Universidad Autónoma de Querétaro

Facultad de Ingeniería

Monocular Depth Estimation with Convolutional Neural Networks on Embedded Systems

Tesis

Que como parte de los requisitos para obtener el Grado de Maestro en Ciencias en Inteligencia Artificial

Presenta

Edgar Rodrigo López Silva

Dirigido por

Dr. Jesús Carlos Pedraza Ortega

Querétaro, Qro. a 26 de agosto de 2021

Universidad Autónoma de Querétaro

Faculty of Engineering

Master of Science in Artificial Intelligence

Monocular Depth Estimation with Convolutional Neural Networks on Embedded Systems

Thesis

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Artificial Intelligence

by
Edgar Rodrigo López Silva

Supervisor
Dr. Jesús Carlos Pedraza Ortega

SYNOD

Dr. Jesús Carlos Pedraza Ortega
President

Dr. Juan Manuel Ramos Arreguín
Secretary

Dr. Marco Antonio Aceves Fernández
Vocal

Dr. Saúl Tovar Arriaga
Alternate

M.S. Luis Rogelio Román Rivera
Alternate

Centro Universitario, Queretaro, Qro.
August 2021
Mexico

This thesis is dedicated to my Parents …

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Jesús Carlos Pedraza Ortega, for his endless support during this two-year journey. His guidance and advices have allowed me to develop myself not only professionally, but personally as well. I must say it has been really an honor and a privilege to have worked this master thesis project under his leadership.

I want to also thank my synod members and professors for their support and feedback in our quest of learning the basics of research. It's truly inspiring to learn surrounded by highly qualified professors who undoubtedly made us feel part of the scientific community.

Overall, I am deeply grateful to Universidad Autonoma de Queretaro for giving me the chance of completing my master studies in such a great institution. Last but not least, I want to thank to the Consejo Nacional de Ciencia y Tecnologia for the financial support that helped finance this project.

Abstract

Monocular depth estimation is becoming a very interesting problem in computer vision to solve due to the several tasks that require as an input the spatial structure of a scene, such as 3D reconstruction, 3D object detection, localization and mapping. The most effective techniques for monocular depth estimation are based on large deep learning-based architectures that cannot be deployed on systems with limited computational resources and therefore preventing its use in application fields where the advantages of monocular cameras (i.e., low cost, small size, low weight and low-energy consumption) could also be exploited. Under this context, the research of low-latency deep learning architectures for monocular depth estimation is a very promising topic for which just a few methods have been proposed until now. In this master thesis, a very low-latency fully convolutional network is proposed. The quantitative results on the NYU-Depth V2 dataset show that the proposed method is 1.6x faster than the state-of-the art related method while also reducing the RMSE metric by 1.16%.

Resumen

La estimación de profundidad monocular se está convirtiendo en un problema muy interesante de resolver en la visión por computadora debido a las diversas tareas que requieren como entrada la estructura espacial de una escena, como la reconstrucción 3D, la detección de objetos 3D, la localización y el mapeo. Las técnicas más efectivas para la estimación de la profundidad monocular se basan en grandes arquitecturas basadas en aprendizaje profundo que no se pueden implementar en sistemas con recursos computacionales limitados y, por lo tanto, impiden su uso en campos de aplicación donde las ventajas de las cámaras monoculares (es decir, de bajo costo, tamaño pequeño, etc. bajo peso y bajo consumo de energía) también podrían aprovecharse. En este contexto, la investigación de arquitecturas de aprendizaje profundo de baja latencia para la estimación de la profundidad monocular es un tema muy prometedor para el que hasta ahora solo se han propuesto pocos métodos. En esta tesis de maestría, se propone una red totalmente convolucional de muy baja latencia. Los resultados cuantitativos en el conjunto de datos NYU-Depth V2 muestran que el método propuesto es 1.6 veces más rápido que el método relacionado con el estado de la técnica y, al mismo tiempo, reduce la métrica RMSE en un 1.16%.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Depth estimation is considered one of the most fundamental problems in the field of *computer vision* as it is an essential function for the realization of several tasks such as localization, mapping, motion planning, 3D object detection and augmented reality applications. Overall, depth estimation refers to the set of techniques and algorithms designed to obtain a representation of the spatial structure of a scene. In other words, the main objective of the depth estimation methods is to achieve a representation of the *absolute* or *relative* distance from the 3D sensor to each point of the scene of interest (Zhao, Sun, Zhang, Tang, & Qian, 2020).

It is well known that depth estimation from digital images has been based primarily on a stereoscopic vision approach, which requires a pair of images captured from the same scene but from different angles in order to triangulate the 3D position of each pixel in the image. Nonetheless, in the last few years, there has been an increasingly interest in *monocular depth estimation*, i.e., using a single RGB input image. The latter refers to a very complex and inherently ambiguous problem since there is no correlation between the intensity value (color saturation) and the *relative depth* associated with each pixel in an RGB image (Yang, 2017).

The task of estimating relative depth maps from a single RGB image can be understood as a highly non-linear transformation that requires the extraction of relevant features from the input RGB image to generate a representation of latent space, from which, the relative depth map corresponding to the input image can be reconstructed. Considering the above, pioneering studies in this line of research began to consider *machine learning* as a promising approach to solve the problem in question. The main reason of the latter has been due to the fact that machine learning is precisely the branch of *artificial intelligence* that studies approximation methods to estimate highly complex functions through the development of statistical and connectionist models of certain phenomena for which a large amount of data is available (Bhoi, 2019).

In particular, recent advances in *deep learning* related to *convolutional neural networks* (CNNs) have shown that estimating relative depth from a single input image is feasible, a factor which has further aroused the interest of the scientific community (Eigen, Puhrsch, & Fergus, 2014).

1.1 Problem description

As mentioned previously, depth estimation is considered an essential function for perception subsystems that require obtaining a representation of the spatial structure of a scene. A clear example is that of embedded processor-based *autonomous systems*, whose main tasks such as localization, mapping, motion planning and 3D obstacle detection depend closely on a previous prediction of a depth map.

Despite the fact that there have been recent advances in depth estimation from a single RGB image, the state-of-the-art deep learning architectures have been found to be too large for real-time inference on an embedded platform (Poggi, Aleotti, Tosi, & Mattoccia, 2018). This makes it impossible to deploy these emerging algorithms in application fields where the computational resources (e.g., processing power, RAM, etc.) are very restricted. Moreover, it should be noted that almost any practical software system consists of a complex pipeline where several programs are executed in a multi-process approach. The latter implies that the embedded processor would not be exclusively dedicated to the monocular depth estimation task, an aspect that further aggravates the problem in question.

At the time of writing this thesis, very few studies have been proposed to tackle the monocular depth estimation problem using low-latency deep learning architectures. From this perspective, it can be said that there is a lack of research efforts on the convergence of monocular depth estimation and the *edge computing* paradigm.

