BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION COMPUTER SCIENCE

DIPLOMA THESIS

Leve: A Lightweight Monocular Depth Estimation Model Exploring Knowledge Distillation via Foundation Models

Supervisor
University Teacher, Dr. Anca Andreica

Author
Drăgan Andrei

2024

ABSTRACT

An important backbone of many computer vision tasks lies in how well depth is estimated from given images or videos, making depth estimation a well-studied topic in recent years. However, besides its importance in many of the current industries such as medicine, augmented reality or self-driving cars, a major interest towards depth estimation was motivated by the latest advancements in deep learning techniques, combined with the improvements of the existing hardware, through which cheaper and faster alternatives can be delivered.

In our work, we focus on building such a solution, by creating *Leve*, a lightweight model capable of producing competitive results against the current literature. We achieve this by implementing various optimization strategies such as leveraging transfer learning for the encoder and relying on skip connections for the decoder, experimenting with depth-wise separable convolutions, and first and foremost, investigating the influence of utilizing knowledge distillation.

Throughout this paper, before diving deeper into defining depth estimation and knowledge distillation, together with the theory behind some different popular associated techniques, we start by presenting a high-level overview of the most popular depth estimation networks and their limitations.

We then continue by introducing our proposed lightweight model, which is based on an encoder-decoder type of architecture, making use of deep-dual resolution networks and a novel and efficient upsampling block. Subsequently, we highlight the three datasets used in our work, following up by explaining the training and evaluation methodology.

Finally, we designate a dedicated chapter for our experiments where we showcase the app used to test our models, the different refinement strategies deployed when building *Leve*, how it stands against the current competition, and more importantly, what generalization and accuracy improvements can be obtained by applying knowledge distillation when relying on the most recently introduced foundation model Depth-Anything. We finish up by drawing our conclusions and suggesting further research that can be done following our work.

Contents

Chapter 1

Introduction

The concept of determining depth from stand-alone images, also known as depth estimation, has been a well-studied topic in the last decade, not because of its sole capabilities, but its crucial role within more complex domains such as tracking and mapping algorithms used in medicine for endoscopy procedures [1], robotic surgeries [2], scene understanding through obstacle detection [3], autonomous navigation [4] for self-driving cars, 3D object detection for augmented reality [5] and many other computer vision tasks ([6], [7], [8]).

While depth can be inferred with the help of stereo cameras or lasers, both of these options are relatively expensive or don't produce the best results. For instance, the Kitti dataset [9], a popular dataset used in computer vision, contains sparse and poorly annotated depth maps, even though they were taken using a costly LIDAR sensor attached to a moving car.

Hence, based on the latest progress made towards deep learning and the higher computational power available nowadays, relying on convolutional neuronal networks (CNNs) has become more appealing. It was proved that such methods are way cheaper than the previous alternatives and produce better results, where a single 2D RGB image can be quickly and accurately translated into a dense depth map, where each pixel within the picture is associated with a numeric value describing the distance from the camera, an approach also known as monocular depth estimation (MDE).

Therefore, thorough research has been put into creating better and more performant models able to deliver impressive results. However, this comes at a cost, since the performance of a network and its architectural complexity are proportional to each other, leading to a higher demand for computational resources. Nonetheless, most domains stated in the first paragraph that could benefit from the advantages of monocular depth estimation don't have access to such capable hardware, resulting in the need to create faster and more lightweight solutions.

While there are still remarkable discoveries targeting the creation of large and complicated models which can deliver high accuracy no matter the context they are put in (i.e. the recently released Depth-Anything [10] which achieved SOTA results), there has been an increasing interest for creating lightweight networks which are faster, yet performant enough to be utilized in the current industries.

In this paper, we introduce *Leve*, an efficient model which delivers compelling results against the existing competition, while exploring different strategies to improve its accuracy without sacrificing performance. Thus, we come up with the following contributions:

- Firstly, we present a novel encoder-decoder architecture, by analysing the influence of several enhancing techniques such as leveraging transfer learning methods through various pre-trained encoders or examining how choosing different skip connections can affect or not the decoding phase. Then, we experiment with multiple options for our decoder's upsampling block based on mixed combinations of convolution operations and choose the best one by considering the quantitative, but also the qualitative results.
- Secondly, we test how our light architecture challenges the existing literature, comparing its balance between accuracy and speed against other lightweight models and more complex pioneer ones.
- Thirdly, we study how applying knowledge distillation techniques relying on a foundation model as a teacher impacts the overall precision of our model and test its generalization potential on two other datasets completely unrelated to the training one.
- Lastly, we make our code public on Github at [11] and provide a simple web application [12] where others can test our model by submitting images which are translated into depth maps in a matter of seconds.

Chapter 2

Related Work

2.1 Monocular Depth Estimation

Generating depth maps based on single RGB images has been initially achieved by relying on probabilistic graphical models and hand-crafted features [13]. However, after deep neural networks proved to be highly efficient in a lot of computer vision tasks, they also gained popularity in the domain of monocular depth estimation (MDE), with Eigen et al. [14] being among the first to propose a supervised deep-learning approach. While there were some attempts to come up with unsupervised solutions [15], there is no doubt that obtaining depth maps by comparing the difference between the pixels' values of the ground truth and the ones of the produced output was the preferred choice.

Therefore, many types of supervised neuronal networks were introduced, one of the most popular being characterised by an encoder-decoder architecture. As the name implies, this type of architecture consists of an encoder comprising multiple layers, in charge of extracting meaningful features from the input image by encoding it towards more compact and abstract representations. In contrast, the decoder aims to reconstruct the encoder's obtained result to its original format, while utilizing the information gathered from the extracted features.

With time, a lot of enhancements were proposed on top of this type of architecture to improve the overall model's accuracy. Farooq et al. introduced in [16] the idea of dividing the predicted depth range into multiple adaptive bins, the pixels in the depth map having a value influenced by a linear combination of the bin middle figures, while Yan et al. suggested in [17] a self-attention mechanism inspired by the well-known paper "Attention is all you need" [18]. Another widespread improvement attributable to transfer learning techniques was the replacement of the classical encoder with a truncated pre-trained one from other computer vision tasks such as image classification or image segmentation, with the likes of Ibraheem et al. obtaining remarkable results in [19] using an unmodified DenseNet-169 encoder [20].

Gradually, with each new refinement the accuracy of the models increased, yet one major drawback was that most of them performed well only in the context of the dataset they were trained on, hence the desire to build a foundation model.

2.2 Foundation Models

Foundation models are Artificial Intelligence networks trained on a broad range of data, which can be used in various downstream tasks [21]. They became highly popular in the last couple of years, due to their ability to showcase great performance no matter the context they are faced with. However, while the world has seen huge gains in the creation of a variety of capable foundation models in the domain of natural language processing (NLP) [22], such as BERT, or the well-known ChatGPT, little progress has been made in the field of computer vision, especially in the case of monocular depth estimation.

As stated by the creators of MiDaS [23], one of the first monocular depth estimation foundation models, a big challenge is caused by the lack of enough labelled datasets that can complement each other and cover the diversity of the visual world. When building MiDaS, Ranft et al. tried to overcome these issues by training their network on five complementary datasets and testing its zero-shot capability on unseen ones. While MiDaS has become a state-of-the-art model in many contexts, it still performs extremely poorly in some scenarios.

Such limitations were recently addressed by Yang et al. when introducing Depth-Anything [10], a new foundation model for monocular depth estimation, capable of handling any image in any context provided. To achieve this, they designed a model trained on 1.5M labelled images to constitute the basis of a data engine responsible for collecting and automatically labelling around 62M new images, which were used to self-train the initial model.

2.3 Lightweight Models

Although these foundation models focus on improving the overall accuracy, they require huge computational power and high inference speeds, making them an unsuitable solution for small mobile embedded devices crucial in today's industries such as aerial navigation [24], augmented reality [25] or medicine [1].

That's why the need to develop fast depth estimation models emerged, resulting in the creation of a multitude of techniques to build lightweight architectures. In line with the trend of facilitating transfer learning, Wofk et al. introduced FastDepth

[26], a CNN network which utilizes the efficient MobileNets [27] as a pre-trained encoder, in combination with pruning strategies to further improve the efficiency of the model. Michael et al. proposed in [28] a method for guiding the upsampling process of the decoder with the help of the input image, while Dong et al. suggested in [29] a dual encoder-decoder architecture based on skip connections, addressing the issue of popular pre-trained encoders that downsample the image resolution too much resulting in loss of spatial information.

2.4 Knowledge Distillation

While all of the methods mentioned above obtained a good balance between accuracy and efficiency, they focus only on the lightweight model. We believe that by leveraging knowledge distillation, we can obtain better accuracy and generalization capabilities without sacrificing the complexity and inference speed of the deployed architecture.

Even though Junjie et al. delivered promising results in [30] adopting such a technique, to our understanding, there is not much research in regards to using knowledge distillation, particularly in the domain of monocular depth estimation and especially using the capabilities and advantages offered to us by the current foundation models.

Chapter 3

Monocular Depth Estimation

Fundamentally, given a series of images, depth estimation aims to correlate every pixel from within each image, with a numeric value representing the relative or sometimes the absolute distance from the camera to the objects inside the scene, resulting in a better understanding of the environment.

While previous approaches relied on multiple shots with the same image taken from different angles to fully comprehend the spatial surroundings, today's advancements made deep learning techniques able to determine the depth just from a single 2D RGB image, resulting in cheaper, yet accurate enough solutions, approach also known as monocular depth estimation. However, when employing such methods, one has to be careful regarding the data the network is trained on, with supervised learning still being the most efficient strategy when building depth estimation models. Fortunately, there are a lot of datasets available nowadays that provide pairs of RGB images and their annotated depth maps, including some pioneer ones such as NYU-Depth V2 [31] and Kitti [9], which promote a continuous competition between hundreds of architectures, state-of-the-art models still being developed every year as shown in [32] and [33] where an up-to-date ranking with the most precise models for each dataset is showcased. These rankings are built by taking into account a series of quantitative metrics explained in more detail in chapter 8, most of them basing their formula on the difference between the pixels' values of the predicted depth map and its equivalent ground truth.

3.1 Encoder-Decoder Architecture

This type of architecture has seen great success in the last couple of years in multiple computer vision tasks such as image segmentation or image classification, and more recently, in the existing works from within the field of monocular depth estimation ([19], [26], [29], [28]).

The first half of an encoder-decoder architecture is characterised by an encoder responsible for learning and extracting as much information as possible from the input image by reducing its resolution and increasing the number of features, each retaining certain details about the original picture.

The final output of the encoder is then taken through a series of upsampling blocks which make up the decoder, in charge of rebuilding the image to its original or desired shape, by utilising the information condensed through the encoding phase.

3.2 Transfer Learning

While the encoder can be designed and trained from scratch, using transfer learning techniques and counting on an already pre-trained encoder from other domains has proved more efficient, MobileNets [27] becoming a common choice for fast monocular depth estimation, thanks to their competitive accuracy and high inference speeds. In addition, they gained even more popularity with the introduction of the more performant versions: MobileNetV2 [34] and MobileNetV3 [35] which increased the overall accuracy and efficiency.

However, to fully maximize MobileNets, one has to use most of its layers from the encoding phase, which reduces the input image to 1/32 of its original size. Nevertheless, this comes at a cost, which was also pointed out by Dong et al. in [29]: it's important to consider how much the image is compressed because even though better details are extracted the lower the resolution is, big reductions produce heavy losses of spatial information.

