The SSIM of two images x_i and y_i is defined as follows:

$$SSIM(x, y) = [l(x, y)]^{\alpha} \cdot [c(x, y)]^{\beta} \cdot [s(x, y)]^{\gamma}, \tag{3.3}$$

where l, c and s refer to the luminance, contrast and structure of x_i and y_i respectively. The luminance is calculated through mean intensity which is defined as follows:

$$l(x, y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2\mu_y^2 + c_1}, \tag{3.4}$$

where μ_x and μ_y are the are the averages of x_i and y_i respectively. c_1 is a constant defined as:

$$c_1 = (k_1 L)^2, \tag{3.5}$$

where L is the dynamic range of the pixel values and k_1 is a constant of usually 0.01. The contrast is calculated the following:

$$c(x, y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2\sigma_y^2 + c_2}, \tag{3.6}$$

where σ_x and σ_y are the are the variances of x_i and y_i respectively. c_2 is a constant defined as:

$$c_2 = (k_2 L)^2, \tag{3.7}$$

where k_1 is a constant of usually 0.03. The structure is defined by the following equation:

$$s(x, y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}, \tag{3.8}$$

where σ_{xy} is the covariance of x_i and y_i and c_3 is a constant defined as:

$$c_3 = \frac{c_2}{2}, \tag{3.9}$$

The SSIM definition given in Equation 3.3 contains three parameters α, β and γ. These parameters are generally set 1 which simplifies the SSIM definition to the following:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(\sigma_{xy} + c_2)}{(\mu_x^2\mu_y^2 + c_1)(\sigma_x^2\sigma_y^2 + c_2)}, \tag{3.10}$$

The results of our experiments can be seen in Figure 3.7. From the figure it is apparent that the L1 loss is the best choice as it is not outperformed by any of the other loss functions. The L1+SSIM loss had s similar progression but ultimately fell short of the results achieved by using L1 exclusively. Overall, this is a desired outcome as L1 is computationally inexpensive and performs better than other more complex combination losses, for our network.

3.4.5 Ablation Study: Real-World Applications

Since our network is designed for supervised training the applications need to be within the area of scenes covered by the existing datasets. Continuing with the NYUv2 dataset which contains data on indoor scenes, one of the main real-world applications would be augmented reality. In AR monocular depth estimation is used to properly place objects in the scene as well as to perform occlusion handling. The depth estimation is used to determine the relative distance of objects in the scene and from that the proper spatial placement can be determined. Using MDE to perform occlusion handling can improve realistic object placement including hiding the AR object behind real objects in the scene and determining what part of the AR object would be visible if it were real. With a rapid increase of AR applications on mobile devices as well as wearable gear, like the Microsoft HoloLense, the need for fast and resource cheap neural networks grows.

Another application for MDE on mobile devices is for a number of image filters. From filters that achieve the Bokeh effect to filters that apply effects to the scene such as under water effects or a scene scan animation [59]. While some of those utilize AR elements they remain image filters at their core. With Bokeh effects the depth estimation is used to distinguish the foreground from the background which can then be used to blur the background. For more advanced filters and AR effects the depth estimation is used to modify the filter in a way that makes the effect appear more realistic by applying darker colours or shadows.

Regardless of AR or image filters and effects, they are increasingly being used on mobile devices. While the compute capabilities of mobile devices have increased significantly over the last years they are still unable to match the performance of dedicated graphics processing units commonly used to train and use neural networks. Furthermore, since both mobile devices and other wearable AR gear are designed to run on battery power the power consumption of neural networks needs to be as low as possible. In our experiments we have shown that we were able to achieve a significant reduction in computational complexity which in turn also reduces the power consumption.

3.5 Conclusion

In this work, we proposed a novel lightweight architecture for monocular depth estimation. We proposed a novel encoder architecture as well as two different decoder architectures. Our model is able to greatly reduce the size and complexity of current state-of-the-art lightweight networks. There is a slight reduction in accuracy but we believe that the trade off is worth it. Furthermore, we believe that this model would profit from unsupervised learning environments and is more suitable for real-time learning. Despite monocular depth estimation being the main focus of the network proposed here, we believe that it can be of significant use in other branches as well.

Chapter 4

A Lightweight Self-Supervised Training Framework for Monocular Depth Estimation

4.1 Overview

Depth estimation is of great interest for various sectors, for example robotics or wearable augmented reality gear. Monocular depth estimation is of particular interest due to its low complexity and cost. Research in recent years as shifted away from supervised learning towards unsupervised or self-supervised approaches. While there have been great achievements, most of the research has focused on large heavy networks which are highly resource intensive which makes them unsuitable for systems with limited resources. We are particularly concerned about the increased complexity during training that current self-supervised approaches bring. In this section, we propose a lightweight self-supervised training framework which utilizes computationally cheap methods to compute ground truth approximations. In particular, we utilize a stereo pair of images during training which are used to compute photometric reprojection loss and a disparity ground truth approximation. Due to the ground truth approximation our framework is able to remove the need of pose estimation and the corresponding heavy prediction networks that current self-supervised methods have. In the experiments, we have demonstrated that our framework is capable of increasing the generator's performance at a fraction of the size required by the current state-of-the-art self-supervised approach.

4.2 Introduction

Depth estimation is a fundamental and ill-posed problem of computer vision. There is a great interest in many areas from scene reconstruction [41] to augmented reality [23]. It has long been a key point of research, however, most traditional methods require multiple view points. Supervised learning of large deep convolutional neural networks overcame this issue. [15, 29, 39]. In addition, CNNs are used in combination with passive sensors, cameras, which are usually cheaper and lighter than their active counterparts like LIDAR. However, supervised learning has the problem of requiring large amounts of labeled data for training. This data is available to some degree for certain specific scenarios such as interior scenes with the NYUv2 dataset[56]. While there is data available there are several different scenes that are not at all or only limitedly covered in the datasets available. In order to allow for more versatile training, of various settings, unsupervised approaches were developed in recent years, the two most prominent being Godard et al. [22] and Zhou et al. [69]. Both of these

approaches require two or more images during training but not during inference. From these proposed works many new and improved training architectures were developed both unsupervised [25, 44] and self-supervised [23].

