DEPARTMENT OF INFORMATION AND ELECTRONIC ENGINEERING

POSTGRADUATE PROGRAM
WEB INTELLIGENCE

Monocular Depth Estimation Using Deep Neural Models

MSc THESIS

of

ANGELOS KOUTSOPOULOS

Supervisor : Konstantinos Diamantaras
Professor, IHU

Thessaloniki, February 2024

Preface

This MSc thesis, titled “Monocular depth estimation using deep neural models”, was prepared in the context of fulfilling the required conditions for obtaining my master’s degree in Web Intelligence from the Department of Information and Electronic Engineering of the International Hellenic University based in Thessaloniki.

Having worked on data analytics and descriptive statistics in my BSc thesis, I wanted to expand my knowledge in a similar field: Machine Learning. I also knew about “Draive”, the department’s Machine Learning team with Mr. Konstantinos Diamantaras as the supervisor professor. Thus, I approached him for possible MSc thesis topics in Machine Learning to help the team in future projects. From the suggested topics, monocular depth estimation immediately caught my interest with its visually appealing outputs, so I decided to put it to the test.

Abstract

Depth Estimation is the process of calculating the distance of each pixel from the camera, which is essential for computer vision tasks, such as autonomous driving, Augmented Reality, and robotics. With the rise of Machine Learning, early models predicted depth with images from two cameras, like a pair of eyes. This approach required additional equipment and data, so research has shifted toward estimating depth with only one camera. It lacks stereoscopic vision, however, and is therefore challenging to predict depth accurately. To address this issue, scientists implemented various techniques in their models to recover depth. The present thesis compares BANet, LapDepth, and PixelFormer: three state-of-the-art Deep Learning models with different architectures that estimate depth from a single image. The aim is to train them from scratch on a large dataset and compare the results to determine the winner. For objective evaluation, two different criteria were used: SILog loss and RMSE. In addition, two more datasets test the models through subjective evaluation. The results show that PixelFormer, with its complex architecture, predicts depth more accurately among the three models.

Keywords: BANet, DIODE, computer vision, convolutional neural networks, deep learning, KITTI, LapDepth, machine learning, monocular depth estimation, PixelFormer, Python

Περίληψη

Η εκτίμηση βάθους (depth estimation) είναι η διαδικασία υπολογισμού της απόστασης κάθε pixel από την κάμερα, η οποία είναι απαραίτητη για εργασίες όρασης υπολογιστή (computer vision), όπως η αυτόνομη οδήγηση, η Επαυξημένη Πραγματικότητα και η ρομποτική. Με την άνοδο της Μηχανικής Μάθησης, τα πρώτα μοντέλα εκτιμούσαν το βάθος με εικόνες από δύο κάμερες, όπως δύο μάτια. Αυτή η προσέγγιση απαιτούσε πρόσθετο εξοπλισμό και δεδομένα, για αυτό οι έρευνες έχουν στραφεί προς την εκτίμηση του βάθους με μία μόνο κάμερα. Στερείται στερεοσκοπικής όρασης, ωστόσο, και είναι, επομένως δύσκολο να εκτιμήσει με ακρίβεια το βάθος. Για να αντιμετωπίσουν αυτό το ζήτημα, οι επιστήμονες εφάρμοσαν διάφορες τεχνικές στα μοντέλα τους για να ανακτήσουν το βάθος. Η παρούσα διπλωματική εργασία συγκρίνει τα BANet, LapDepth και PixelFormer: τρία υπερσύγχρονα μοντέλα Βαθιάς Μάθησης με διαφορετικές αρχιτεκτονικές που εκτιμούν το βάθος από μία μόνο εικόνα. Στόχος είναι να εκπαιδευτούν από την αρχή σε ένα μεγάλο σύνολο δεδομένων και να γίνει σύγκριση των αποτελεσμάτων για τον καθορισμό του νικητή. Για αντικειμενική αξιολόγηση χρησιμοποιήθηκαν δύο διαφορετικά κριτήρια: SILog loss και RMSE. Επιπλέον, δύο ακόμη σύνολα δεδομένων κάνουν testing μέσω υποκειμενικής αξιολόγησης. Τα αποτελέσματα δείχνουν ότι το PixelFormer, με την πολύπλοκη αρχιτεκτονική του, εκτιμά το βάθος με μεγαλύτερη ακρίβεια μεταξύ των τριών μοντέλων.

Λέξεις Κλειδιά: BANet, DIODE, όραση υπολογιστή, συνελικτικά νευρωνικά δίκτυα, βαθιά μάθηση, KITTI, LapDepth, μηχανική μάθηση, εκτίμηση απόστασης με μία κάμερα, PixelFormer, Python

Acknowledgments

At this point, I would like to express my deepest gratitude to my supervisor, professor Konstantinos Diamantaras, for his unwavering guidance through each stage of this project and for his continuous support and encouragement. The achieved result could not have been possible without his invaluable assistance and coordination.

Next, I would like to thank the members of “Draive” for granting me access to and helping me with Titan server to perform the necessary experiments. Dionysis Miaris from “Draive” further guided me through the project with his knowledge of Machine Learning, and I thank him for his valuable feedback.

Last but not least, a big thank you to my family and friends for their unparalleled support and encouragement throughout the duration of my thesis project study and work.

Table of Contents

List of Figures

List of Tables

Abbreviations

1D	One-dimensional
2D	Two-dimensional
3D	Three-dimensional
AI	Artificial Intelligence
AR	Augmented Reality
BDE	Binocular Depth Estimation
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
DE	Depth Estimation
DIODE	Dense Indoor/Outdoor Depth Estimation
GB	Gigabyte
GPU	Graphics Processing Unit
IDE	Integrated Development Environment
IHU	International Hellenic University
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	Laser Imaging Detection and Ranging
MB	Megabyte
MDE	Monocular Depth Estimation
NLP	Natural Language Processing
ReLU	Rectified Linear Unit
RGB	Red-Green-Blue
VR	Virtual Reality

1

Introduction

1.1 Depth Estimation

Computer applications, such as autonomous driving, Augmented Reality (AR), and robotics, are becoming an essential part of modern society. They reduce traffic accidents, integrate digital objects into the physical world, or work in hazardous environments. To perform these tasks, they require three-dimensional (3D) maps. In autonomous driving, they warn the vehicle of other vehicles, pedestrians, and obstacles. In AR, they act as a guide for placing virtual objects relative to real ones in the 3D world. Lastly, in robotics, they help the robot understand its environment so it can navigate around and interact with objects. Other fields include Virtual Reality (VR), photography, medical imaging, and geospatial mapping [1].

One approach acquires depth data directly from the environment with special equipment. An example is Laser Imaging Detection and Ranging (LiDAR) sensors, such as the one in Figure 1.1. These devices fire laser pulses and measure the time it takes for them to return [2]. Although precise, they are expensive and produce maps that are missing some data. Moreover, since they operate with rotating mechanisms to scan their surroundings, they may not be properly aligned, or the surface texture may not reflect their laser [3].

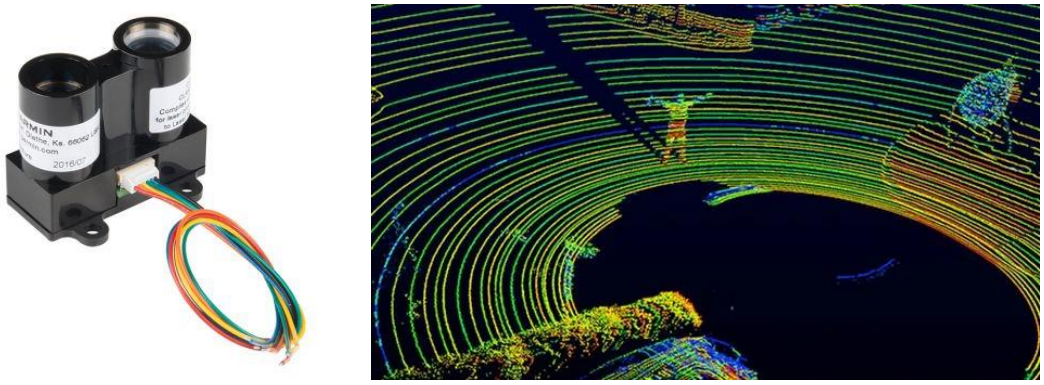

Figure 1.1: LiDAR sensor function [4] [2]

On the contrary, regular cameras obtain images or videos much more easily. With the thrive of Machine Learning in recent years, research is geared toward estimating depth maps from Red-Green-Blue (RGB) images [5]. This method, called Depth Estimation (DE), feeds an RGB image to a Machine Learning model and calculates the distance of each pixel from the camera, giving a depth map as output [6]. Figure 1.3 shows an RGB image on the left and its depth map on the right, rendered with the jet colormap.

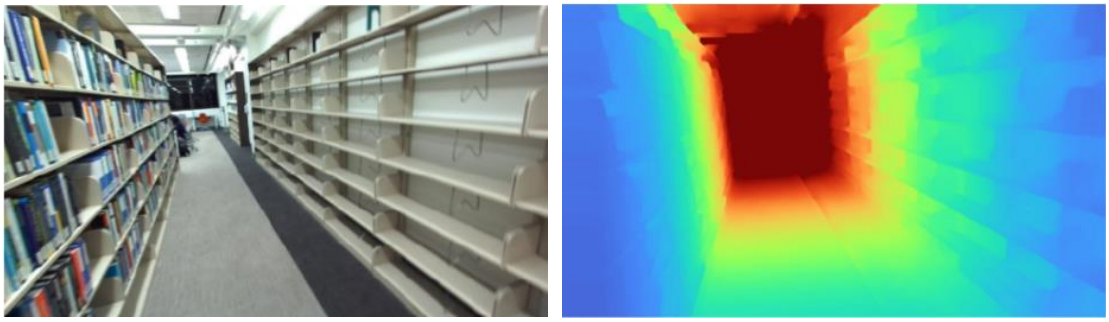

Figure 1.2: An RGB image and depth map [7]

Colormaps visualize the results of Machine Learning models in an aesthetically pleasing way. In the example of Figure 1.2, shades of blue indicate that a pixel is closer to the camera, while shades of red indicate that it is far away. Colormaps greatly affect the representation of data, so choosing the most appropriate one is crucial. Some present features in a meaningful way, while others obscure data and lead to false assumptions. Apart from jet, Figure 1.3 features other popular colormaps for depth maps: inferno (black, purple, yellow shades), viridis (dark blue, green, yellow shades), and gray (black, gray, white shades) [8].

Figure 1.3: Colormaps for DE [8]

In humans, the brain compares the differences in the image it receives from the left eye with that of the right eye to estimate depth. These two images are called a stereo pair [2]. In addition, humans use seven cues. The first, occlusion, occurs when one object is partially hidden behind another, meaning it is farther away. Humans also use perspective when they see parallel lines that meet in the distance. Size and texture gradient play a major role too. An object's size changes depending on the distance, and its texture becomes denser as it increases. Another cue, the atmospheric cue, refers to the object getting blurry as it moves away. In addition, human vision considers an object's shadow or the shadow it casts on other objects. The last cue, height, means objects far from the camera appear closer to the horizon.

Nevertheless, human perception is flawed to a degree, as it is sometimes difficult to determine the exact distance. The key factor lies in prior knowledge. The visual system searches among an infinite number of geometrically possible 3D scenes for the one that makes the most sense in the real world. For a computer system that lacks these functions, this task is even more challenging, as different, unique 3D scenes can generate the same two-dimensional (2D) image [9]. As an example, Figure 1.4 shows two cubes of different sizes that produce the same 2D image.

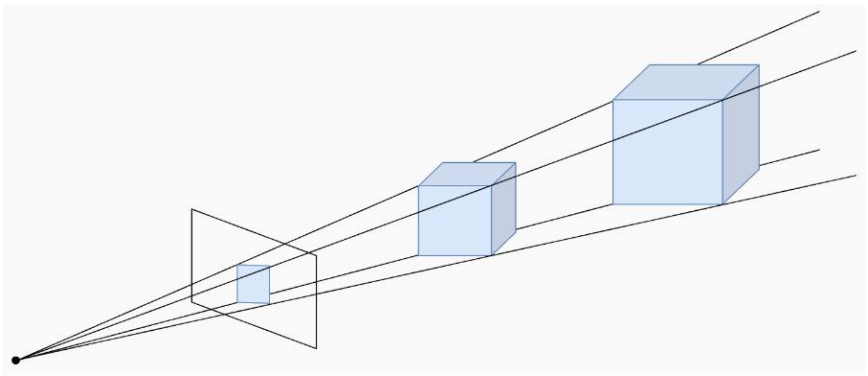

Figure 1.4: Different 3D scenes produce the same 2D image [10]

Early Machine Learning models emulated the function of two eyes, using stereo image pairs and human-like cues to estimate depth. Figure 1.5 features an example of a stereo pair from two cameras, one on the left and one on the right, and their combined result at the bottom. In this image, the red hue represents the part captured by the left camera, while the cyan represents the right. Binocular Depth Estimation (BDE) models take two images from slightly different

viewpoints as input and use triangulation to calculate each pixel's depth. The diagram on the right side of Figure 1.5 presents the concept of triangulation, where I(L) and I(R) are a stereo pair from the left and right cameras respectively. This approach, however, requires a lot of resources and data [3] [11].

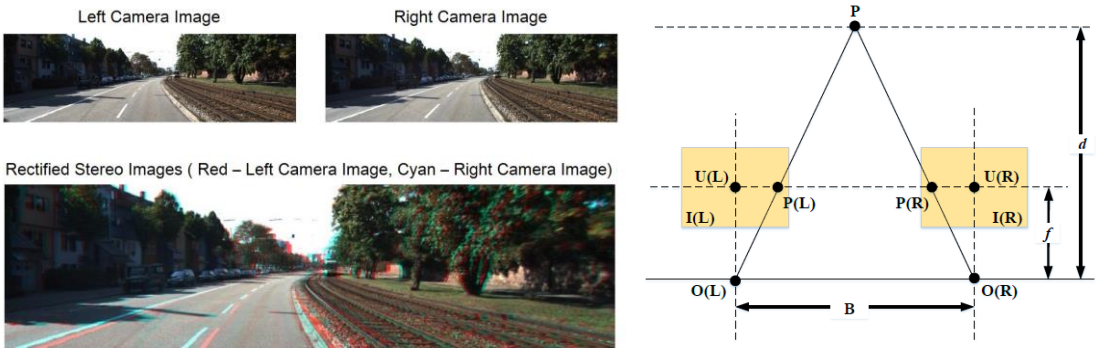

Figure 1.5: A stereo image pair [6] [12]

Monocular Depth Estimation (MDE) uses images or video sequences from only one camera. Its simplicity and low cost compared to BDE, which requires additional equipment, have increased the demand for such models. Regular images lack the stereoscopic vision advantage of stereo images. The dimension of depth is lost in the process of depicting the 3D world in two dimensions. MDE models attempt to recover it to construct the depth map. Figure 1.6 demonstrates their basic architecture, where the model receives a single image as input and produces a depth map [6].

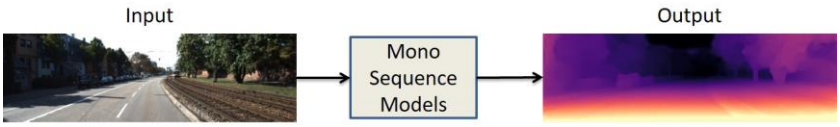

Figure 1.6: Basic MDE model architecture [6]

Table 1.1: Depth map generation methods

Method	Equipment	Advantages	Disadvantages
LiDAR	LiDAR sensor	Highly accurate	Costly, produces sparse maps
BDE	Two cameras	Stereoscopic vision	Costly, requires a lot of resources
MDE	One camera	Simple, low cost	Dimension of depth is lost

Table 1.1 provides an overview of the three methods discussed: LiDAR, BDE, and MDE. For this thesis, the MDE approach was chosen. MDE models are simpler to install and run and do not require special equipment. Thus, it is easy to create a mini dataset to test their performance in various real-world scenarios, even with a simple smartphone camera.

To estimate depth, models first train on a large dataset. They usually consist of either outdoor or indoor RGB images and their corresponding ground truth depth maps. In Machine Learning, ground truth refers to the solution to a specific problem that the model aims to achieve.

Outdoor environments are commonly found in autonomous driving, robotics, 3D architecture, and AR/VR. A typical outdoor image contains objects close to the camera up to a kilometer or more away. Sensors, such as LiDAR, obtain depth information for the training data, which as discussed, are expensive and produce sparse maps. Furthermore, LiDAR sensors have a limited range of about 100 meters. This means the models also predict depth up to 100 meters [13] [14]. Figure 1.7 shows an RGB image and depth map from the KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute) dataset, popular for its exterior scenes captured from a moving vehicle. Missing information appears as blue gaps in the depth map. However, this does not mean they are close to the camera. Similarly, the blue segment at the top, mostly consisting of the sky, is not close either, rather it contains no depth information.

Figure 1.7: Outdoor image and depth map [15]

Applications for indoor datasets include robotics, AR/VR, and 3D photography. These datasets cover various parts of a house (kitchen, living room, hallways, bathrooms, etc.) and other interior spaces (offices, meeting rooms, classrooms, libraries, etc.). Figure 1.8 shows an image and its depth map from NYU-Depth V2, a popular indoor dataset. This diversity in environments poses challenges in DE. Items associated with one room, for example, kitchen appliances, might not appear in other areas, like bathrooms or classrooms. Textures also vary from room to room. Thus, models trained in a specific indoor setting might perform poorly on another. Additionally, the equipment used to generate the training data struggles to capture complexities and small objects, such as utensils [7].

Figure 1.8: Indoor image and depth map [16]

1.2 Objective and Contribution

The main challenge in MDE is to produce accurate depth maps from a single camera, which lacks the stereoscopic vision of BDE. Inaccurate depth maps can cause an inability to interact with the environment in AR and robotics and even lead to fatal accidents in the case of autonomous driving.

This thesis explores three state-of-the-art Deep Learning models for MDE: BANet, LapDepth, and PixelFormer. The main objective is to train them from scratch on a large dataset and compare the results to determine the winner. The code will be adjusted to load the dataset into the models and extract statistics and depth maps. Different values for the hyperparameters will be tested to find the combination that yields the best statistics for each model. Two small-scale datasets for testing the models will collect additional depth maps. One of them is a new dataset, created specifically for this thesis. The results will be compared both objectively and subjectively. More specifically, the SILog loss and RMSE criteria will ensure an objective evaluation, whereas the visual quality of the predicted depth maps will provide a subjective evaluation.

The model with the best performance, based on objective and subjective evaluation, will be declared the winner. It will also be delivered to “Draive”, International Hellenic University’s (IHU) Machine Learning team, which specializes in autonomous driving, to be used in future projects. The model will assist an autonomous vehicle in making better decisions, such as which direction to steer to avoid obstacles on the road. Figure 1.9 displays Draive’s logo.

Figure 1.9: Draive logo [17]

In other words, the thesis's contribution is summarized as follows:

1. Train BANet, LapDepth, and PixelFormer on a large dataset.
2. Test different values for the hyperparameters to find the best combination.
3. Perform depth estimation on two more datasets to collect additional depth maps.
4. Compare the results objectively through two criteria: SILog loss and RMSE.
5. Compare the results subjectively through the visual quality of the predicted depth maps.
6. Determine the winning model based on objective and subjective evaluation.

1.3 Chapter Organization

Chapter 2 analyzes various MDE methods and techniques as they evolved in three distinct periods: the Early Period, the Machine Learning Period, and the Deep Learning Period. The last period, in particular, discusses BANet, LapDepth, and PixelFormer among other models.

Chapter 3 explains the structure of Deep Learning models, a branch of Machine Learning, and two popular Deep Learning architectures: DenseNet and ResNeXt. Moreover, it demonstrates the process of training a model and two problems that can arise: underfitting and overfitting. Lastly, this chapter provides the evaluation metrics that measure a DE model's performance.

Chapter 4, divided into two subsections, details the tools needed to train and evaluate BANet, LapDepth, and PixelFormer. More specifically, the first subsection focuses on the Python programming language, libraries, frameworks, and Integrated Development Environments (IDE). The second one introduces the remote server "Titan", which will perform the experiments, as well as various applications for accessing and working on it.

Chapter 5 reports on the contribution of the current thesis. It starts by analyzing all datasets for the upcoming experiments. The rest of the chapter focuses on BANet, LapDepth, and PixelFormer. For each model, it investigates its architecture in more detail, explains the training setup, and compares the results to determine the winner.

Chapter 6, the final chapter of the thesis, discusses the conclusions and possible plans for future work.

2

Related Work

2.1 Introduction

This chapter presents the bibliographic retrospective of relevant research in MDE. It starts from the Early Period, progresses to the Machine Learning Period, and finally, to the Deep Learning Period. Each one explores various techniques and methodologies of the time. The Deep Learning Period, in particular, features state-of-the-art models, including BANet, LapDepth, and PixelFormer. Figure 2.1 provides the highlights of the three periods examined in the following subsections.