1.2 Justification

Currently, there are two popular approaches for depth estimation. On the one hand, the LIDAR (Light Detection and Ranging) sensor is considered highly precise and reliable. Nevertheless, it is quite expensive (around $75,000 USD), which makes it unsuitable for low-budget projects (You *et al.*, 2020). Additionally, the LIDAR sensor is a fragile component that consists of an electromechanical assembly that requires periodic maintenance and calibration. On the other hand, stereo-cameras are also commonly used to estimate depth through triangulation techniques. However, this approach requires a laborious hardware setup and continuous calibration schemes (Larsson, 2019).

The limitations and disadvantages of the approaches above-mentioned have driven the research and development of maintenance-free, cheaper and power-efficient alternatives for depth estimation. Among these alternatives, monocular depth estimation is currently regarded as one of the most promising approaches to that end. The latter has been mainly due to the fact that deep learning-based methods have ultimately shown outstanding performance in computer vision problems and monocular depth estimation is no exception. As already mentioned, recent developments have demonstrated that relative depth information can be recovered from a single RGB image using end-to-end deep learning techniques (Zhao *et al.*, 2020).

The main purpose of developing monocular depth estimation techniques is not to replace the previously mentioned approaches but to provide a low-cost alternative for specific use cases. Moreover, the development of low-latency monocular depth estimation algorithms would enable its use in application fields where the advantageous characteristics of monocular cameras such as high-energy efficiency (i.e., low-energy consumption), small size and low weight could be exploited. These camera features are expected to keep improving continuously during the next decade and more importantly, many common devices and machines are now capable of supporting monocular camera functionalities (e.g., eyeglasses, smartphones, smartwatches, drones, motor vehicles, etc.). These are just a few examples to illustrate that adding monocular cameras to common devices is becoming a disruptive trend in many sectors involved with the research and development of goods and services.

On the other hand, low-latency monocular depth estimation techniques could contribute to the *democratization* of software-based systems that rely on a 3D perception pipeline. For instance, the technology behind some advanced active safety features in vehicles that would help to prevent or mitigate road crashes is currently quite expensive mainly due to the LIDAR sensors that are now being used (You *et al.*, 2020).

Similarly, in recent years, research efforts have been made to use augmented reality devices for cognitive prosthesis to help blind people navigate unfamiliar spaces (Liu, Stiles, & Meister, 2018). The main drawback of the latter is also the price of the augmented reality device. These are just a few examples where low-cost depth estimation techniques, such as monocular depth estimation, have the potential to democratize technologies that could improve human safety and ultimately increase the quality of life.

1.3 Hypothesis

By using deep learning techniques, it is feasible to decrease the error metric for relative depth estimation from monocular images in embedded systems.

1.4 Objectives

1.4.1 General objective

Develop, implement and evaluate an algorithm for relative depth estimation from monocular images in an embedded system using deep learning-based methods.

1.4.2 Specific objectives

- Develop, implement and train the deep learning algorithm for relative monocular depth estimation using a personal computer.
- Select the embedded system with the necessary specifications to carry out the inference stage.
- Define the evaluation metrics relevant to relative monocular depth estimation.
- Develop standard performance tests that can verify that the error metric has been improved.
- Execute the standard performance tests on a personal computer and on an embedded system.
- Validate the obtained results by comparing against some established method.

1.5 Scope and limitations

1.5.1 Scope

- Follow a hand-crafted design approach to develop low-latency deep learning models for a pixel-level regression task.
- Select a state-of-the-art encoder to perform the feature extraction process.
- Train the proposed models using a single benchmark dataset for monocular depth estimation.
- Perform the model validation in the standard way as defined by prior state-of-the-art studies (i.e., using official splits).
- The model deployment considers a single pre-selected embedded system.

1.5.2 Limitations

- Low-latency architecture design from scratch is out of the scope of this thesis.
- Lack of a high-performance computer limited the ability to carry out exhaustive model trainings.
- Inability to use the google colab tool to accelerate the model training since its usage is limited to smaller datasets (<15Gb).
- No post-processing steps to improve latency will be considered since it is well known that such techniques could impact the model performance.
- K-fold cross validation is not considered due to the lack of computational resources.
- Hyperparameter optimization techniques are not considered due to lack of high-end computational resources and time constraints.
- No video capture devices were considered for testing.

1.6 Thesis organization

The above introduction offers a brief overview of the research topic, describes the problem that is addressed in this thesis and the motivation behind it. Chapter 2 presents the literature survey of classical and state-of-the-art studies that are related to the specific research topic. Chapter 3 describes the theoretical background and fundamental concepts that are relevant to monocular depth estimation. In this sense, Chapter 3 establishes the formal framework that supports the research methodology.

Chapter 4 gives a detailed description of the scientific method's steps that were developed to test the hypothesis that was initially stated. Chapter 5 presents the quantitative and qualitative experimental results that were obtained by evaluating several monocular depth estimation models on the validation dataset on an embedded system. Furthermore, the obtained results are compared against a well-established state-of-the-art method. Lastly, a thorough discussion of the obtained quantitative and qualitative results is presented. In particular, the results are analyzed and interpreted from a theoretical standpoint in order to explain the causal relationships that were found between the different factors (i.e., architectural design and hyperparameter values) that govern the performance of low-latency deep learning algorithms for monocular depth estimation.

Finally, Chapter 6 presents the conclusion of the research in terms of the contributions and provides the future work that may further improve the methodology.

CHAPTER 2

Literature Review

This chapter first describes the classic literature that is relevant to three-dimensional reconstruction in order to provide enough context to the reader. The second section of this chapter presents the state-of-the-art research papers that are relevant to monocular depth estimation. Likewise, the methods that use low-latency algorithms to solve the problem in question are highlighted.

2.1 Three-dimensional reconstruction

As already stated, recovering 3D information from a scene represents one of the most fundamentals problems in the field of computer vision and its solution can be found through very diverse techniques. According to (Rocchini, Cignoni, Montani, Pingi, & Scopigno, 2001), the techniques for 3D reconstruction can be divided into two large groups: contact acquisition and non-contact acquisition (see Fig. 2.1).