Most recent research has revolved around big heavy networks both for unsupervised and self-supervised networks. There are lightweight approaches out there such as MiniNet [44] and PyDNet [49]. Both MiniNet [44] as well as MonoDepthV2 [23] utilize secondary networks during training to boost their performance. In both cases the networks are utilized to provide pose estimation between the images. These secondary networks are large heavy networks which are not utilized during inference. However, they drastically increase the complexity during training. We propose a novel self-supervised lightweight training framework, Figure 4.1, which reduces the training complexity while still maintaining an increase in performance. Our novel self-supervised framework consists of three distinct parts, the generator, the discriminator and the disparity calculation. The generator is the target network which makes the disparity predictions and whose performance is to be increased. The discriminator brings adversarial training which benefits the generator's performance. The self-supervision is added through the use of ground truth approximation via conventional disparity map calculation. Our framework uses a stereo pair of images at training time to compute the ground truth approximation while the generator is given one of the images to make a disparity prediction. That prediction is then compared to the calculated disparity giving the self-supervised loss. Our training framework is targeted specifically for lightweight generators. Our novel self-supervised framework is able to boost the generator's performance while still maintaining a low complexity during training. Our novel framework eliminates the need for pose estimation entirely making the large secondary networks used by MonoDepthV2 and MiniNet unnecessary. We do still utilize a secondary network, the discriminator but it is significantly smaller and less complex than the pose estimation networks. While our novel approach does not provide the same improvements as the current state-of-the-art approach MonoDepthV2, it is able to boost performance requiring about 3.21% the size of MonoDepthV2. This size and complexity decrease is particularly noticeable in a significant reduction of training time. Our approach is able to complete each epoch at approximately $1/5$ the time that MonoDepthV2 takes.

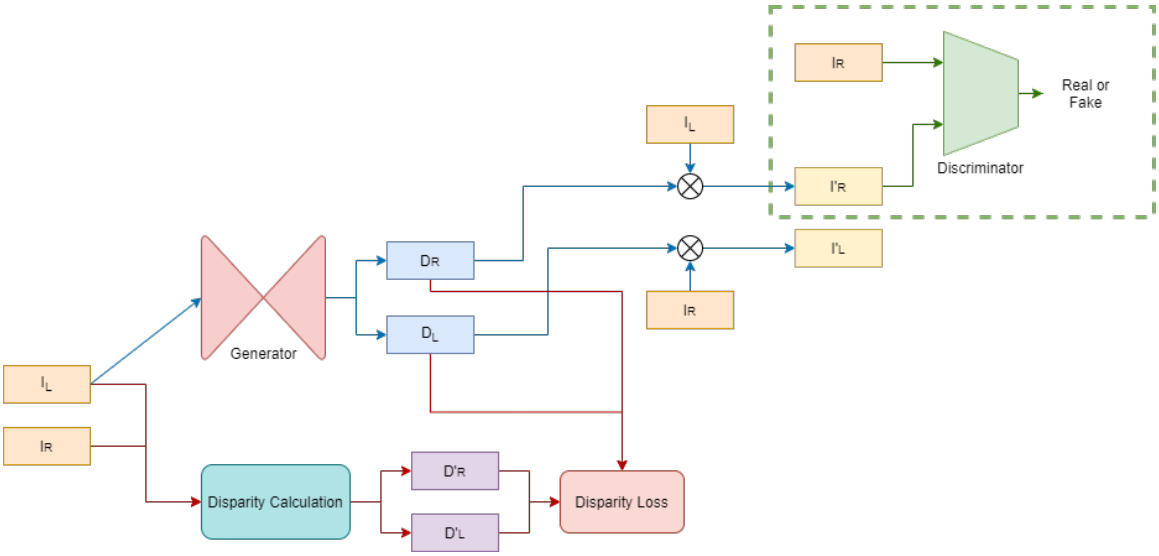

Figure 4.1: Our proposed Self-Supervised framework overview. The blue arrows represent the work of [22] resulting in the photometric loss with left-right consistency. The green arrows represent the work of [25] giving adversarial training. The red arrows show our proposed addition to provide self-supervision by calculating disparity maps for both the left and right image.

4.3 Self-Supervised Framework

4.3.1 General Framework Explanation

Our novel training framework exploits the existing need for a stereo image pair at training time that current unsupervised methods have. Unlike current state-of-the-art self-supervised approaches there is no need for pose information at training time or pose estimation through secondary networks. Instead we propose to compute a ground truth approximation from the stereo images. A general overview of our proposed framework is given in Figure 4.1.

In the figure the three key components of our network become apparent. The generator, the discriminator and the disparity calculation. The generator is the network which makes the disparity predictions needed to compute the back projection of the stereo input. It is also the network whose performance is to be increased by our framework. The discriminator provides adversarial learning, further explained in 4.3.3. The disparity calculation provide the ground truth approximation. Our proposed framework maintains the advances made by MonoDepth [22] utilizing their left-right consistency and photometric loss calculation as a basis for our framework.

Furthermore, inspired by Groenendijk et al. we utilize their insight into adversarial learning with regards to unsupervised monocular depth estimation to optimize our framework further. Our proposed disparity calculation is the key component which improves network performance and provides self-supervised learning. We utilize conventional methods to compute a disparity map estimation which can be compared to the generators predicted disparities to provide self-supervised loss. While the inherent ambiguity of disparity map calculation, due to for example occluded pixels, leads to a computation that is not 100% accurate it is still sufficient to provide the lightweight generator additional information during training. In addition, while not being a true ground truth, the computation of disparity maps is inexpensive and trivial compared to the secondary networks used for pose estimation by MonodepthV2[23] and MiniNet[44]. While our framework still utilizes a secondary network, the discriminator, it is significantly smaller and less complex than the heavy networks used by MonodepthV2 and MiniNet. The three main components of our framework are laid out in detail below.