To combat such limitations, we further explore other types of pre-trained encoders, shifting our attention from image classification tasks to image segmentation, specifically to deep dual-resolution networks (DDRNets) [36], which present a lightweight solution having as a backbone a dual architecture. The way they work is that instead of relying on a single information flow, they impose two parallel branches: one that downsamples the image to at most 1/8 of its original size, and another one that extracts more valuable information at lower resolutions; both branches keeping a constant connection between them through multiple bilateral fusions. By adopting the lightest DDRNet version (DDRNet-23-Slim), Rudolph et al. managed to find in [28] a good balance between accuracy and performance.

3.3 Skip Connections

Counting on skip connections is another popular strategy used to improve the accuracy of a model. Essentially, instead of relying on a single straightforward flow, which is prone to forgetting prior information the deeper a network is, output feature maps from preceding layers are reinstated through additions or different fusion techniques. This idea was supported especially by Huang et al. in [20] which obtained significantly better results when utilising a dense set of skip connections, where all layers were connected between each other. However, it is important to note that in general, such connections are made between layers that work with feature maps having similar shapes.

In the case of monocular depth estimation, a common approach is to employ skip connections between layers from within the encoder and their associated decoding blocks from a size perspective, using the encoded feature maps to better guide the upsampling process.

3.4 Depth-Wise Separable Convolutions

Finally, a well-known strategy that usually improves the inference speed which we are going to explore as well in our paper, is to utilize depth-wise separable convolutions. The mechanism behind these is that the single classical convolution operation is split into a depth-wise convolution followed by a point-wise convolution, a combination which decreases the overall number of multiplications performed, such technique gaining a lot of popularity in many computer vision and monocular depth estimation tasks ([34], [29]).

In general, for a normal convolution operation, if we have a tensor with a size of $[H \times W]$ pixels and F_1 number of features that we want to translate into a tensor with the same resolution but with F_2 number of features, we would use a set consisting of F_2 kernels having the following shape: $[K \times K \times F_1]$. Each such kernel would then shift through the whole width and height of the input tensor and perform $F_1 \cdot K^2$ multiplications at every step, producing an overall complexity of $H \cdot W \cdot F_1 \cdot K^2$ multiplications for a single kernel and $H \cdot W \cdot F1 \cdot F2 \cdot K^2$ for the whole convolution.

In contrast, a depth-wise separable convolution will first treat separately every feature channel from the input tensor, each one being multiplied against a $[K \times K \times 1]$ kernel, resulting in an intermediate output with the same shape as the input tensor at the cost of $H \cdot W \cdot F1 \cdot K^2$ multiplications. Then, to modify the number of features from F_1 to F_2, this intermediate output goes through a classical 1×1 convolution with a complexity of $H \cdot W \cdot F_1 \cdot F_2 \cdot 1^2$ multiplications, leading to a total number of $H \cdot W \cdot F_1 \cdot (K^2 + F_2)$ multiplications, thus making the depth-wise separable convolution faster by a factor of $\frac{K^2 * F_2}{K^2 + F_2}$ then the regular one. For a better understanding, Fig. 3.1 illustrates a full depth-wise separable convolution operation.

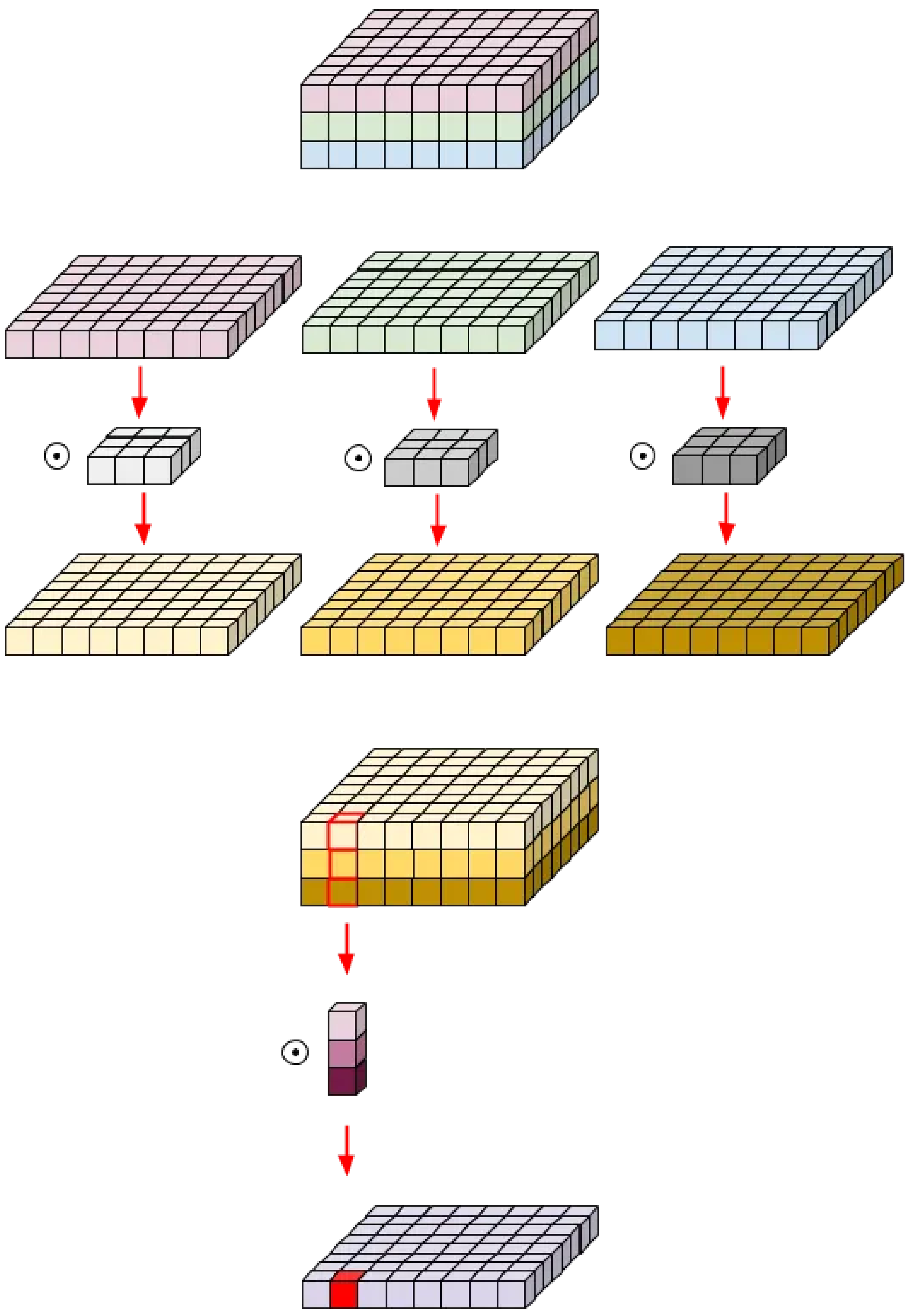

Figure 3.1: A depth-wise separable convolution operation.

Figure taken from [37]

Chapter 4

Knowledge Distillation

We believe it's worth dedicating an entire chapter to this part due to the extensive experiments that we perform in our work leveraging such a technique and the interesting observations that we highlight by adding knowledge distillation to our lightweight model.

In essence, knowledge distillation is the process in which, during training, information is transferred to a smaller model named the student from a bigger and more complex one known as the teacher. As stated in [38] there are three main types of knowledge distillation: response-based knowledge distillation where only the last layers of the teacher matter, feature-based knowledge distillation where alongside the final layers, intermediate layers are also taken into consideration, and relation-based knowledge distillation where besides the sole results of different layers from within the teacher, the relation between the produced feature maps is considered.

By employing knowledge distillation methods, the student model learns to mimic the behaviour of its teacher, resulting in better accuracy without sacrificing inference speed. This can be achieved by altering the way the loss function is computed during training, such that in addition to the error between the result of the student and the associated ground truth, the one between the student's output and the teacher's output is also taken into consideration, the output of the teacher being usually known as *knowledge*.

Motivated by the words of Beyer et al. in [39] which affirm that "a good teacher is patient and consistent", we wanted to rely on an already pre-trained network capable of producing outstanding results no matter the circumstances it faces, hence we started looking at foundation models in the field of monocular depth estimation. Moreover, since our proposed model is trained on NYU-Depth V2 [7], we also aimed for our teacher to deliver state-of-the-art performance on the same dataset.

This is the reason why Depth-Anything [10] captivated our attention. Thanks to their ingenious data engine possessing the capacity to gather and annotate unlabelled data automatically, Depth-Anything is trained on about 62M images taken from a comprehensive range of environments making it one of the most powerful models that can produce high-accuracy depth maps on any image no matter the context, outperforming MiDaS [23], another significant foundation model in the domain of monocular depth estimation. Furthermore, as mentioned in [32], in April 2024, Depth-Anything occupies second place regarding quantitative metrics on evaluation criteria against the NYU-Depth V2 dataset.

While Hu et al. produced significant improvements in [30] by relying on a teacher with a similar architecture as the student and employing different training techniques such as leveraging auxiliary labelled and unlabelled data, our approach is slightly different. Inspired by Han et al. in [1] which proved that foundation models produce similar outcomes against fine-tuned models, we focus on how using a teacher such as Depth-Anything may influence the generalization abilities of the student model. Moreover, since Depth-Anything has a more complex architecture than the one in our lightweight model, we will resort to sharing only response-based knowledge as shown in Fig. 4.1.

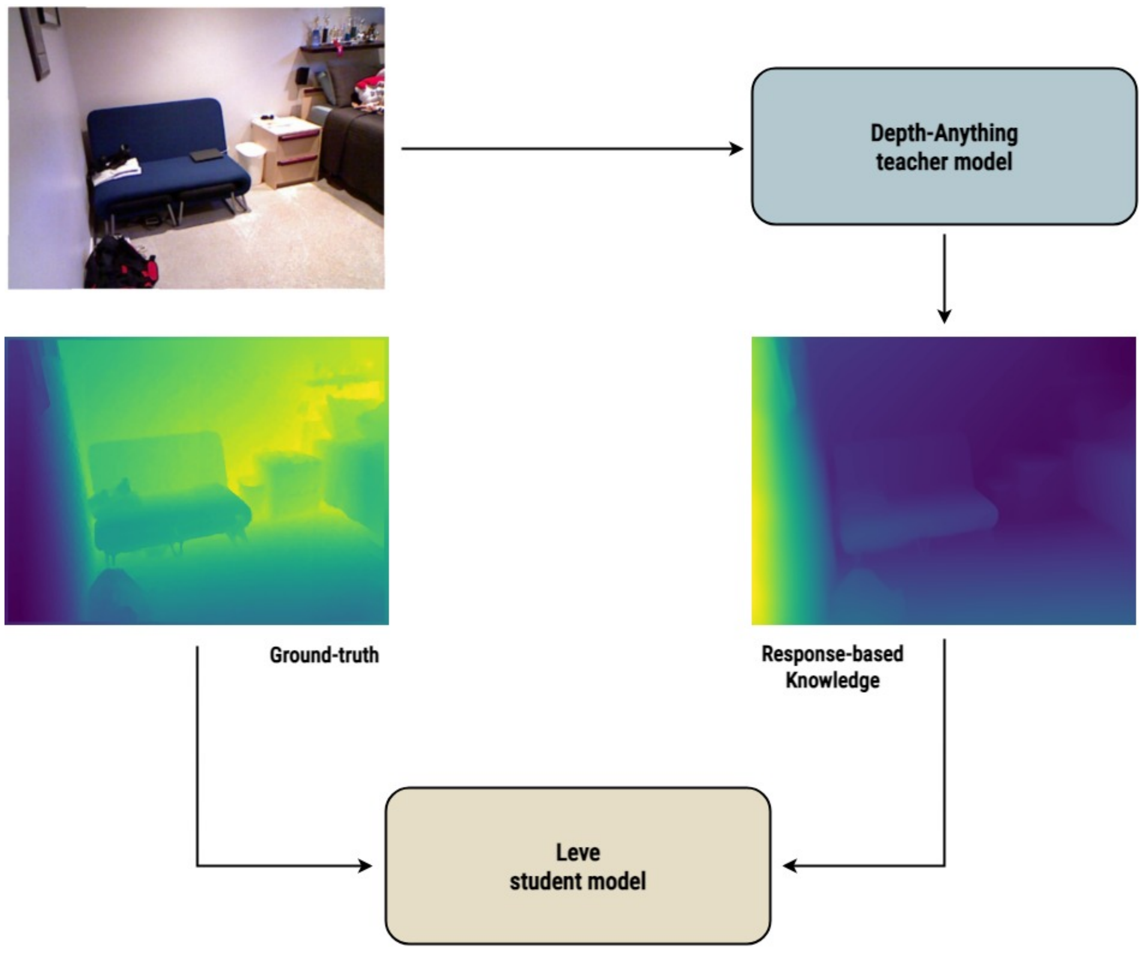

Figure 4.1: The proposed teacher-student architecture which leverages knowledge distillation via Depth-Anything as a foundation teacher model.
Figure generated with draw.io [40]