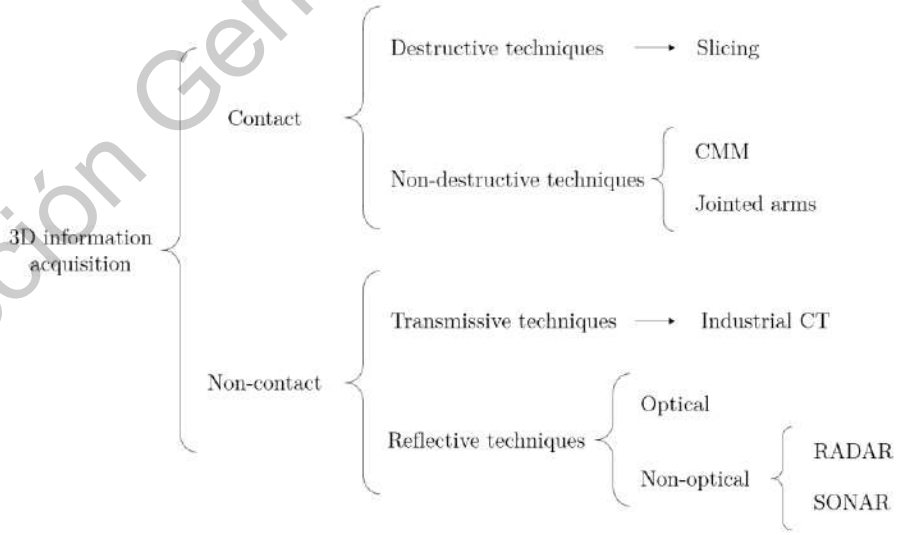


Fig. 2.1: Taxonomy of 3D information acquisition techniques. Adapted from (Rocchini *et al.*, 2001).

The contact acquisition techniques are divided into destructive and non-destructive. The destructive techniques include slice-based methods by which it is possible to reduce the dimension of the analysis by successively sectioning an object into 2D shapes. On the other hand, the non-destructive techniques consist in using prismatic or revolute instruments that record the coordinates of the object using a probe (Giancola, Valenti, & Sala, 2018).

In contrast, the non-contact techniques are of special interest since these avoid any physical contact with the object to be measured and, therefore, eliminate any risk of inducing mechanical stress or damage to it. Within this group of techniques, there are transmissive and reflexive. The former uses the projection of electromagnetic signals toward the object to identify changes in density within it, while the latter process the reflection of the signal emitted by the object. In particular, reflective techniques can be non-optical and optical. Non-optical techniques refer to those that use signals that are not included in the visible and infrared spectrum, e.g., RADAR-based systems use radio waves as the type of electromagnetic signal (Giancola *et al.*, 2018).

On the contrary, optical techniques operate in the visible and infrared wavelength spectrum to obtain information from a scene. In turn, optical techniques for shape acquisition can be divided into passive and active methods (see Fig. 2.2). Passive methods use the reflection of natural light on a given target to measure its 3D shape. In other words, passive methods do not interfere with the object being measured, they just use a sensor to measure the radiation reflected by the surface of the object and infer its 3D structure through image understanding. For instance, stereoscopy uses the theory of triangulation and epipolar geometry to search for homogeneous multi-camera features to reconstruct a 3D shape (Giancola *et al.*, 2018).

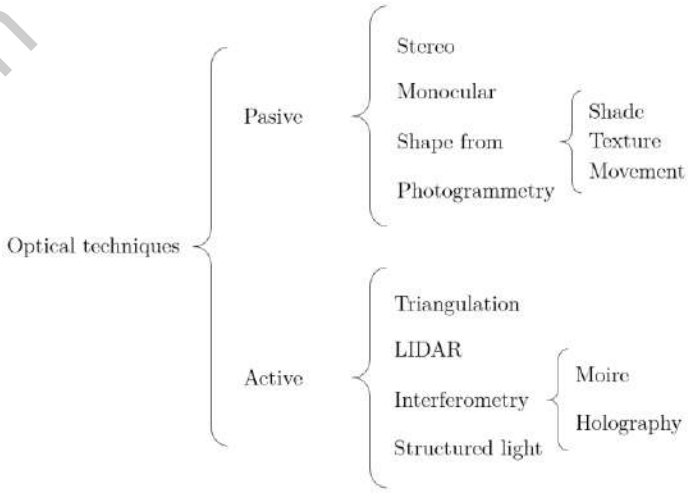


Fig. 2.2: Classification of optical techniques. Adapted from (Rocchini *et al.*, 2001).

On the other hand, the monocular approach exploits machine learning methods for passive depth estimation using a single RGB input image, thus avoiding acquisitions from multiple points of view for this purpose (Poggi *et al.*, 2018).

By contrast, active methods are characterized for using an external light source that provides additional information to improve the acquisition of 3D shapes. These techniques are based on the disturbance of the detected environment to infer the corresponding depth maps. Among the most relevant active methods are LIDAR systems, whose operating principle is the same used in TOF (Time-of-Flight) systems. The operating principle consist on estimating depth maps by measuring the delay time between the emission of an electromagnetic pulse and its reception through the reflected signal. Interferometry-based methods are another example of active techniques. These consist on the projection of fringe patterns to estimate the 3D shapes and a subsequent iterative spatial refinement of the projected pattern (Giancola *et al.*, 2018).

Despite the effectiveness and high reliability of the active techniques, passive techniques have been gaining greater interest in recent years. In large part, this is due to the fact that the implementation of passive optical techniques is much cheaper since they do not require specific lighting conditions for their operation (Molleda, 2008).

2.2 State-of-the-art

This thesis work aims to study the monocular passive technique by using low-latency machine learning methods for relative depth estimation. This passive technique is better known as monocular depth estimation and consists in predicting relative a depth map from a single RGB input image. As mentioned above, the monocular technique is regarded as an emerging research topic and is currently considered as a promising approach due to its advantages over other alternatives. Furthermore, the recent advances in the field of Artificial Intelligence have boosted the research and development of machine learning algorithms for monocular depth estimation (Bhoi, 2019).

The initial works on relative depth estimation using monocular images are based on traditional machine learning algorithms. In particular, Saxena, Chung, and Ng (2006) proposed the first study related to monocular depth estimation by using hand-crafted image features and discriminative training of a MRF (Markov Random Field) model. However, their method was characterized by poor performance in uncontrolled environments. On the other hand, Karsch, Liu, and Kang (2012) suggested a method based on K-NN (K-Nearest Neighbors) and SIFT Flow for depth estimation in images with static background. Despite

the progress that these studies represented, both methods are known to require laborious alignment procedures.

As already mentioned, it was not until the implementation of deep learning techniques that significant advances were made in this line of research (see Table 2.1). One of the first studies that proposed this approach was (Eigen *et al.*, 2014). Their proposal consists in using two coupled convolutional neural networks to infer the depth associated with each pixel in an image. This work also stood out as the first to introduce the concept of multi-scale information in monocular depth estimation.

Later studies, such as (Liu, Shen, Lin, & Reid, 2016), focused on obtaining sharper visual transitions by incorporating a CRM (Conditional Random Field) model as a regularization stage. At the same time, Laina, Rupprecht, Belagiannis, Tombari, and Navab (2016) proposed the use of fully convolutional networks (also known as FCNs) with residual connections for monocular depth estimation. In contrast to the previous studies, this approach does not require additional post-processing or other refinement steps. From this point in time, the fully convolutional network architecture began to establish itself as one of the most promising deep learning-based methods in this line of research.