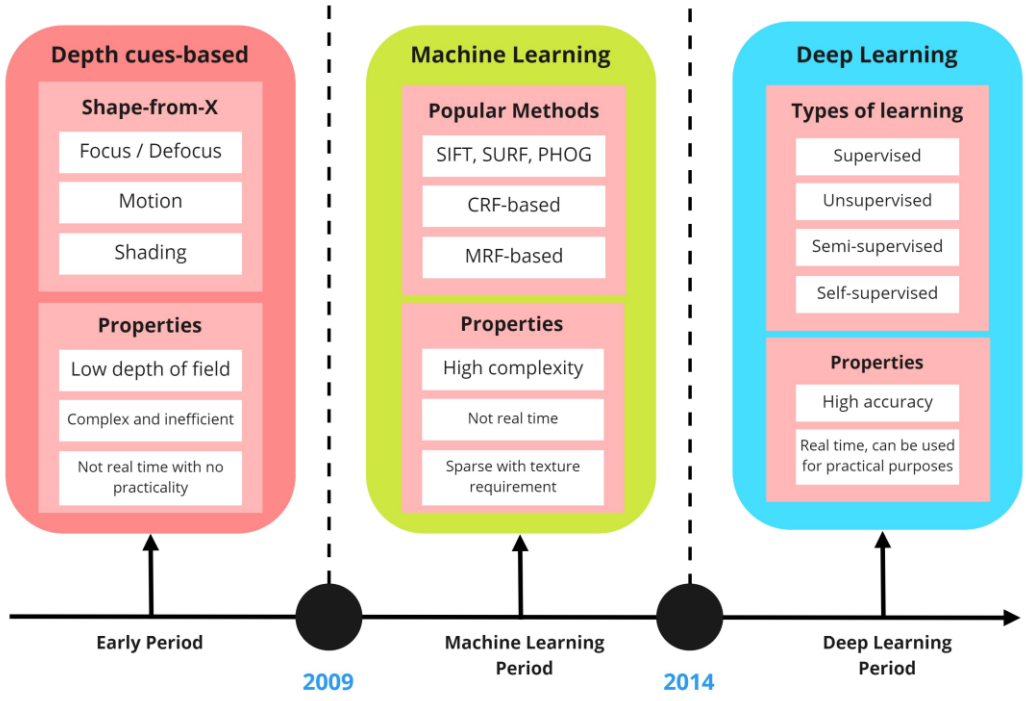

Figure 2.1: DE evolution [13]

2.2 Early Period (Before 2009)

One of the earliest systems, back in 2006, divided an image into small patches and assigned a depth value to each. For this task, the researchers incorporated two types of features: absolute and relative. Absolute features estimated the absolute depth of a patch. To achieve this, they examined the image resolution and object size. They also considered the features of the four immediate neighbors of a specific patch to calculate its depth. On the other hand, relative features estimated relative depths. These features studied the depth difference between two neighboring patches to discover how they were related. Apart from the immediate neighbors, their system compared patches that were not immediate neighbors to detect similarities. The walls of a building, for example, had a correlating depth. Figure 2.2 illustrates an example of their method. The input image is on the left, followed by the ground truth, and the prediction of two different approaches. The first one used the Gaussian function to model the depth distribution, and the second one, which performed better, used the Laplacian function. Moreover, since the ground truth maps were captured with a 3D laser scan with a range of 81 meters, the system's estimation capabilities were limited to this range. Objects further than this distance received the maximum depth value. Nevertheless, their method yielded good results in a variety of indoor and outdoor settings [18].

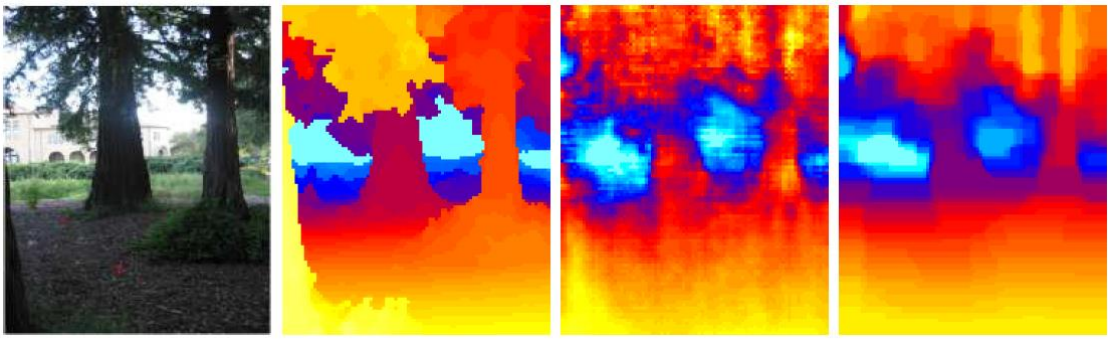

Figure 2.2: MDE using the Gaussian and Laplacian distribution [18]

Later, in 2008, a group of scientists converted 2D indoor images to 3D for 3D TV entertainment, such as movies and gaming. Their method was fairly simple yet effectively represented a variety of settings. First, the algorithm extracted the foreground. To extract the background, they detected wall borders and corners with a corner detector. The algorithm then split the background into horizontal (floor and ceiling) and vertical surfaces (walls, doors). Finally, it assigned a depth value to each pixel depending on its geometric features. Consequently, objects in the foreground had lower depth values than the background. From the generated depth map, they collected stereo image pairs, which they imported into an image manipulation program to

obtain a 3D image. The result could be viewed with 3D glasses [19]. Figure 2.3 shows an example of an input image on the right, the depth map in the middle (black for “far”, white for “close”), and the 3D image on the right.

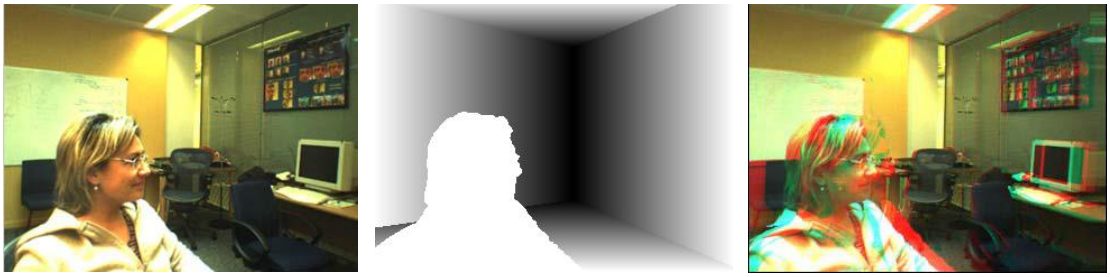

Figure 2.3: MDE for converting a 2D image to 3D [19]

2.3 Machine Learning Period (2009 – 2014)

One of the works of this period, in 2011, proposed an algorithm that divided an outdoor image into homogenous segments to extract the sky segment and the ground segment. The sky was easier for the system to locate. It always appeared at the top of an image, stretched across a large area, and had a distinctive color (blue or white in the morning, yellow or orange in the evening, black at night). In addition, since it was far away, the entire sky segment took the maximum depth value. The ground segment was more difficult to identify because of its varying colors: black roads, green grass and trees, blue seas and rivers. However, since the ground was always at the bottom of an image, this was a useful indicator. This segment did not have a fixed depth value, but it gradually increased as it approached the sky. The in-between segments took a depth value between the sky and the ground. Figure 2.4 shows the results of their algorithm on a selected input image on the left. The middle one displays the image divided into segments with the sky segment at the top and the ground segment at the bottom. The last image presents the estimated depth map after the segmentation process. Here, the pink color translates to “far”, which is the sky, and the red shades with gradually increasing depth translate to “close”, which is the ground. The rest of the shades belong to the in-between segments. Despite the aesthetically pleasing depth maps, the algorithm could not be evaluated due to the lack of ground truth data [20].

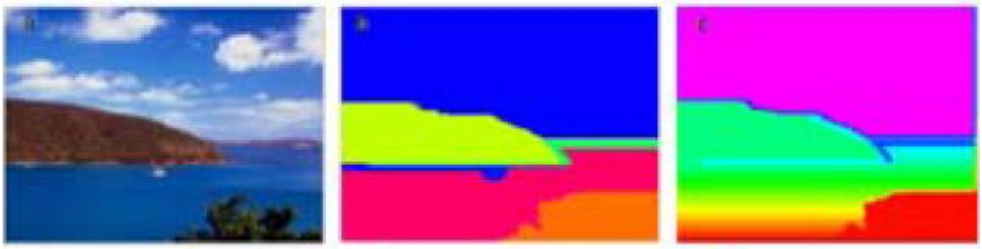

Figure 2.4: MDE based on image segmentation [20]

Another work, in 2013, proposed a framework that separated an RGB image into overlapping 16x16 patches and extracted features for each patch based on the brightness and coordinates of a pixel. A brighter pixel meant the patch was closer to the camera. Additionally, the researchers assumed that pixels closer to the top of an image generally have higher depth values than those closer to the bottom. The framework then applied filters and distribution functions with location, scale, and shape as parameters to assign a depth value to each patch. Finally, to form the predicted depth map, it glued all the patches together by averaging the overlapping pixels between two patches [21]. Figure 2.5 visualizes the results, where the input image is on the left, the ground truth in the middle, and the prediction on the right (red for "far", blue for "close"). The framework performed quite well and discovered small details, such as the pole structure in the Figure.

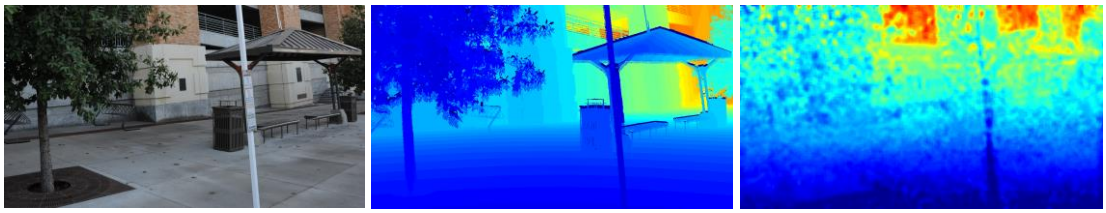

Figure 2.5: MDE based on extracting features from overlapping patches [21]

2.4 Deep Learning Period (2014 – Present)

This period introduced an innovative methodology for MDE: predicting depth maps from a single image through a Deep Learning network. Eigen et al. were the first ones to propose this method in 2014 [12]. Since then, many more networks have followed the same principle. Some of the newer ones include DenseDepth in 2019, BANet and LapDepth in 2021, and PixelFormer in 2022.

New, popular indoor and outdoor datasets were created for MDE during this period: DIODE, NYU-Depth V2, and KITTI. DIODE (Dense Indoor/Outdoor Depth Estimation) consists of two

datasets: one from indoor and one from outdoor environments. The former contains a total of 9,000 indoor images of 1024x768 resolution with a range of 50 meters, while the latter contains 17,000 outdoor images of the same resolution with a range of 350 meters. NYU-Depth V2 (or simply NYUv2) focuses on indoor environments. It contains 120,000 images of 640x480 resolution with a range of 10 meters. KITTI is the most widely used dataset for driving scenarios. It contains 90,000 images of 1241x375 resolution with a range of 120 meters [6] [22] [23]. Figure 2.6 features an image and a depth map from each dataset.

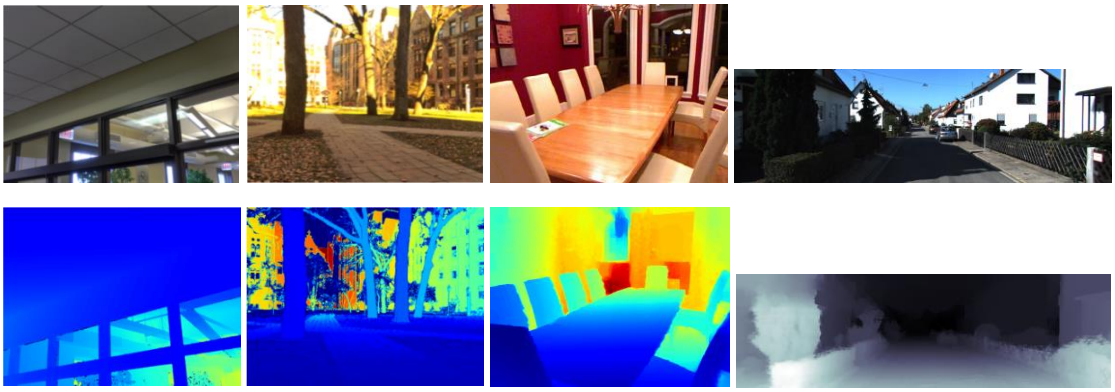

Figure 2.6: Popular datasets for MDE [16] [22]

Table 2.1: Popular datasets for MDE in the Deep Learning Period

Dataset	Scenario	Images	Resolution	Range
DIODE/Indoor	Indoor	9,000	1024x768	50m
DIODE/Outdoor	Outdoor	17,000	1024x768	350m
NYUv2	Indoor	120,000	640x480	10m
KITTI	Outdoor	90,000	1241x375	120m

Table 2.1 summarizes key information for each dataset. For comparison purposes, this subsection only includes visual results from KITTI, which was common to all five models.

Scientists of this period invented a new solution to improve performance and accuracy in the training stage. This solution, called image augmentation, increases the dataset with new images by applying various techniques to the existing ones, such as random crop, random rotation, horizontal flip, and color jitter. Random crop randomly selects a part of an image and cuts out the rest. Random rotation rotates the image randomly. Horizontal flip flips the image horizontally. Color jitter modifies the brightness, contrast, saturation, and hue of an image. Figure 2.7 showcases the four mentioned techniques: random crop, random rotate, horizontal

flip, and color jitter with the original image on the left. Image augmentation can combine two or more of these techniques simultaneously, for example, random rotate with color jitter, for more complex results [24].

Original Random crop Random rotation Horizontal flip Color jitter

Figure 2.7: Image augmentation techniques [24]

2.4.1 Eigen et al. (2014) [25]

In 2014, scientists presented a new method for estimating depth from a single image: a Deep Learning network. This network consisted of two components: a coarse and a fine network.

The coarse network examined the image from a global perspective to construct a coarse depth map. To understand a scene, it extracted important features with cues, such as the location of an object, the arrangement of a setting, and parallel lines meeting in the distance. Due to image transformations, the coarse network produced a depth map with a small area clipped from each edge.

The fine network looked at the image from a local perspective with a range of 45x45 pixels at a time and spotted details, such as object and wall edges. The output of the coarse network was fed into the fine network to refine it with these local features and produce the final depth map.

The scientists also applied various image augmentation techniques to the training set: random rotation, random crop, color jitter, and horizontal flip.

Their model was trained on both the indoor and outdoor environments of the NYU-Depth V2 and KITTI datasets respectively. It required 2.6 days to complete training on NYU-Depth V2 and 1.8 days on KITTI with an NVIDIA GTX Titan Black GPU (Graphics Processing Unit). This GPU was first released on February 18th, 2014, and had a memory size of 6GB (gigabyte) [26].

Figure 2.8 shows an example of the experiments Eigen et al.'s model performed on KITTI, where the input image is at the top, the ground truth in the middle, and their prediction at the bottom (white for "far", black for "close"). The results, although blurry, demonstrated the potential of Deep Learning networks in MDE and improved performance.

Figure 2.8: MDE with the Eigen et al. model [25]

2.4.2 DenseDepth (2019) [27]

DenseDepth aimed to define clearer object boundaries in its depth maps than the blurry, low-resolution result of previous models. To achieve this, DenseDepth functioned with an encoder and a decoder and the help of transfer learning. This Deep Learning technique employed a pre-trained encoder in its architecture. DenseDepth's pre-trained model was originally designed for image classification: a Machine Learning problem where the model identifies the object depicted in an image [28].

DenseDepth's DE procedure was very simple. First, the image was fed to the pre-trained encoder to extract features. The output then entered the decoder, which constructed the final depth map. The scientists concluded that the success of their model despite the simplicity lied in the careful design of the encoder-decoder structure and the avoidance of complex components that do not improve performance.

They also carefully considered what image augmentation techniques to apply to the training set, selecting only two: horizontal flip and color jitter.

Their model was trained on the two challenging datasets, NYU-Depth V2 and KITTI, of indoor and outdoor environments respectively. Training with NYU-Depth V2 required 20 hours, while with KITTI 9 hours on four NVIDIA Titan Xp GPUs. This GPU first appeared on April 6th, 2017 and had a memory size of 12GB [29].

Figure 2.9 displays DenseDepth's results on KITTI with the input image on the left and the prediction on the right (black for "far", yellow for "close"). With its simple architecture and low requirements, DenseDepth produced higher quality depth maps compared to previous state-of-the-art models.

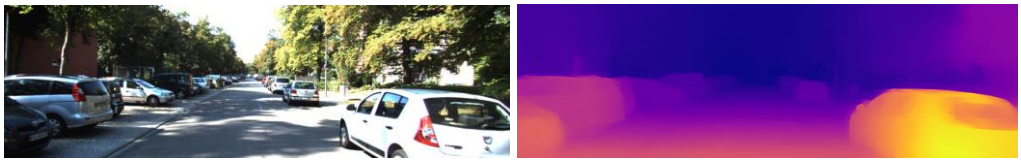

Figure 2.9: MDE with DenseDepth [27]

2.4.3 BANet (2021) [22]

The Bidirectional Attention Network (BANet), as its name suggests, is a network with two directions: forward and backward. Five stages extract features from the input image and feed them to the next stage. D2S (depth-to-space) modules at each stage process the features they receive in either a forward or backward manner. These features are then collected to generate the final depth map.

Each stage extracts different types of features. The first stage holds a general view of the image. The next focuses on the ground or the road. The third and fourth examine objects, such as cars and people, and calculate their distance from the camera. The final stage collects information about parallel lines that meet in the distance.

The only two image augmentation techniques the researchers applied to the training set were horizontal flip and color jitter.

The model was trained on DIODE's indoor and outdoor datasets, as well as KITTI's driving dataset with an NVIDIA Tesla V100 GPU. This GPU was released on March 27th, 2018 and had a memory size of 32GB [30].

BANet performed well against other models of its time. Figure 2.10 features their results on KITTI, where the input image appears at the top, the ground truth in the middle, and the prediction at the bottom (red for "far", blue for "close").

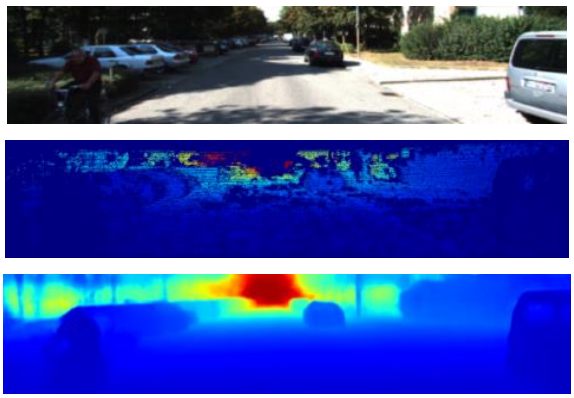

Figure 2.10: MDE with BANet [22]

2.4.4 LapDepth (2021) [31]

LapDepth functions with an encoder and a decoder. First, the input image passes through multiple blocks in the encoder and shrinks to 1/16 of its original size. The model then learns color-depth correlations from the features it collects, using geometry.

The decoder side contains a Laplacian pyramid. Laplacian pyramids decompose images at multiple low-resolution scales [32]. This implementation helps LapDepth understand the structure of a scene by storing local details, such as general depth information and object boundaries at each level of the pyramid. The results are progressively combined through the levels to form the final depth map.

LapDepth's Laplacian pyramid has five levels. The scientists conducted experiments with a six-level and a seven-level pyramid as well. The performance improved slightly at the expense of GPU memory, so they decided to keep the five-level pyramid as the primary architecture.

They also applied various image augmentation techniques to the training set: random crop, random rotation, horizontal flip, and color jitter.

LapDepth was trained on KITTI and NYU-Depth V2 for their vast outdoor and indoor environments respectively. It required 16 hours to complete training with four NVIDIA Titan Xp GPUs. This GPU first appeared on April 6th, 2017 and had a memory size of 12GB [29].

Figure 2.11 shows LapDepth's results on KITTI, where the input image is at the top, the ground truth in the middle, and the prediction at the bottom (black for "far", white for "close"). Compared to previous state-of-the-art models, in which object boundaries appeared blurry, LapDepth defined sharper edges, even around thin pillars and traffic signs.

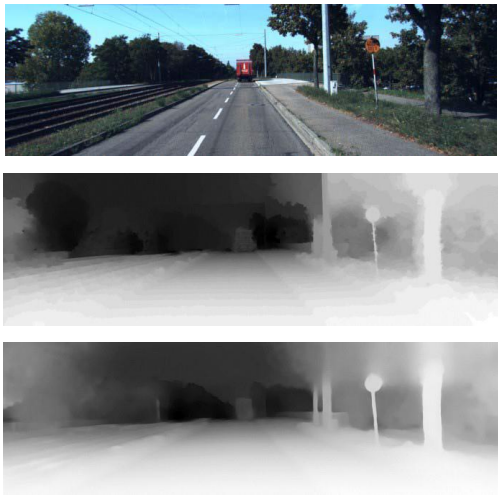

Figure 2.11: MDE with LapDepth [31]

2.4.5 PixelFormer (2022) [23]

PixelFormer uses a transformer in its architecture. A transformer receives a sequence of elements as input and, as the name suggests, transforms it into another sequence. In Natural Language Processing (NLP) applications, for example, the sequence is a sentence or a document. The transformer receives the text and discovers relationships between different elements (or words in this example) to translate it into another language [33].

Apart from NLP, transformers are also found in computer vision. PixelFormer, in particular, uses SwinTransformer, created in 2021 by a different group of scientists. Instead of words, this transformer works with pixels to extract features. More specifically, the image first enters the SwinTransformer in PixelFormer to extract features at multiple scales: 1/4, 1/8, 1/16, and 1/32. These features are very coarse with only general information about the scene.

The Pixel Query Initializer (PQI) module, then, accumulates the global scene information from the features at scale 1/32. This output passes through the Bin Center Predictor (BCP) that discretizes depth into intervals. A third module, the Skip Attention Module (SAM), also receives the output of the PQI module along with the coarse features to enhance resolution with finer details. Lastly, PixelFormer processes the output of SAM and BCP to construct the final depth map.