4.3.2 The Generator

The generator represents the target network of our framework. The purpose of our framework is to improve it's performance in monocular depth estimation. While our self-supervised framework is indiscriminate about the generator's architecture on a macroscopic level it is specifically designed to improve lightweight generator's performance and not heavy ones. The generator is trained to make disparity map predictions. The generator only ever receives one of the stereo images as input, commonly the left one. Based on the single image input the generator predicts a left and a right disparity map, d^l and d^r. While a single prediction for the left and right disparity is possible, optimal performance is achieved when a number of predictions are made for each side. These predictions are made at different resolutions, often called scales, commonly these resolutions are, 1:1, 1:2, 1:4, 1:8 and 1:16. The number of scales depends on the generators architecture. The predictions are then used with the stereo image pair given to compute projected, or reconstructed, versions of those inputs. Using the warping function f_w the left and right images can be reconstructed in the following way:

$$\tilde{I}^r = f_w(d^r, I^l), \tag{4.1}$$

$$\tilde{I}^l = f_w(d^l, I^r), \tag{4.2}$$

where $\tilde{I}^r$ and $\tilde{I}^l$ are the reconstructed images and I^r and I^l are the training images give. In an optimal case the reconstructed images should be identical to the original inputs. While this is not possible with current methods, a well performing generator should predict disparities d^l and d^r so that the reconstructed images are as close as possible to the originals. Therefore, the purpose of our self-supervised framework is to increase the generators ability to predict accurate disparities.

4.3.3 Adversarial Learning

Inspired by Groenendijk et al. [25] we utilize adversarial learning to improve the generators performance. Their thorough research into the impact of different discriminator designs and GAN methods provides a good knowledge base for our framework. They determined that without modifying the generator's architecture, with the addition of normalization layers, the utilization of adversarial learning had a negative impact on the generator's performance. In particular the usage of lightweight discriminators in correspondence with the WGAN architecture [6] proved to be particularly harmful. However, in order to maintain our lightweight design we use a small 4-layer discriminator consisting of three regular convolutions and a single fully connected layer. Inspired by Groenendijk et al., our discriminator's convolution layers are set up as follows, all three use a kernel size of three with a stride of 2. The first layer increases the channel dimension from three to 64, the second doubles that to 128 and the third increases it further to 256. The linear layer consists of 4096 neurons which make a single prediction, whether the input is fake or real. While initial tests using only adversarial learning in an unsupervised setting proved Groenendijk et al.'s conclusions correct we determined in our experiments that the utilization of a lightweight WGAN architecture in combination with out self-supervision proved to be effective in increasing network performance. While the generator is only fed one input image it makes two predictions, d^l and d^r. These disparities are then used to reconstruct the stereo pair images to produce the reconstructed images $\tilde{I}^l$ and $\tilde{I}^r$ respectively. The discriminator only examines the original and reconstructed right images I^r and $\tilde{I}^r$. The loss for the generator, L_w^G, and the discriminator L_w^D are defined as:

$$L_w^G = \mathbb{E}[D(\tilde{I}^r)], \tag{4.3}$$

$$L_w^D = \mathbb{E}[D(I^r) - D(\tilde{I}^r)] + \lambda \Omega_{GP}, \tag{4.4}$$

where Ω_{GP} is the gradient penalty and λ is set to 10 as defined by WGAN-GP [26].

4.3.4 Disparity Calculation

The disparity calculation is the key component for our self-supervised framework. Due to the disparity calculation we are able to remove the need for pose estimation completely, which as mentioned above removes the need for a secondary network for pose estimation. This allows us to reduce the frameworks complexity significantly. While there are several approaches to compute disparity maps we propose the use of the Block Matching algorithm. It is simple and efficient, and the data available in the dataset utilizes a pair or stereo cameras which are calibrated allowing for very efficient disparity computation. Our framework uses a block size of 15 pixels as we have found that this performs best in terms of increasing the generators performance. Furthermore, the Sum of Absolute Differences is used to locate similar blocks and get the disparity value. Our framework computes two disparity approximations one for the left view and one for the right. This is done because the generator also predicts a left and a right disparity map. In order to compute these two disparity maps we first utilize the left image as source and the right image as offset to compute the first disparity. The second is then computed by reversing the order, the right image is the source and the left image is the offset.

4.3.5 Disparity Loss

In order to utilize the compute disparity maps to provide the generator with information we compare the computed disparities to the predicted one. We evaluate the generators performance based on the L1-norm. We performed similar testing as the one detailed in 3.4.4 and achieved similar results. In our tests we explored the L1, L2 and SSIM as well as the combinations of them but did not achieve better performance than with the use of the pure L1-norm. As previously mentioned this is a desirable outcome as the L1-norm is computationally efficient. Two separate losses are computed, one for the left prediction and one for the right prediction. Those are then combined in a weighted matter defined as:

$$L_{Self} = \alpha L_{left} + (1 - \alpha) L_{right}, \tag{4.5}$$

where L_{left}, L_{right} are the losses for the left and right image respectively and α is the weight constant for the left image. The right image's weight is $(1-\alpha)$ to achieve a total weight of 1 or 100%. In our tests we found that using an equal weighting, $\alpha = 0.5$, resulted in the best performance increase. The weighted total is then added to the sum of photometric reconstruction loss for each predicted scale. The photometric reconstruction loss as proposed by Monodepth is defined as follows:

$$L_{photometric} = \sum_{s=1}^{n} C_s, \tag{4.6}$$

where C_s is defined as:

$$C_s = \alpha_{ap}(C_{ap}^l + C_{ap}^r) + \alpha_{ds}(C_{ds}^l + C_{ds}^r) + \alpha_{lr}(C_{lr}^l + C_{lr}^r), \tag{4.7}$$

where C_{ap} is the Appearance Matching loss which evaluates how similar the reconstructed image looks to the orginal. C_{ds} enforces smooth disparities and C_{lr} enforces the left and right disparities to be consistent. The formulation for the Appearance Matching loss C_{ap} is:

$$C_{ap}^l = \frac{1}{N}\sum_{i,j} \alpha \frac{1 - SSIM(I_{ij}^l, \tilde{I}_{ij}^l)}{2} + (1-\alpha)||I_{ij}^l - \tilde{I}_{ij}^l||, \tag{4.8}$$

where α is a constant set to 0.85. The disparity smoothness loss C_{ds} is defined by the following equation:

$$C_{ds}^l = \frac{1}{N}\sum_{i,j} |\partial_x d_{ij}^l| e^{-||\partial_x I_{ij}^l||} + |\partial_y d_{ij}^l| e^{-||\partial_y I_{ij}^l||}, \tag{4.9}$$

where ∂d is the gradient of the disparities and ∂I is the image gradient which is used to have edge awareness in the loss calculation. The left-right consistency loss C_{lr} can be defined by the following:

$$C_{lr}^l = \frac{1}{N}\sum_{i,j} |d_{ij}^l - d_{ij+d_{ij}^l}^r|. \tag{4.10}$$

This left-right consistency is a modified L1-norm to enforce the consistency between the left and the right predicted disparity map. As evident from Equation 4.7 all the above equations are computed for both the left image and the right image.