Subsequent works to (Laina *et al.*, 2016) are mostly based on FCNs and have been focused on improving the accuracy metric in various ways, e.g., through the inclusion of semantic information in the training stage (Gurram, Urfalioglu, Halfaoui, Bouzaraa, & López, 2016), the use of dilated convolution techniques to improve the overall robustness of the architecture (Fu, Gong, Wang, Batmanghelich, & Tao, 2018) or by defining novel perceptual loss functions to improve the retention of local details (Kumari, Jha, Bhavsar, & Nigam, 2019). Common to the deep learning-based methods mentioned above is the supervised learning strategy, which requires *labeled data* during the training stage.

Table 2.1: State-of-the-art methods for non-efficient monocular depth estimation.

Author	Model	Architecture	Learning strategy
(Saxena *et al.*, 2006)	MRF	N/A	Supervised
(Eigen *et al.*, 2014)	CNN	GCSN-LFSN	Supervised
(Liu *et al.*, 2016)	CNN	DCNF	Supervised
(Godard *et al.*, 2017)	CNN	FCN	Unsupervised
(Gurram *et al.*, 2018)	CNN	DSC-DRN	Supervised
(Fu *et al.*, 2018)	CNN	DORN	Supervised
(Pilzer *et al.*, 2018)	CNN	CycleGAN	Unsupervised

On the other hand, the unsupervised learning strategy is the orthogonal approach to the one described above. This approach attempts to exploit the potential of deep learning methods without the use of labeled data. In this sense, (Godard, Mac Aodha, & Brostow, 2017) proposed depth estimation as an image reconstruction problem, for which they define a new loss function capable of imposing consistency between the disparities produced between the right and left images. Similarly, (Pilzer, Xu, Puscas, Ricci, & Sebe, 2018) proposed a novel method for depth estimation based on an antagonistic cyclical generative network (CGAN or CycleGAN). Both unsupervised methods previously mentioned take advantage of direct or indirect measurement of disparity as a previous step for depth estimation.

However, all the previously mentioned state-of-the-art methods stand out for using highly complex convolutional neural networks whose inference stage cannot be deployed in real-time on systems with limited computational resources. From this perspective, very few studies have been proposed to tackle the monocular depth estimation problem using low-latency convolutional architectures (see Table 2.2).

On the one hand, Poggy *et al.* (2018) adopts an unsupervised learning strategy to train a low-latency pyramidal convolutional network (PyD-Net) on the KITTI dataset. Their study is the first proposal in this research area to achieve a significant reduction in the number of learnable parameters which enabled them to dramatically decrease the execution time. Overall, their learning strategy is based on (Godard *et al.*,2017) and compared to such work, their model is almost 94% smaller.

Similarly, Wofk, Ma, Yang, Karaman, and Sze (2019) propose a lightweight fully convolutional network that uses the MobileNet feature extractor as the encoder and a low-latency decoder. Also, the authors suggest the use of additional model compression algorithms to further decrease the number of learnable parameters and inference time. Their proposed architecture (without the compression stage) is smaller than (Poggy *et al.*, 2018) and is evaluated on the NYU-Depth V2 dataset instead.

Table 2.2: State-of-the-art methods for efficient monocular depth estimation.

Author	Model	Architecture	Learning strategy
(Poggy *et al.*, 2018)	CNN	PyD-Net	Unsupervised
(Wofk *et al.*, 2019)	CNN	low-latency FCN	Supervised
(Wang *et al.*, 2020)	CNN	DepthNet Nano	Supervised

In contrast to the studies above described, Wang, Famouri, and Wong (2020) have recently proposed a compact convolutional neural architecture obtained by employing a human-machine collaborative design strategy. Their proposal is smaller than the typical studies that use large deep learning architectures but even so, it is not as lightweight and fast as the convolutional architectures proposed in (Poggy *et al.*, 2018; Wofk *et al.*, 2019).

This thesis project addresses the monocular depth estimation problem using a novel low-latency fully convolutional network that yields better performance (both at run time and in error) when compared to (Wofk *et al.*, 2019) and which is considered the most efficient state-of-the-art method from those shown in Table 2.2.

CHAPTER 3

Theoretical Framework

This chapter presents the theory related to artificial intelligence that is relevant to this master thesis. Specifically, this chapter offers a brief overview of the fundamentals of deep learning and a detailed introduction to *fully convolutional networks* which are the cornerstone of this research topic. Furthermore, some concepts related to low-latency convolutional layers for deep learning at the edge are also outlined. The last section of this chapter presents the standard problem formulation for monocular depth estimation.

3.1 Machine learning

Machine learning is the subfield of artificial intelligence that studies computer algorithms that are capable of finding patterns in large amounts of raw data. In other words, machine learning is the *data-driven approach* to artificial intelligence (Goodfellow, Bengio, & Courville, 2016).

According to Mitchell (1997), the concept of *machine learning* can be formally defined as follows:

> "A computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**."

At the same time, machine learning can be divided into two different subsets of techniques: *classical machine learning* and *representation learning*. On the one hand, classical machine learning profoundly depends on the representation of the data that is to be analyzed. The latter implies that it is first necessary to design the correct features to extract for a specific task. Nonetheless, knowing what features need to be extracted can be very difficult. On the other hand, representation learning solves the above- mentioned problem by using machine learning to also find the proper representation of the input data and not

just the mapping from representation to output. This means that some machine learning algorithms can be used for feature extraction in order to obtain the desired representation. However, designing algorithms for learning features can usually be as complicated as solving the task by the classical machine learning approach (Goodfellow *et al.*, 2016).

Deep learning is a representation learning approach that uses multilayer artificial neural networks to avoid the quoted difficulty by allowing the computer program to build complex representations on top of simpler representations. Fig. 3.1 shows how deep learning is a subset of machine learning and also representation learning (Goodfellow *et al.*, 2016).

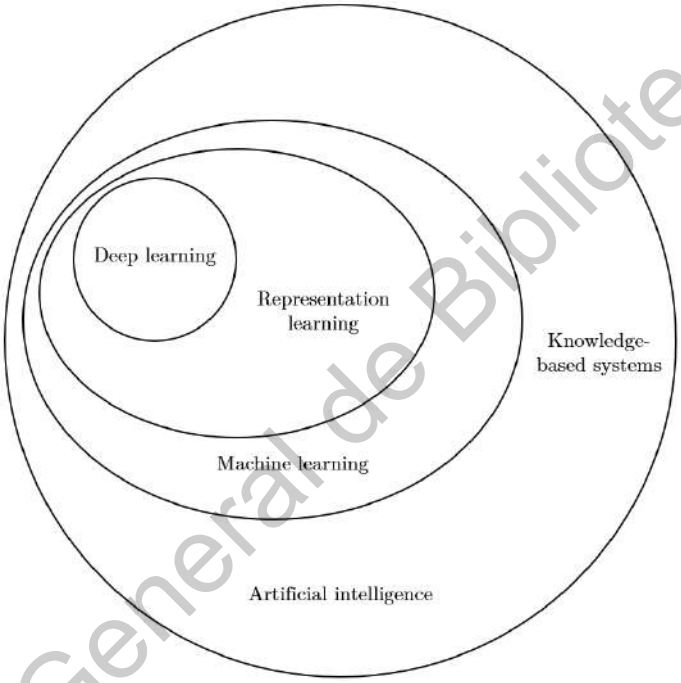


Fig. 3.1: Subfields of artificial intelligence. Adapted from (Goodfellow *et al.*, 2016).