The scientists applied various image augmentation techniques, including random crop, horizontal flip, and color jitter.

PixelFormer was trained on the popular NYU-Depth V2 and KITTI datasets for 10 hours with four NVIDIA A100 GPUs. This GPU, first released on May 14th, 2020, had a memory size of 40GB [34].

Figure 2.12 presents PixelFormer's results on KITTI with the input image on the left and the resulting depth map on the right (yellow for "far", dark blue for "close"). Next to the depth map is a zoomed-in area defined by the white rectangle, where PixelFormer captured small details at a great distance. The model displayed significant improvement over other state-of-the-art models in both KITTI and NYU-Depth V2. At the time of its release, it ranked 1st on KITTI's official website.

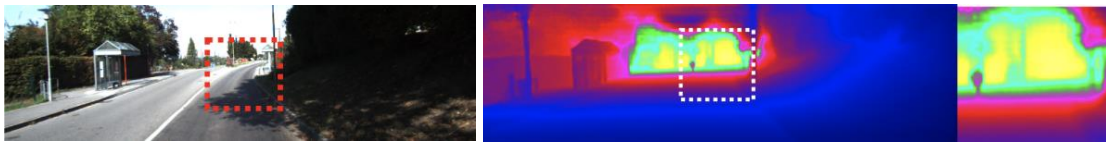

Figure 2.12: MDE with PixelFormer [23]

2.5 Conclusion

MDE is a problem that has concerned scientists over the years. Early Period systems were simple and employed techniques, such as image segmentation and neighboring patches. Although the resulting depth maps lacked detail, they were nonetheless informative and served the needs of their time.

The amount of detail captured in depth maps increased during the Machine Learning Period. In addition, systems of this period applied similar, simple techniques to the previous one (image segmentation, neighboring patches). The scientists trained them on a variety of datasets, some of which lacked ground truth depth maps, so it was impossible to evaluate or compare the results.

Table 2.2: State-of-the-art models of the Deep Learning Period

Year	Model	Architecture	Dataset	NVIDIA GPU	Duration
2014	Eigen et al.	Coarse to Fine	NYUv2, KITTI	(1) GTX Titan Black	1.8 - 2.6 days
2019	DenseDepth	Transfer Learning	NYUv2, KITTI	(4) Titan Xp	9 - 20 hours
2021	BANet	Bidirectional	DIODE, KITTI	(1) Tesla V100	N/A
2021	LapDepth	Laplacian Pyramid	NYUv2, KITTI	(4) Titan Xp	16 hours
2022	PixelFormer	Transformer	NYUv2, KITTI	(4) A100	10 hours

Table 2.2 provides highlights of the models explored in the Deep Learning Period subsection. Scientists trained them on popular datasets, such as NYU-Depth V2 for indoor environments and KITTI for outdoor spaces. This made it possible to visually compare the results, in which depth maps evolved from being blurry to capturing small details, such as thin pillars and traffic signs, even at a great distance.

Table 2.2 further provides information about each model's architecture. The methods were vast, ranging from simple (coarse to fine network, transfer learning) to more advanced and complex (bidirectional network, Laplacian pyramid-based, utilizing a transformer). Although different, they produced depth maps in a similar manner: they first detected coarse information and then added finer details. As their complexity increased, the number of GPUs also increased from one to four, but as a result, training duration dropped from a few days to a few hours depending on the GPU capabilities.

3

Theoretical Background

3.1 Introduction

Machine Learning, a field of Artificial Intelligence (AI), trains models that learn from data, similar to how the human brain works. Models learn gradually and become more accurate over time at making predictions. During training, they recognize patterns and evaluate their performance through useful metrics [35].

Deep Learning is a field of Machine Learning. It is inspired by the structure of the human brain, which contains millions of neurons that work together to learn from large amounts of data. Similarly, Deep Learning neurons operate interconnected within an artificial neural network, also known as a deep neural network. These networks consist of three components: an input layer, one or more hidden layers, and an output layer. As Figure 3.1 demonstrates, the first hidden layer receives input from the input layer, and the output layer receives input from the last hidden layer to produce the final output [36].

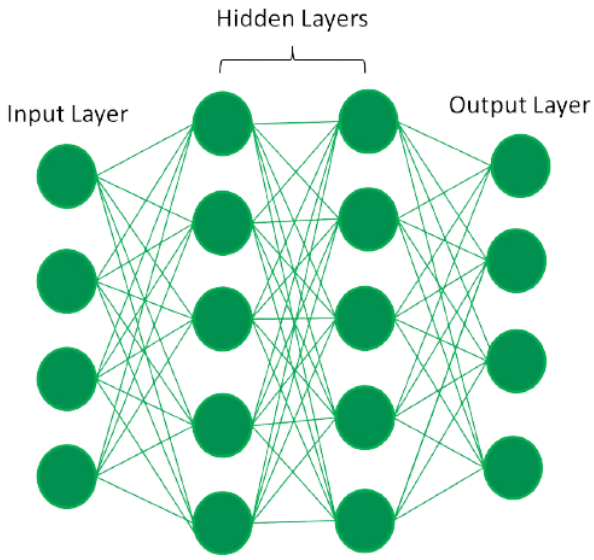

Figure 3.1: Basic structure of a neural network [36]

This chapter covers the theoretical background for understanding the structure and function of MDE models. First, it explores Convolutional Neural Networks (CNNs) and its key components. It also takes a look at well-known types of CNN that MDE models employ in their backbone. Next, it explains the process of training a CNN, including different types of datasets, parameters used, and the serious problems of underfitting and overfitting. Lastly, the chapter examines the evaluation metrics by which scientists compare the performance of DE models.

3.2 CNN architecture

A CNN is a type of Deep Learning network, suitable for computer vision tasks. It processes images through layers that extract features and, in the case of MDE, build depth maps. As shown in Figure 3.2, these layers can be convolution, pooling, and fully connected. A typical architecture includes stacks of convolution and pooling layers, one after the other, followed by fully connected layers, and the activation function at the end of the final fully connected layer. The first few layers collect general features, but as the image progresses deeper into the network, it detects more complex features [37].

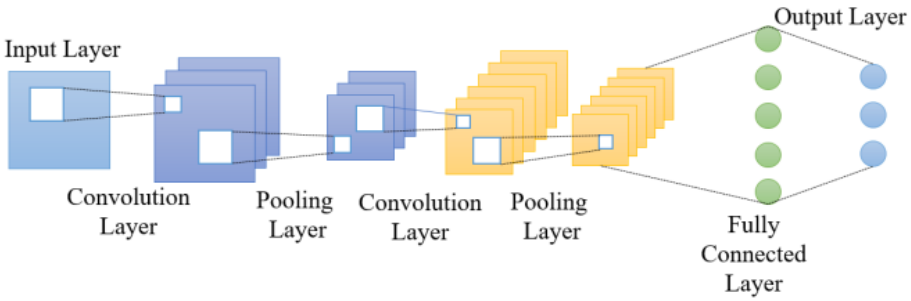

Figure 3.2: Basic CNN architecture [38]

3.2.1 Convolution Layer

This core component of a CNN extracts features from a 2D image. As shown in Figure 3.3, 2D images consist of an array of numbers that store each pixel's color value. In this example, the array stores values between 0 and 255, depending on the brightness.

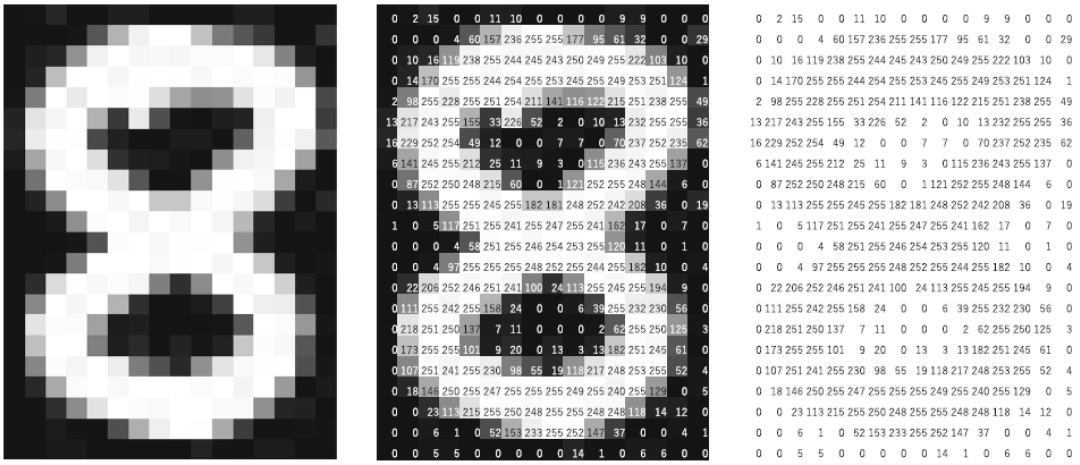

Figure 3.3: How the computer views an image [37]

In the operation of convolution, a small array of numbers, called a kernel, traverses the input image's array of numbers, called a tensor. Each number in the kernel is multiplied by the corresponding number in the tensor, element-wise. The resulting values are summed up into the corresponding position of a new array, called a feature map. The process continues by applying more kernels of various sizes (usually 3x3, 5x5, or 7x7) to the input tensor. Different kernels represent different features and thus generate different feature maps. Figure 3.4 demonstrates an example of a 3x3 kernel moving along a tensor with a stride of 1, the most common choice for convolution layers. The stride determines how far the kernel will move in the next step. In addition, Figure 3.5 illustrates how two different 3x3 feature maps filter the input image on the

right. The former, whose second row contains zeros, detects horizontal lines, while the latter, whose second column contains zeros, detects vertical lines [37].

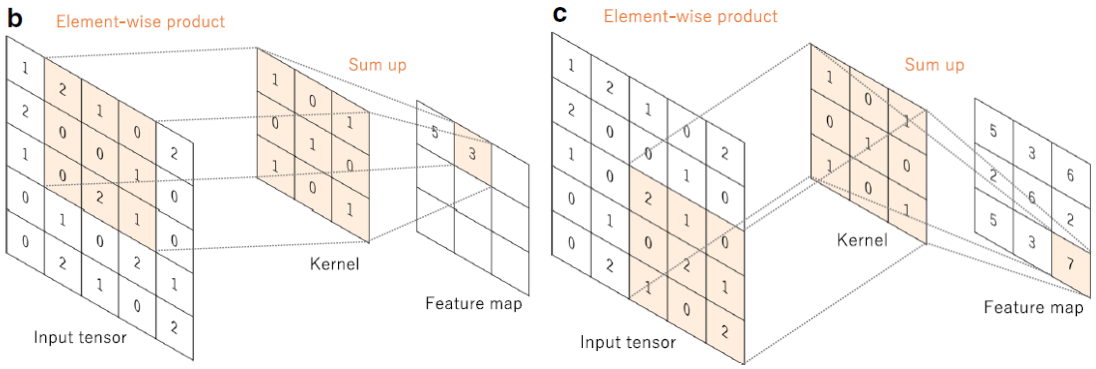

Figure 3.4: Convolution operation example [37]

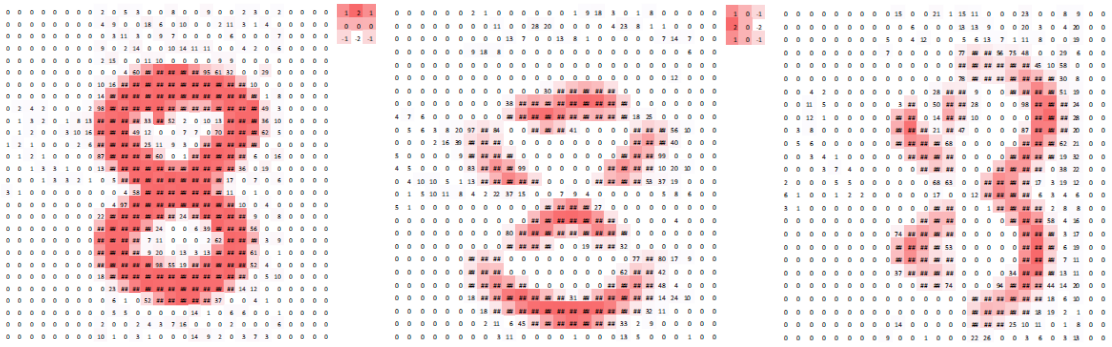

Figure 3.5: Example of feature extraction [37]

3.2.2 Pooling Layer

Convolution layers specify the exact location of a feature in an image. Slight variations of the same image due to image augmentation (for example random crop, random rotate) would generate different feature maps for the same feature and potentially lead to incorrect output.

Pooling layers receive the output of convolutional layers and reduce its dimensions to retain only important information while discarding redundant details. In other words, they generalize the feature maps so that the network can detect a particular feature anywhere in the image regardless of shifts and distortions. As a result, these types of layers do not learn new features.

Pooling layers work similarly to convolution layers, where a small, usually 2x2 window, called a filter, traverses the feature map with a set stride, usually 2. This way, it defines the areas for the pooling process. Based on the pooling method, it calculates a new value for each area, reducing the dimensions of the feature map by 2 in the case of the 2x2 filter.

The three most popular pooling methods are max pooling, average pooling, and global pooling. Max pooling selects the maximum value of each area. It is best suited for dark images, as it selects the brightest pixels. On the other hand, average pooling calculates the average value of each area. It is best suited for smoothing out rough edges. Figure 3.6 demonstrates these two methods in an example with a 4x4 feature map and a 2x2 filter. Max pooling selects the maximum value of each of the four areas, while average pooling calculates the average value. Global pooling works together with either max or average pooling. Global max pooling selects the maximum value of all feature maps and returns it as a single element, while global average pooling calculates and returns the average value of all feature maps [39].

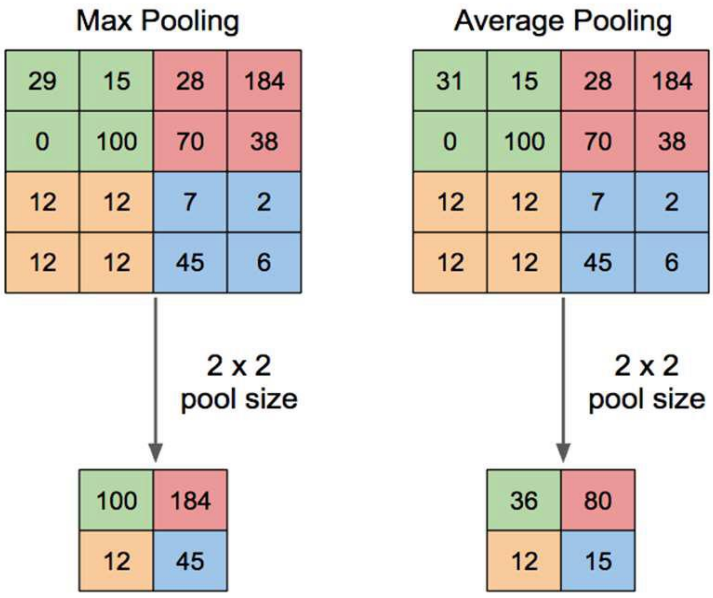

Figure 3.6: Max pooling and average pooling [40]

3.2.3 Fully Connected Layer

Typically, after the final convolution or pooling layer, the feature maps are flattened. In other words, they are converted into a one-dimensional (1D) array, called a vector. One or more fully connected layers then receive the vector. These layers are usually found at the end of a CNN with the task of connecting every input to every output to form the final result [37]. Figure 3.7 visualizes how fully connected layers pass data forward, connecting each input to each output until the final fully connected layer, which forms the result.

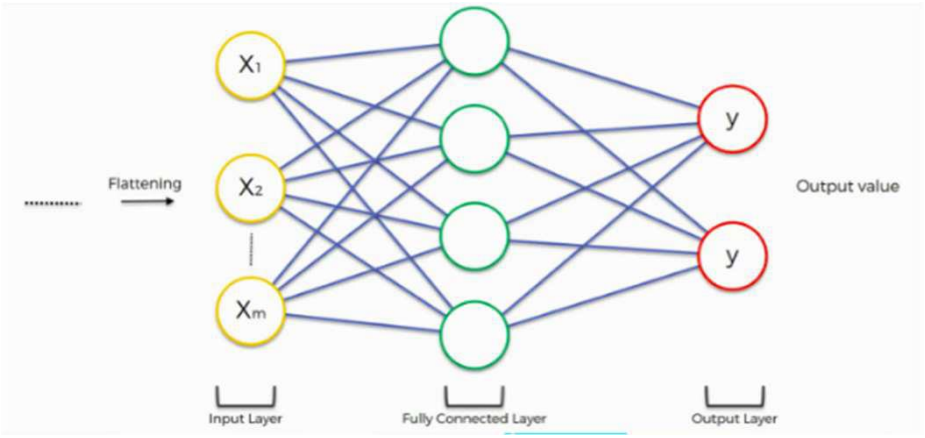

Figure 3.7: Fully connected layers example [40]

3.2.4 Activation Function

Since a CNN works similarly to the human brain, the activation function decides whether a neuron will be activated or not. If activated, the output of that particular neuron is passed as input to the next neuron. Otherwise, the neuron is not activated, and its output is not passed to the next one. This way, the network only learns useful information.

The activation function is a non-linear function due to similarities in the structure of the human brain and to limit the neuron's output to a certain range to avoid computational problems. Figure 3.8 displays two popular choices for the activation function: Rectified Linear Unit (ReLU) on the left and softmax on the right. ReLU produces 0 as output for all negative values, while softmax produces outputs in the range of 0-1 [41].

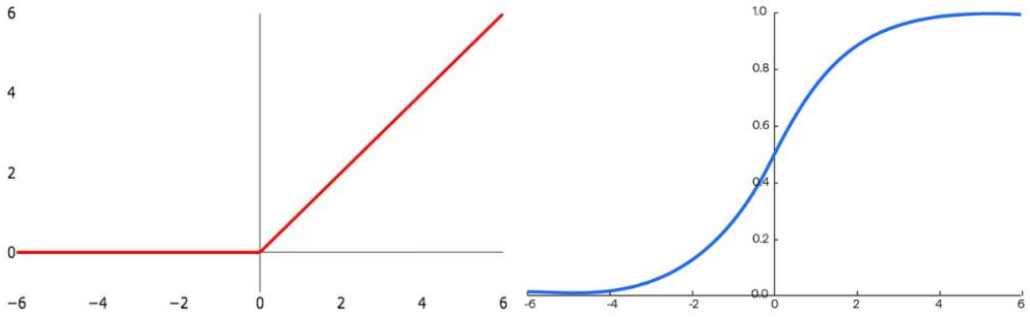

Figure 3.8: ReLU and softmax activation functions [41] [42]

3.3 Types of CNN

MDE models usually employ specific types of CNN in their backbone. Two of them are DenseNet and ResNeXt. These CNNs, which help MDE models learn new features, were originally trained on ImageNet, a large image database, for object identification [22] [31].

3.3.1 DenseNet

The Dense Convolutional Network (DenseNet) was proposed in 2016. It uses short connections between layers to make deep learning networks even deeper and more effective in training. In this CNN, every layer is connected to all deeper layers. This means the 1st layer is connected to the 2nd, 3rd, 4th, and so on, the 2nd is connected to the 3rd, 4th, 5th, and so on. In addition, each layer receives the input of all preceding layers and delivers its feature maps to subsequent layers. This way, the maximum amount of information flows in the network.

DenseNet's core components are dense blocks and transition layers. Dense blocks contain two convolution layers, one with a 1x1 kernel, and one with a 3x3 kernel, which are repeated multiple times in a loop. Transition layers include a convolution layer with a 1x1 kernel and an average pooling layer with a 2x2 filter.

In DenseNet, the image first passes through a convolution layer with a 7x7 kernel and a stride of 2, then a max pooling layer with a 3x3 filter and a stride of 2. A series of dense blocks and transition layers follow in the same order until the final dense block. At the end of the network is a global average pooling layer and a fully connected layer with the softmax activation function. Figure 3.9 displays the above architecture. DenseNet variants include DenseNet-121 with 121 layers, DenseNet-169 with 169 layers, DenseNet-201 with 201 layers, and DenseNet-264 with 264 layers [43].

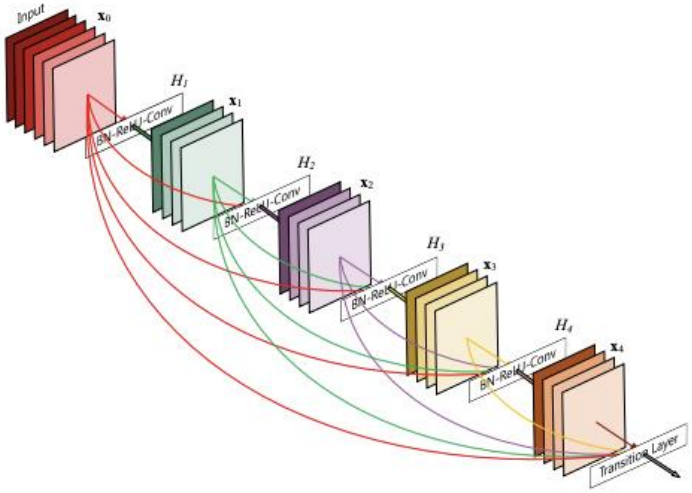

Figure 3.9: DenseNet architecture [43]

3.3.2 ResNeXt

This architecture was proposed in 2017 to reduce the complexity of Deep Learning networks. To achieve this, it introduced a new dimension called "cardinality", which refers to the number of transformations. As shown in Figure 3.10, ResNeXt performs 32 transformations. Thus, the cardinality is also 32.