Combining the generator loss of Equation 4.3 with the photometric reconstruction loss of Equation 4.6 and our self supervised loss from Equation 4.5 we get the following complete loss for the generator:

$$L_G = L_{photometric} + \phi_G L_w^G + \tau_G L_{Self}, \tag{4.11}$$

where ϕ_G and τ_G are weight constants set to 0.1 and 1.0 respectively.

4.4 Experiments

4.4.1 Experimental Setup

In order to ensure a fair comparison between our framework with current state-of-the-art methods it and consequently the generators are trained and evaluated on the KITTI[21] dataset, sample input images can be seen in Figure 4.2. In particular the Eigen split [16] of the KITTI dataset is used. The Eigen split consists of 22.6K training images, 888 validation images and 697 test images. The implementation of our framework as well as tested generator networks is done in Pytroch [48]. Our training set up follows that of Groenendijk et al. [25] to ensure fair comparison. This means that all models are trained over 50 epochs in mini-batches of 8 using an Adam optimizer [37] and plateau learning rate scheduler [51] with an initial learning rate of 10^{-4}. None of the models used in our framework have any pre-trained weights. The training and evaluation of this model is performed on a Nvidia GTX 1080Ti. Models are evaluated quantitatively using the following metrics widely used by other methods [22, 25, 49], Absolute Relative Distance (Abs Rel), Squared Relative Distance (Sq Rel), Root Mean Squared Error (RMSE), Root Mean Squared Logarithmic Error (RMSE log), and accuracy within threshold t (δt , with t $\in [1.25, 1.25^2, 1.25^3]$).

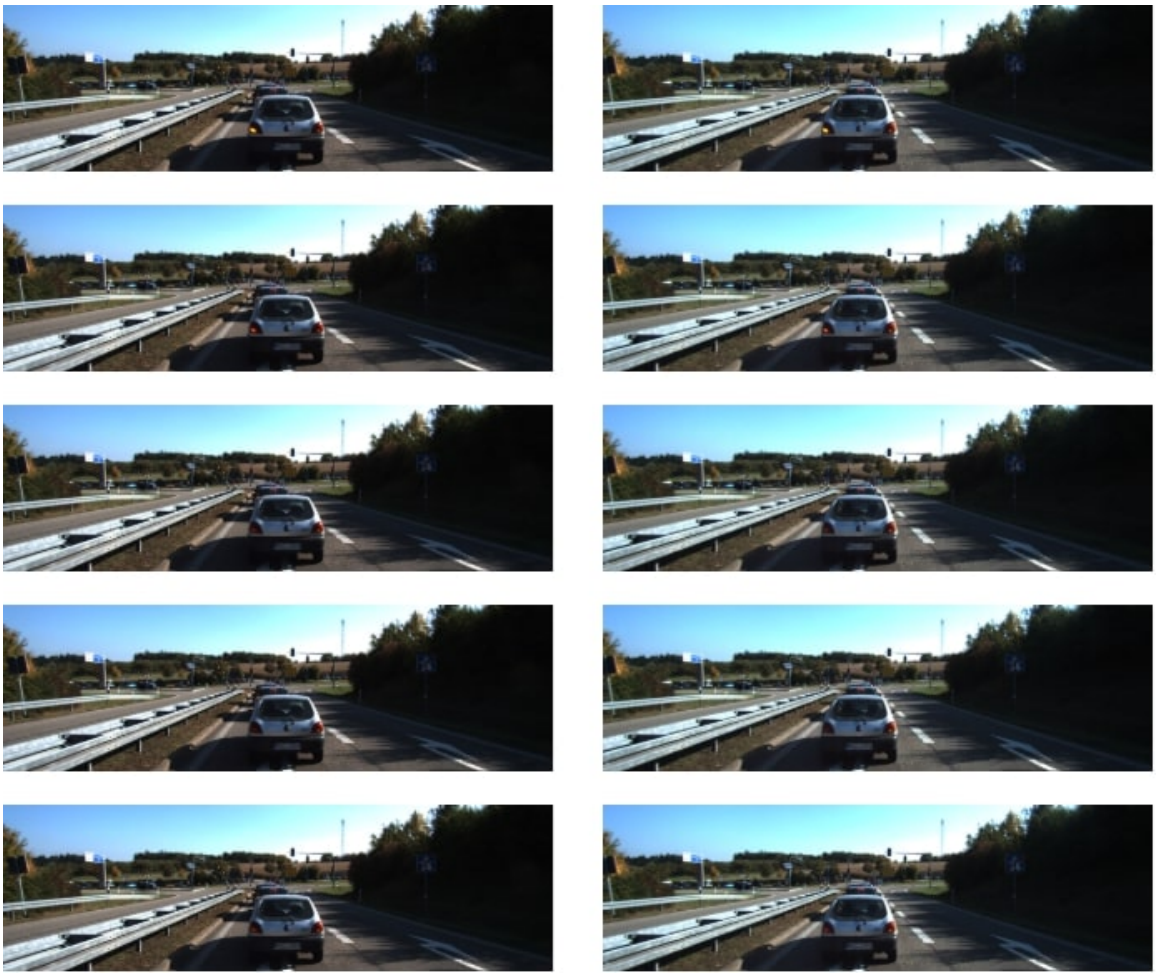

Figure 4.2: Sample images taken from the KITTI dataset. Left and right view are on the Left and Right respectively. Samples clearly show that the samples or extracted from a video capture.