Chapter 5

Proposed Method

In this chapter, we present the novel building design of our proposed network called *Leve*, which means *light* in Latin, capable of producing competitive results against state-of-the-art alternatives, by leveraging multiple techniques that can improve the overall model's predictions, while not sacrificing its performance.

As stated in chapter 3, motivated by all the accomplishments it had in the field of monocular depth estimation, an encoder-decoder type of architecture is also used as the base of our lightweight model, making use of the advantages offered by different strategies such as transfer learning techniques or relying on skip connections from the encoder's intermediate layers to act as guiding points throughout the decoding phase. An illustration of the proposed architecture can be observed in Fig. 5.1.

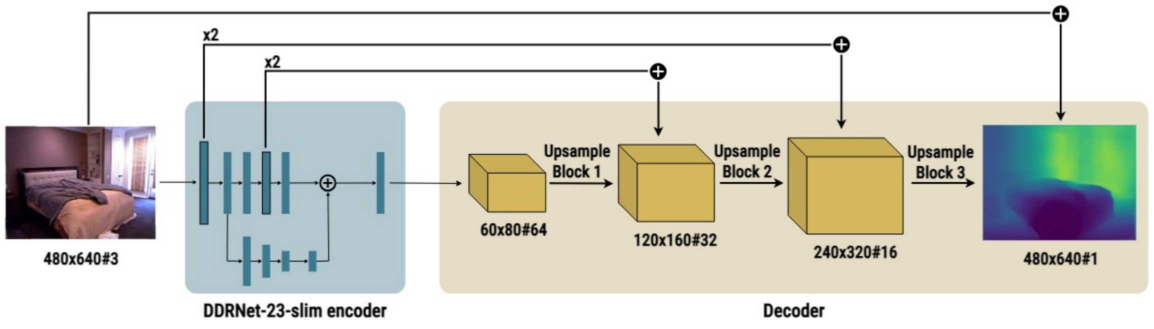

Figure 5.1: The proposed architecture for *Leve*.
Figure generated with draw.io [40]

While for the encoding component, we adopt the pre-trained and slightly modified DDRNet-23-Slim encoder from [28], our decoder is based on a novel, yet straightforward and lightweight upsampling block (Fig. 5.2), which aims to deliver compelling results, without amplifying too much the overall complexity of the whole network.

Each upsampling block consists of two branches. The first one takes as input the network's main tensor obtained from the original image, scales it up by a factor of 2 through bilinear interpolation and sends it through a **layered convolution block (LCB)**. On the second branch, if necessary, the tensor obtained from the associated skip connection is resized to match the desired output resolution through the same bilinear interpolation technique, after which is sent through a simple 1×1 convolution operation that adjusts its number of features. Ultimately, the post-LCB and pre-LCB tensors from the first branch are summed up together with the altered tensor obtained from the second branch, and the whole result goes through one more final 1×1 convolution operation.

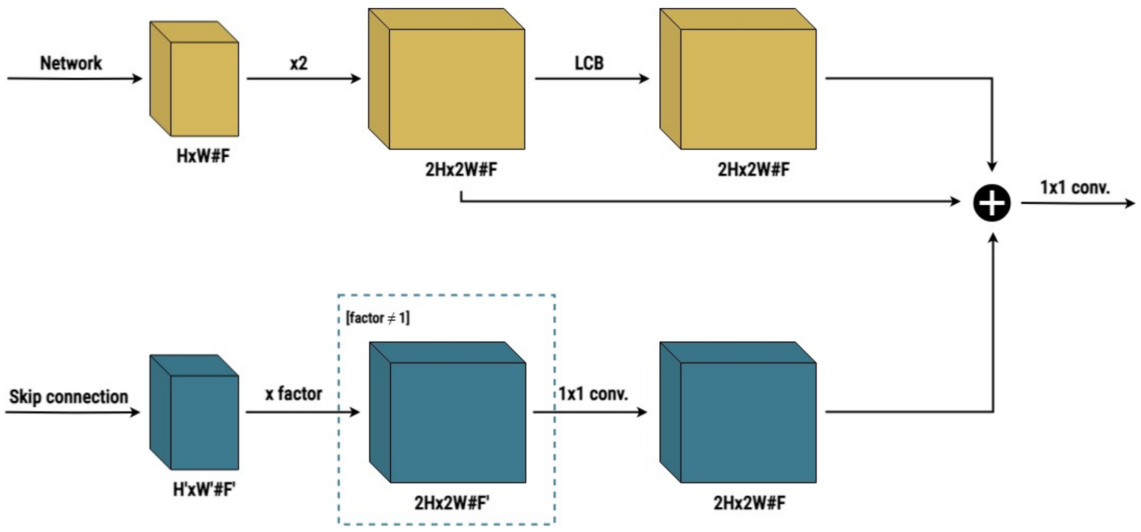

Figure 5.2: The proposed high-level architecture of the upsampling block.
Figure generated with draw.io [40]

As one of the key points for our decoder, we tested a lot with how to design the layered convolution block (**LCB**), since we wanted to keep it lightweight, yet performant enough to be able to decode as much information as possible. While we didn't change the fact that each convolution operation is followed by a batch normalization and a ReLU activation function [41], since such a technique has proved to be the most efficient in the current literature, we experimented with the sizes and the number of the actual convolution operations within the **LCB**, leading us to conclude that the best candidate for our **LCB** is constituted by two 3×3 convolutions one after each other, whose architecture can be seen in Fig 5.3.

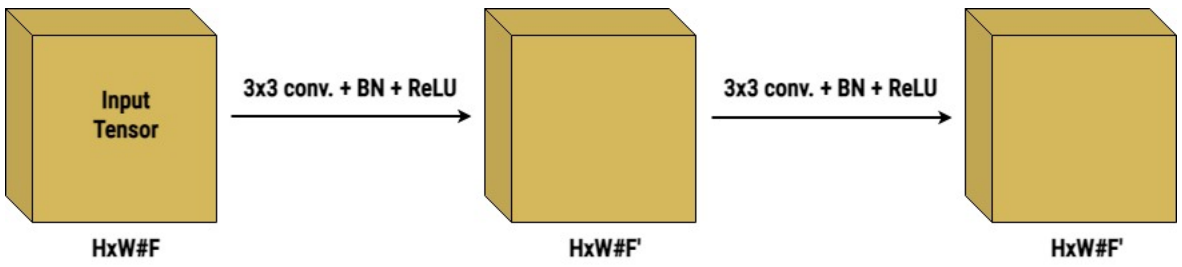

Figure 5.3: The chosen architecture for the layered convolution block (**LCB**).
Figure generated with draw.io [40]

Chapter 6

Datasets Used

Aiming to create a lightweight student model that can deliver high accuracy and fast inference times against the existing works, we train our network on a popular dataset used in the domain of monocular depth estimation. However, to further test the impact of knowledge distillation techniques and the generalization influence a teacher foundation model can have on the student, we search for other datasets unrelated to the first one, resulting in the following selection:

6.1 NYU-Depth V2

NYU-Depth V2 dataset [31] was released around 2012 and it has become a pioneer in the field of monocular depth estimation ([42], [43], [44]) and semantic segmentation ([45], [46], [47]).

It contains a variety of indoor video sequences recorded through a Microsoft Kinect camera at a resolution of 640×480 pixels from an RGB and Depth perspective, covering residential and commercial buildings from 3 different US cities. As stated by Ibraheem et al. in [19], it sums up to 120k training images paired with their corresponding densely depth maps and 694 similar samples for testing.

However, to speed up the training process, we use a reduced subset introduced by Ibraheem and Wonka in [19], including only 50.688 samples out of the 120k labelled ones. For the testing part, we follow the same procedure as Rudolph et al. in [28], which relies on the testing split presented by Eigen et al. in [14] containing 694 RGB images and their related densely filled depth maps as ground truths.

One thing to mention is that when processing the training dataset, to obtain consistent results in meters, the pixel values comprising the validation depth maps should be multiplied by 10, while the ones from the training depth maps need to be divided by 1000. Moreover, another detail important to our method is that the maximum depth of the NYU-Depth V2 dataset is 10 meters.

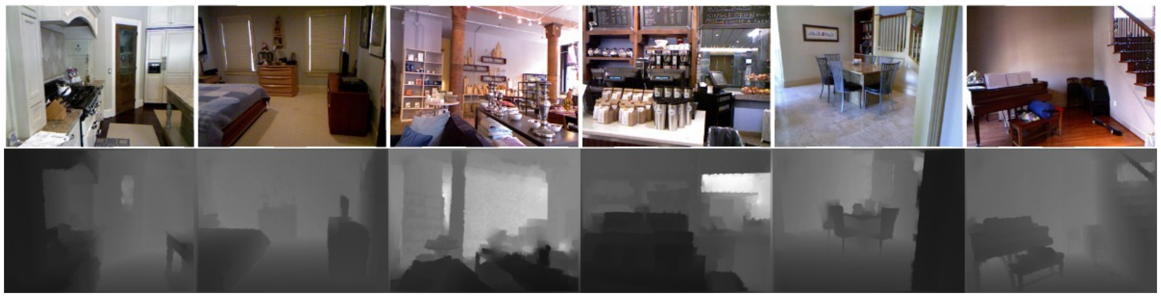
Figure 6.1: Images taken from the NYU-Depth V2 dataset and their associated ground truth depth maps.
Figure generated with draw.io [40]

6.2 Kitti

Kitti [9] is another popular dataset for object detection tasks, released around the same time as NYU-Depth V2. It contains pairs of outdoor scenes captured by a camera attached to a moving vehicle and their related depth maps produced by laser scans with the help of a LIDAR sensor. However, due to the sparse nature of the ground truths, different colourization techniques are required to fill in the blank pixels, such as the one proposed by Levin et al in [48]. Over the years, the Kitti dataset was divided into multiple reduced and modified versions, the favoured one in the field of monocular depth estimation ([49], [50], [42]) being the one proposed in [14].