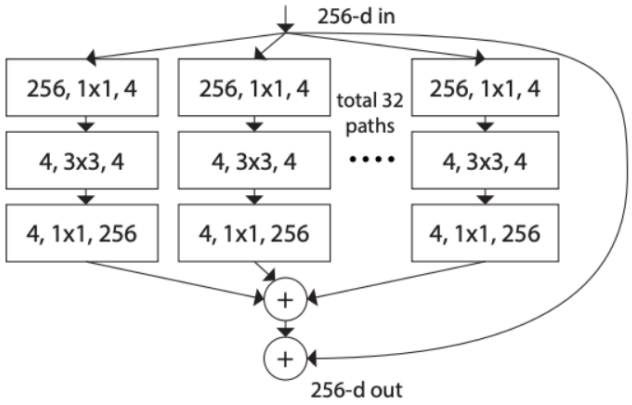

Figure 3.10: ResNeXt architecture [43]

ResNeXt's basic component is a cardinality block. It transforms the image through three convolution layers: one with a 1x1 kernel, one with a 3x3 kernel, and one with a 1x1 kernel.

In this network, the image first passes through a convolution layer with a 7x7 kernel and a stride of 2, then a max pooling layer with a 3x3 filter and a stride of 2. A series of cardinality blocks follow before ResNeXt applies global average pooling. At the end of the network is a fully

connected layer with the softmax activation function. ResNeXt variants include ResNeXt-50 with 50 layers and ResNeXt-101 with 101 layers [43].

3.4 Training a CNN

CNNs train on a large set of images, called the training set. To evaluate their performance, they require two additional datasets: the validation set, and the test set. Typically, MDE models train with RGB images and ground truth depth maps. Hyperparameters also need to be set before the process begins. The most important ones are number of epochs, batch size, learning rate, momentum, and weight decay. However, serious overfitting and underfitting problems can affect model performance.

3.4.1 Dataset Splits

The training set is the data from which the model trains and learns features to decide or predict an outcome. The more diverse the images, the more accurately the model will predict every possible future scenario. Usually, this dataset contains about 80% of the total data and is the largest of the three. The training data is repeatedly fed to the model, which continues to learn features and improve over time.

The validation set is a separate dataset with different images. Each time the training data is re-fed to the model, the validation set evaluates its performance. Given a validation image, the model exploits the features it currently knows to predict an outcome. This provides useful information to configure the model accordingly. This set, however, does not teach the model any new features. If there is no test set, it usually occupies the remaining 20% of the total data. Otherwise, this percentage is reduced to about 10%, leaving the remaining 10% for the test set.

The test set determines whether the model truly performs well in new, unknown scenarios. In other words, it is the final evaluation step. Thus, the model encounters it only after it completes the training stage. If there is no separate test set, the validation set is used, but it is not a good practice. As mentioned, if there is a test set, it usually occupies 10% of the total data [44].

Figure 3.11 illustrates the training, validation, and test set splits when there is a test set. As shown, the training set contains most of the data, while the validation and tests set are of roughly equal, much smaller size.

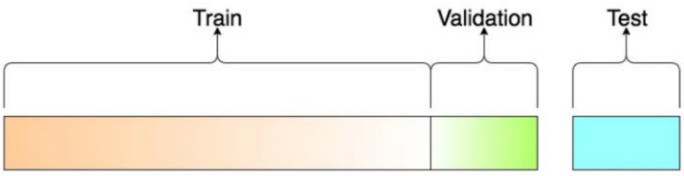

Figure 3.11: Training, validation, test set splits [44]

3.4.2 Training Parameters

The number of epochs determines how many times the training data will pass through the model. One epoch means the entire training dataset passes through only once. After each epoch, the model learns more features and updates its internal parameters. The number of epochs varies by model and task. Some require only 10, while others require hundreds, thousands, or more to train successfully.

Due to computer memory limitations, the training data cannot pass through the model all at once. Instead, it is divided into smaller groups, called batches. After all batches have been fed into the model, an epoch is complete [45].

The loss function calculates the difference between the ground truth and the prediction, reporting how accurately a model makes predictions. An optimization algorithm (or optimizer) iteratively searches for the local or global minimum to minimize the loss function. Deep Learning models achieve this by utilizing the learning rate parameter to control how fast or slow they learn. As shown in Figure 3.12, the goal is to reach the lowest point of the valley, which corresponds to the global minimum. A high learning rate (α in the Figure) takes large steps down the slope but risks skipping the lowest point as it bounces back and forth on the curve. A low learning rate is safer, as it carefully descends the slope. However, it takes longer to reach the bottom. Typical learning rate values include 0.001, 0.003, 0.01, 0.03, 0.1, and 0.3 [46].

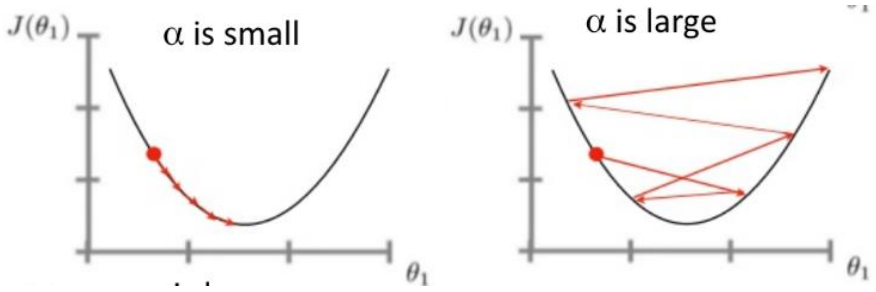

Figure 3.12: Low and high learning rate [46]

Momentum is closely related to learning rate. It makes the learning process faster while keeping it stable. Deep Learning scientists set it to 0.9 without further experimentation. However, this value has not been proven to fit all cases [47].

Weight decay is a small value that regularizes the loss function to reduce the serious problem of overfitting while improving model performance [48].

3.4.3 Underfitting and Overfitting

Underfitting and overfitting are the two main reasons for poor model performance. Ideally, a successful model can generalize. This means it can make accurate predictions on new, unknown data it will encounter in the future. Figure 3.13 demonstrates these two problems.

Underfitting occurs when the model is too simple to capture complex features, and thus, it does not generalize well on unknown data. It can also occur when the training set is too small to teach the model enough features. An underfit model performs poorly on both the training set and the validation set. One way to identify this problem is through high training and validation loss values. To solve this, the model needs to be more complex so that it can detect complex features. A higher number of epochs may also eliminate the problem. If the issue persists, the model may not be suitable for the task.

Overfitting occurs when the model memorizes complex patterns in the training set so well that it fails to generalize on unknown data. An overfit model will have significantly higher validation loss than training loss. Prevention methods include increasing the number of training data, stopping training early, performing data augmentation, and applying the weight decay parameter [49].

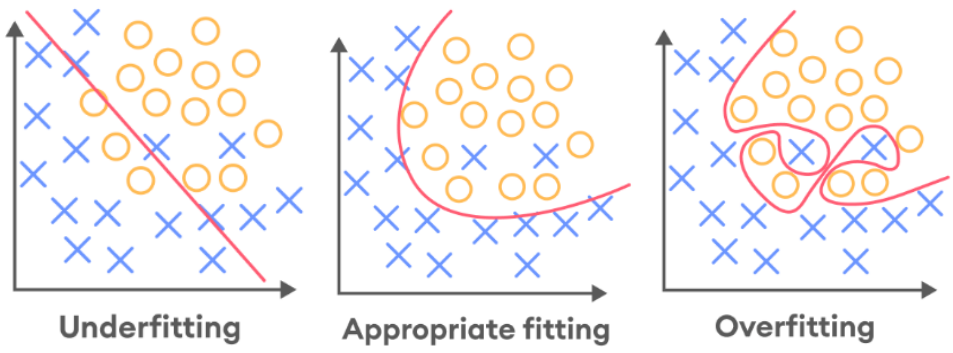

Figure 3.13: Underfitting and overfitting [49]

3.5 *Evaluation Metrics*

For the DE problem, specifically, Eigen et al. introduced five metrics to evaluate a DE model's performance: absolute relative error (Abs-Rel) (3.1), square relative error (Sq-Rel) (3.2), root mean square error (RMSE) (3.3), logarithm root mean square error (RMSE-log) (3.4), and accuracy under a threshold (δ_t) (3.5). To this day, scientists compare DE models, using the above metrics. Smaller values in the first four indicate that the model performs better, while a higher value in the last metric is preferred. The formulas are listed below, given a predicted depth map "pred", a ground truth depth map "gt", and the number of pixels in all images "P". Common values for the threshold include 1.25, 1.25^2, and 1.25^3 [12].

$$Abs - Rel = \frac{1}{P}\sum_{i \in P}\frac{|pred_i - gt_i|}{gt_i} \tag{3.1}$$

$$Sq - Rel = \frac{1}{P}\sum_{i \in P}\frac{|pred_i - gt_i|^2}{gt_i} \tag{3.2}$$

$$RMSE = \sqrt{\frac{1}{P}\sum_{i \in P}|pred_i - gt_i|^2} \tag{3.3}$$

$$RMSE - log = \sqrt{\frac{1}{P}\sum_{i \in P}|\log(pred_i) - \log(gt_i)|^2} \tag{3.4}$$

$$\delta_t = \frac{1}{P}|\sum_{i \in P}\max\left(\frac{pred_i}{gt_i}, \frac{gt_i}{pred_i}\right) < t| \times 100\% \tag{3.5}$$

As the formulas indicate, Abs-Rel (3.1) computes the sum of the absolute differences of the prediction and the ground truth divided by the ground truth. Sq-Rel (3.2) is similar to Abs-Rel, but computes the squared differences of the prediction and the ground truth. RMSE (3.3) computes the root of the sum of the squared differences of the prediction and the ground truth. RMSE-log (3.4) is similar to RMSE, but adds the logarithm of the prediction and the ground truth to the formula. Lastly, δ_t is the percentage of pixels with property max < threshold.

Eigen et al. also proposed the scale-invariant loss in the log space (SILog loss) (3.6) as the loss function for MDE. It compares the relationship between a pixel in the predicted depth map and the corresponding pixel in the ground truth, regardless of the scale difference between the prediction and the ground truth. The formula is shown below, given a predicted depth map "pred", a ground truth depth map "gt", and the number of pixels "P". "λ" is a hyperparameter with acceptable values between 0 and 1 [25]. Modern MDE approaches still employ the SILog

loss function to this day [22] [23] [31].

$$SILog\ loss = \frac{1}{P}\sum_{i \in P}(\log(pred_i) - \log(gt_i))^2 - \frac{\lambda}{P^2}(\sum_{i \in P}(\log(pred_i) - \log(gt_i)))^2 \quad (3.6)$$

3.6 Conclusion

This chapter explored Deep Learning's core architecture, CNNs, delving into three types of layers (convolution layer, pooling layer, fully connected layer) and their function. It also presented two commonly used activation functions: ReLU and softmax. Next, it reviewed DenseNet and ResNeXt, two popular CNNs scientists employ as the backbone of DE models.

The next subsection explained the complete training process of a CNN. Models are trained on the training set and evaluated on the validation and test sets. Some parameters adjust the duration and speed of training, while others prevent the serious problems of underfitting and overfitting. Underfitting occurs when the model is too simple or the data size is too small. On the other hand, overfitting occurs when the model makes very accurate predictions on the training set but fails to generalize on the validation set.

In the last subsection, this chapter discussed five metrics scientists use to compare DE models: Abs-Rel, Sq-Rel, RMSE, RMSE-log, where smaller values are better, and δ_t, where a higher value is better. It also presented the SILog loss function, which scientists still use as the loss function for their models.

4

Platforms and Programming Tools

4.1 Introduction

This chapter lists the platforms and tools to conduct experiments on the three selected MDE models for this thesis: BANet, LapDepth, and PixelFormer. Since their source code was written in Python, that language will be installed along with PyCharm, a popular IDE for viewing, writing, and editing Python scripts. PyCharm will also visualize all the graphs, images, and depth maps for the thesis.

The second part of the chapter introduces "Titan", a powerful remote server in the Department of Information and Electronic Engineering on which the three models will be trained, evaluated, and compared. In particular, it takes a look at Titan's specifications and several tools for working remotely on the server, such as WinSCP, PuTTY, and GitHub.

4.2 Python Machine Learning Tools

Python is a popular object-oriented programming language with a range of applications, such as software and web development, data science, and Machine Learning. It is versatile, fast, free to download, and easy to learn and read with English-like syntax. Developers run code line by line, thus detecting errors faster, and do not need to initialize variables, thus writing programs faster with fewer lines. In the fields of data science and Machine Learning, its data visualization libraries create charts, such as line charts, bar charts, pie charts, and histograms, and allow researchers to build deep learning algorithms. The first version of Python was released in 1991. Python 1.0 followed in 1994, then Python 2.0 in 2000, and finally, Python 3.0 in 2008.

Python offers more than 137,000 libraries, the most widely used of which are Matplotlib, Pandas, NumPy, and OpenCV-Python. Matplotlib visualizes 2D and 3D data in various types

of charts. Pandas manipulates data structures, such as tables and arrays. NumPy supports linear algebra operations for arrays. Lastly, OpenCV-Python handles image processing features, such as reading and writing images, creating 3D environments, and working with videos. Figure 4.1 displays the above and some other popular Machine Learning libraries.

Figure 4.1: Python's Machine Learning libraries [50]

Python also contains a variety of frameworks. A Python framework is a collection of packages and modules that help developers build programs faster since the framework handles the low-level details for them. A module is a piece of code, and a package is a set of modules. PyTorch is a Machine Learning framework based on the Torch library. Scientists prefer it in Deep Learning research to build models that extract information from images and videos. The models run on either a CPU or a GPU.

PyCharm is a popular Python IDE developed by JetBrains. The free, open-source Community Edition is suitable for small applications. Its paid Professional Edition includes the full set of features for building large-scale applications. Some of them are code completion, code inspection, a debugger, and support for frameworks and Machine Learning libraries, such as Matplotlib and NumPy [51].

Conda is an open-source package and environment manager originally for Python applications but now supports more languages, including R, Java, C, and C++. As a package manager, it finds, installs, and updates packages. As an environment manager, it sets up environments with different versions of Python to save, load, and switch between if a package requires a specific version [52].

4.3 Working Remotely on Titan Server

Titan is a server on the premises of the Department of Information and Electronic Engineering, located in the Alexandrian Campus of IHU. This powerful machine runs the Ubuntu 20.04.5 LTS operating system. On the hardware side, it uses an AMD Ryzen 9 5900X 12-Core Processor and an NVIDIA Titan Xp GPU, first released on April 6th, 2017 with a memory size of 12GB [29]. Students and academics access the server from their local computer through their user accounts and several tools for remote work.

Titan's GPU's CUDA version is 11.5. CUDA (Compute Unified Device Architecture) is a parallel programming platform and programming model developed by NVIDIA for NVIDIA GPUs. It speeds up the execution of tasks, as they run in parallel with other tasks instead of in a sequence. Thus, many scientists prefer it for Deep Learning research. CUDA also supports many programming languages, such as Python, C, C++, Fortran, and MATLAB [53].

WinSCP (Windows Secure Copy) is a free, open-source Secure File Transfer Protocol client for Microsoft Windows. It sets up an encrypted connection between a local Windows computer and a remote server for secure file transfer. WinSCP supports alternative protocol options, such as the unencrypted File Transfer Protocol and WebDAV. To access a remote server, users must first enter their login credentials. After the authentication process, they can upload and download files through the user-friendly interface, divided into two sections. The left side displays the directories and files on the local computer, while the right side displays everything on the remote server. WinSCP additionally offers a text editor for opening, editing, and saving files directly on the server [54].

PuTTY is a free, open-source terminal emulator. It establishes a secure connection between two devices typically of different operating systems, such as a local Windows computer and a remote Linux server. Available protocols include secure shell, Telnet, Secure Copy Protocol, and more. Users enter commands in the terminal that are executed on the server as if they were working directly on it [55]. Despite its data transferring capabilities, uploading and downloading files is only possible via the terminal, as it does not provide a graphical user interface. WinSCP can be used instead for this task. It also supports PuTTY integration, but users need to log in separately [56].

GitHub is a free, cloud-based software development platform. It uses Git, a free, open-source versioning control system that allows multiple users to collaborate on files and track all changes. After setting up a GitHub account, users store their files in a special folder, called a repository, which also contains version history information. Collaborators download a copy of the repository, called a clone, to their local computer. After modifying one or more files locally, they commit the changes along with a brief description. Next, they "push" the committed files

to the online repository on GitHub. They can also “pull” committed files from this repository. Collaborators can easily revert to previous versions when needed. GitHub Desktop is an application for Windows and Mac devices that allows users to perform the above actions through the application instead of the browser [57].

4.4 Conclusion

This chapter, divided into two subsections, reviewed all the applications of the current thesis for training, evaluation, and comparison of the three selected models: BANet, LapDepth, and PixelFormer. The first subsection provided information about the Python programming language, useful libraries, such as Matplotlib and Pandas, the PyTorch framework, the PyCharm IDE, and the Conda package and environment manager. The second subsection introduced the Titan server and three applications for accessing and working on it remotely: WinSCP, PuTTY, and GitHub.

5

Experimental Results

5.1 Introduction

This chapter covers the contribution of the current thesis. It begins by presenting the datasets on which the selected models, BANet, LapDepth, and PixelFormer will be trained and evaluated. One of them is a new dataset created specifically for this thesis.

As mentioned, the experiments will be conducted on the remote Titan server. For each model, and according to the papers' instructions, the Conda package and environment manager will create three different environments. In each one, it will install a specific version of Python along with the appropriate PyTorch and CUDA versions and any other necessary libraries. The models will train in their specified environment with different hyperparameter values to find the best combination. The SILog loss and RMSE metrics will be collected and plotted on line graphs showing how they change after each epoch for objective evaluation. Moreover, samples of the resulting depth maps will be visualized along with the corresponding ground truths for visual, subjective comparison. The last subsection of this chapter compares the three models' performance to select the best one.

Table 5.1: Quantitative comparison on the KITTI dataset
(The arrows' direction indicates better performance)

Model	Abs-Rel↓	Sq-Rel↓	RMSE↓	RMSE-log↓	$\delta_t < 1.25$↑	$\delta_t < 1.25^2$↑	$\delta_t < 1.25^3$↑
BANet	0.082	0.181	3.300	0.084	0.937	0.988	0.996
LapDepth	0.059	0.212	2.446	0.091	0.962	0.994	**0.999**
PixelFormer	**0.051**	**0.149**	**2.081**	**0.077**	**0.976**	**0.997**	**0.999**

Table 5.1 compares the three models' metrics after the original creators trained them on the KITTI dataset, which was common to all. The arrows' direction indicates better performance (smaller values for the first four metrics, higher values for the last three). According to the statistics, PixelFormer is expected to perform best, followed by LapDepth in second place, and BANet in third.

5.2 Datasets

The following three datasets will be used for the experiments: the outdoor images of DIODE (DIODE/Outdoor), a collection of images from the KITTI dataset (KITTI selection), and the new dataset, named "IHU". More specifically, DIODE/Outdoor will be used as a training, validation, and test set, while the other two will be used as test sets to monitor the models' performance in new, unknown scenarios.

5.2.1 DIODE/Outdoor

DIODE/Outdoor, along with DIODE/Indoor, is a subset of DIODE, created in 2019. DIODE is the first dataset to capture indoor and outdoor images with the same equipment, the FARO Focus S350 laser scanner, shown in Figure 5.1. The particular scanner can adapt to both indoor and outdoor environments with a range of 350 meters. Moreover, with the RGB camera close to the depth laser, there is seemingly no difference between the RGB images and depth maps it produces. Compared to other datasets, DIODE provides higher quality RGB images and depth maps with greater scene diversity.

Figure 5.1: FARO Focus Laser Scanner S350 [58]

DIODE/Outdoor, in particular, includes a large number of outdoor scenes, such as city streets, large residential buildings, hiking trails, parking lots, parks, forests, and river banks at short, medium, and long distances. The data was collected from three different cities with most of the scenes containing many objects near and far from the camera. Since each scan required 11 minutes to complete, the creators detected inconsistencies between some of the RGB images and their corresponding depth maps. If, for example, a vehicle moved away before the depth scan was complete, this would incorporate differences between the RGB image and the resulting depth map. Thus, they removed such scenes from the dataset. They also created a mask covering the sky and transparent objects, such as windows, for each depth map [59].

The creators divided DIODE/Outdoor into a training and a validation set. The training set contains 16,884 RGB images and their corresponding depth maps and masks, while for the validation set, this number is 446. Each RGB image has a resolution of 1024x768. The depth maps and masks are NumPy arrays of the same dimensions. Figure 5.2 presents a sample of the DIODE/Outdoor subset. The first two columns refer to the training set with the RGB image on the left and the corresponding depth map on the right. Similarly, the last two columns refer to the validation set with the RGB image on the left and the corresponding depth map on the right. The black color in the depth maps represents the mask.

The total size of DIODE/Outdoor is 124GB. For the thesis's experiments, the DIODE/Outdoor training set will be used as the training set to train the three models. Its validation set will be used as both the validation and the test set.