4.4.2 Comparison to State-of-the-Art Self-Supervised Solution

We compare our proposed framework with the current state-of-the-art self-supervised approach MonodepthV2 [23]. The frameworks are evaluated on the quantitative metrics above as well on the size increase, parameters in million, at training time. These comparisons are intended to determine the trade of between size and improved generator results. The detailed comparison is given in Table 4.1. The training size in the table is evaluated based on the size increase of both our proposed method and that of MonoDepthV2. The results given in the highlighted rows are taken from Aleotti et

al. [4] since they have previously performed a quantitative analysis of training both PyDNet and FastDepth using MonoDepthV2.

From the results given it becomes apparent that both our novel self-supervised framework as well as MonoDepthV2 are able to improve the overall performance of the generator for both PyDNet and FastDepth. While MonoDepthV2 achieves an overall better performance it requires a significant increase in size and complexity to do so. Our framework is able to achieve competitive performance and is able to surpass MonoDepthV2 for some metrics. Furthermore, our proposed method is able to achieve this with a size increase of only ∼3.21% of the size increase from MonoDepthV2. In addition, this significant difference in size and complexity becomes quite apparent when comparing training time of the two frameworks. When training PyDNet for 50 epochs, MonoDepthV2 requires approximately 2500 seconds per epoch equaling ∼30 hours total. In comparison our framework requires around 475 seconds per epoch totaling ∼6.5 hours.

Table 4.1: Comparison of our proposed self-supervised approach with MonoDepthV2. Comparison based on increased network performance as well increased overall size based on parameters at training time. Highlighted results taken from [4] as they have performed a quantitative analysis of PyDNet trained on MonoDepthV2.

Generator	Supervision	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑	Training Size (Params. in M.)
PyDNet ([49])	L-R Unsupervised	0.163	1.399	6.253	0.262	0.759	0.911	0.961	1.970
PyDNet	MonoDepthV2	0.154	1.307	**5.556**	0.229	**0.812**	**0.932**	**0.970**	+11.690
PyDNet	Proposed	**0.145**	**1.154**	5.775	**0.238**	0.792	0.924	0.969	**+0.375**
FastDepth	L-R Unsupervised	0.370	5.868	9.859	1.084	0.614	0.781	0.868	4.02
FastDepth	MonoDepthV2	**0.156**	**1.260**	**5.628**	**0.231**	**0.801**	**0.930**	**0.971**	+11.690
FastDepth	Proposed	0.182	2.221	6.340	0.688	0.787	0.904	0.946	**+0.375**

4.4.3 Ablation Study: Framework Parts Evaluation

In order to determine the efficacy of the different parts of our framework as well as to determine the need for them we evaluate each part individually as well as together. We use pure unsupervised MonoDepth as a baseline and then evaluate the addition of each part. Each step is evaluated on three generator architectures. Two lightweight ones, PyDNet and FastDepth and one heavy VGG based one. While the specific use case for our framework lies with the improvement of lightweight generators we feel the need to explore the impact on heavy networks as well. The results of the comparison can be found in Table 4.2. In the table 'GAN' refers to the addition of adversarial learning through the discriminator, 'Self' refers to the addition of self-supervised

learning through disparity map calculation and 'Self+GAN' refers to the complete proposed framework structure. Training for each part and generator combination is performed over 50 epochs using the same settings mentioned above, this ensures a fair comparison between them.

The results clearly show improved performance for both lightweight generators when using the complete framework. Furthermore, when only self-supervision is used the performance is increased in terms of *delta* threshold accuracies but reduced for the other metrics. The addition of adversarial learning reduces overall performance, which is in line with the results of Groenendijk et al., however when combined with self-supervision we achieve the best performance for lightweight generators. The heavy network experienced an overall negative impact from each individual part as well as the complete framework. This is to be expected as our framework is target for lightweight generators. Overall, these test highlight that our proposed framework is able to improve overall network performance for lightweight generators.

Table 4.2: Evaluation of different framework components, compared with baseline. Baseline is given by training networks on [22] training framework exclusively. The framework components are evaluated for one heavy network and two lightweight networks. PyDNet is given two baselines one from the original paper and one from a Pytorch [48] implementation used for our framework.

Network Architecture	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
VGG Baseline	**0.151**	**1.325**	**5.876**	**0.246**	**0.791**	**0.920**	**0.965**
VGG GAN	0.152	1.357	6.003	0.249	0.788	0.917	0.963
VGG Self	**0.151**	1.335	5.951	0.248	0.789	**0.920**	0.964
VGG Self + GAN	0.159	1.467	6.152	0.256	0.784	0.915	0.961
PyDNet Baseline ([49])	0.163	1.399	6.253	0.262	0.759	0.911	0.961
PyDNet Baseline (Pytorch)	**0.145**	1.246	5.798	0.242	0.786	0.923	0.968
PyDNet GAN	0.160	1.462	6.156	0.256	0.770	0.910	0.962
PyDNet Self	0.149	1.316	5.810	0.245	**0.792**	0.923	0.967
PyDNet Self + GAN	**0.145**	**1.154**	**5.775**	**0.238**	**0.792**	**0.924**	**0.969**
FastDepth Baseline	0.370	5.868	9.859	1.084	0.614	0.781	0.868
FastDepth GAN	0.207	2.680	6.824	0.731	0.760	0.890	0.938
FastDepth Self	0.193	2.416	6.440	0.696	0.785	0.900	0.943
FastDepth Self + GAN	**0.182**	**2.221**	**6.340**	**0.688**	**0.787**	**0.904**	**0.946**

4.4.4 Ablation Study: Real-World Application

As previously mentioned, there are a number of datasets available to train models on monocular depth estimation. While they contain a significant amount of sample data for both indoor and outdoor scenes there is still a large quantity of possible scenes

not contained in those datasets. This makes it important to be able to conduct continued training after deployment for several applications. Since previous self-supervised approaches required large amounts of resources at training time, this was not possible for applications deployed on resource weak systems.