In our approach, we use Kitti only for testing, so similarly to how we did in the case of the NYU-Dpeth V2, we follow the same testing procedure as Rudolph et al. in [28], relying on the Kitti testing split introduced by Eigen et al. in [14] which consists of 697 image-depth pairs.

It's important to note that the images from the Kitti dataset have a resolution of 1241×376 pixels and a maximum depth of 80 meters.

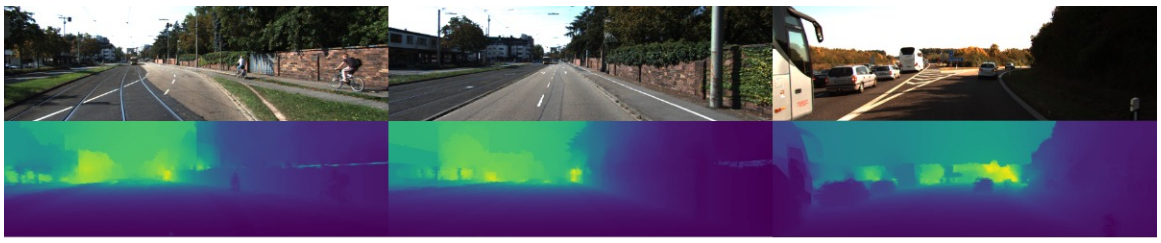
Figure 6.2: Images taken from the Kitti testing split introduced by Eigen et al. in [14] and their associated ground truth depth maps.
Figure generated with draw.io [40]

6.3 EndoSLAM

The EndoSLAM dataset [51] gained huge popularity in the last couple of years as a solution for monocular depth estimation in endoscopic videos, due to the difficulty of creating such types of datasets in the medical field. It contains both endoscopy recordings and synthetic data, being divided into multiple datasets that target the colon, the small intestine and the stomach.

For our work, similarly to Kitti, we rely on EndoSLAM only for testing purposes to check the generalization capabilities of knowledge distillation techniques. Thus, we use a subset of around 36k samples that contains pairs of RGB images and associated ground truth maps covering all three organs, from which we extract 200 images targeting each organ, summing up to a total of 600 images.

The images from the EndoSLAM dataset have a resolution of 320×320 pixels. Since no maximum depth is specified, we empirically choose it to be 1 meter.

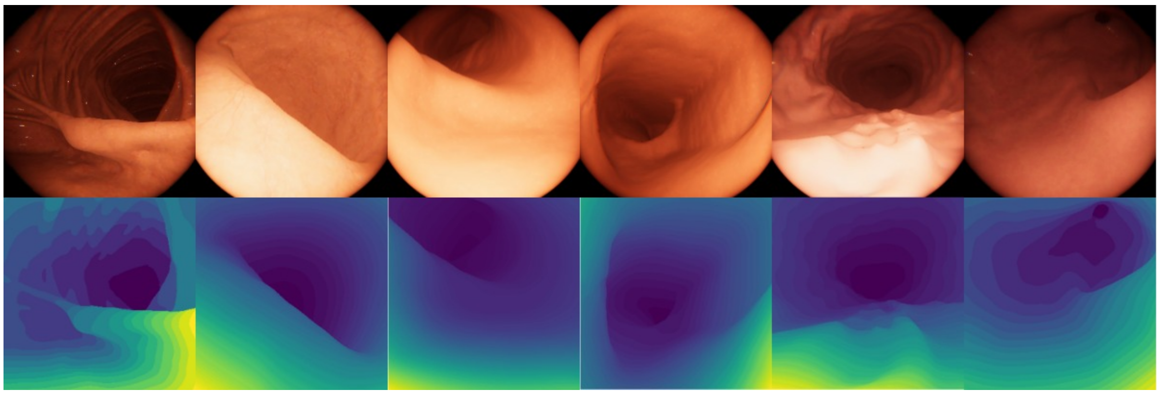

Figure 6.3: Images taken from the custom EndoSLAM dataset and their associated ground truth depth maps.

Figure generated with draw.io [40]

Chapter 7

Training

Our training approach is very similar to the one introduced by Ibraheem and Wonka [19], which proved to perform successfully not only in [19], but in other related works as well ([28]).

7.1 Training Details

We train our models on Google Colab's V100 GPU [52] for 20 epochs having a batch of size 8. As an optimizer, we use Adam [53], with a learning rate of 0.0001, which decreases by 1% every 5 epochs.

We believe that by reducing the interferences with the original dataset through resizing or cropping the input images, we can deliver higher accuracy. Thus, our main and most performant lightweight model which we compare against the existing literature is trained on the original resolution of the NYU-Depth V2 dataset of 480×640 pixels, requiring around 11 hours of training for the sole lightweight model and around 40 hours for the full architecture where knowledge distillation is applied. However, for faster training times, when doing experiments between different versions of our proposed models we decrease the resolution of each sample to half the original size (240×320) using bilinear interpolation.

During training, within an epoch, every 1000 samples, we record the validation loss and the quantitative metrics presented in chapter 8, resulting in seven such recordings per epoch, which are stored using Weight & Biases [54]. We save checkpoints every epoch and choose our best model as being the checkpoint which delivered the lowest root-mean-squared error on the validation subset.

7.2 Data Augmentation

To enhance the generalization performance and diminish over-fitting, in all our training sessions we apply two types of data augmentation which proved to deliver significant improvements in [19] and [28].

In general, a popular strategy is to alter the orientation of the input image by rotating it to a certain degree or flipping it vertically or horizontally. However, due to the unique nature of monocular depth estimation, switching the floors with the ceilings through a vertical flip or rotating the image might worsen the learning process. Thus, we only apply a random horizontal flip with a probability of 50%, accompanied by the trivial random colour channel swap with a similar probability.

7.3 Loss Function

To better understand the formulas used when computing our final loss function, we denote the following notations:

- $target$ = the densely annotated ground truth map
- $output$ = the depth map generated by our proposed lightweight model
- $knowledge$ = the depth map produced by Depth-Anything and shared with the student in the process of knowledge distillation

Since we rely on knowledge distillation, our final loss function is based on two main components: the loss function of the student ($L_{student}$) and the loss function of the teacher ($L_{teacher}$), the balance between them being dictated by a factor of ϵ.

$$\begin{aligned} L(output, target, knowledge) = (1-\epsilon) * L_{student}(output, target) \\ + \epsilon * L_{knowledge}(output, knowledge) \end{aligned} \tag{7.1}$$

7.3.1 The Student Model

While there are various versions of loss terms employed in the domain of monocular depth estimation, we settled down to the ones which are widely used in the existing literature and proved to perform well. Hence, we present the following three loss functions:

1. The most classical one computes the average absolute difference between each pixel value from both depth maps. This is also known as the pixel-wise L1 loss [19] and is defined as follows:

$$L_{depth}(output, target) = \frac{\sum_p^P |output_p - target_p|}{P} \tag{7.2}$$

   where P denotes the total number of pixels from the image, and p represents a single pixel.

2. Following a similar structure, the gradient loss [19] computes the average absolute difference between the O_x and O_y components of each pixel value from both depth maps, resulting in sharper and clearer edges around the objects.

$$L_{grad}(output, target) = \frac{\sum_p^P |g_x(output_p, target_p)| + |g_y(output_p, target_p)|}{P} \tag{7.3}$$

$$g_x(output_p, target_p) = g_x(output_p) - g_x(target_p) \tag{7.4}$$

$$g_y(output_p, target_p) = g_y(output_p) - g_y(target_p) \tag{7.5}$$

where g_x and g_y represent the partial derivative of a pixel p with respect to the O_x and O_y axes.

3. The last one that proved to be a good loss term in image reconstruction and depth estimation tasks is based on the Structural Similarity term (SSIM) [55] and has the following form:

$$L_{SSIM}(output, target) = \frac{1 - SSIM(output, target)}{2}. \tag{7.6}$$

It's also worth noting that with larger values of the pixels from within the ground truth depth maps, those loss terms also become bigger. Thus, a normalization of the ground truth is necessary. As found by [19], using the inverse of the depth seems to solve this problem, meaning that the ground truth depth map $target$ becomes $target = max\_depth/target$, where max_depth is the maximum possible depth of the whole dataset. Moreover, to minimize the significant impact of L_{depth}, Ibraheem et al. [19] found empirically that multiplying it with a coefficient of 0.1 produces the best results, hence we adopt a similar approach. Finally, by combining everything we've presented above, the overall loss term used when training our lightweight model against the ground truth maps has the following form:

$$L_{student}(output, target) = 0.1 * L_{depth} + L_{grad} + L_{SSIM} \tag{7.7}$$

7.3.2 The Teacher Model

While we could've adopted the same loss function when taking into account the loss between the *output* and the *knowledge* as Hu et al. did in their work [30], since there is some discrepancy between the architectures of our student and the Depth-Anything teacher, together with the fact that the pixels of the ground truths represent absolute metric values, while Depth-Anything provides only relative depth estimation, we first normalize the *output* and the *knowledge* by dividing each map with the maximum value of its pixels, thus ensuring pixel values that are only between 0 and 1, and then apply a trivial pixel-wise L1 loss as follows:

$$L_{teacher}(output, knowledge) = \frac{\sum_{p}^{P} |outputN_p - knowledgeN_p|}{P} \tag{7.8}$$

where P denotes the total number of pixels from the image, p represents a single pixel and $outputN$ and $knowledgeN$ are the normalized versions of $output$ and $knowledge$ respectively.

Finally, we observed that the teacher model's knowledge doesn't require applying the inverse of the depth previously mentioned, since Depth-Anything already produces depth maps where the value of a pixel is inversely proportional to its distance from the camera.

Chapter 8

Evaluation

For testing, we use the NYU-Depth V2 and Kitti test splits introduced by [14], together with the EndoSLAM's custom dataset presented in chapter 6. However, it's important to consider that each dataset has a unique resolution associated with its images and ground truths. To tackle this issue and guarantee a fair comparison between our solutions and those existing in the literature, before an input image is sent through a model, it is resized to the resolution the model was originally trained on. While a popular strategy in such a case is to randomly crop the image to the desired size, this can lead to major inconsistencies in the evaluation, as highlighted by [28], motivating us to rescale the image through bilinear interpolation. Nonetheless, by doing this, the output will have the same rescaled resolution, so it's worth noting that before evaluating it against the ground truth we reshape it back to the original size using the same technique.

Another thing that we should take into consideration as mentioned by Eigen et al. in [14] is defined by the inconsistencies that can occur at the borders of an image. This is why, when evaluating the output against its ground truth, based on each dataset, we account only for the following pixels' ranges: [$(24 : 616) \times (20 : 460)$] for NYU-Depth V2, [$(45 : 1196) \times (128 : 381)$] for Kitti and [$(0 : 320) \times (0 : 320)$] for EndoSLAM.

Additionally, inspired by [19], for more rigorous evaluation metrics, we compute the average between the results produced by the output of the original image, combined with the ones obtained by the output of the vertically flipped input image.

8.1 Quantitative Metrics

We present three popular quantitative metrics used in prior works [14], which we are going to employ when evaluating our models:

1. The first one is the **Root Mean Squared Error (RMSE)** which quantifies the average magnitude of the mean squared pixel-wise difference between the out-

put and the target ground truth and has the following formula:

$$RMSE = \sqrt{\frac{\sum_{output,target}^{N}(\frac{1}{P} * \sum_{p}^{P}(output_p - target_p)^2)}{N}} \tag{8.1}$$

where P denotes the total number of pixels from the image, p represents a single pixel, and N is the total number of samples of type $(target, output)$ from the dataset.