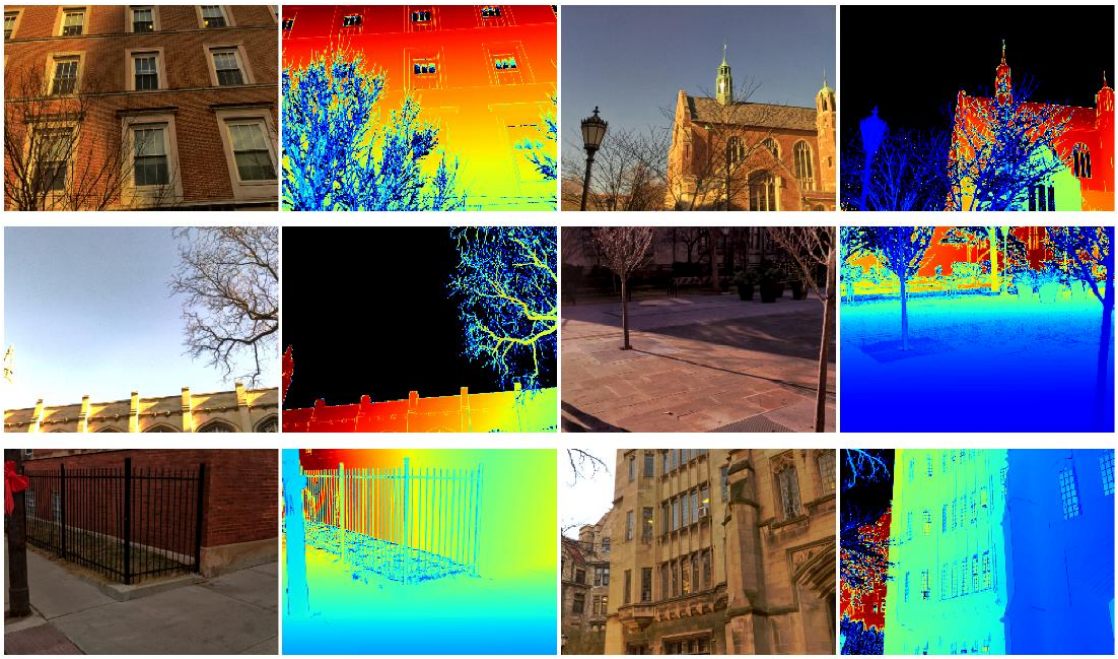

Figure 5.2: DIODE/Outdoor samples [59]
(Columns 1 and 3: RGB images. Columns 2 and 4: corresponding depth maps)

5.2.2 KITTI Selection

The KITTI dataset is mostly intended for autonomous driving applications. A Volkswagen station wagon with mounted equipment, as shown in Figure 5.3, drove around Karlsruhe, Germany, recording 6 hours of driving scenarios during morning hours. The setup included a Velodyne HDL-64E LiDAR sensor with a range of 120 meters and a PointGray Flea2 FL2-14S3C-C color camera among other devices for other tasks, such as BDE and object detection. The entire procedure lasted several days, from the 26th to the 30th of September 2011, and one on the 3rd of October of the same year [60].

Figure 5.3: KITTI recording equipment [60] [61] [62]

For MDE, the creators divided KITTI into a training set, a validation set, and a test set. The training set contains 94,000 RGB images and their corresponding depth maps in PNG format. They excluded the bottom part of the images, in which the vehicle's hood appeared, and the top part with the sky region, resulting in a resolution of 1241x375. The validation set contains 1,000 RGB images and depth maps, and the test set contains 500 RGB images without depth maps.

For this thesis, the validation set of KITTI, named "KITTI selection" for the experiments, will be used as a test set to evaluate the models' performance for the following reasons. The first one is that the test set itself does not include depth maps. Thus, comparing them with the model's predictions would be impossible. The second reason is that KITTI, overall, contains similar images to DIODE/Outdoor, such as vehicles, trees, roads, and other objects. Figure 5.4 depicts a sample from the KITTI selection dataset with the RGB images on the left and the depth maps on the right. The total size of this dataset is 802MB.

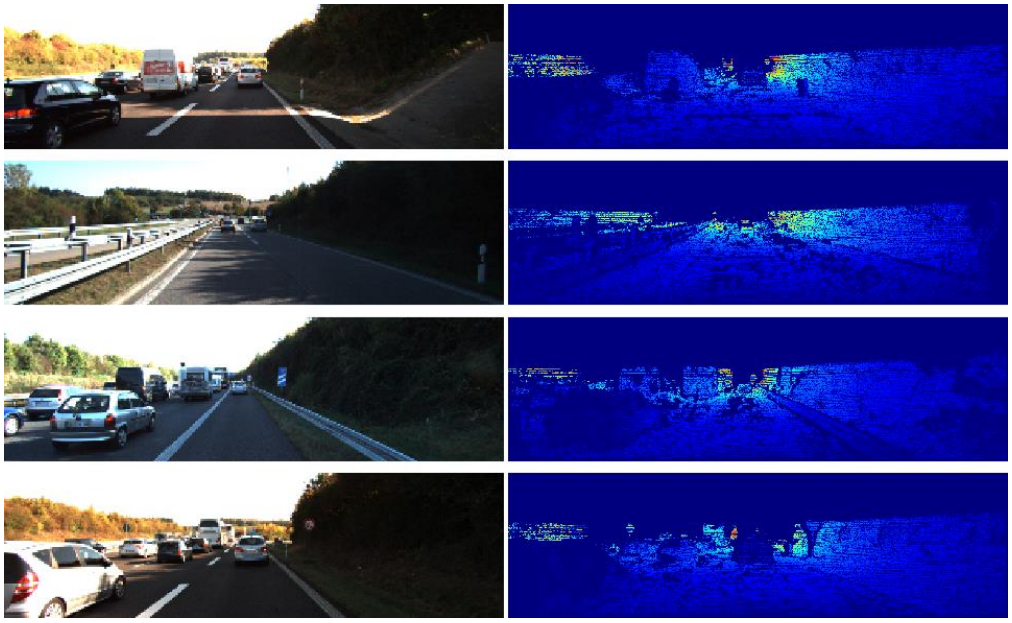

Figure 5.4: KITTI selection samples [60]
(Column 1: RGB images. Column 2: corresponding depth maps)

5.2.3 IHU

The IHU dataset was created specifically for this thesis. A Samsung Galaxy A13 5G smartphone, as shown in Figure 5.5, was the only equipment. The particular phone, released in 2022, has a triple rear camera. The second camera is the main, 50-megapixel camera, which collected the data. The other two, 2-megapixel cameras, which were not used, blur the background to capture artistic close-ups of people or objects. Thus, they were not suitable for creating a high-quality, long-range dataset [63].

Figure 5.5: Samsung Galaxy A13 5G [63]

IHU contains 30 RGB images taken after a walk around the Alexandrian Campus of IHU on the 19th of October 2023 from 6 p.m. to 7:30 p.m. The aspect ratio was set to 4:3, resulting in 3060x4080 resolution RGB images. They were then resized to 1024x768 with a total size of 4MB and DIODE's resolution into consideration to ease the models' DE process. Moreover, they depict similar outdoor environments to DIODE/Outdoor with objects, such as vehicles, roads, trees, and signs. Figure 5.6 features some examples of the IHU dataset. Due to its small size and the lack of ground truth depth maps, it will only be used as a test set. IHU will collect additional useful information on the models' ability to estimate depth maps.

Figure 5.6: IHU samples

5.3 Training BANet

The Bidirectional Attention Network, BANet, is based on the CNN architecture, employs DenseNet-161 as its backbone, and has 35×10^6 parameters. The overall architecture, explained in Chapter 2, is shown in Figure 5.7. As mentioned, BANet processes the images both forwards and backwards, using D2S (depth-to-space) modules in each of the five stages. These modules contain a convolution layer with a 1x1 kernel, an average pooling layer with a 32x32 filter, a fully connected layer, and a second convolution layer with a 3x3 kernel. D2S modules use the softmax activation function. Their output is fed to a convolution layer with a 9x9 kernel. At the end of the fifth stage, all feature maps are collected and processed with a convolution layer with a 3x3 kernel, ending with the softmax activation function before estimating the final depth map.

BANet was chosen as the first model to experiment with despite its lowest statistics in KITTI compared to the other two models. This is because BANet is currently one of the few models trained and tested on DIODE, as the dataset itself is also quite recent. Its implementation is easy to understand and execute without further complex preprocessing or code alterations, as BANet is 100% compatible with DIODE's RGB images and NumPy files. In general, it will provide a simple first look at working with this dataset. The whole process will prove useful in configuring the other two models to achieve better results, as BANet is not expected to outperform them.

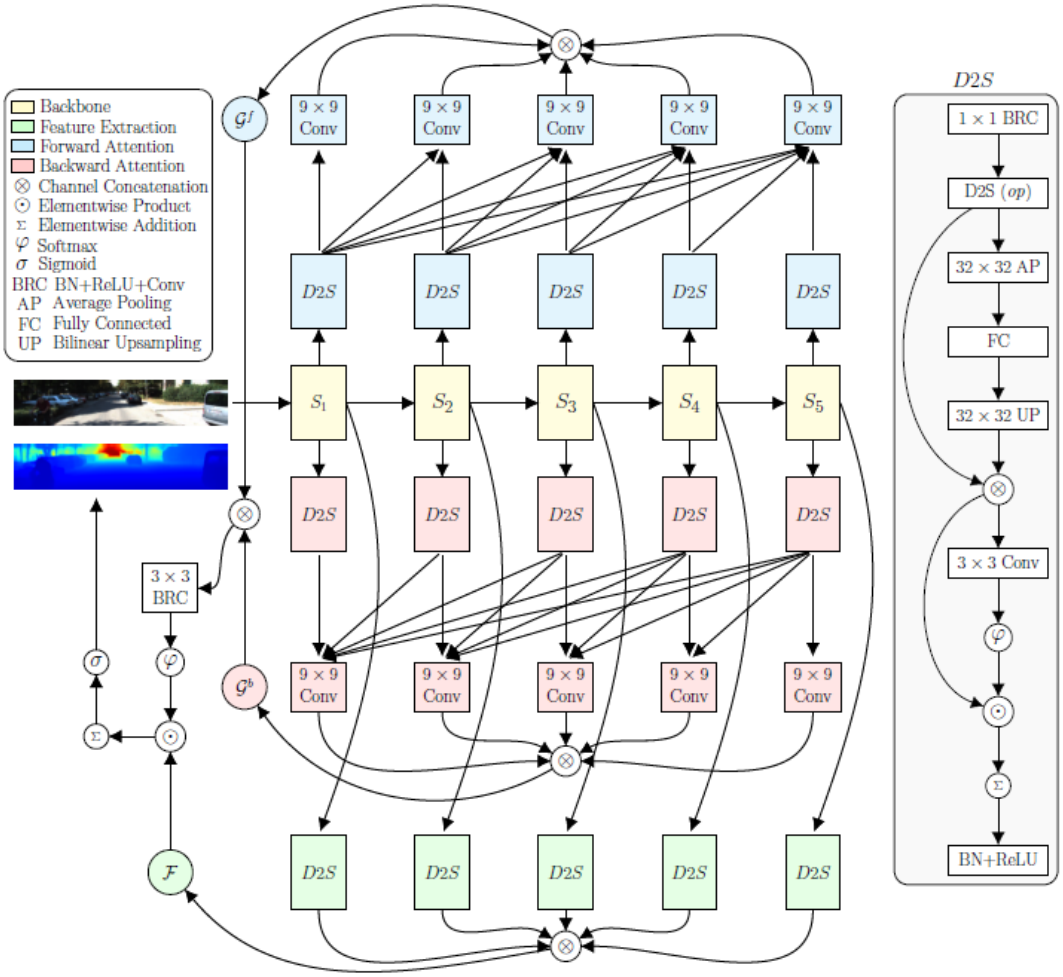

Figure 5.7: BANet architecture [22]

5.3.1 Experimental Setup

Conda installed the appropriate environment for BANet with Python 3.6.8, PyTorch 1.8.0 with CUDA 10.2, and all other necessary Python libraries. The model performed several tests with

different training hyperparameters and image augmentation techniques to find the combination that minimizes the loss function.

Following the paper's strategy, the number of epochs was set to 50 for all experiments. Batch sizes of 8, 16, and 32 were tested with 16 resulting in better performance as opposed to the paper's proposal of 32. Moreover, according to the paper, the SILog loss function was chosen with Adam as the optimizer. The learning rate was kept at 1e-4 with a momentum of 0.9 and a weight decay of 0. Both smaller and higher values were tested for these three hyperparameters, but none yielded better results. The depth range was set to 300 meters, similar to the paper.

Several image augmentation techniques were also tested: random flip, random crop, random rotate, and color jitter. However, none of the above improved performance. Ultimately, the only image augmentation technique used was center crop, which cut out a small portion of the images.

Table 5.2: Final experimental setup for BANet

Environment setup			Hyperparameter configuration				
Python	PyTorch	CUDA	Epochs	Batch size	Learning rate	Momentum	Weight decay
3.6.8	1.8.0	10.2	50	16	1e-4	0.9	0

Table 5.2 summarizes the final experimental setup for BANet. The environment setup includes the versions of Python, PyTorch, and CUDA used, while the hyperparameter configuration presents the values that minimize the loss function.

5.3.2 Experimental Results

The above configuration resulted in a validation loss of 47.975, BANet's lowest on DIODE/Outdoor after all experiments were completed. It achieved the specific loss value in epoch 22 with the corresponding RMSE value of 0.025 during the validation process.

Both the SILog loss and RMSE were monitored during each epoch for the training and validation set. Figure 5.8 displays them per epoch for BANet. The blue line refers to the training set, while the red line refers to the validation set. The line graph on the left presents the SILog loss per epoch. The training loss smoothly decreases from 65 in the first epoch to 40 in the last epoch. The validation loss starts at a value lower than the training loss, between 50 and 55 in the first epoch and reaches values less than 50 in the last epoch. From epoch 25 onwards, overfitting occurs. However, the code saved the model's state in time, thus avoiding the problem. Additionally, the line graph on the right displays the RMSE metric per epoch. As it

appears, the training error fluctuates between higher and lower values in each epoch. In contrast, the validation error remains consistently close to 0.025.

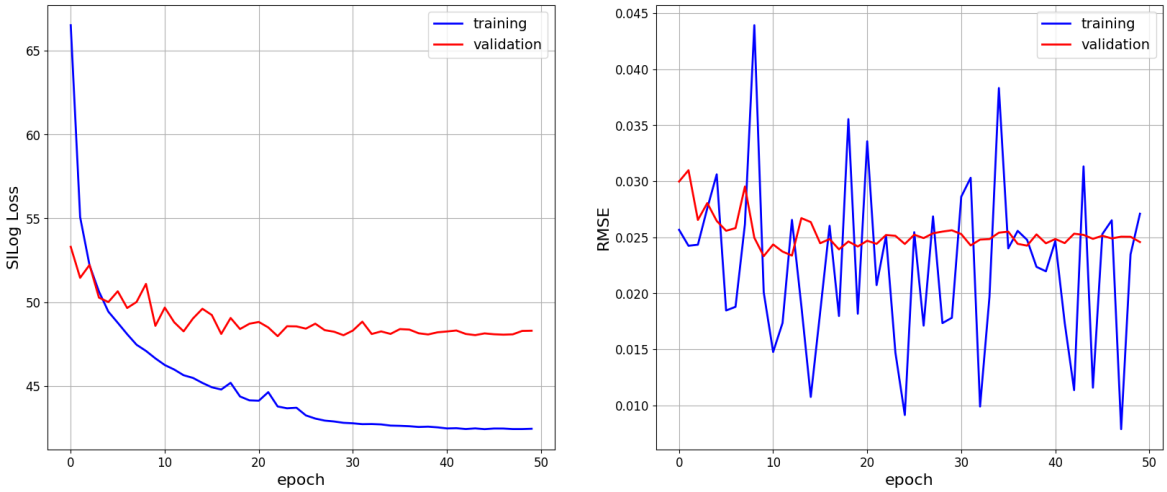

Figure 5.8: SILog loss and RMSE per epoch for BANet

The three Figures below (Figure 5.9, Figure 5.10, and Figure 5.11) visualize a sample of the predictions on the three datasets: DIODE/Outdoor, KITTI selection, and IHU respectively. Each one provides further observations and a better understanding of the model's behavior in various outdoor scenarios.

Figure 5.9 features the results on the DIODE/Outdoor validation set. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. The ground truths can be visually compared to the model's output due to their similar depth range: 350 and 300 meters respectively. Upon closer inspection, it is clear that BANet struggles with the details in the architecture of the buildings and even fails to detect the bench in the third image. Nevertheless, it captures the outlines of trees and cars quite well. BANet, also, does not seem to accurately predict depth along buildings but is better in open spaces.

Figure 5.10 features the results on KITTI selection. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. In KITTI selection's case, the ground truths have a range of 120 meters. Thus, they cannot be visually compared to the model's output's 300m range, which was trained according to the range of DIODE/Outdoor. Instead, the ground truths serve only as a reference. In this dataset, BANet completely fails to capture objects, such as cars and trees. Although it detects areas far from the camera (in red color), it produces outputs too blurry to understand what the scenes depict.

Figure 5.11 features the results on IHU. Columns 1 and 3 display the input images, while columns 2 and 4 display the estimated depth maps. Due to the lack of ground truth depth maps,

the results can only be interpreted visually. Although BANet detects buildings, it loses details of their architecture or even the building itself if some other object, such as tree branches and bushes, partially obscures it. Other scenes are too complex for the model to understand, such as the second image in column 3.

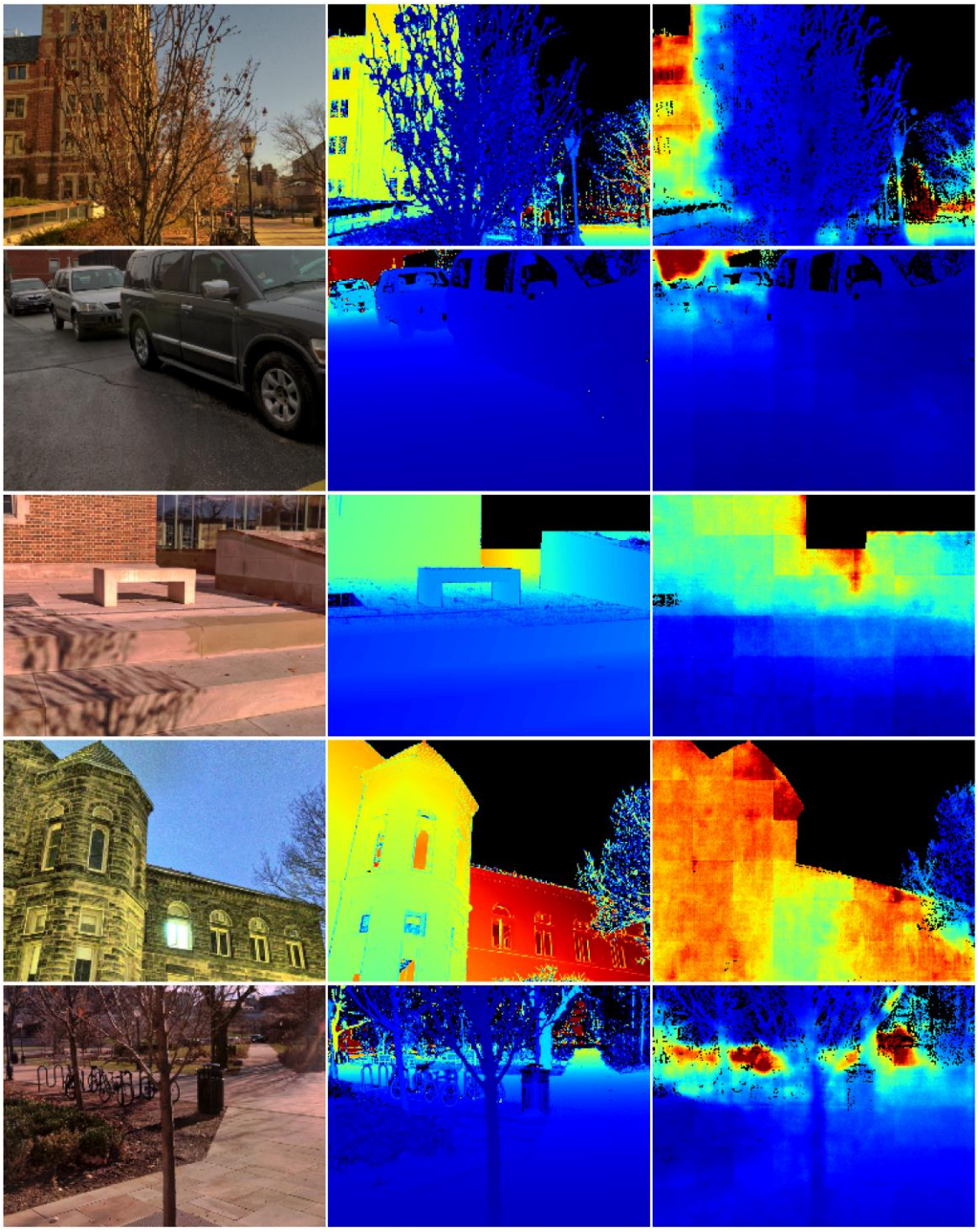

Figure 5.9: Prediction on DIODE/Outdoor with BANet
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