Robotics is one of the fields with several applications where there is not enough scene data available to perform adequate training of the network. In particular, robotic systems that are being deployed underwater or on other planets. While current and previous rovers such as the Mars rovers Curiosity [2, 8] and Perseverance [3, 7] or the lunar rovers Chang'e 4 [42] and Lunokhod 2 [54] have collected a large number of images from the surface of Mars and the Moon, this data is largely raw captured footage and not collected in a usable training set. Furthermore, the relatively small distances traversed by the rovers only amounts to a small scene diversity for the respective settings. The rovers currently deployed by NASA, Curiosity and Perseverance, have highly advanced camera sensor suites, using multiple cameras and additional sensors such as lasers to analyze their surroundings. Due to these vision sensors as well as a number of other sensors, to analyze samples collected amongst other uses, the rovers have a high power consumption and are also physically quite large, the size of a small car. While these rovers are invaluable for current scientific research on Mars, there is also a need for much smaller and more autonomous rovers for both Mars and lunar missions. These smaller rovers are much more concerned about being lightweight and reducing power consumption. One such rover is the Audi Lunar Quattro (ALQ) [1] which is the size of a large remote controlled car. Using the ALQ as an example, it is equipped with far less sensors compared to it's larger counter parts, it's vision system consists of two stereo cameras as well as a telephoto zoom camera. In order for these rovers to maneuver autonomously accurate depth estimation is needed. Since high accuracy is needed to adequately avoid obstacles, self-supervision is the superior choice compared to unsupervised, reduced accuracy, and supervised, limited training data. The small size limits both the power and compute resources making the use of lightweight neural networks a key concern. As mentioned, there is limited training data available for the scenes these rovers will be exposed to. This means that continued training while the rover is in use might be necessary to achieve the accuracy needed. With current self-supervised approaches this would be unfeasible since the compute resources required by MonoDepthV2 or MiniNet would increase training time drastically as well as consume large amounts of power. Our proposed approach would mitigate this issue, as shown in our experiments

we are able to enhance network performance at marginal cost to size and compute resources. This means that the rover's vision systems can be continuously trained to optimize performance for the target setting at minimal cost of compute resources and power consumption.

4.4.5 Ablation Study: PyDNet vs FastDepth Design Efficacy

Both FastDepth and PyDNet represent the state of the art lightweight networks in their respective training domain. FastDepth for supervised learning and PyDNet for unsupervised or self-supervised. The significantly different network design of the two networks begs the question of efficacy in each other's domain. FastDepth utilizes a distinct encoder-decoder design using the lightweight encoder MobileNet and a lightweight decoder consisting of depthwise-separable convolutions. PyDNet on the other hand does not use any lightweight convolutions at all but rather has a compact design, fusing the encoder and decoder. We aim to evaluate the efficacy of PyDNet in the supervised training environment as well as the efficacy of FastDepth in the un- or self-supervised training settings.

First, we evaluate the efficacy of both lightweight networks in the supervised training section. Since FastDepth is the state-of-the-art for this settings we will follow the training and evaluation approach outlined by Wofk et al. [65]. We will utilize the NYUv2 dataset for training along with the SVG optimizer with a learning rate of 0.01, a momentum of 0.9 and a weight decay of 0.0001. Since FastDepth was designed for this training environment no modifications need to be made. PyDNet requires some minor modification as it's purpose is to perform a series of disparity predictions. We maintain the overall architecture but modify the output layer of the prediction to output a 1D output for the depth map and remove the sigmoid activation function. In addition we will only evaluate the largest scale predicted by PyDNet which is 1:2. The results of the evaluation are given in Table 4.3. The results for FastDepth are taken directly from the paper [65]. The evaluation is performed on two metrics, RMSE and the $\delta 1$ accuracy. We perform training on an input size of 256x256 size as PyDNet's architecture requires the input to be divisible by 64. In addition, both the network size and complexity is compared. Using the table below it is apparent that FastDepth is significantly outperforming PyDNet in terms of RMSE and $\delta 1$.

Table 4.3: Network performance comparison between PyDNet and FastDepth on the NYUv2 dataset using supervised learning.

Network	RMSE ↓	$\delta 1$ ↑	Parameters (M) ↓	MACs (G) ↓
PyDNet	1.214	0.373	**1.97**	2.47
FastDepth	**0.599**	**0.775**	3.93	**0.96**

Second, we evaluate the network's performance for unsupervised training. As PyDNet is the state-of-the-art network for this training environment we will perform the evaluation based on it's training settings. We will use the unsupervised Left-Right Consistency training proposed by Godard et al. [22]. We train using stereo inputs from the KITTI dataset and the Adam [37] optimizer with an initial learning rate of 10^{-4}. In terms of modifications, PyDNet does not need any modifications for this training as this is it's intended training environment. We also did not perform any modifications for FastDepth which is why it will only be trained and evaluated based on it's full scale output. The evaluation results can be seen in Table 4.4. The results for PyDNet are taken from the original paper [49]. We perform the testing on an input size of 256x512. From the table it becomes apparent that for unsupervised learning PyDNet significantly outperforms FastDepth in both RMSE and $\delta 1$.

Table 4.4: Network performance comparison between PyDNet and FastDepth on the KITTI dataset using the Eigen Split. Training performed using the unsupervised Left-Right Consistency method.

NetWork	RMSE ↓	$\delta 1$ ↑	Parameters (M) ↓	MACs (G) ↓
PyDNet	**6.253**	**0.759**	**1.97**	4.93
FastDepth	9.859	0.614	4.02	**1.93**

The test results clearly show distinct performance differences between the two networks. While FastDepth's unsupervised performance is worse than that of PyDNet it is able to achieve a usable performance. In comparison, PyDNet performs very poorly for supervised training and is not able to match FastDepth's performance. It also needs to be noted that if FastDepth is to be used in self-supervised settings such as MonoDepthV2 or our proposed method, it is able to compete with PyDNet's results. This leads to the conclusion that FastDepth is the superior design choice for supervised and most self-supervised training settings where as PyDNet performs particularly well in unsupervised settings. When deciding what network architecture

to utilize two more things need to be considered, size and complexity. PyDNet is smaller in terms of parameters as it requires around half the parameters that FastDepth needs. However, FastDepth has the advantage of having significantly lower complexity at 40% the complexity of PyDNet. This means that the design decision lies with both the target application as well as the resources available on the system. If the system is limited in storage and RAM and near real-time performance is not needed then a design similar to that of PyDNet should be chosen. If however, the system is limited by power consumption or near real-time inference is needed then a design following that of FastDepth, using lightweight convolutions, should be used.