3.2 Artificial neural networks

An artificial neural network (ANN) is a machine learning technique that is somehow inspired on the mechanism of learning in biological organisms. It is also commonly regarded as a *connectionist approach* due to the use of a set of processing units called *neurons* which are connected to each other (Haykin, 2008).

More formally, an artificial neural network can be considered as a computational graph composed of nodes that interconnect which each other for the transmission of information from an input layer to an output layer. The inputs to a neuron are scaled through parameters

known as synaptic weights which serve as the strength of the connections. These parameters need to be adjusted during the training phase to find the mapping of interest (Haykin, 2008).

The simplest ANN model consists of a single neuron as illustrated in Fig. 3.2.

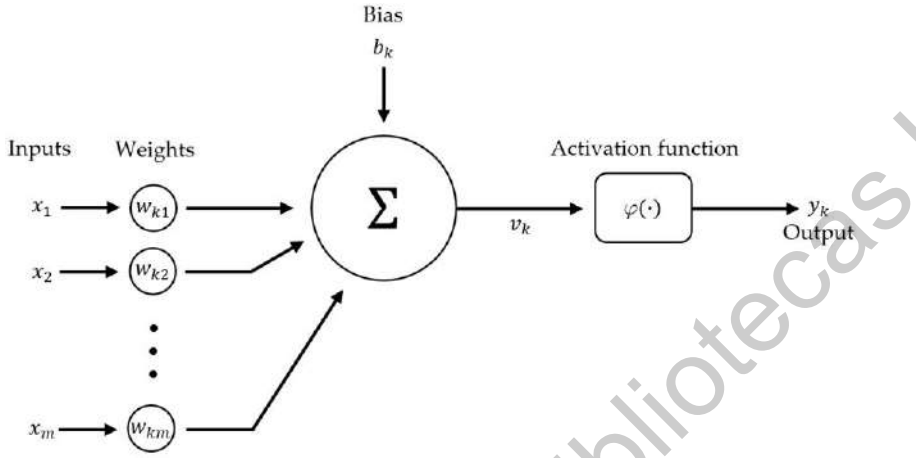


Fig. 3.2: Non-linear model of a neuron. Adapted from (Haykin, 2008).

In general, for a vector x containing the input signals $(x_1, x_2, \dots, x_m)$, a vector of synaptic weights w, a bias value b and an activation function $\varphi(\cdot)$, the output y of a neuron is given by the equation in (1) (Haykin, 2008).

$$y = \varphi\left(b + \sum_{i=1}^{m} x_i w_i\right) \tag{1}$$

The model shown in Fig. 3.2 is considered non-linear since the activation function $\varphi(\cdot)$ is known to introduce the required non-linearities to the output. It should be noted that the most common activation function is the rectifier function better known as ReLU (Haykin, 2008). A more detailed discussion on activation functions is presented in section 3.4.2.

As mentioned before, several neurons can be interconnected with each other to create a specific structure which is commonly referred to as an *architecture*. The simplest kind of architectures consist of an *input layer* of source nodes that feed signals to the next layer which can be either an *output layer* or a *hidden layer*. In both cases, the network is considered to be a *feed-forward* type. Overall, an artificial neural network receives inputs from other neurons in an analogous way such as the biological nervous system. Moreover, a neural network is said to be *fully or densely connected* when every neuron in each layer of the network is

interconnected to every other neuron in the next layer (Haykin, 2008). A fully connected feed-forward layer with one hidden layer is shown in Fig. 3.3.

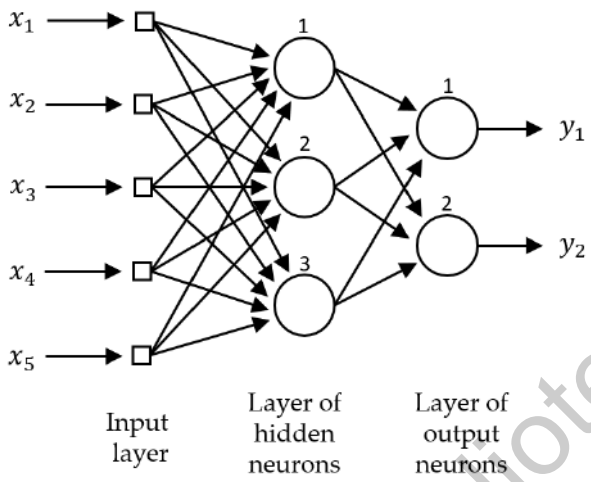


Fig. 3.3: Feed-forward neural network with a single hidden layer. Adapted from (Haykin, 2008).

3.3 Deep learning

Artificial neural networks with more than one hidden layer are referred to as *deep neural networks* or *deep learning algorithms* (see Fig. 3.4). Although the latter is the most general definition of deep learning, it should be noted that not all deep learning models explicitly use neurons as hidden layers (see Section 3.5) (Goodfellow *et al.,* 2016).

As already mentioned before, deep learning is considered a representation learning technique that is capable of breaking down a desired complex mapping into simpler non-linear mappings. Similar to a FFNN (feed-forward neural network) with one hidden layer, the simpler mappings are linked together by composition to approximate the desired transformation (Goodfellow *et al.,* 2016).

The feed-forward neural network illustrated in Fig. 3.3 can be converted into a deep learning model by just adding a hidden layer (see Fig. 3.4). Recall that each of the hidden layers extracts progressively abstract features from the input signal. Overall, the training stage helps the model to determine which parameter values on the hidden layers are useful for explaining the relationships in the input data (Goodfellow *et al.,* 2016).

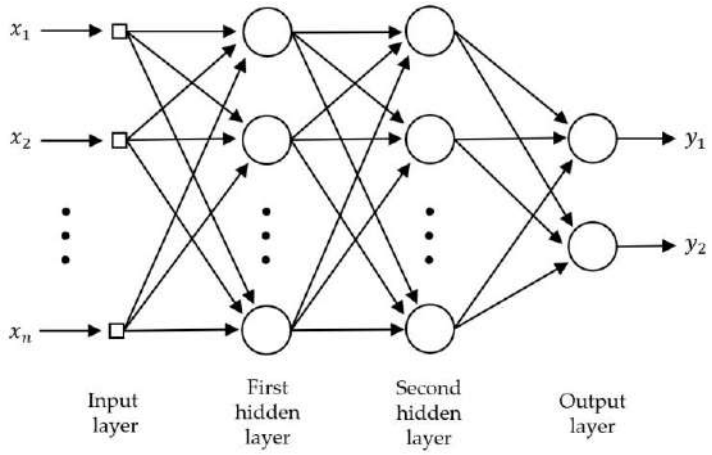


Fig. 3.4: A feed-forward neural network with two hidden layers is the most simple deep learning model. Adapted from (Haykin, 2008).