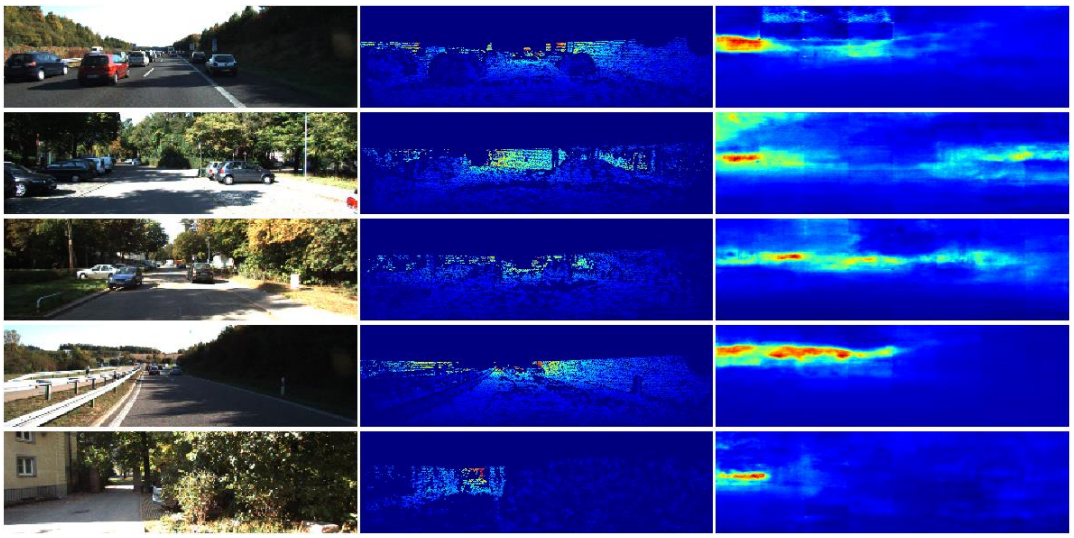

Figure 5.10: Prediction on KITTI selection with BANet
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

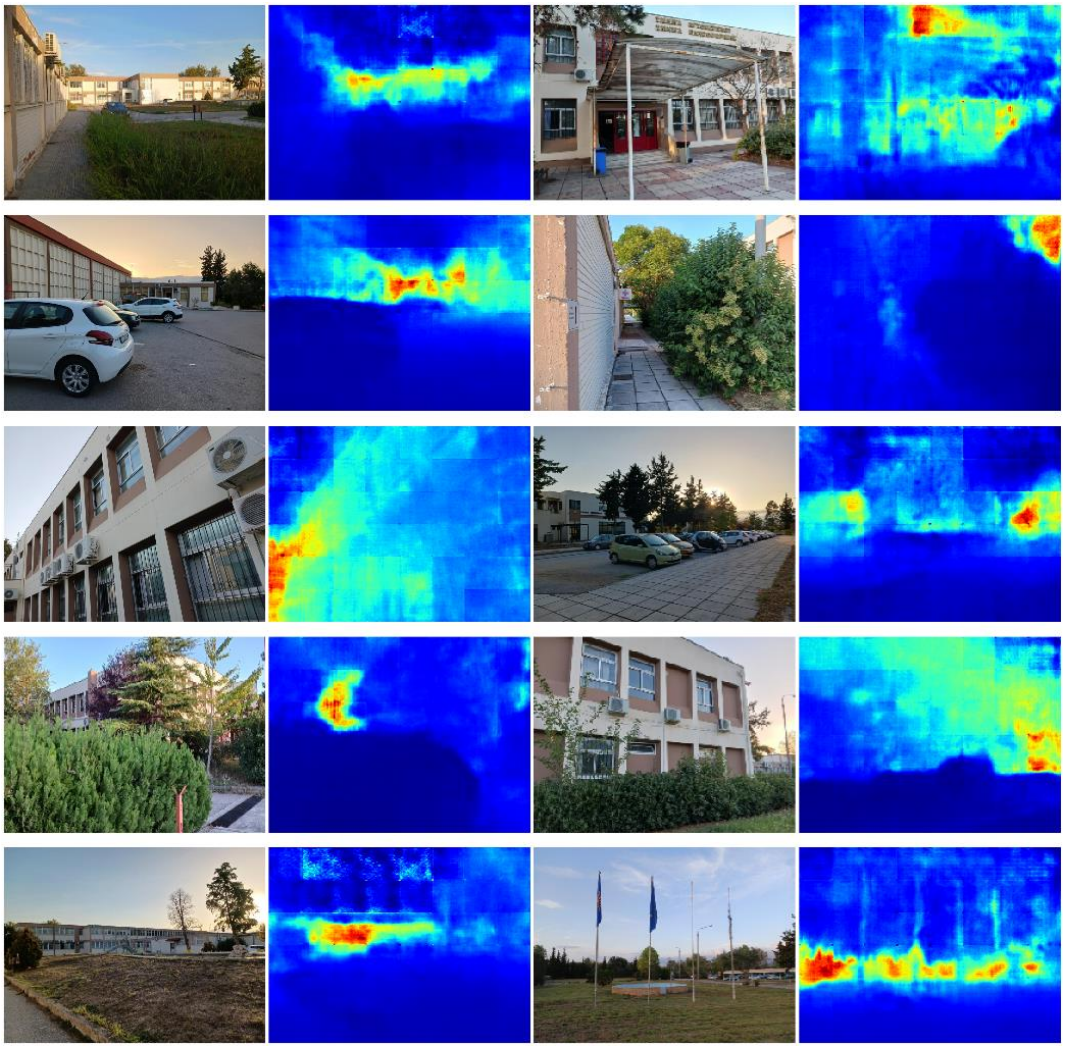

Figure 5.11: Prediction on IHU with BANet
(Columns 1 and 3 show the RGB images. Columns 2 and 4 show the predictions)

As expected, BANet did not achieve competitive results. Despite its fairly decent performance on DIODE/Outdoor, BANet struggles when it encounters a completely different dataset. Even in the case of KITTI selection, which contains roads and open spaces, it constructs blurry depth maps where all objects are missing. Although IHU's depth maps are of better quality, the model does not capture details and sometimes does not render scenes properly.

5.4 Training LapDepth

LapDepth is based on the CNN architecture, employs ResNeXt-101 as its backbone, and has $73x10^6$ parameters. The overall architecture, explained in Chapter 2, is given in Figure 5.12. As mentioned, its key feature is the five-level Laplacian pyramid. The input image progresses through all levels, which decompose it at multiple low-resolution scales. Moreover, each level utilizes multiple convolution layers with a 3x3 kernel [31].

LapDepth was chosen as the second model to experiment with, a bit more complex than BANet but still simple and easy to understand and work with. The instructions are also straightforward, and Titan can execute the code without further problems. Moreover, LapDepth is more modern, newer than BANet, and presents an alternative DE method: the Laplacian pyramid, which would be interesting to test. Its original statistics on KITTI are also quite good, with its depth maps capturing objects clearly, even thin ones, such as traffic signs. Thus, it is expected to perform better than BANet. However, since the original developers did not design LapDepth for training on DIODE, it is unclear how the model will behave.

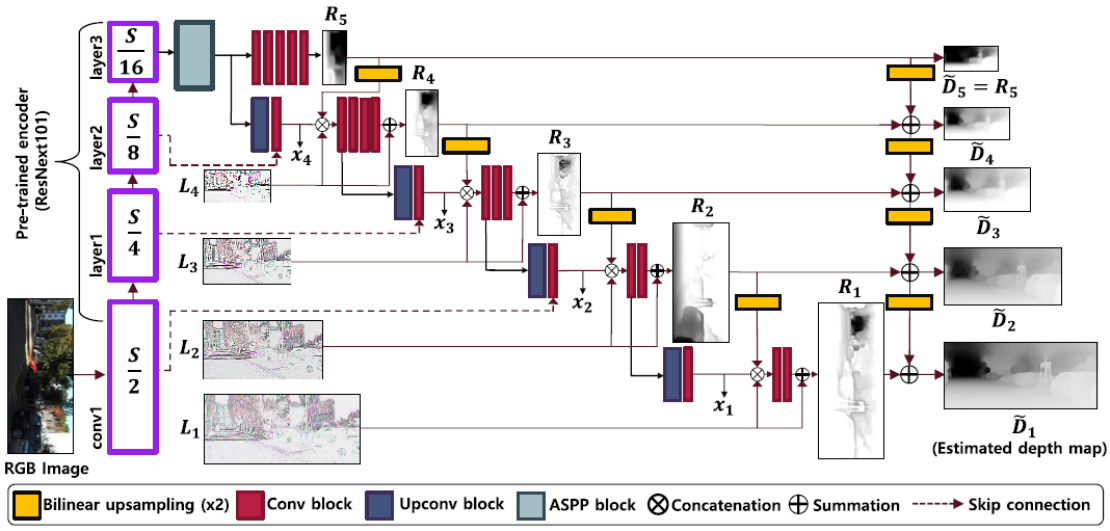

Figure 5.12: LapDepth architecture [31]

5.4.1 Experimental Setup

Conda installed the appropriate environment for LapDepth with Python 3.7, PyTorch 1.6.0 with CUDA 9.2, and all other necessary Python libraries. The model performed several tests with different training hyperparameters and image augmentation techniques to find the combination that minimizes the loss function.

Similarly to BANet, the number of epochs was set to 50 for all experiments. Batch sizes of 8 and 16 were tested with 16 resulting in better performance as in the paper. Due to hardware limitations, Titan could not perform experiments with a batch size of 32. Moreover, according to the paper, the SILog loss function was chosen with AdamW as the optimizer. The learning rate was kept at 1e-4 with a momentum of 0.9. Both smaller and higher values were tested for these two hyperparameters, but none yielded better results. Lastly, 1e-1 was chosen for weight decay after experimenting with several values.

Several image augmentation techniques were also tested: random crop, random horizontal flip, and color jitter. Random crop cut out important parts of the images, leading to poor performance. Thus, only random horizontal flip and color jitter were used.

Table 5.3: Final experimental setup for LapDepth

Environment setup			Hyperparameter configuration				
Python	PyTorch	CUDA	Epochs	Batch size	Learning rate	Momentum	Weight decay
3.7	1.6.0	9.2	50	16	1e-4	0.9	1e-1

Table 5.3 summarizes the final experimental setup for LapDepth. The environment setup includes the versions of Python, PyTorch, and CUDA used, while the hyperparameter configuration presents the values that minimize the loss function.

5.4.2 Experimental Results

The above configuration resulted in a training loss of 16.862, LapDepth's lowest on DIODE/Outdoor after all experiments were completed. It achieved the specific loss value in epoch 48 with the corresponding RMSE value of 1.937 during the validation process.

LapDepth's code does not provide any insight into the validation loss, so it was impossible to store it. Even after manually adding the function to the code for the validation process, it produced runtime errors, and the model aborted training. Nevertheless, the code fully stored the RMSE metric during each epoch for both the training and validation set.

Figure 5.13 displays the SILog loss and RMSE per epoch for LapDepth. The blue line represents the training set, while the red line represents the validation set. The line graph on the left displays the SILog loss per epoch. Training is unstable, with the loss fluctuating each epoch between higher and lower values. This behavior was expected since the graphs in the paper display the same abrupt loss shifts. Additionally, the line graph on the right displays the RMSE metric per epoch. As it appears, the fluctuation persists for both the training and the validation sets. The red line (validation set) always stays above the blue line (training set).

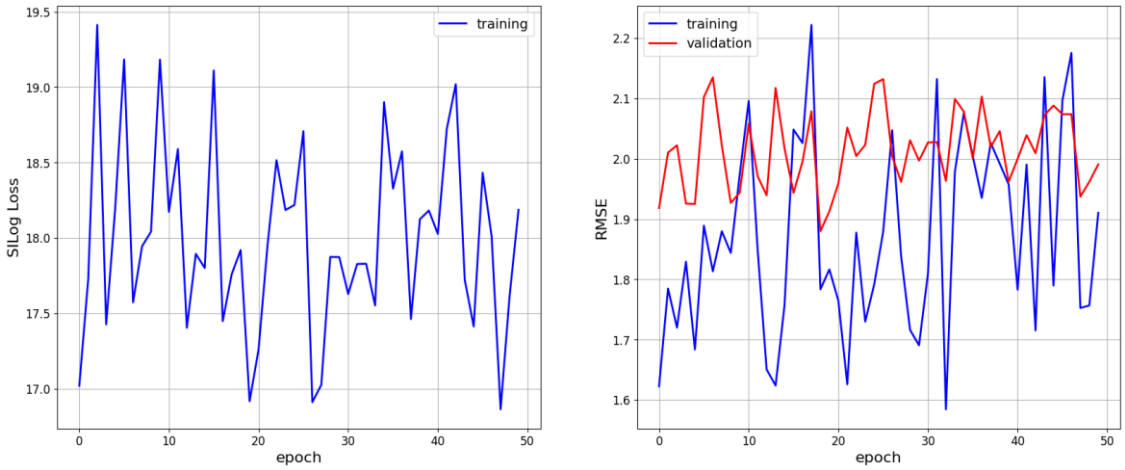

Figure 5.13: SILog loss and RMSE per epoch for LapDepth

The three Figures below (Figure 5.14, Figure 5.15, and Figure 5.16) visualize a sample of the predictions on the three datasets: DIODE/Outdoor, KITTI selection, and IHU respectively. Each one provides further observations and a better understanding of the model's behavior in various outdoor scenarios.

Figure 5.14 features the results on the DIODE/Outdoor validation set. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. The ground truths can be visually compared to the model's output due to their similar depth range: 350 and 300 meters respectively. Upon closer inspection, LapDepth captures slightly more detail in building architecture than BANet, mainly windows and corners. It also better distinguishes pixels that are far from those that are close and recognizes cars, trees, bushes, and cables. They still appear blurry but not to BANet's extent. Nevertheless, LapDepth still makes mistakes in estimating depth, especially along the walls of buildings.

Figure 5.15 features the results on KITTI selection. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. In KITTI selection's case, the ground truths have a range of 120 meters. Thus, they cannot be visually compared to the model's output's 300m range, which was trained according to the range of

DIODE/Outdoor. Instead, the ground truths serve only as a reference. Unlike BANet, which failed to capture objects in this dataset, LapDepth detects buildings along with their windows. Although it also draws the outlines of trees and cars, they are barely visible. In addition, LapDepth has a better understanding of the scene than BANet, assigning depth values to areas far and close to the camera more accurately.

Figure 5.16 features the results on IHU. Columns 1 and 3 display the input images, while columns 2 and 4 display the estimated depth maps. Due to the lack of ground truth depth maps, the results can only be interpreted visually. LapDepth produces clearer depth maps than BANet, defining building edges, corners, and windows more accurately. It also detects cars, trees, and traffic signs. Although the output is somewhat blurry, it is still possible to understand the scene and even detect fine details. The main problem with this model occurs when objects in the image are too far away, for example, the first image in column 3, and the third image in column 1. In this case, the model incorrectly assumes that the buildings are close to the camera and depicts them in shades of blue.

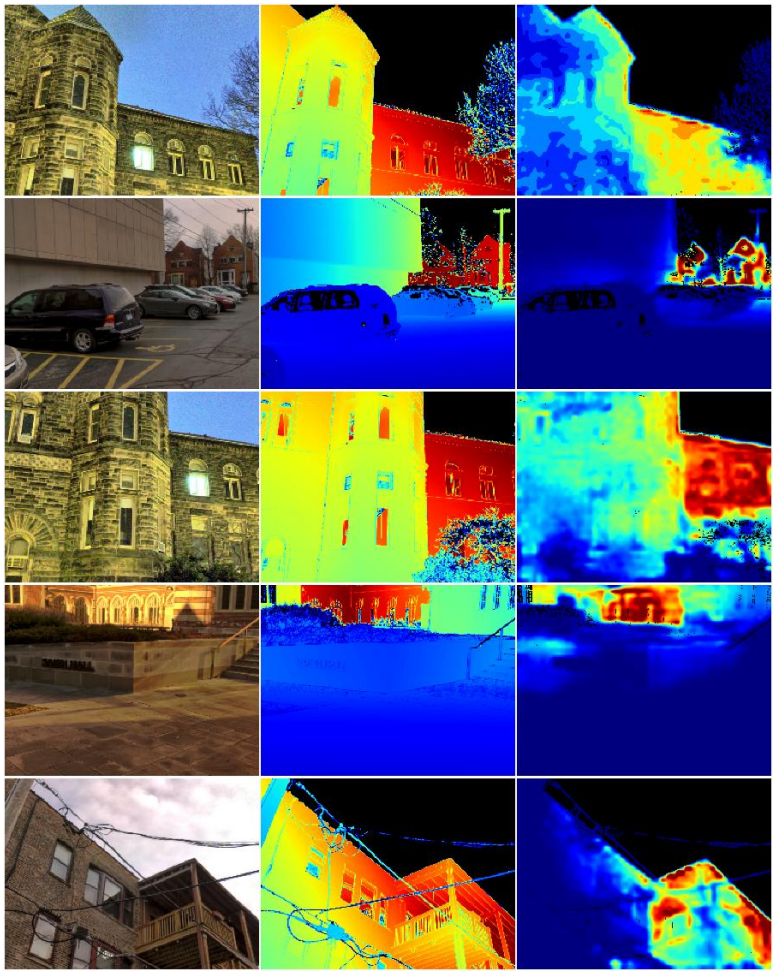

Figure 5.14: Prediction on DIODE/Outdoor with LapDepth
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

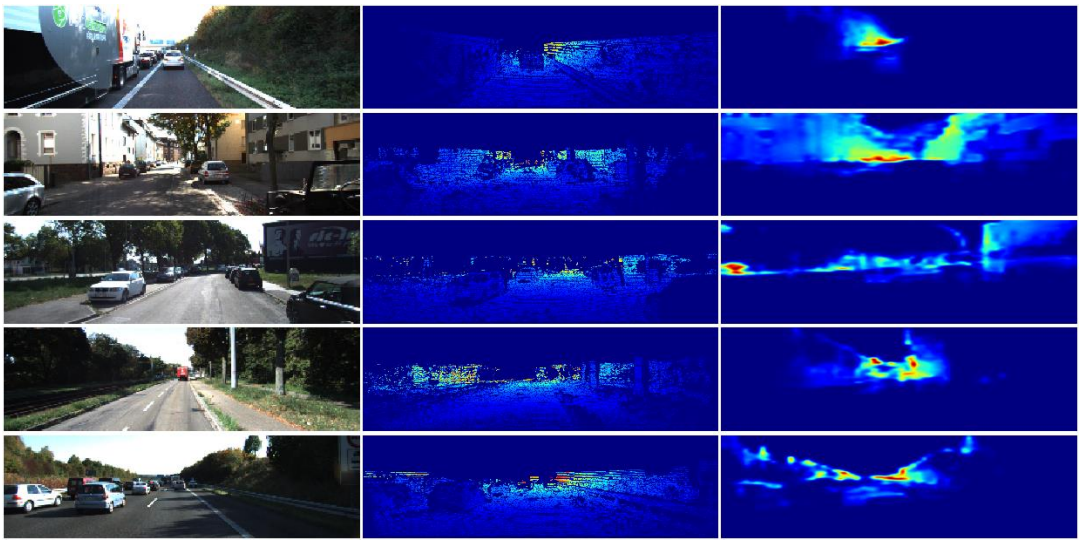

Figure 5.15: Prediction on KITTI selection with LapDepth
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

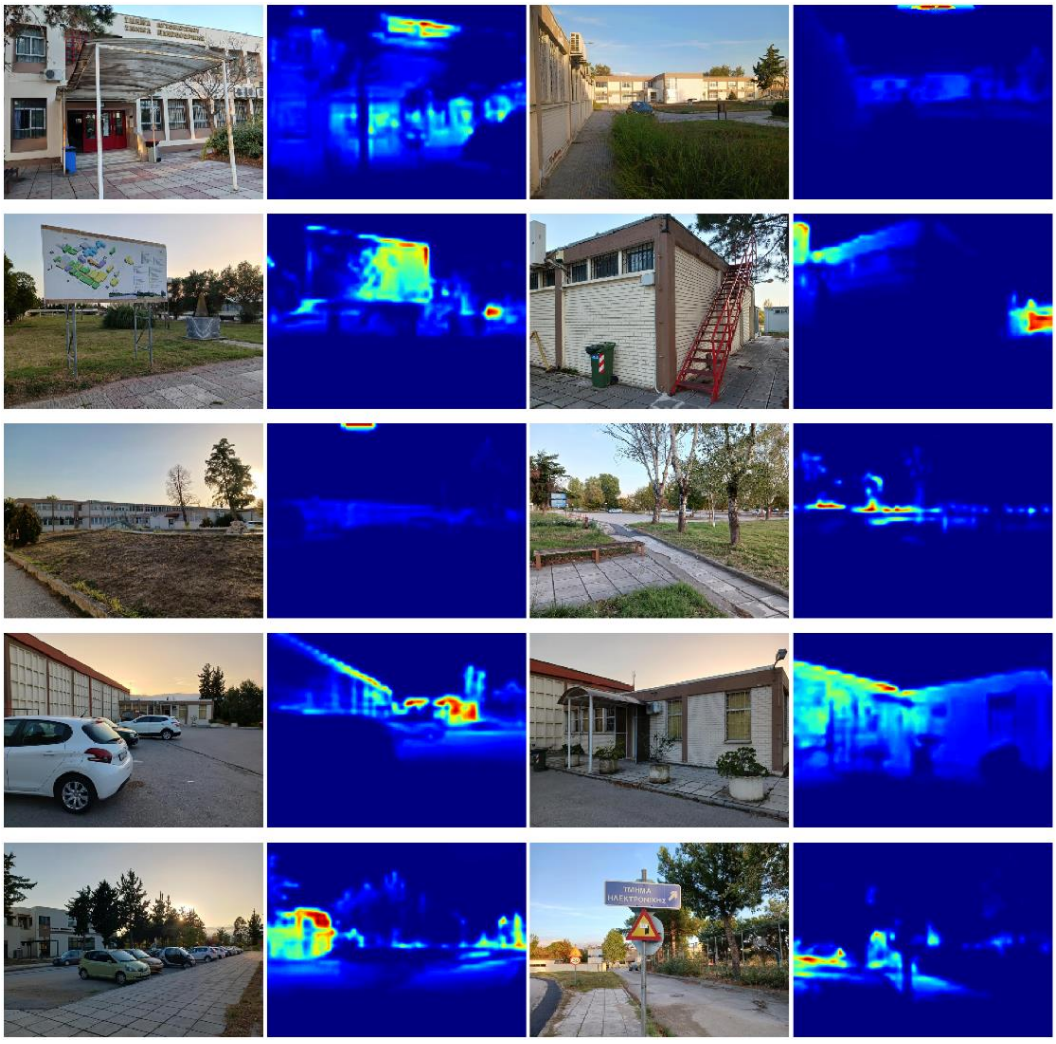

Figure 5.16: Prediction on IHU with LapDepth
(Columns 1 and 3: RGB images. Columns 2 and 4: predictions)

Overall, LapDepth did not adapt well to training on DIODE/Outdoor. A similar problem with BANet is its inability to predict depth accurately along walls. LapDepth is, however, better at adding details to building architecture, mainly windows and corners. The depth maps this model produces are slightly blurry, but it is easy to understand what they depict. Even on KITTI selection and IHU, the datasets it has never encountered before, LapDepth performs decently, with the exception of buildings that are too far away.