4.5 Conclusion

In this work, we proposed a novel lightweight self-supervised training framework for monocular depth estimation. Our framework specifically targets lightweight networks and aims to increase their performance at marginal cost to size and complexity. We leverage the existing need for stereo image pairs to be given at training time to compute disparity ground truth approximations. Our method eliminates the need for pose estimation that other state-of-the-art frameworks have. For this reason we are able to significantly reduce the overall complexity of self-supervised learning. Our novel framework is able to compete with the current state-of-the-art method in terms of performance while significantly reducing the size. While we cannot exceed current performance in all metrics we believe that the trade off between size and performance is worth it.

Chapter 5

Conclusion & Future Work

5.1 Overview

The approaches proposed in this thesis provide a significant reduction in size and complexity to current state-of-the-art methods while maintaining a competitive accuracy. The proposed lightweight network as well as the lightweight self-supervised training framework can be used in future research and applications. We give detailed explanations about the two approaches as well as possible real world applications for both of them.

5.2 Main Contributions

Our research addresses the need for lightweight deep neural networks for monocular depth estimation. We highlight the issue with current network design and the limitation it poses on weaker systems. We propose two distinct methods to solve this issue, one is a lightweight neural network specifically designed and optimized for monocular depth estimation. The second is a lightweight self-supervised training framework that is designed to improve the performance of lightweight neural networks.

The two key contributions are summarized as follows:

- **First:** We proposed a novel lightweight encoder as well as two decoders specifically designed for the task of monocular depth estimation. Specifically, our novel encoder takes advantage of the state-of-the-art DiCENet. We optimized our encoder in design to be as small as possible but we ensured that our design was able to take advantage of pre-trained weights to increase performance. In addition, we determined the optimal decoder design for our encoder and proposed two decoders one DiCEUnit based and one depthwise-separable convolution based. We highlight the drawbacks of utilizing DiCEUnits outside of DiCENet which led to the second decoder being proposed. Overall, our lightweight network is able to significantly reduce both size and complexity when compared with state-of-the-art lightweight networks. Experimental results showed that, despite complexity and size reduction, our network can still achieve competitive results.

- **Second:** We proposed a lightweight self-supervised training framework specifically designed to boost the performance of lightweight networks. We exploit the need for stereo images at training time that current unsupervised approaches have. Our framework utilizes the stereo images to compute a ground truth approximation which can be used to increase the networks performance. Furthermore, we used adversarial training, for which we used a lightweight discriminator, to further increase network performance. Unlike other state-of-the-art self-supervised methods, we do not require a secondary network for pose estimation since our method does not utilize any pose. Due to these advances we were able to design a training framework that has minimal impact on complexity while increasing overall network performance.

5.3 Conclusion

In this thesis we proposed two distinct solutions for the task of lightweight monocular depth estimation. Both of our methods are able to reduce size and complexity while maintaining competitive results. Our lightweight network is able to reduce the size by roughly 70% and the complexity by 50% compared to the current state-of-the-art method while having an accuracy reduction of less than 6%. The self-supervised

training framework we proposed is able to boost network performance. Other self-supervised approaches achieve higher performance boosts but our framework achieves a competitive increase in performance at just about 3% of the size of other methods.

Appendix A

List of Abbreviations

In this appendix we give a list of abbreviations used in the text.

A

Abs Rel: Absolute Relative Distance
ALs: Activation Layers
ANNs: Artificial Neural Networks
AQL: Audi Lunar Quattro
AR: Augmented Reality

C

CLs: Convolution Layers
CNNs: Convolutional Neural Networks

D

DenseNet: Densely Connected Convolutional Networks
DiCENet: Dimension-wise Convolutions for Efficient Networks
DL: Deep Learning
DNNs: Deep neural networks

E

EDA: Encoder-Decoder Architecture

F

FCLs: Fully Connected Layers

G

GANs: Generative Adversarial Networks

L

LiDAR: Light Detection and Ranging
$\mathbf{Log_{10}}$: Mean Squared Logarithmic Error
LSGAN: The Least Squares Generative Adversarial Network

M

MAC: Multiply-Accumulate
MDE: Monocular Depth Estimation

N

NCC: Normalized Cross Correlation

R

RaGAN: Relativistic GAN
ReLU: Rectified Linear Unit
ResNet: Residual Neural Network
RMSE: Root Mean Squared Error
RMSE log: Root Mean Squared Logarithmic Error

S

SAD: Sum of Absolute Differences
SSD: Sum of Squared Differences
SSIM: Structural Similarity Index
Sq Rel: Squared Relative Distance

W

WGAN: Wasserstein Generative Adversarial Network

Bibliography

[1] Moon rover meet the audi lunar quattro. *PTScientists*.

[2] Mars curiosity rover. *NASA*, 2011.

[3] 2020 mission perseverance rover. *NASA*, 2020.

[4] F. Aleotti, G. Zaccaroni, L. Bartolomei, M. Poggi, F. Tosi, and S. Mattoccia. Real-time single image depth perception in the wild with handheld devices. *Sensors*, 21:15, 12 2020.

[5] I. Alhashim and P. Wonka. High quality monocular depth estimation via transfer learning. *arXiv*, 2018.

[6] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

[7] J. Bell, J. Maki, G. Mehall, M. Ravine, M. Caplinger, Z. Bailey, S. Brylow, J. Schaffner, K. Kinch, M. Madsen, et al. The mars 2020 perseverance rover mast camera zoom (mastcam-z) multispectral, stereoscopic imaging investigation. *Space Science Reviews*, 217(1):1–40, 2021.

[8] J. F. Bell III, A. Godber, S. McNair, M. Caplinger, J. Maki, M. Lemmon, J. Van Beek, M. Malin, D. Wellington, K. Kinch, et al. The mars science laboratory curiosity rover mastcam instruments: Preflight and in-flight calibration, validation, and data archiving. *Earth and Space Science*, 4(7):396–452, 2017.

[9] J. Chang, Y. Choi, T. Lee, and J. Cho. Reducing mac operation in convolutional neural network with sign prediction. *Proceedings of International Conference on Information and Communication Technology Convergence (ICTC)*, pages 177–182, 2018.