2. The second one is the **Mean Absolute Relative Error (REL)** which computes the average between the mean relative pixel-wise absolute difference between the output and the target ground truth and has the following formula:

$$REL = \frac{\sum_{output,target}^{N}(\frac{1}{P} * \sum_{p}^{P} \frac{|(output_p - target_p)|}{target_p})}{N} \tag{8.2}$$

3. The third and last one is called the **Threshold Accuracy (δ)**, which computes the average percentage of pixel values between the output and the ground truth having a ratio within the threshold δ. In general, some popular values for δ are 1.25, $(1.25)^2$, $(1.25)^3$.

$$\Delta = \frac{\sum_{output,target}^{N}(\frac{1}{P} * \sum_{p}^{P} threshold(output_p, target_p, \delta))}{N} \tag{8.3}$$

$$\text{threshold}(x, y, \delta) = \begin{cases} 1 & \text{if } \max\left(\frac{x}{y}, \frac{y}{x}\right) < \delta \\ 0 & \text{otherwise} \end{cases}$$

8.2 Inference Testing

The last important bit of our evaluation methodology is related to how we test the inference speed and the number of frames per second (FPS) for each model. Since Google Colab's environment provides a lot of inconsistency regarding such tasks, we test each model on the local GPU (NVIDIA GeForce GTX 1650 Ti with 4GB of dedicated memory), by sending through the network a randomly generated tensor of 640×480 pixels and measuring the time it takes to produce an output. We do this operation around 200 times per test and we perform several such tests to finally select a round-up metric value towards each model converges.

Chapter 9

Experiments & Results

9.1 Building Leve

In this section, we present all the meaningful experiments that we have performed and their associated results that led us to different decisions towards building the final version of *Leve*, which is described in more detail in chapter 5. In the end, we also showcase how the best version of *Leve* stands as a competitive candidate against not only state-of-the-art lightweight models, but also against some more complex pioneer depth estimation networks.

9.1.1 The Encoder

As stated in chapter 5, we maximize the advantages of transfer learning strategies and adopt the pre-trained DDRNet-23-Slim encoder from [28]. Although we have already presented some reasons why we avoid the widely-used MobileNets, we still wanted to test DDRNet-23-Slim against MobileNetV3, the most performant MobileNet so far. As shown in Fig. 9.1 and Table 9.1, although the MobileNetV3-based architecture is faster, it lacks a lot in quantitative metrics, making DDRNet-23-Slim the preferred option for a better balance between performance and accuracy.

Encoder	RMSE	REL	δ_1	δ_2	δ_3	Inference Speed	FPS
MobileNetV3	0.671	0.206	0.681	0.914	0.976	0.023 seconds	43
DDRNet-23-Slim	0.516	0.142	0.814	0.958	0.988	0.037 seconds	27

Lower values are better	Higher values are better

Table 9.1: A quantitative comparison between the two *Leve* models using the pre-trained MobileNetV3 encoder and DDRNet-23-Slim respectively.

The layout and the format of the table are inspired by [29]

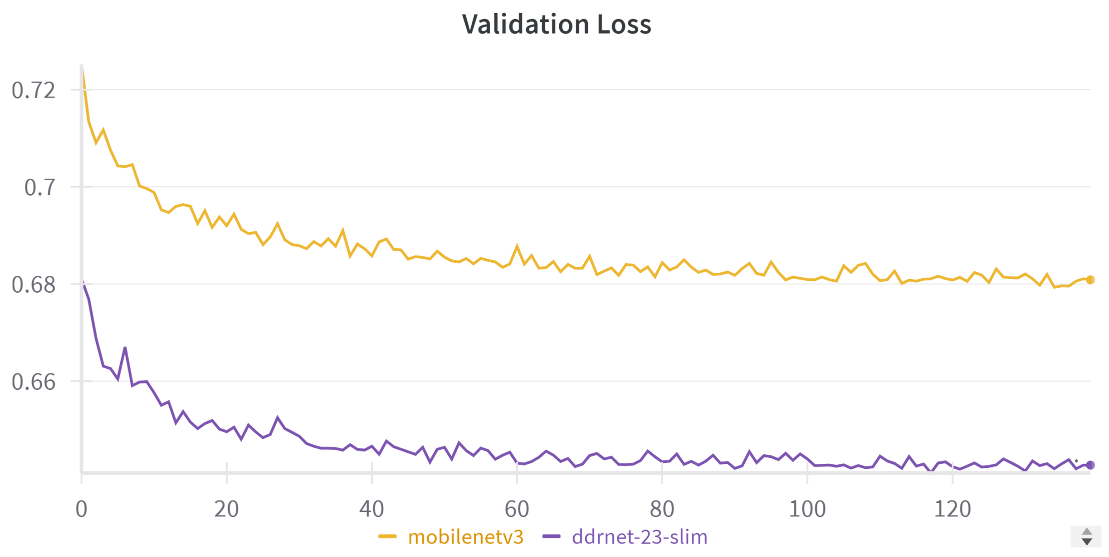

Figure 9.1: The evolution of the validation loss for the two *Leve* models relying on MobileNetV3 as a pre-trained encoder and DDRNet-23-Slim respectively.

Figure generated from Weights & Biases [54]

9.1.2 The Decoder

For our decoder, we experimented a lot with what types and how many convolutions should make up the layered convolution block (**LCB**). In the end, we settled down to the following 4 versions:

- Only one 3x3 convolution [*3x3*]
- A 3x3 convolution, followed by a 1x1 convolution [*3x3+1x1*]
- Two 3x3 convolutions [*3x3+3x3*]
- Two 5x5 convolutions [*5x5+5x5*]

To choose the most suitable one, we performed a head-to-head comparison between them from a quantitative, but also a qualitative point of view. As we can observe in Table 9.2, while the [*5x5+5x5*] delivers marginally better quantitative results, it's significantly slower than the others forcing us to cross it out from our consideration. This leaves us with the other 3 alternatives: *[3x3+1x1]*, *[3x3+3x3]* and *[3x3]*, which based on only their evaluation metrics, produce highly similar outcomes, similarity that can also be noticed by looking at the development of their validation loss during training (Fig. 9.3). However, we believe it's worth considering the qualitative results as well shown in Fig. 9.2, and analysing fine details such as background objects and their edges. In the end, based on all of these, we concluded that *[3x3+3x3]* is the best option for our layered convolution block.

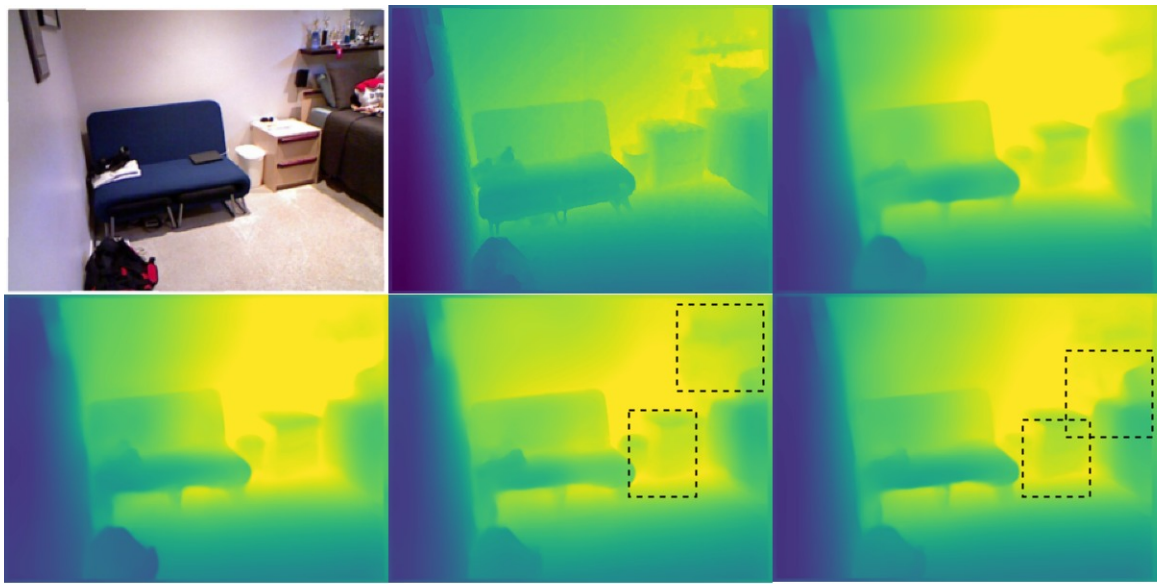

Figure 9.2: A qualitative comparison between the *Leve* models that use the suggested layered convolution blocks (**LCBs**). From left to right, top to bottom, we have the original image, its ground truth and the outputs of [*3x3*], [*3x3+1x1*], [*3x3+3x3*] and [*5x5+5x5*] in this order. As we can see, the last two images ([*3x3+3x3*] and [*5x5+5x5*]) capture the most details over the background objects and the objects' edges.

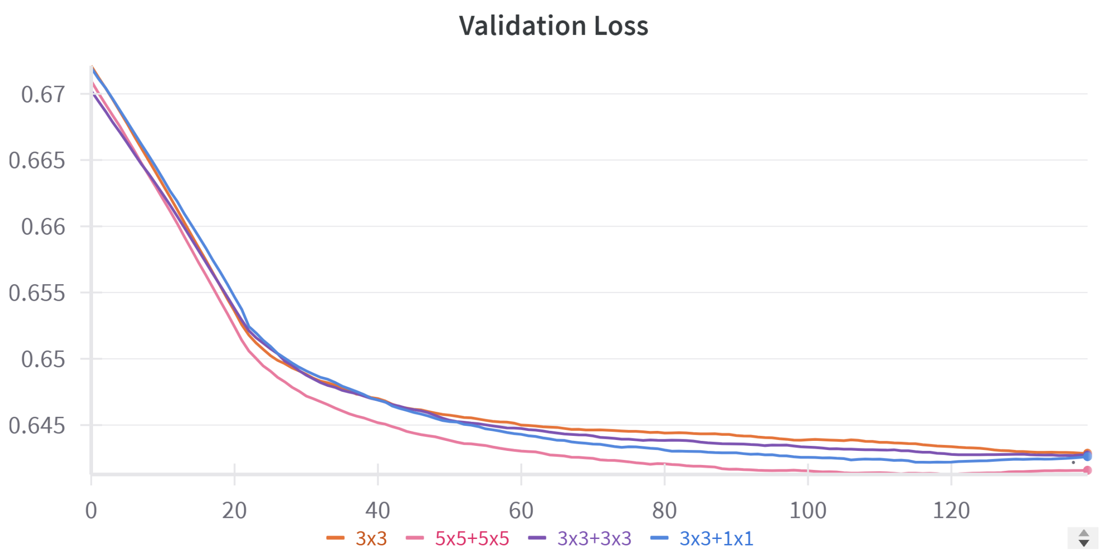

Figure 9.3: The evolution of the validation loss for the *Leve* models that use the suggested layered convolution blocks (**LCBs**) over the 20 epochs of training.

Figure generated from Weights & Biases [54]

Layered Convolution Block	RMSE	REL	δ_1	δ_2	δ_3	Inference Speed	FPS
3x3	0.515	0.143	0.817	0.957	0.988	0.030 seconds	34
3x3+1x1	0.515	0.143	0.818	0.959	0.988	0.033 seconds	29.5
3x3+3x3	0.516	0.142	0.814	0.958	0.988	0.037 seconds	27
5x5+5x5	0.514	0.140	0.816	0.959	0.989	0.055 seconds	17.5

Lower values are better	Higher values are better

Table 9.2: A quantitative comparison between the *Leve* models that use the suggested layered convolution blocks (**LCBs**).