5.5 Training PixelFormer

PixelFormer is also based on the CNN architecture, employs SwinTransformer as its backbone, and has $270x10^6$ parameters. The overall architecture, explained in Chapter 2, is shown in Figure 5.17. As mentioned, its key component is the transformer. The image enters SwinTransformer to extract features at multiple low-resolution scales, which three different modules process afterwards: the Pixel Query Initializer (PQI), the Bin Center Predictor (BCP), and the Skip Attention Module (SAM). More specifically, the PQI module accumulates the global scene information from the coarsest scale with global average pooling and convolution. The BCP module discretizes depth into intervals with global average pooling. Lastly, the SAM module enhances resolution with finer details by using convolution. At the end of the network is a convolution operation, followed by the softmax activation function to predict the final depth map.

Due to its high complexity, PixelFormer was left until last after experimenting with the two previous, simpler models. Its excellent performance on KITTI, compared to BANet and LapDepth, makes it a competitive candidate for testing on DIODE/Outdoor. Moreover, it is newer than the other two models examined in this thesis, proposing a different DE method: the transformer. For the above reasons, PixelFormer is expected to outperform both BANet and LapDepth. However, it was not designed to be trained on DIODE and, as in the case of LapDepth, it is unclear how it will behave.

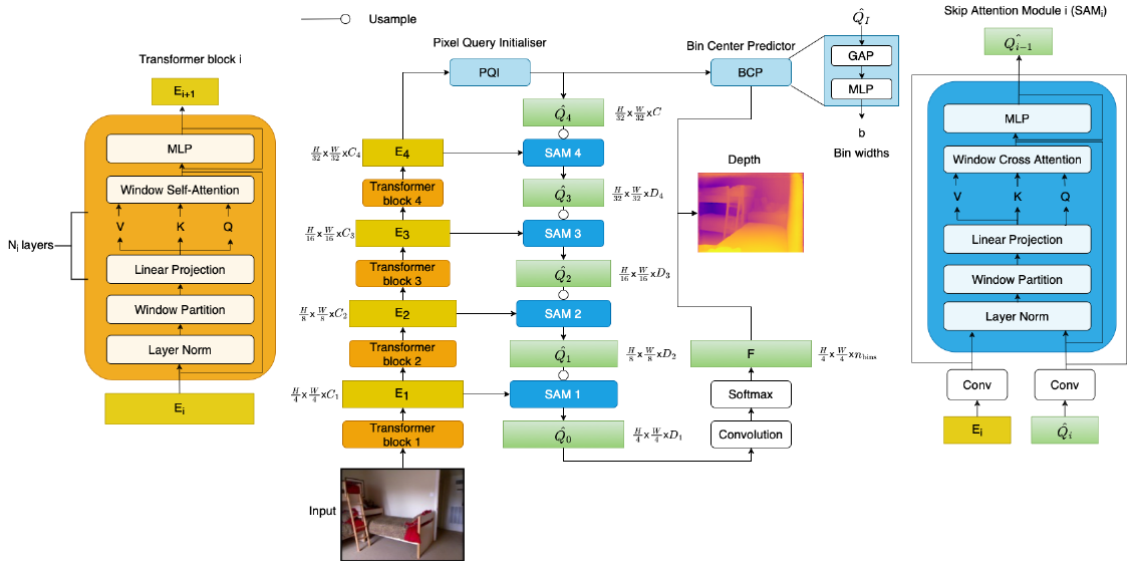

Figure 5.17: PixelFormer architecture [23]

5.5.1 Experimental Setup

Conda installed the appropriate environment for PixelFormer with Python 3.8, PyTorch 1.10.0 with CUDA 10.2, and all other necessary Python libraries. The model performed several tests with different training hyperparameters and image augmentation techniques to find the combination that minimizes the loss function.

Similarly to BANet and LapDepth, the number of epochs was set to 50 for all experiments. Due to hardware limitations, Titan can only perform experiments with a batch size of 8 for this model. Moreover, according to the paper, the SILog loss function was chosen with Adam as the optimizer. The learning rate was kept at 4e-5 with a momentum of 0.9. The weight decay was changed to 1e-1 after testing several smaller and higher values.

Only two image augmentation techniques were tested: random crop and color jitter, both resulting in poor performance. Ultimately, no image augmentation techniques were used for this model.

Table 5.4: Final experimental setup for PixelFormer

Environment setup			Hyperparameter configuration				
Python	PyTorch	CUDA	Epochs	Batch size	Learning rate	Momentum	Weight decay
3.8	1.10.0	10.2	50	8	4e-5	0.9	1e-1

Table 5.4 summarizes the final experimental setup for BANet. The environment setup includes the versions of Python, PyTorch, and CUDA used, while the hyperparameter configuration presents the values that minimize the loss function.

5.5.2 Experimental Results

The above configuration resulted in a validation loss of 43.252, PixelFormer's lowest on DIODE/Outdoor after all experiments were completed. It achieved the specific loss value in epoch 5 with the corresponding RMSE value of 0.112 during the validation process.

Both the SILog loss and RMSE were monitored during each epoch for the training and validation set. Figure 5.18 displays them per epoch for PixelFormer. The blue line refers to the training set, while the red line refers to the validation set. The line graph on the left presents the SILog loss per epoch. Training seems to flow smoothly but with minimal loss reduction. Furthermore, the validation loss remains at significantly higher values (around 45) than the training loss (around 10), indicating signs of overfitting even from the first epoch. Additionally, the line graph on the right displays the RMSE metric per epoch. As it appears, both the training and validation error fluctuate between higher and lower values in each epoch with the validation error always below the training error.

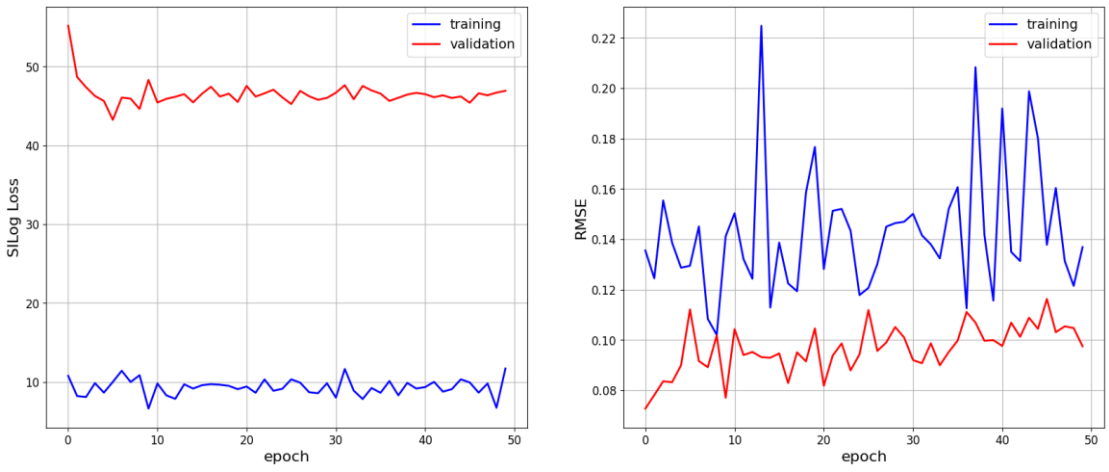

Figure 5.18: SILog loss and RMSE per epoch for PixelFormer

The three Figures below (Figure 5.19, Figure 5.20, and Figure 5.21) visualize a sample of the predictions on the three datasets: DIODE/Outdoor, KITTI selection, and IHU respectively. Each one provides further observations and a better understanding of the model's behavior in various outdoor scenarios.

Figure 5.19 features the results on the DIODE/Outdoor validation set. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. The ground truths can be visually compared to the model's output due to their similar depth range: 350 and 300 meters respectively. Upon closer inspection, PixelFormer renders each scene in exquisite detail, especially building architecture, and defines sharp outlines around objects. Nevertheless, it suffers from the same problem as BANet and LapDepth, struggling to accurately predict depth along building walls.

Figure 5.20 features the results on KITTI selection. The input images are on the left with the ground truth depth maps in the middle and the estimated depth maps on the right. In KITTI selection's case, the ground truths have a range of 120 meters. Thus, they cannot be visually compared to the model's output's 300m range, which was trained according to the range of DIODE/Outdoor. Instead, the ground truths serve only as a reference. In this dataset, PixelFormer continues to detect objects in great detail, such as cars, trees, and railings. It also assigns depth values more accurately since KITTI consists of roads and open spaces.

Figure 5.21 features the results on IHU. Columns 1 and 3 display the input images, while columns 2 and 4 display the estimated depth maps. Due to the lack of ground truth depth maps, the results can only be interpreted visually. PixelFormer fully understands each scene, capturing trees, bushes, cars, flags, signs, and even the tiniest details on buildings. Contrary to DIODE/Outdoor, depth transitions more smoothly from blue to red hues along building walls as the distance increases.

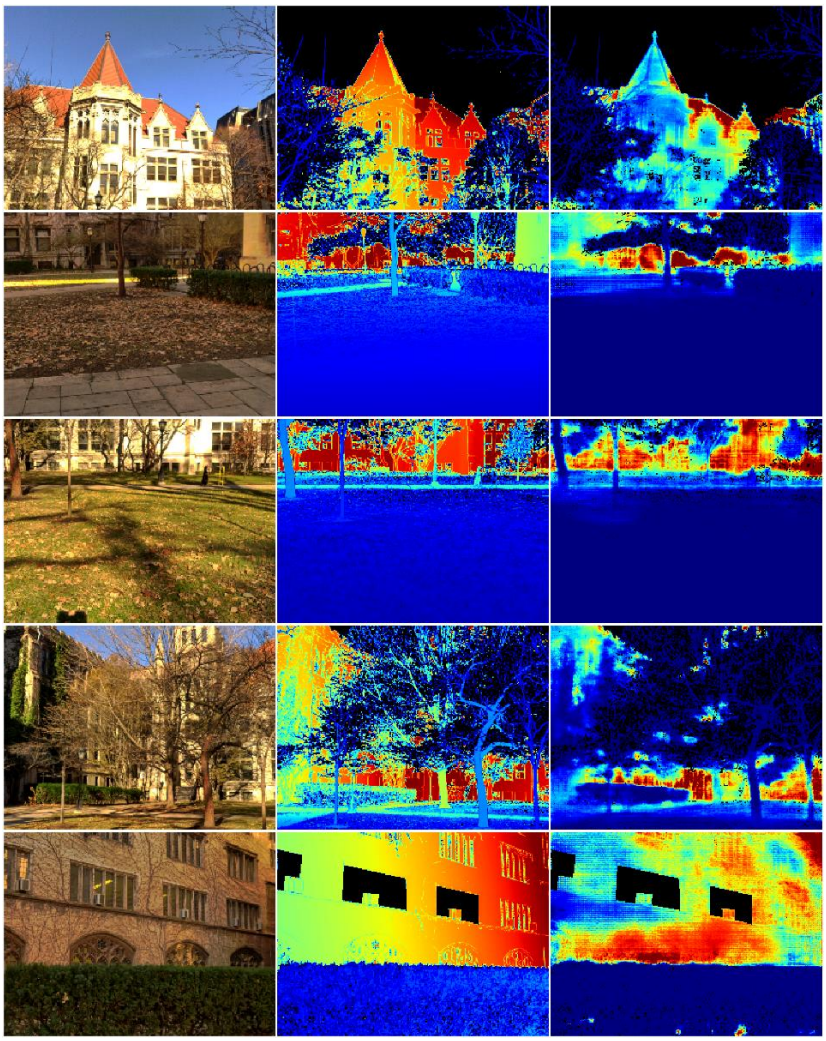

Figure 5.19: Prediction on DIODE/Outdoor with PixelFormer
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

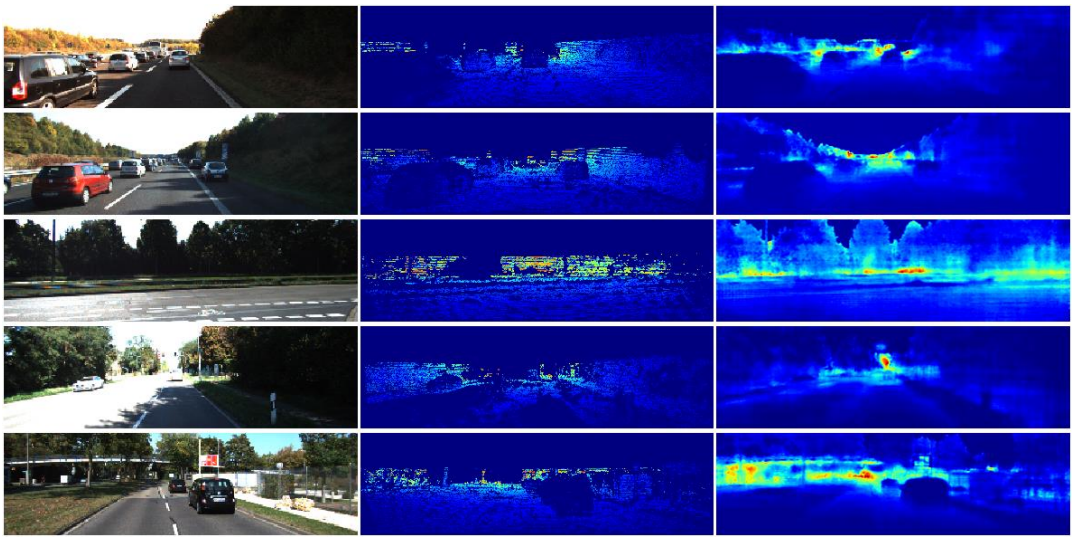

Figure 5.20: Prediction on KITTI selection with PixelFormer
(Column 1: RGB images. Column 2: ground truths. Column 3: predictions)

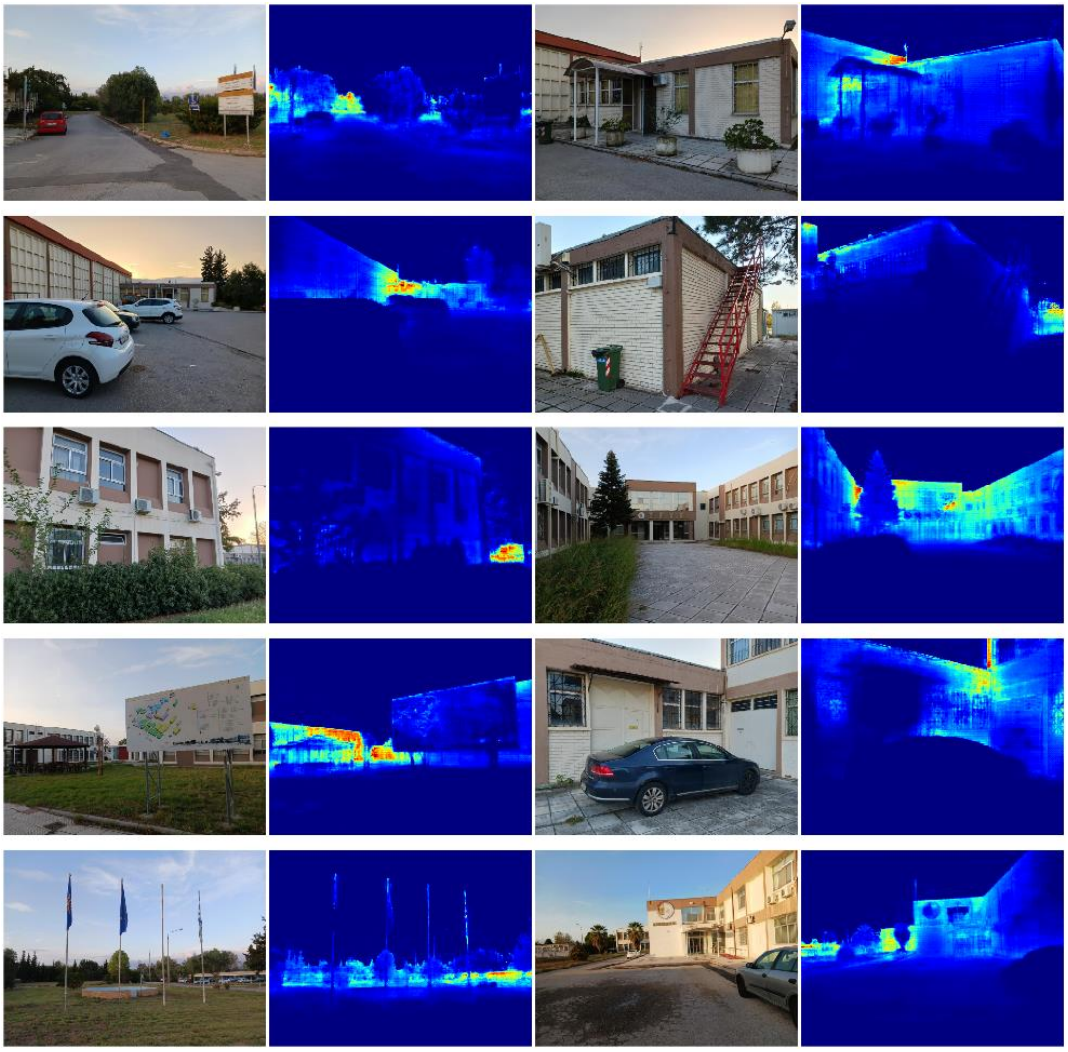

Figure 5.21: Prediction on IHU with PixelFormer
(Columns 1 and 3: RGB images. Columns 2 and 4: predictions)

Overall, PixelFormer adapted well to DIODE, although it was not designed to train on it. It faced the same difficulties as BANet and LapDepth, predicting depth along walls imprecisely, but it achieved competitive results as expected. The model produces aesthetically pleasing depth maps on all three datasets, adding the slightest detail to the architecture of buildings even at a great distance. It also detects thin objects, such as flags, signs, trees, and bushes. In the IHU dataset, in particular, it predicts depth along walls more smoothly compared to BANet and LapDepth and even compared to DIODE/Outdoor, from which it was inspired.

5.6 Final Model Comparison

Understanding the architecture and code of each model required up to a month with the difficulty level increasing from BANet to LapDepth and PixelFormer. During this time, the

code was adjusted to work with DIODE/Outdoor and extract statistics and NymPy files for comparing the plots and depth maps. The next step was to test different values for the hyperparameters to find the combination that minimized the loss function. With 50 epochs, each model completed training after approximately a day and a half of testing a single value. Although this was a relatively short waiting time, an extra month was needed to try all possible values for a single model. Technical problems arose during that time, such as frequent power outages in the Alexandrian Campus, which housed Titan. This delayed the completion of the experiments significantly.

Table 5.5: Experimental setup comparison

Model	Environment setup			Hyperparameter configuration				
	Python	PyTorch	CUDA	Epochs	Batch size	Learning rate	Momentum	Weight decay
BANet	3.6.8	1.8.0	10.2	50	16	1e-4	0.9	0
LapDepth	3.7	1.6.0	9.2	50	16	1e-4	0.9	1e-1
PixelFormer	3.8	1.10.0	10.2	50	8	4e-5	0.9	1e-1

Table 5.5 compares the final experimental setup of BANet, LapDepth, and PixelFormer, which yielded the best results for each model. The environment setup includes the versions of Python, PyTorch, and CUDA used, where PixelFormer employs a more recent version of Python and PyTorch than BANet and LapDepth. The hyperparameter configuration presents the values that minimize the loss function, which differ slightly for each model.

First, the models were compared objectively, through the SILog loss and RMSE graphs. BANet's loss decreased smoothly in both the training and validation sets. The code saved the model's state in time to avoid overfitting. This resulted in a validation loss of 47.975 in epoch 22 with an RMSE value of 0.025, placing BANet third among the three models. LapDepth faced unstable training with fluctuations in each epoch and no information about the validation loss. Nevertheless, it resulted in a training loss of 16.862 in epoch 48 with an RMSE value of 1.937. Even without the validation loss, LapDepth could be placed second, despite its high RMSE value, because its training loss is significantly lower than that of BANet. This leaves first place for PixelFormer, which exhibited severe overfitting during training. It achieved, however, pretty decent results in epoch 5 with a validation loss of 43.252 and an RMSE value of 0.112.