[10] P.-Y. Chen, A. H. Liu, Y.-C. Liu, and Y.-C. F. Wang. Towards scene understanding: Unsupervised monocular depth estimation with semantic-aware representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2624–2632, 2019.

[11] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[13] C. Dong, C. C. Loy, K. He, and X. Tang. Image super-resolution using deep convolutional networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(2):295–307, 2016.

[14] 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 *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[15] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. *Proceeding of IEEE International Conference on Computer Vision (ICCV)*, 2015.

[16] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.

[17] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep ordinal regression network for monocular depth estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[19] M. Gadelha, S. Maji, and R. Wang. 3d shape induction from 2d views of multiple objects. In *2017 International Conference on 3D Vision (3DV)*, pages 402–411. IEEE, 2017.

[20] B. Gecer, S. Ploumpis, I. Kotsia, and S. Zafeiriou. Ganfit: Generative adversarial network fitting for high fidelity 3d face reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1155–1164, 2019.

[21] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[22] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 270–279, 2017.

[23] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019.

[24] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[25] R. Groenendijk, S. Karaoglu, T. Gevers, and T. Mensink. On the benefit of adversarial training for monocular depth estimation. *Computer Vision and Image Understanding*, 190:102848, Jan 2020.

[26] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5769–5779, Red Hook, NY, USA, 2017. Curran Associates Inc.

[27] R. A. Hamzah, R. A. Rahim, and Z. M. Noh. Sum of absolute differences algorithm in stereo correspondence problem for stereo matching in computer vision application. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 1, pages 652–657, 2010.

[28] C. Han, H. Hayashi, L. Rundo, R. Araki, W. Shimoda, S. Muramatsu, Y. Furukawa, G. Mauri, and H. Nakayama. Gan-based synthetic brain mr image generation. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 734–738. IEEE, 2018.

[29] Z. Hao, Y. Li, S. You, and F. Lu. Detail preserving depth estimation from a single image using attention guided networks. *Proceedings of International Conference on 3D Vision (3DV)*, 2018.

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[32] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[33] A. Ignatov, R. Timofte, M. Denna, and A. Younes. Real-time quantized image super-resolution on mobile npus, mobile ai 2021 challenge: Report. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 2525–2534, June 2021.

[34] F. Isensee, P. Jaeger, S. Kohl, J. Petersen, and K. Maier-Hein. nnu-net: a self-configuring method for deep learning-based biomedical image segmentation. *Nature Methods*, 18:1–9, 02 2021.

[35] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

[36] P. Kamencay, M. Breznan, R. Jarina, P. Lukac, and M. Radilova. Improved depth map estimation from stereo images based on hybrid method. *Radioengineering*, 21, 04 2012.

[37] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[38] A. Kuhl. Comparison of stereo matching algorithms for mobile robots. *Centre for Intelligent Information Processing System*, pages 4–24, 2005.

[39] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. *Proceedings of Fourth International Conference on 3D Vision (3DV)*, 2016.

[40] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[41] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala, and T. Aila. Noise2noise: Learning image restoration without clean data. *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 80:2965–2974, 2018.

[42] C. Li, Z. Wang, R. Xu, G. Lv, L. Yuan, Z. He, and J. Wang. The scientific information model of chang'e-4 visible and near-ir imaging spectrometer (vnis) and in-flight verification. *Sensors*, 19(12):2806, 2019.

[43] F.-F. Li, J. Johnson, and S. Yeung. Lecture 9: Cnn architectures, Apr 2019.

[44] J. Liu, Q. Li, R. Cao, W. Tang, and G. Qiu. Mininet: An extremely lightweight convolutional neural network for real-time unsupervised monocular depth estimation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 166:255–267, Aug 2020.

[45] F. Ma and S. Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[46] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2017.

[47] S. Mehta, H. Hajishirzi, and M. Rastegari. Dicenet: Dimension-wise convolutions for efficient networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2020.

[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[49] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia. Towards real-time unsupervised monocular depth estimation on cpu. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5848–5854. IEEE, 2018.

[50] M. Poggi, F. Tosi, and S. Mattoccia. Learning monocular depth estimation with unsupervised trinocular assumptions. In *2018 International conference on 3d vision (3DV)*, pages 324–333. IEEE, 2018.

[51] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[52] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[53] A. Rosebrock. Imagenet: Vggnet, resnet, inception, and xception with keras, Mar 2017.

[54] A. A. Siddiqi. *A Chronology of Deep Space and Planetary Probes 1958-2000*. Monograph in Aerospace History. Nasa, 2002.

[55] L. Sifre. *Rigid-Motion Scattering For Image Classification*. PhD thesis, Ecole Polytechnique, 2014.

[56] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[57] T. Silva. An intuitive introduction to generative adversarial networks (gans), Jan 2018.

[58] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[59] Snap Inc. Snaptchat lense studio, 2018.

[60] Y. Taigman, A. Polyak, and L. Wolf. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*, 2016.

[61] S.-S. Tung and W.-L. Hwang. Depth extraction from a single image and its application. In *Pattern Recognition-Selected Methods and Applications*. IntechOpen, 2019.

[62] N. d. Tyson. In the beginning. *Natural History*, Sep 2003.

[63] U. Udofia. Basic overview of convolutional neural network (cnn), Feb 2018.

[64] G. Wang and H. Ju. A disparity map extraction algorithm for lunar rover bh2. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 4, pages 385–389, 2009.

[65] D. Wofk, F. Ma, T. Yang, S. Karaman, and V. Sze. Fastdepth: Fast monocular depth estimation on embedded systems. *Proceedings of International Conference on Robotics and Automation (ICRA)*, pages 6101–6108, 2019.

[66] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 340–349, 2018.

[67] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6848–6856, 2018.

[68] F. Zhao, Q. Huang, and W. Gao. Image matching by normalized cross-correlation. In *IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, volume 2, pages II – II, 06 2006.

[69] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017.

[70] Y. Zhou, G. G. Yen, and Z. Yi. Evolutionary compression of deep neural networks for biomedical image segmentation. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):2916–2929, 2020.