The procedure to adjust the learnable parameters for any of the previously described neural networks is known as *training phase*. At the same time, this learning process can be carried out under two main different approaches: *supervised* and *unsupervised* (Goodfellow *et al.*, 2016).

On the one hand, the supervised learning approach consists in learning the most appropriate synaptic weights of the model by searching the function that maps an input to an output based on examples or labeled training data that is part of a dataset. Broadly speaking, every sample of the labeled training data is an ordered pair formed by an input object and its corresponding output value (Goodfellow *et al.*, 2016).

In contrast, the unsupervised learning approach takes place when no labeled training data is used for the training phase, that is, the input vectors are known but not their corresponding output values. Although this approach is ultimately gaining some popularity, it is less used compared to the supervised approach. Furthermore, there are some variants of the main approaches such as the *semi-supervised*, *self-supervised* and *weakly-supervised* strategies. Although in essence these novel learning strategies are different from the supervised and unsupervised approaches, they inherit and combine the theoretical principles from those (Goodfellow *et al.*, 2016).

In any case, the learning process can be formulated in terms of the minimization of associated *loss function*. In broader terms, the loss function is an objective function that is used to assess the performance of the model for a given set of the learnable parameter values obtained during the training phase. The error is typically used to measure the performance for artificial neural networks and deep neural networks (Goodfellow *et al.*, 2016).

The minimization of the loss function is carried out by using *optimization algorithms* (e.g., *stochastic gradient descent*, *Adam*, *Adamax*, just to mention a few). Overall, for models with several layers it is required to employ an algorithm known as *backpropagation* to consistently compute the gradients of the model. These gradients are then used by the optimization algorithm to minimize the error and update the learnable parameters values of the model (Goodfellow *et al.*, 2016).

Every time an epoch of the training phase ends, it is highly recommended to evaluate the current trained model on the validation dataset (*validation phase*) in order to continuously analyze its performance and decide whether or not the hyperparameters values are a good selection. Ultimately, the performance to be reported is the one obtained on the validation dataset, which contains samples that are not used for training (Goodfellow *et al.*, 2016).

After the training and validation phases have finished, it is possible to use the obtained model to perform predictions on unseen data that is not available on the selected dataset. This stage is known as the *inference phase* (Goodfellow *et al.*, 2016).

3.4 Convolutional neural networks

Convolutional neural networks (CNNs) are neural networks specially designed for processing grid-structured input data that has noticeable spatial dependencies in local regions (e.g., image data and time-series data). In general, any deep neural network that use a convolutional operation in at least one layer is considered a convolutional neural network. Nonetheless, most of these neural networks use the convolution operation in more than one layer (Aggarwal, 2018).

There are three main operations present in a CNN: *convolution*, *activation functions* and *pooling*. These operations can be thought of as different kind of layers within a convolutional neural network. In this sense, it is very common that most CNNs contain multiple groups of layers, where each group is composed by stacking the operations mentioned above (Aggarwal, 2018).

Other important operations commonly used in current convolutional architectures are *batch normalization* and *drop-out*. These operations are used as *regularization techniques* and will be discussed in section 3.7.

3.4.1 Convolutional layers

A *convolution* is a mathematical operation that quantifies the overlap between two functions $x(\cdot)$ and $w(\cdot)$ when one function is flipped and shifted by t. Such operation is given by (2) (Goodfellow *et al.*, 2016):

$$s(t) = \int x(a)w(t-a)\,da \tag{2}$$

In simpler terms, $s(t)$ represents a weighted average operation that measures the overlap between the two involved functions at a moment t. In machine learning terminology, the function $x(\cdot)$ is referred to as the *input* and the second function $w(\cdot)$ as the *kernel*. The output of this operation is commonly denoted as the *feature map* (Goodfellow *et al.*, 2016).

The convolution operation as used in convolutional neural networks is also often referred to as the *cross-correlation* operation. Now, since the focus of this thesis project is related to image data, it is required to rewrite (2) as the discrete expression given in (3):

$$[\mathrm{S}]_{i,j,d} = \sum_{a=-\Delta}^{\Delta}\sum_{b=-\Delta}^{\Delta}\sum_{c}[\mathrm{X}]_{i+a,j+b,c}[\mathrm{W}]_{a,b,c,d} \tag{3}$$

where W is the convolutional kernel or filter that contains the learnable parameters, X is the input image or input tensor, i denotes the vertical pixel index and j denotes the horizontal pixel index. The indices a, b and c allow to cover both the entire spatial resolution and depth of the input image for an offset Δ. The index d allows to support multiple channels in both the input X and the feature map S (Zhang, Lipton, Li, & Smola, 2021). A basic illustration of the convolution operation applied to a 2D image is shown in Fig. 3.5.

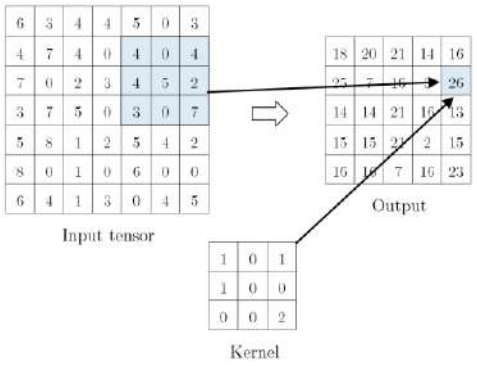


Fig. 3.5: Convolution operation applied to a 2D image. Adapted from (Aggarwal, 2018).

Broadly speaking, the input tensor X is convolved with a kernel W by sliding the latter over each region of the former. During the sliding process, the convolutional operation can be understood as an element-wise product between the filter and the defined region (also called window) on the input tensor, followed by the addition of each result to obtain a single number (see Fig. 3.5) (Aggarwal, 2018).

In general, the convolutional layers enable a process known as *automatic feature extraction* of the input tensor. The latter means that when the convolutional filters are given the appropriate parameters values for a specific task (i.e., classification, regression, etc.), these are capable of extracting the most relevant features from the input tensor to build a hidden representation that contains valuable information of it. In general, filters in the initial layers of a convolutional neural network tend to extract more primitive features, while filters in intermediate or final layers are capable of building complex compositions from those primitive features (Aggarwal, 2018).

On the one hand, the convolutional layers have the effect of reducing the spatial resolution of the input tensor. Moreover, it should be noted that the input tensor and the filter must have the same number of channels. The latter implies that each *k*th slice of the filter operates on the *k*th channel of the input tensor to generate a 'partial' feature map. All resulting 'partial' feature maps are added together in a traditional way to obtain a *single* feature map. Since its common to increase the number of channels during a feature extraction process, it is necessary to apply *n* different filters to the input tensor to produce an output with depth equal to *n* (see Fig. 3.6) (Aggarwal, 2018).