The layout and the format of the table are inspired by [29]

9.1.3 Depth-Wise Separable Convolutions

We also studied how the overall inference speed improves by utilising depth-wise separable convolutions. To explore this idea in our approach, we relied on the previously mentioned [*3x3*] layered convolution block (**LCB**) based architecture, in which we replaced the simple 3×3 convolution with a 3×3 depth-wise separable one ([*3x3-dw-separable]*). As shown in Table 9.3 the [*3x3-dw-separable*] delivers only slight improvements in the inference speed, with the cost of worse quantitative metrics. Motivated by this, and the visible qualitative differences from Fig. 9.4 which look better in the case of the [*3x3*], we decided there is not worth leveraging the advantages offered by depth-wise separable convolutions in our context.

Layered Convolution Block	RMSE	REL	δ_1	δ_2	δ_3	Inference Speed	FPS
3x3	0.515	0.143	0.817	0.957	0.988	0.030 seconds	34
3x3-dw-separable	0.520	0.148	0.810	0.957	0.988	0.030 seconds	33

Lower values are better	Higher values are better

Table 9.3: A quantitative comparison between the two *Leve* models that use the [*3x3-dw-separable*] LCB and the [*3x3*] LCB respectively.

The layout and the format of the table are inspired by [29]

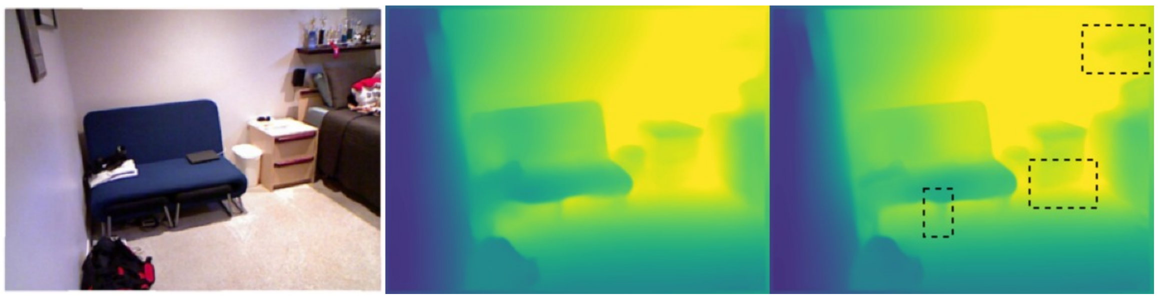

Figure 9.4: A qualitative comparison between the two *Leve* models that use the [*3x3-dw-separable*] LCB and the [*3x3*] LCB respectively. From left to right, there is the original image and the outputs of [*3x3-dw-separable*] and [*3x3*] in this order.

9.1.4 Skip Connections

Additionally, we also explored the idea of how choosing what layers within the encoder make up the skip connections impacts the model's accuracy. Besides the main architecture presented in Fig. 5.1 ([*main-skip-connections*]), we experiment with another version shown in Fig. 9.5 ([*alternative-skip-connections*]) and a version which doesn't rely at all on skip connections ([*no-skip-conections*]).

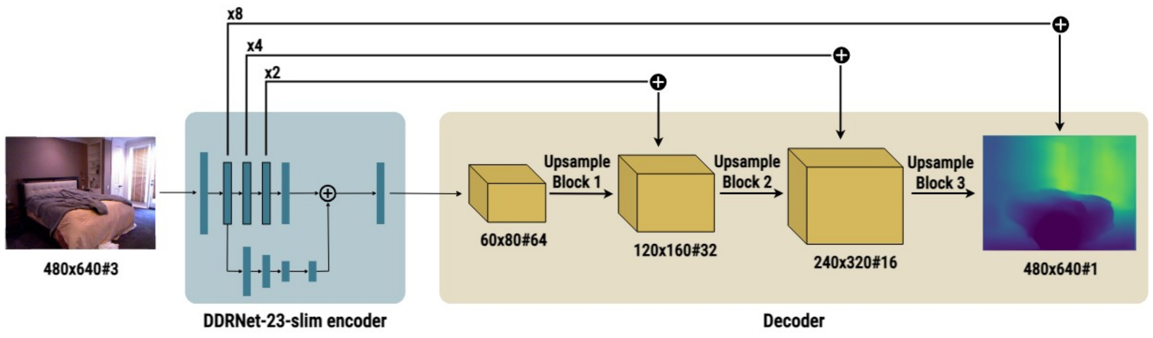

Figure 9.5: An alternative architecture for *Leve* with different skip connections.

Figure generated with draw.io [40]

As can be observed from Table 9.4 and Fig. 9.6 not utilising skip connections delivers the worst results. On the other hand, since the differences between the other two versions are negligible, we took inspiration from the idea promoted in [28], which uses only the original input image to guide the upsampling process no matter its location within the network's layers. Thus, we keep [*main-skip-connections*] for our primary model, where we pick layers placed towards the beginning of the architecture and as similar as possible to the block's size.

Method	RMSE	REL	δ_1	δ_2	δ_3	Inference Speed	FPS
main-skip-connections	0.516	0.142	0.814	0.958	0.988	0.037 seconds	27
alternative-skip-connections	0.513	0.143	0.815	0.959	0.989	0.040 seconds	25
no-skip-connections	0.518	0.146	0.813	0.955	0.988	0.034 seconds	29

Lower values are better	Higher values are better

Table 9.4: A quantitative comparison between the three *Leve* models that employ different strategies regarding the utilised skip connections.

The layout and the format of the table are inspired by [29]

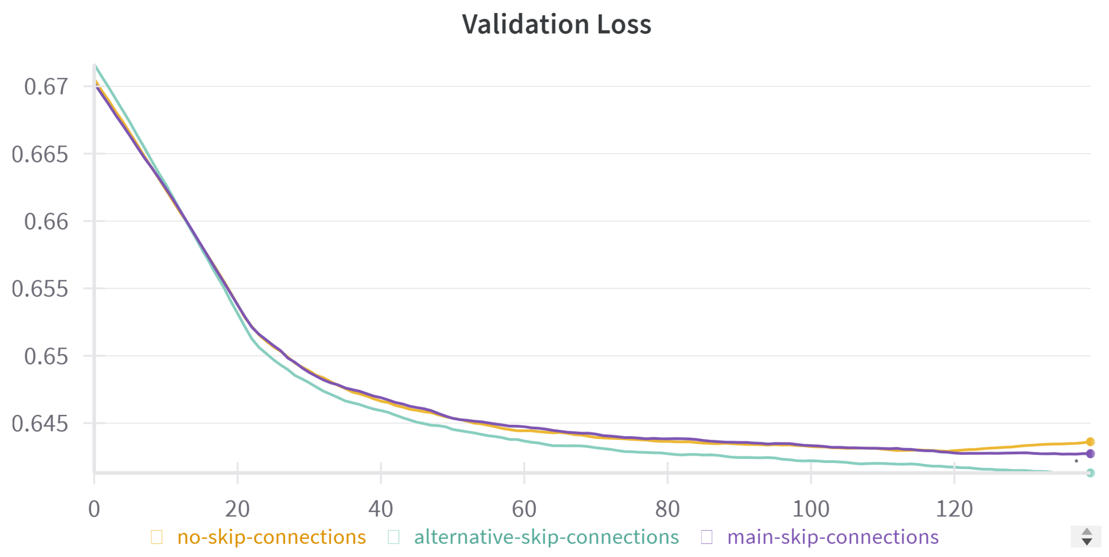

Figure 9.6: The evolution of the validation loss between the three *Leve* models that employ different strategies regarding the utilised skip connections.

Figure generated from Weights & Biases [54]

9.1.5 Final Results

Method	RMSE	REL	δ_1	δ_2	δ_3	Inference Speed	FPS
FastDepth [26]	0.663	0.219	0.701	0.912	0.971	0.0022 seconds	44
Junjie et al. [30]	0.511	0.147	0.813	0.958	0.989	0.0697 seconds	14.5
Rudolph et al. [28]	0.478	0.128	0.839	0.969	0.992	0.0460 seconds	21.5
Leve (240x320)	0.516	0.142	0.814	0.958	0.988	0.037 seconds	27
Leve (480x640)	0.495	0.136	0.829	0.964	0.990	0.037 seconds	27
JointDepth [56]	0.561	0.158	0.769	0.945	0.987	-	-
SARPN [57]	0.489	0.133	0.831	0.962	0.989	-	-
AdaBins [16]	0.430	0.119	0.871	0.975	0.994	-	-

Lower values are better	Higher values are better

Table 9.5: A quantitative comparison between *Leve* and the existing literature. At the top of the table there are lightweight networks, while underneath our models, we listed a few pioneer complex alternatives, which were too demanding when evaluating their inference speed. A detailed breakdown of each evaluation metric can be found in chapter 8.

The layout and the format of the table are inspired by [29]

Ultimately, we analyze how *Leve* stands against some of its lightweight competition, but also against some pioneer models that aren't built to optimize performance. As explained in chapter 7, for efficiency purposes, when comparing different alternatives of *Leve* between one another we trained them at half the size of the NYU-Depth V2 dataset's images. However, our most performant version and the one we focus on is trained on the original resolution of 480×640 pixels.

We can observe in Table 9.5 that our model produces competitive results in terms of finding the best balance between accuracy and performance when compared to other light models and even outperforms some of the staple more complex models that couldn't even be tested from a speed perspective due to their highly demanding nature.

9.2 Exploring Knowledge Distillation

While the high-level idea of how our full teacher-student architecture works when employing knowledge distillation is described in detail in chapter 4, the way we experiment with different versions comes down to how we choose the ϵ coefficient defined in the loss function from 7.1, settling down to the following three main models:

- *teacher-student-10* - with $\epsilon = 0.10$, the teacher's knowledge having a small influence of only 10% on the overall training process.
- *teacher-student-25* - with $\epsilon = 0.25$, which we believe as being the most balanced version, the teacher's knowledge influencing the overall training process in a proportion of 25%.
- *teacher-student-50* - with $\epsilon = 0.50$, where the teacher's knowledge is as important as the ground truth having an overall influence of 50%.

We continue by making a head-to-head comparison of all these three models against our main [*3x3+3x3*] lightweight *Leve* model. By only looking at the evolution of the validation loss we can see in Fig. 9.7 that the results are mostly similar, with [*3x3+3x3*] performing expectedly the best among all of them on the NYU-Depth V2 dataset, since no matter how effective a teacher model is, its knowledge cannot outperform the ground truth. However, to further test the generalization improvements knowledge distillation may deliver, we will look at various quantitative results on each proposed dataset described in chapter 6.