Table 5.6: Quantitative comparison on DIODE/Outdoor
(The arrows' direction indicates better performance)

Model	Epoch	SILog loss↓	RMSE↓
BANet	22	47.975	**0.025**
LapDepth	48	N/A	1.937
PixelFormer	5	**43.252**	0.112

Table 5.6 provides the statistics listed above, including the best SILog loss and RMSE each model achieved and in which epoch. PixelFormer did so in epoch 5, much earlier than the other models.

Next, the models were compared visually and subjectively through the predicted depth maps. Figure 5.22 displays their performance on IHU. The input images are in the top row with the models' output in the following three rows (BANet in the second row, LapDepth in the third, and PixelFormer in the fourth). BANet produced poor quality depth maps. They were blurry, and in some cases, it was impossible to understand the scene. LapDepth greatly outperformed BANet with higher quality depth maps. They were less blurry, and it was easier to interpret what they depicted. PixelFormer added even more detail to its depth maps, which were clear and more aesthetically pleasing.

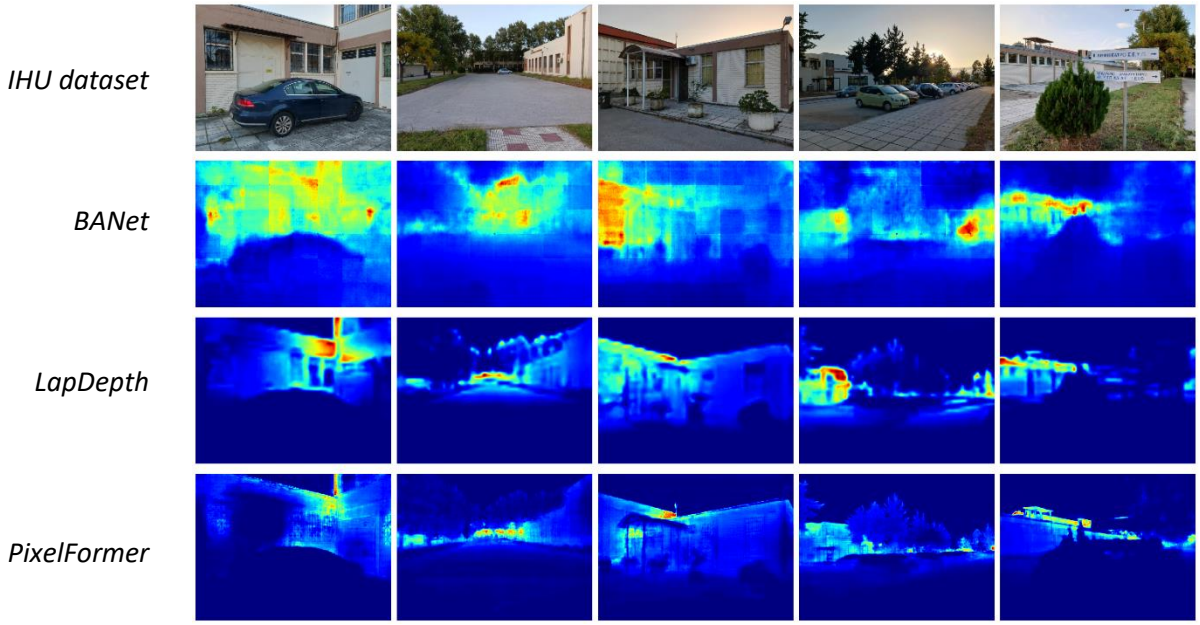

Figure 5.22: Visual comparison on IHU

The objective and subjective evaluation shows that PixelFormer is clearly the winning model for its excellent performance against BANet and LapDepth. For a dataset of such complexity

as DIODE/Outdoor, which contains buildings, windows, small architectural details, trees, and other objects, a complex model, like PixelFormer, proved to be the most suitable.

All three models had different impacts on the thesis and served different purposes. BANet was more friendly to someone new to MDE, who wanted to practice and learn without minding the quality of the depth maps. LapDepth was best suited for those familiar with MDE, who wanted to improve their skills while obtaining decent enough depth maps. PixelFormer was more difficult to understand, and therefore, more intended for those with a deeper knowledge of MDE, who were looking for high-quality depth maps.

5.7 Conclusion

This chapter investigated the training, validation, and test results of the three models, BANet, LapDepth, and PixelFormer on three datasets: DIODE/Outdoor, KITTI selection, and IHU. DIODE/Outdoor was used as a training, validation, and test set, while the other two were used for testing purposes only. PixelFormer outperformed the other two models with a validation loss of 43.252, RMSE of 0.112, and detailed depth maps. Thus, it was declared as the winning model of the thesis. Its success lies in its complex architecture, making it suitable for training on the complex DIODE/Outdoor dataset.

6

Conclusions and Future Work

6.1 Conclusions

This thesis compared three state-of-the-art Deep Learning models for MDE: BANet, LapDepth, and PixelFormer. The results determined which one predicted depth more accurately based on the SILog loss and RMSE statistics for objective evaluation, and the visual quality of the produced depth maps for subjective evaluation. An overview of DE was presented, as well as the main challenge MDE models face in predicting depth accurately since monocular images lack the stereoscopic vision of stereo pairs.

Previous related research demonstrated the evolution of MDE over three periods: the Early Period, the Machine Learning Period, and the Deep Learning Period. Early Period approaches were simple and lacked detail, but they served the needs of their time. Machine Learning Period approaches were a little more complex and captured more detail in the produced depth maps. Deep Learning Period approaches, as the name suggests, employed Deep Learning networks to detect even more details. Researchers trained their models on popular datasets, such as NYU-Depth V2 and KITTI, adding various image augmentation techniques to achieve the desired result. BANet, LapDepth, and PixelFormer were among the models discussed in this period, which are based on CNN architecture.

A CNN is a type of Deep Learning network that processes images through convolution, pooling, and fully connected layers. DenseNet and ResNeXt are two popular types of CNNs that MDE models employ in their backbone. Successful training requires splitting the dataset into the training, the validation, and the test set. Moreover, various hyperparameters, including the number of epochs, batch size, learning rate, momentum, and weight decay, control the training process. Serious underfitting and overfitting problems may also occur. After the training stage,

five metrics evaluate an MDE model's performance: Abs-Rel, Sq-Rel, RMSE, RMSE-log, and δ_t. MDE models usually prefer the SILog loss as the loss function.

Several platforms and tools were installed for the necessary experiments on the three selected models of the thesis. PyCharm was selected as the main programming environment since the creators of the models developed the code in Python. IHU's server, Titan, trained these models from scratch. WinSCP, PuTTY, and GitHub provided access to the server for remote work.

Three datasets were used for the experiments: DIODE/Outdoor, KITTI selection, and IHU. DIODE/Outdoor served as the training, validation, and test set, while the other two were only used as test sets. IHU, in particular, was a new dataset created specifically for this thesis with images from the Alexandrian Campus inspired by DIODE/Outdoor.

The experimental stage of the thesis started with BANet, the simplest of the three and one of the only models that its developers trained and tested on DIODE. After configuring several hyperparameters and image augmentation techniques, BANet achieved a validation loss of 47.975 and an RMSE value of 0.025 in epoch 22. In the SILog loss per epoch line graph, the training and validation curves decreased smoothly, and the code saved the model's state before overfitting. The RMSE per epoch line graph displayed fluctuations in the training curve, while the validation set remained more stable around values closer to zero. The predicted depth maps demonstrated BANet's struggle to understand datasets it had never seen before.

LapDepth was chosen next, as it was newer, a little more complex than BANet, and its designers achieved state-of-the-art results on the KITTI dataset. After configuring several hyperparameters and image augmentation techniques, LapDepth achieved a training loss of 15.440 and an RMSE value of 1.937 in epoch 48. The code did not provide information about the validation loss, and errors occurred after manually adding the function. Even without this information, both the SILog loss and RMSE per epoch line graphs showed its unstable training. Nevertheless, LapDepth produced decent depth maps and understood new, unknown datasets, capturing details albeit blurry.

PixelFormer was kept for last because of its extremely complex architecture. This model was expected to outperform the other two due to the excellent results its developers achieved on the KITTI dataset. After configuring several hyperparameters and image augmentation techniques, PixelFormer achieved a validation loss of 43.252 and an RMSE value of 0.112 in epoch 5. The SILog loss per epoch line graph exhibited a severe overfitting problem from the beginning of the training, while the RMSE per epoch line graph displayed fluctuations. However, the predicted depth maps captured even the tiniest details in the buildings' architecture and defined clear object boundaries.

The final comparison demonstrated PixelFormer's superior performance over BANet and LapDepth in both objective and subjective evaluation. The reason for its success lied in its

complex architecture, suitable for training on the complex DIODE/Outdoor dataset. Other conclusions include BANet being more intended for those new to MDE, LapDepth to those familiar with MDE, and PixelFormer to those with a deeper knowledge of MDE.

6.2 Future Work

The winning model, PixelFormer, will be delivered to "Draive" for an autonomous driving project they are working on. PixelFormer will assist the vehicle in making better decisions, such as the direction to steer.

Other plans for future work could include training the three models on DIODE/Indoor to investigate their response to indoor environments. Since scientists continue to develop new MDE models, they could be added to the research. Moreover, other methods, such as estimating depth from monocular videos, could be explored.

One issue not addressed in this thesis was KITTI having a different range than DIODE/Outdoor: 120 meters and 350 meters respectively. Thus, the ground truth depth maps were used as a reference, not for actual comparison with the predictions. A future goal is to convert the range of the predicted depth maps to 120 meters and compare the results anew, objectively rather than subjectively.

7

References

[1] U. Kulkarni, S. B. Devagiri, R. B. Devaranavadagi, S. Pamali, N. R. Negalli and V. Prabakaran, "Depth Estimation using DNN Architecture and Vision-Based Transformers," in *ITM Web of Conferences*, EDP Sciences, 2023.

[2] B. Raj, "Depth Estimation," Medium, 16 February 2019. [Online]. Available: https://medium.com/beyondminds/depth-estimation-cad24b0099f. [Accessed 18 October 2023].

[3] M. Poggi, F. Tosi, K. Batsos, P. Mordohai and S. Mattoccia, "On the synergies between machine learning and binocular stereo for depth estimation from images: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 9, pp. 5314-5334, 2021.

[4] "LIDAR-Lite v3," GRobotronics, 14 November 2016. [Online]. Available: https://grobotronics.com/lidar-lite-v3.html?sl=en. [Accessed 13 October 2023].

[5] X. Chen, R. Zhang, J. Jiang, Y. Wang and G. Li, "Self-supervised monocular depth estimation: Solving the edge-fattening problem," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023.

[6] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif and D. Puig, "Monocular depth estimation using deep learning: A review," *Sensors*, vol. 22, no. 14, p. 5353, 2022.

[7] C.-Y. Wu, Q. Gao, C.-C. Hsu, T.-L. Wu, J.-W. Chen and U. Neumann, "InSpaceType: Reconsider Space Type in Indoor Monocular Depth Estimation," *arXiv preprint arXiv:2309.13516*, 2023.

[8] "Colormaps," HoloViews, 31 March 2018. [Online]. Available: https://holoviews.org/user_guide/Colormaps.html. [Accessed 24 October 2023].

[9] A. Mertan, D. J. Duff and G. Unal, "Single image depth estimation: An overview," *Digital Signal Processing*, vol. 123, p. 103441, 2022.

[10] G. S. Brizuela, "Scale invariant log loss mathematical proof," 13 June 2022. [Online]. Available: https://guillesanbri.com/Scale-Invariant-Loss/. [Accessed 30 November 2023].

[11] A. Bhoi, "Monocular depth estimation: A survey," *arXiv preprint arXiv:1901.09402*, 2019.

[12] Y. Ming, X. Meng, C. Fan and H. Yu, "Deep learning for monocular depth estimation: A review," *Neurocomputing*, vol. 438, pp. 14-33, 2021.

[13] P. Vyas, C. Saxena, A. Badapanda and A. Goswami, "Outdoor monocular depth estimation: A research review," *arXiv preprint arXiv:2205.01399*, 2022.

[14] M. A. Reza, J. Kosecka and P. David, "Farsight: Long-range depth estimation from outdoor images," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 4751-4757.

[15] A. W. Bergman, D. B. Lindell and G. Wetzstein, "Deep adaptive lidar: End-to-end optimization of sampling and depth completion at low sampling rates," in *2020 IEEE international conference on computational photography (ICCP)*, IEEE, 2020, pp. 1-11.

[16] M. Pan, H. Zhang, J. Wu and Z. Jin, "SELF-DISTILLATION NETWORK FOR INDOOR AND OUTDOOR MONOCULAR DEPTH ESTIMATION".

[17] "draive," Draive, 21 October 2021. [Online]. Available: https://draive.gr. [Accessed 23 November 2023].

[18] A. Saxena, S. Chung and A. Ng, "Learning depth from single monocular images," *Advances in neural information processing systems*, vol. 18, 2005.

[19] X.-L. Deng, X.-H. Jiang, Q.-G. Liu and W.-X. Wang, "Automatic depth map estimation of monocular indoor environments," in *2008 International Conference on MultiMedia and Information Technology*, IEEE, 2008, pp. 646-649.

[20] Y. Salih, A. S. Malik and Z. May, "Depth estimation using monocular cues from single image," in *2011 National Postgraduate Conference*, IEEE, 2011, pp. 1-4.

[21] C.-C. Su, L. K. Cormack and A. C. Bovik, "Depth estimation from monocular color images using natural scene statistics models," in *IVMSP 2013*, IEEE, 2013, pp. 1-4.

[22] S. Aich, J. M. U. Vianney, M. A. Islam and M. K. B. Liu, "Bidirectional attention network for monocular depth estimation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 11746-11752.

[23] A. Agarwal and C. Arora, "Attention attention everywhere: Monocular depth prediction with skip attention," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 5861-5870.

[24] S. Tanwar, "Image Augmentation," Medium, 4 August 2021. [Online]. Available: https://medium.com/analytics-vidhya/image-augmentation-9b7be3972e27. [Accessed 2 November 2023].

[25] D. Eigen, C. Puhrsch and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," *Advances in neural information processing systems*, vol. 27, 2014.

[26] "NVIDIA GeForce GTX TITAN BLACK," TECHPOWERUP, [Online]. Available: https://www.techpowerup.com/gpu-specs/geforce-gtx-titan-black.c2549. [Accessed 3 November 2023].

[27] I. Alhashim and P. Wonka, "High quality monocular depth estimation via transfer learning," *arXiv preprint arXiv:1812.11941*, 2018.

[28] K. Sanghvi, "Image Classification Techniques," Medium, 8 May 2020. [Online]. Available: https://medium.com/analytics-vidhya/image-classification-techniques-83fd87011cac. [Accessed 30 October 2023].

[29] "NVIDIA TITAN Xp," TECHPOWERUP, [Online]. Available: https://www.techpowerup.com/gpu-specs/titan-xp.c2948. [Accessed 3 November 2023].

[30] "NVIDIA Tesla V100 PCIe 32 GB," TECHPOWERUP, [Online]. Available: https://www.techpowerup.com/gpu-specs/tesla-v100-pcie-32-gb.c3184. [Accessed 3 November 2023].

[31] M. Song, S. Lim and W. Kim, "Monocular depth estimation using laplacian pyramid-based depth residuals," *IEEE transactions on circuits and systems for video technology*, vol. 31, no. 11, pp. 4381-4393, 2021.

[32] E. L. Denton, S. Chintala and R. Fergus, "Deep generative image models using a laplacian pyramid of adversarial networks," *Advances in neural information processing systems*, vol. 28, 2015.

[33] Y. Gavrilova, "Transformers in ML: What They Are and How They Work," Serokell, 11 April 2023. [Online]. Available: https://serokell.io/blog/transformers-in-ml. [Accessed 4 November 2023].

[34] "NVIDIA A100 SXM4 40 GB," [Online]. Available: https://www.techpowerup.com/gpu-specs/a100-sxm4-40-gb.c3506. [Accessed 3 November 2023].

[35] "3 Types of Machine Learning You Should Know," Coursera, 26 October 2023. [Online]. Available: https://www.coursera.org/articles/types-of-machine-learning. [Accessed 5 November 2023].

[36] "Introduction to Deep Learning," GeeksforGeeks, 14 April 2023. [Online]. Available: https://www.geeksforgeeks.org/introduction-deep-learning/. [Accessed 5 November 2023].

[37] R. Yamashita, M. Nishio, R. K. G. Do and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, pp. 611-629, 2018.

[38] H. Gu, Y. Wang, S. Hong and G. Gui, "Blind channel identification aided generalized automatic modulation recognition based on deep learning," *IEEE Access*, vol. 7, pp. 110722-110729, 2019.

[39] "Pooling in a CNN: Pooling Layers Explained," KnowledgeHut, 5 September 2023. [Online]. Available: https://www.knowledgehut.com/blog/data-science/pooling-layer. [Accessed 8 November 2023].

[40] M. Yani, S. S. M. Budhi Irawan and S. M. Casi Setiningsih, "Application of transfer learning using convolutional neural network method for early detection of terry's nail," *Journal of Physics: Conference Series*, vol. 1201, no. 1, p. 12052, 2019.

[41] V. Jain, "Everything you need to know about "Activation Functions" in Deep learning models," Medium, 30 December 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253. [Accessed 6 November 2023].

[42] "Softmax Function," BotPenguin, 16 August 2023. [Online]. Available: https://botpenguin.com/glossary/softmax-function. [Accessed 10 November 2023].

[43] V. Kurama, "A Review of Popular Deep Learning Architectures: DenseNet, ResNeXt, MnasNet, and ShuffleNet v2," Paperspace, 22 june 2020. [Online]. Available: https://blog.paperspace.com/popular-deep-learning-architectures-densenet-mnasnet-shufflenet/. [Accessed 10 November 2023].

[44] T. Shah, "About Train, Validation and Test Sets in Machine Learning," Medium, 6 December 2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7. [Accessed 12 November 2023].

[45] "What is Epoch in Machine Learning?," Simplilearn, 7 November 2023. [Online]. Available: https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-is-epoch-in-machine-learning. [Accessed 13 November 2023].

[46] C. Liu, "5 Concepts You Should Know About Gradient Descent and Cost Function," KD nuggets, 16 September 2022. [Online]. Available: https://www.kdnuggets.com/2020/05/5-concepts-gradient-descent-cost-function.html. [Accessed 13 November 2023].

[47] C. R. Wolfe, "Why 0.9? Towards Better Momentum Strategies in Deep Learning.," Medium, 26 February 2021. [Online]. Available: https://towardsdatascience.com/why-0-9-towards-better-momentum-strategies-in-deep-learning-827408503650. [Accessed 14 November 2023].

[48] S. Yang, "Deep learning basics — weight decay," Medium, 4 September 2020. [Online]. Available: https://medium.com/analytics-vidhya/deep-learning-basics-weight-decay-3c68eb4344e9. [Accessed 14 November 2023].

[49] "Overfitting and underfitting in machine learning," SuperAnnotate, 17 October 2022. [Online]. Available: https://www.superannotate.com/blog/overfitting-and-underfitting-in-machine-learning. [Accessed 14 November 2023].

[50] A. Kalbande, "Best Python Libraries For Machine Learning," Fireblaze AI School, 18 June 2020. [Online]. Available: https://www.fireblazeaischool.in/blogs/python-libraries-for-machine-learning/. [Accessed 19 November 2023].

[51] "What is Python?," AWS, [Online]. Available: https://aws.amazon.com/what-is/python/. [Accessed 19 November 2023].

[52] "Conda," Anaconda, Inc., [Online]. Available: https://docs.conda.io/en/latest/. [Accessed 20 November 2023].

[53] F. Oh, "What Is CUDA?," NVIDIA, 10 September 2012. [Online]. Available: https://blogs.nvidia.com/blog/what-is-cuda-2/. [Accessed 2 December 2023].

[54] "WinSCP," Digital Guide IONOS, 12 May 2023. [Online]. Available: https://www.ionos.com/digitalguide/hosting/technical-matters/encrypted-data-transfer-with-winscp/. [Accessed 21 November 2023].

[55] "What is PuTTY? A Comprehensive Guide," Rushax, 16 September 2023. [Online]. Available: https://rushax.com/what-is-putty-a-comprehensive-guide/. [Accessed 22 November 2023].

[56] "PuTTY Home - Free Downloads, Tutorials, and How-Tos," SSH, 19 April 2017. [Online]. Available: https://www.ssh.com/academy/ssh/putty. [Accessed 22 November 2023].

[57] B. Lutkevich, "What is GitHub?," TechTarget, 21 February 2023. [Online]. Available: https://www.techtarget.com/searchitoperations/definition/GitHub. [Accessed 21 November 2023].

[58] "FARO Focus Laser Scanner S350," Geo-matching, 4 October 2019. [Online]. Available: https://geo-matching.com/products/faro-focus-laser-scanner-s350. [Accessed 23 November 2023].

[59] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter and G. Shakhnarovich, "Diode: A dense indoor and outdoor depth dataset," *arXiv preprint arXiv:1908.00463,* 2019.

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

[61] "VELODYNE HDL-64E," Tree Company Corporation, [Online]. Available: https://www.treecomp.gr/hdl-64e.html. [Accessed 25 November 2023].

[62] "TELEDYNE FLIR (POINT GREY) FLEA 2 FL2-14S3C-C, 1.4 MP, 15 FPS, COLOUR, C-MOUNT,," ebay, [Online]. Available: https://www.ebay.com/itm/144538814299. [Accessed 25 November 2023].

[63] "Samsung Galaxy A13 5G Dual SIM (4GB/64GB) Black," skroutz, 1 July 2023. [Online]. Available: https://www.skroutz.gr/s/37783922/Samsung-Galaxy-A13-5G-Dual-SIM-4GB-64GB-Black.html. [Accessed 27 November 2023].