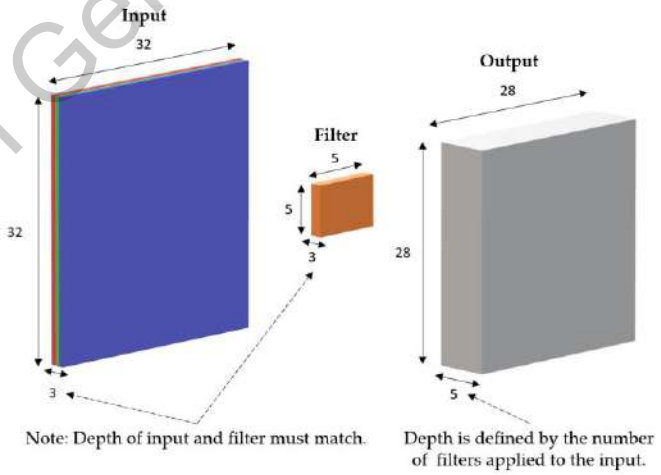


Fig. 3.6: Convolution between an input tensor (32 x 32 x 3) and a kernel of size (5 x 5 x 3). The number of feature maps in the output depends on the number of filters applied. Adapted from (Aggarwal, 2018).

Furthermore, one of the most important properties of a convolutional neural network is its *receptive field*. The receptive field is basically the size of the region in the input that produces the feature map. It is well known that the performance of a CNN can be improved if the size of its receptive field is increased. Overall, a larger receptive field is able to capture complex hidden representations in a larger spatial region of the input image (Aggarwal, 2018).

An important hyperparameter that impacts the size of the receptive field in a CNN is the *stride*. The stride defines the number of pixels the kernel moves when sliding over the input tensor. Overall, its common to select the stride value as 1 or 2 for most applications. For instance, the example shown in Fig. 3.6 uses a stride value of 1. It is also worth mentioning that the reduction in spatial resolution produced by the convolution layer is known to cause the loss of information at the edges of the input image or feature maps. This effect can be decreased by using a technique known as *padding*. This technique consists in adding pixels with a value set to zero all around the edges of the input tensor in order to preserve its spatial resolution after the convolution operation (Aggarwal, 2018).

Another parameter that is commonly used is the *bias*. Similar to conventional artificial neural networks, each convolutional filter can be associated with a bias. Note that the bias value of each filter is also adjusted during the training phase. Finally, the convolutional filters are usually *square* in terms of its spatial dimensions. It is also quite common for the size of a kernel to be small and odd (e.g., 3 or 5). Finally, in order to achieve a clean fit of the convolutional filters with a given stride value, it is highly recommended to work with square images. Otherwise, an additional preprocessing step will be required for proper handling (Aggarwal, 2018).

3.4.2 Activation functions

Similar to traditional ANNs, activation functions are used in convolutional neural networks to introduce *non-linearities* into the linear model. This enables the convolutional neural network's training stage to find a non-linear transformation in the search space (Aggarwal, 2018).

The activation function is applied as an element-wise operation, so it does not change the spatial dimensions of the feature maps. As of today, the most common activation function is called ReLU (Rectified Linear Unit). The main advantages of the ReLU activation function are two-fold: its simplicity that allows fast computations and a reduced likelihood of vanishing gradients (Aggarwal, 2018).

Furthermore, some of its most popular variants, such as Leaky ReLU, improve the ReLU function by preventing the *dying ReLU* problem. The latter refers to the issue when the nodes become irreversibly inactive on any data point (Goodfellow *et al.*, 2016). The functions are shown in Fig. 3.7.

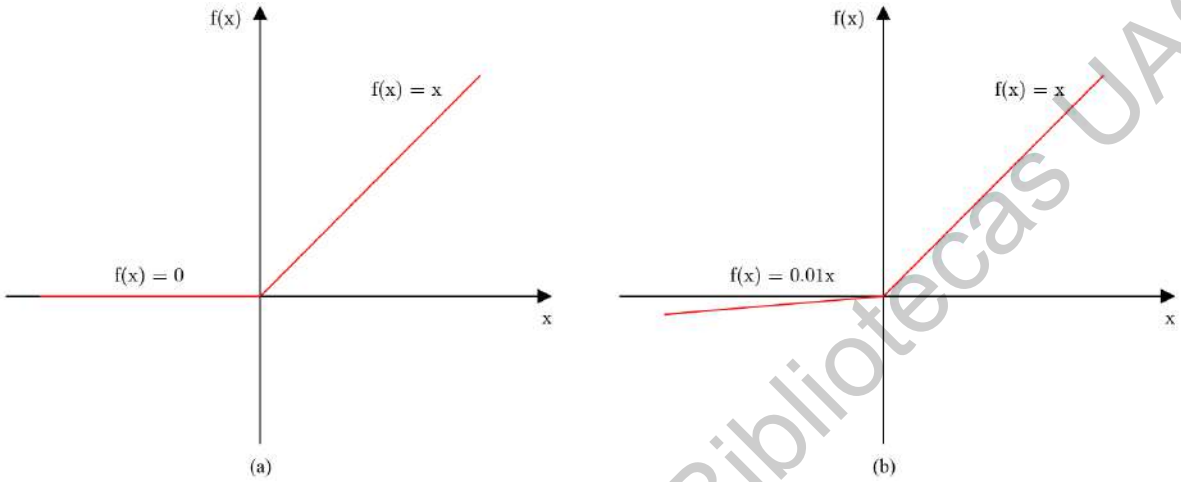


Fig. 3.7: Graphical representation of some activation functions. (a) ReLU and (b) Leaky ReLU.

3.4.3 Pooling layers

Unlike the convolution operation, the *pooling operation* does not modify the number of channels or feature maps. This operation has two purposes: to improve the feature extraction process by alleviating the sensitivity of convolutional layers to location (producing invariance to translation) and to further reduce the spatial resolution of the feature maps (Aggarwal, 2018).

The pooling operation are typically dedicated to calculate either the maximum or the average value in a pooling window. The pooling window refers to a small square region of a feature map. Similar to convolutional layers, the pooling operation consists in sliding the pooling window over all regions in the input feature map from left to right and top to bottom with a specific stride value. It should be noted that this operation works independently on each feature map to generate another feature map (Aggarwal, 2018).

Overall, the *max pooling* operation is widely preferred over the *average pooling* operation for its better performance at feature extraction (Aggarwal, 2018). Fig. 3.8 shows a basic example of max pooling of one feature map with a stride of 2 and its corresponding output.

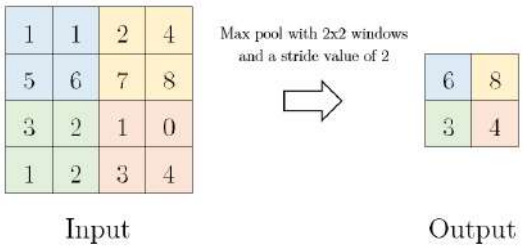


Fig. 3.8: Example of max pooling of one feature map of size 4 x 4 with a stride of 2. Adapted from (Aggarwal, 2018).