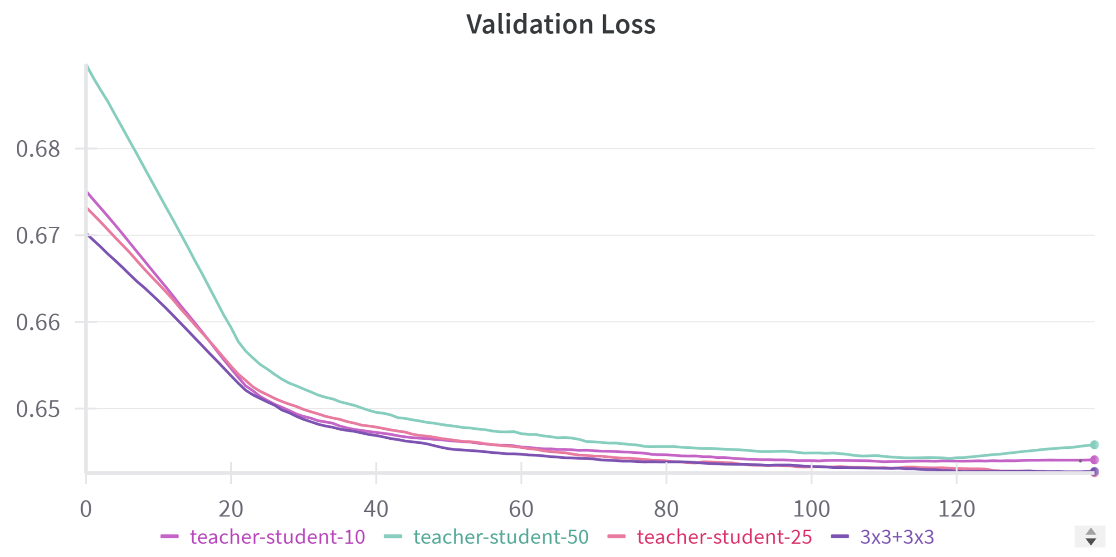

Figure 9.7: The evolution of the validation loss of the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight student model over 20 epochs of training.

Figure generated from Weights & Biases [54]

9.2.1 NYU-Depth V2

The NYU-Depth V2 comprises indoor images at a median depth scale of a maximum of 10 meters. As already mentioned, while we didn't expect any improvements when testing against images from a similar context as the training dataset, due to the high integrity of the ground truth depth maps, surprisingly, [*teacher-student-25*] delivers the best quantitative metrics as shown in Table. 9.6.

Model	RMSE	REL	δ_1	δ_2	δ_3
teacher-student-10	0.523	0.144	0.810	0.955	0.987
teacher-student-25	0.512	0.144	0.817	0.958	0.987
teacher-student-50	0.519	0.145	0.810	0.958	0.988
3x3+3x3	0.516	0.142	0.814	0.958	0.988

Lower values are better	Higher values are better

Table 9.6: A quantitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the NYU-Depth V2 test dataset.

The layout and the format of the table are inspired by [29]

9.2.2 Kitti

The next dataset is the Kitti dataset which, unlike the NYU-Depth V2, consists of outdoor images at higher depth distances reaching a maximum depth of 80 meters. Looking at Table 9.7, while we are pleased to find significant improvements in a majority of metrics for each teacher-student model, out of all three of them, the [*teacher-student-25*] performs the worst.

Model	RMSE	REL	δ_1	δ_2	δ_3
teacher-student-10	11.929	0.777	0.224	0.438	0.628
teacher-student-25	12.484	0.790	0.223	0.431	0.615
teacher-student-50	11.967	0.787	0.229	0.442	0.627
3x3+3x3	12.051	0.835	0.209	0.408	0.600

Lower values are better	Higher values are better

Table 9.7: A quantitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the Kitti test dataset.

The layout and the format of the table are inspired by [29]

9.2.3 EndoSLAM

The final dataset is the EndoSLAM dataset, which consists of endoscopic images targeting three different organs: the colon, the stomach and the small intestine. The uniqueness of this dataset is offered by its small depth scale (tens of centimetres), combined with the fact that no actual objects are present in the frame.

Model	RMSE	REL	δ_1	δ_2	δ_3
teacher-student-10	0.073	35.971	0.151	0.298	0.442
teacher-student-25	0.079	38.001	0.143	0.280	0.416
teacher-student-50	0.073	34.404	0.156	0.303	0.444
3x3+3x3	0.078	45.658	0.153	0.295	0.432

Lower values are better	Higher values are better

Table 9.8: A quantitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the EndoSLAM test dataset.

The layout and the format of the table are inspired by [29]

We can observe in Table 9.8, similarly as in the case of the Kitti dataset, that while we see enhancements in some of the metrics for all three teacher-student models, the [*teacher-student-25*] performs slightly worse than the other two.

9.2.4 Observations

Before coming up with some final comments, we believe we should also take into account the qualitative results depicted in Fig. 9.8, 9.9 and 9.10. While for the datasets introducing completely unexplored circumstances the visual results are far away from getting close to the ground truth images, an overall improvement over the [*3x3+3x3*] lightweight *Leve* model for all three teacher-student architectures is noticeable, even in the case of the NYU-Depth V2 dataset, where subtle details can be pointed out in the produced depth maps.

Overall, we consider this a successful experiment which proves that relying on a foundation model as a guiding teacher can improve the overall accuracy and generalization capabilities of a lightweight model, even in novel environments. However, there is still room for progress and future work which are further discussed in more detail in chapter 10.

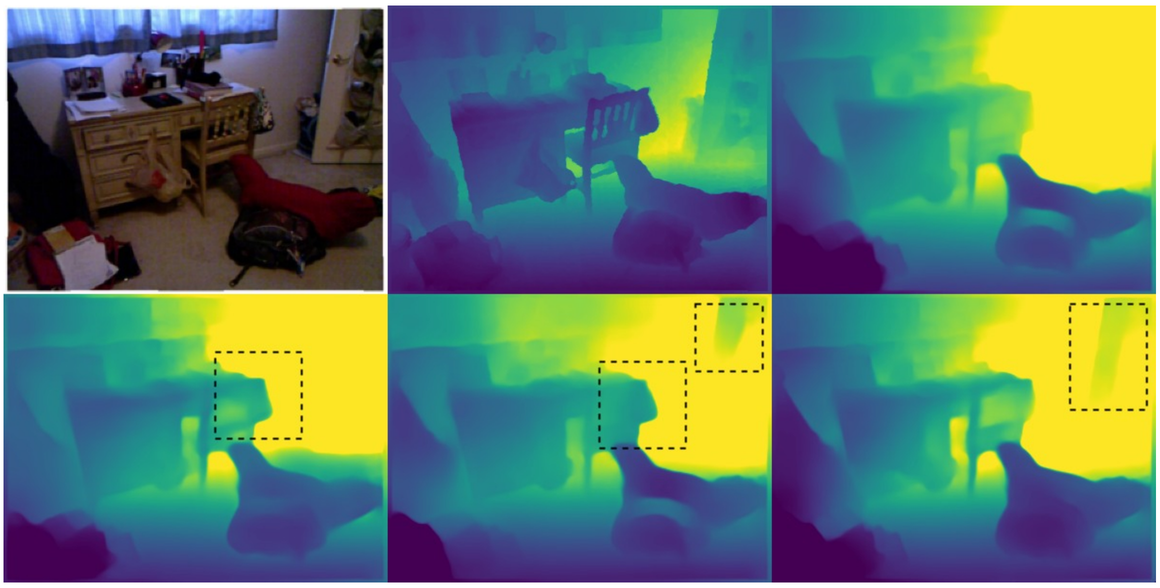

Figure 9.8: A qualitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the NYU-Depth V2 test dataset. From left to right, top to bottom, we have the original image, the ground truth and the outputs of [*3x3+3x3*], [*teacher-student-10*], [*teacher-student-25*] and [*teacher-student-50*] in this order.

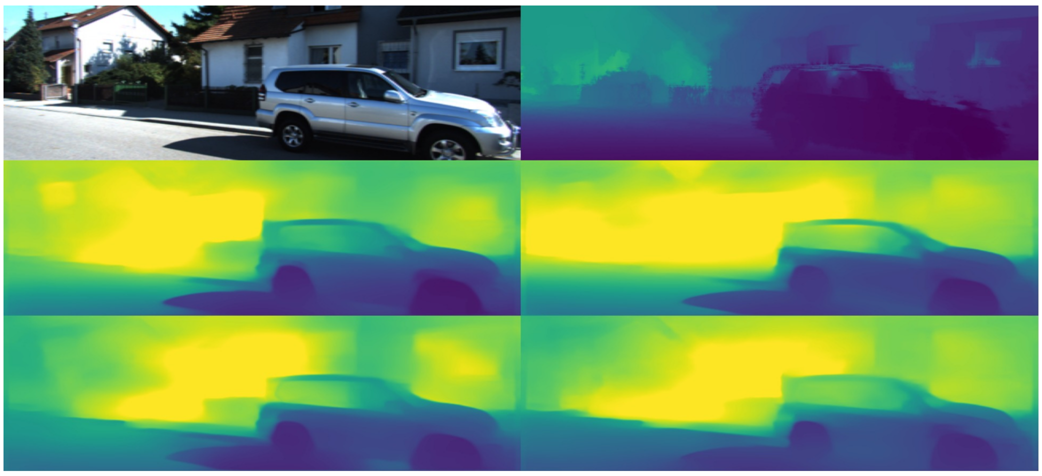

Figure 9.9: A qualitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the Kitti test dataset. From top to bottom, left to right, we have the original image, the ground truth and the outputs of [*3x3+3x3*], [*teacher-student-10*], [*teacher-student-25*] and [*teacher-student-50*] in this order.

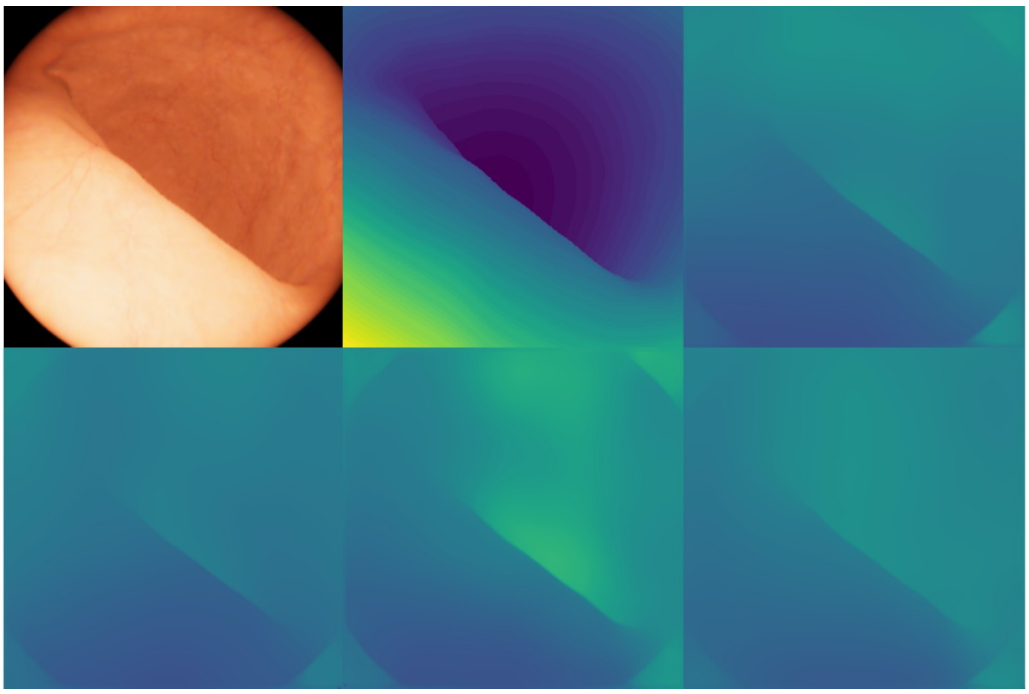

Figure 9.10: A qualitative comparison between the three suggested teacher-student architectures against the [*3x3+3x3*] lightweight *Leve* model for the EndoSLAM test dataset. From left to right, top to bottom, we have the original image, the ground truth and the outputs of [*3x3+3x3*], [*teacher-student-10*], [*teacher-student-25*] and [*teacher-student-50*] in this order.

9.3 The Application

To test the qualitative results of the different versions we've tried so far and moreover, to showcase the functionality of our most performant *Leve* model, we built a web-based application capable of translating images inputted by the users into their associated depth maps within seconds. The application can be accessed at the following link `https://leveai.streamlit.app/`, with its code being published on GitHub at `https://github.com/andrei-dragan/Leve-Web-App`.

Since the main work of this paper is focused on scientific experiments and their results, we didn't plan to overcomplicate the environment in which our application is built. However, we still wanted to have something that could be deployed over the internet and be accessible by many users, without relying on cloud computing resources or a backend to store information. This is why we used Streamlit [58], an open-source Python framework designed especially for AI/ML engineers and data scientists to create dynamic, simple, yet powerful apps to present their work.

At the moment, the application allows JPG, JPEG or PNG images to be uploaded as media that is converted into depth maps. Nonetheless, we built the application with the idea of easily scaling it in the future for recordings or even live video content. To achieve this, we relied on different design patterns such as *Factory* when instantiating the different UI components based on the media type, *Template* when rendering the screens, or *Singleton* to avoid loading the model every time a new inference is performed.

One limitation we encountered when building the application is that at the current time (May 2024), Streamlit allows only CPU execution which limits the performance of *Leve* and other lightweight models, making it impossible at the moment to perform inference on large video formats. We believe that with future improvements and allocated GPU resources, the application can become a lot more powerful and facilitate new feature developments.

Chapter 10

Conclusion & Future Work

In this chapter, we summarize our contributions to this paper and note down some ideas that can be explored in future research, having as a basis our discoveries.

We started by introducing a novel depth estimation model called *Leve*, which is based on an encoder-decoder type of architecture, capable of producing competitive results against state-of-the-art lightweight networks and even against some more complex popular ones, by leveraging different enhancement techniques. We showcased the advantages of relying on transfer learning strategies for the encoding part, highlighting the importance of experimenting with new pre-trained encoders from various computer vision tasks and not sticking to the usual MobileNets. Then, we emphasized that counting on skip connections significantly improves the performance of a model, stressing the fact that combining layers with similar sizes delivers the best results. Finally, we also analysed the impact of depth-wise separable convolutions, which in our case, didn't prove to provide great improvements.

Moreover, we performed an extensive study on the impact of knowledge distillation on our lightweight network, mainly focusing on using Depth-Anything as a foundation model to be the teacher and testing how different degrees of influence during training can impact the final accuracy and generalization capabilities of the student. We concluded that such a technique indeed improves the quantitative and qualitative results, even in environments unrelated to the training dataset.

However, we still believe there is room for improvement and additional research that can be done towards obtaining better results. For instance, in our work, while we focused only on response-based knowledge distillation, due to the incompatibility between our proposed architecture for *Leve* and the one of Depth-Anything, we think that utilizing feature-based or relation-based knowledge distillation as well, can determine a student to behave even more similar to its teacher. In addition, further experiments can be done on the way the loss function is computed when taking into account the knowledge shared by the teacher, since in our case, we didn't try to overcomplicate this aspect and stayed committed to a simple alternative.

Another limitation we encountered was related to the fact that Depth-Anything produced relative and not absolute depth maps. We consider that with future developments, more and more foundation models will appear in the field of monocular depth estimation, leaving room for new studies.

Finally, as we did with our lightweight model when trying different optimization techniques, a similar approach can be done when employing knowledge distillation, such as leveraging auxiliary training data as Hu et al. did in [30] or taking inspiration from language models as Farooq et al. took when relying on transformers in [16].

All in all, huge steps were made in the last couple of years in the field of monocular depth estimation. With the recent exponential rise in interest in artificial intelligence, together with the hardware that improves day by day, we look forward with excitement towards what can be discovered next.

Bibliography

[1] John J. Han, Ayberk Acar, Callahan Henry, and Jie Ying Wu. Depth anything in medical images: A comparative study, 2024.

[2] Beilei Cui, Mobarakol Islam, Long Bai, and Hongliang Ren. Surgical-dino: Adapter learning of foundation models for depth estimation in endoscopic surgery, 2024.

[3] Michele Mancini, Gabriele Costante, Paolo Valigi, and Thomas A. Ciarfuglia. Fast robust monocular depth estimation for obstacle detection with fully convolutional networks, 2016.

[4] Raul de Queiroz Mendes, Eduardo Godinho Ribeiro, Nicolas dos Santos Rosa, and Valdir Grassi. On deep learning techniques to boost monocular depth estimation for autonomous navigation. *Robotics and Autonomous Systems*, 136:103701, February 2021.

[5] Woontack Woo, Wonwoo Lee, and Nohyoung Park. Depth-assisted real-time 3d object detection for augmented reality. 2011.

[6] Shahira Kc, Sagar Tripathy, and A. Lijiya. Obstacle detection, depth estimation and warning system for visually impaired people. 08 2020.

[7] Xiaoxu Liu and Wei Qi Yan. Depth estimation of traffic scenes from image sequence using deep learning. In Han Wang, Wei Lin, Paul Manoranjan, Guobao Xiao, Kap Luk Chan, Xiaonan Wang, Guiju Ping, and Haoge Jiang, editors, *Image and Video Technology*, pages 186–196, Cham, 2023. Springer International Publishing.

[8] Timm Haucke and Volker Steinhage. Exploiting depth information for wildlife monitoring, 2021.

[9] Andreas Geiger, P Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the kitti dataset. *The International Journal of Robotics Research*, 32:1231–1237, 09 2013.

[10] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data, 2024.

[11] Github leve. `https://github.com/andrei-dragan/Leve`.

[12] Web app leve. `https://leveai.streamlit.app/`.

[13] Ashutosh Saxena, Sung Chung, and Andrew Ng. Learning depth from single monocular images. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press, 2005.

[14] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network, 2014.

[15] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue, 2016.

[16] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. Adabins: Depth estimation using adaptive bins. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2021.

[17] Jiaxing Yan, Hong Zhao, Penghui Bu, and YuSheng Jin. Channel-wise attention-based network for self-supervised monocular depth estimation, 2021.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[19] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning, 2019.

[20] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[21] Chunyuan Li, Zhe Gan, Zhengyuan Yang, Jianwei Yang, Linjie Li, Lijuan Wang, and Jianfeng Gao. Multimodal foundation models: From specialists to general-purpose assistants, 2023.

[22] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S. Yu, and Lichao Sun. A comprehensive survey on pretrained foundation models: A history from bert to chatgpt, 2023.

[23] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer, 2020.

[24] Nathaniel Simon and Anirudha Majumdar. Mononav: Mav navigation via monocular depth estimation and reconstruction, 2023.

[25] Ashkan Ganj, Yiqin Zhao, Hang Su, and Tian Guo. Mobile ar depth estimation: Challenges prospects – extended version, 2023.

[26] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems, 2019.

[27] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[28] Michael Rudolph, Youssef Dawoud, Ronja Güldenring, Lazaros Nalpantidis, and Vasileios Belagiannis. Lightweight monocular depth estimation through guided decoding, 2022.

[29] Xingshuai Dong, Matthew A. Garratt, Sreenatha G. Anavatti, and Hussein A. Abbass. Mobilexnet: An efficient convolutional neural network for monocular depth estimation, 2021.

[30] Junjie Hu, Chenyou Fan, Hualie Jiang, Xiyue Guo, Yuan Gao, Xiangyong Lu, and Tin Lun Lam. Boosting light-weight depth estimation via knowledge distillation, 2023.

[31] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 746–760, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[32] Nyu-depth v2 depth estimation benchmark. `https://paperswithcode.com/sota/monocular-depth-estimation-on-nyu-depth-v2`.

[33] Kitti depth estimation benchmark. `https://paperswithcode.com/sota/monocular-depth-estimation-on-kitti-eigen`.

[34] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[35] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019.

[36] Yuanduo Hong, Huihui Pan, Weichao Sun, and Yisong Jia. Deep dual-resolution networks for real-time and accurate semantic segmentation of road scenes, 2021.

[37] Depth-wise separable convolutions - illustration. `https://medium.com/@zurister`.

[38] Xichuan Zhou, Haijun Liu, Cong Shi, and Ji Liu. Chapter 3 - model design and compression. In Xichuan Zhou, Haijun Liu, Cong Shi, and Ji Liu, editors, *Deep Learning on Edge Computing Devices*, pages 39–58. Elsevier, 2022.

[39] Lucas Beyer, Xiaohua Zhai, Amélie Royer, Larisa Markeeva, Rohan Anil, and Alexander Kolesnikov. Knowledge distillation: A good teacher is patient and consistent, 2022.

[40] Draw.io. `https://www.drawio.com/`.

[41] Andrew L" "Maas, Awni Y" "Hannun, and Andrew Y" "Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[42] Luigi Piccinelli, Yung-Hsu Yang, Christos Sakaridis, Mattia Segu, Siyuan Li, Luc Van Gool, and Fisher Yu. Unidepth: Universal monocular metric depth estimation, 2024.

[43] Bingxin Ke, Anton Obukhov, Shengyu Huang, Nando Metzger, Rodrigo Caye Daudt, and Konrad Schindler. Repurposing diffusion-based image generators for monocular depth estimation, 2024.

[44] Mykola Lavreniuk, Shariq Farooq Bhat, Matthias Müller, and Peter Wonka. Evp: Enhanced visual perception using inverse multi-attentive feature refinement and regularized image-text alignment, 2023.

[45] Siddharth Srivastava and Gaurav Sharma. Omnivec: Learning robust representations with cross modal sharing, 2023.

[46] Shaohua Dong, Yunhe Feng, Qing Yang, Yan Huang, Dongfang Liu, and Heng Fan. Efficient multimodal semantic segmentation via dual-prompt learning, 2023.

[47] Daniel Seichter, Benedict Stephan, Söhnke Benedikt Fischedick, Steffen Müller, Leonard Rabes, and Horst-Michael Gross. Panopticndt: Efficient and robust panoptic mapping, 2023.

[48] Anat Levin, Dani Lischinski, and Yair Weiss. Colorization using optimization. *ACM Transactions on Graphics*, 23, 06 2004.

[49] Shengjie Zhu and Xiaoming Liu. Lighteddepth: Video depth estimation in light of limited inference view angles. In *CVPR*, 2023.

[50] Rajeev Yasarla, Manish Kumar Singh, Hong Cai, Yunxiao Shi, Jisoo Jeong, Yinhao Zhu, Shizhong Han, Risheek Garrepalli, and Fatih Porikli. Futuredepth: Learning to predict the future improves video depth estimation, 2024.

[51] Kutsev Bengisu Ozyoruk, Guliz Irem Gokceler, Gulfize Coskun, Kagan Incetan, Yasin Almalioglu, Faisal Mahmood, Eva Curto, Luis Perdigoto, Marina Oliveira, Hasan Sahin, Helder Araujo, Henrique Alexandrino, Nicholas J. Durr, Hunter B. Gilbert, and Mehmet Turan. Endoslam dataset and an unsupervised monocular visual odometry and depth estimation approach for endoscopic videos: Endo-sfmlearner, 2020.

[52] Google colab. `https://colab.google/`.

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[54] Weights & biases. `https://wandb.ai/site/company/about-us`.

[55] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[56] Vladimir Nekrasov, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations, 2019.

[57] Xiaotian Chen, Xuejin Chen, and Zheng-Jun Zha. Structure-aware residual pyramid network for monocular depth estimation, 2019.

[58] Streamlit. `https://docs.streamlit.io/`.