3.5 Fully convolutional networks

A *fully convolutional network* (FCN) is a deep learning architecture that only contains convolution, pooling and upsampling operations, i.e., it does not contain any fully connected layer (see Fig. 3.9). This kind of architecture was initially proposed by (Long, Shelhamer, & Darrell, 2015) as a method to solve the semantic segmentation problem. In general, this kind of networks can be used for pixel-wise continuous regression problems.

Note that a fully convolutional network is sometimes referred to as convolutional autoencoder due to the fact that there is an encoder stage followed by a decoder stage (see Fig. 3.9). Nonetheless, the term convolutional autoencoder is more often used to denote a method for visualization through unsupervised learning (Aggarwal, 2018).

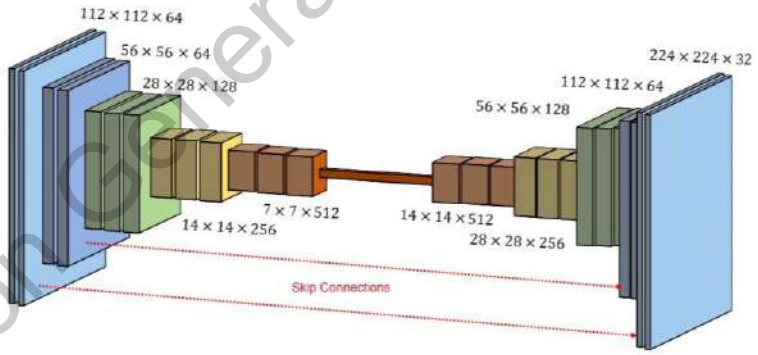


Fig. 3.9: A typical illustration of a fully convolutional network. Adapted from (Long *et al.*, 2015).

The encoder part of the network is basically a feature extractor in charge of obtaining the latent-space representation. This means that any convolutional neural network (without the fully connected layers) can be used. On the other hand, the decoder part consists of several layers of convolution, unpooling and upsampling operations. The objective of these layers is to reconstruct an image or map from the obtained latent-space representation (Aggarwal, 2018).

In particular, FCNs are heavily considered one of the best deep learning-based methods to solve the monocular depth estimation problem since the latter can be directly formulated as a regression problem as shown in section 3.9.

3.6 Low-latency convolutional layers

It is very well known that modern convolutional neural networks have become larger and more complex, which in turn requires much more computational resources not only to train the models but to perform the inference phase. As a consequence, it is almost impossible to deploy state-of-the-art deep learning models on embedded systems whose computational resources (e.g., processing power, RAM, etc.) are very restricted (Howard *et al.*, 2017).

From this perspective, some research in the last few years has focused on the design of low-latency convolutional layers that improve the execution time and memory footprint on resource constrained devices. These research efforts have led to the design of a less expensive operation known as *depthwise separable convolution*. Although this kind of convolution operation has shown promising results, it should be pointed out that the noticeable trade-off between resource efficiency and accuracy still needs to be considered for low-latency models. The latter means that current low-latency deep learning models are not as accurate as the larger and more complex ones (Howard *et al.*, 2017). The following section describes the above-mentioned low-latency convolution operation.

3.6.1 Depthwise separable convolutions

A depthwise separable convolution is a low-latency convolution operation designed to split the standard convolution layer into separate convolution operations. The difference between this and the standard convolution is the number of FLOPs, i.e., the number of multiply-adds operations associated with each approach. In general, a depthwise separable convolution reduces significantly the number of learnable parameters of a standard convolution by factoring the latter into simpler operations (Howard *et al.*, 2017).

Nonetheless, as the number of learnable parameters of a deep learning model decreases, its performance is known to decrease as well. In order to take the maximum advantage of a low-latency model, it is needed to find by experimentation the best hyperparameters that exploit the performance of the model (Zhao *et al.*, 2020).

The separate convolution operations into which a depthwise separable convolution is divided are described as following:

- Depthwise convolution

A *depthwise convolution* reduces the spatial resolution but does not change the depth of the resulting hidden representation. Basically, a kernel of size F_q x F_q x 1 iterates independently on a single channel of the input tensor. The obtained independent feature maps are concatenated together to create the output (Howard *et al.*, 2017).

- Pointwise convolution

A *pointwise convolution* helps to increase the number of channels without impacting the spatial resolution of the resulting hidden representation. In this case, n different kernels of size 1 x 1 x d_q operate on the input tensor. The latter means that each filter has the same depth d_q as the input tensor and iterates through every single pixel of it. Similar to the standard convolution, the pointwise convolution uses n different kernels to create an output of n feature maps (Howard *et al.*, 2017).

As will be explained in the following chapter, these low-latency convolution operations can be used together with other novel operators, such as *channel split* and *channel shuffle* to significantly reduce the network latency (Ma, Zhang, Zheng, & Sun, 2018).

3.7 Regularization techniques

Regularization techniques are used to reduce overfitting during the learning process. This allows the deep learning model to improve its performance on unseen data (Aggarwal, 2018). The most common techniques are described below.

3.7.1 Data augmentation

One of the strongest prerequisites for deep learning is having access to a large-scale dataset. The latter is crucial for a deep neural network to improve its ability to generalize on new data. In this sense, *data augmentation* provides a way to expand the existing dataset by generating new training samples through the transformation of the original samples. The transformation of the original samples consists in applying operations such as flipping, rotation, cropping, translation and scaling (Aggarwal, 2018).

3.7.2 Batch normalization

This technique is implemented as a distinct operation layer within a convolutional neural network. It consists in standardizing the input tensors for each mini-batch in order to adjust the magnitudes of the activations. This adjusting is known to improve the behavior of the gradient updates. The batch normalization operation is performed after the convolution operation and before the activation function (Aggarwal, 2018).

3.7.3 L2 regularization

Also known as *weight decay*, this regularization technique is applied after the gradient updates step. It consists in forcing the values of the learnable parameters to be small but different from zero. It can be considered as a set of penalty values that are added to the loss function that is being optimized during the training phase (Aggarwal, 2018).

3.7.4 Dropout

Similar to the batch normalization technique, this regularization method is implemented as an additional layer that randomly sets inputs to zero during the training phase in order to prevent a phenomenon known as feature co-adaptation that occurs when two or more filters detect the same features repeatedly. In general, dropout avoids this problem by forcing the model to perform the forward propagation using only a subset of the inputs (Aggarwal, 2018).

3.8 Transfer learning

Transfer learning is a deep learning technique where a pretrained model on one dataset is used to initialize the filters of the model that is to be trained on a different dataset (regardless of the type of task). This technique is well-known to improve not only the training time but also the performance of the model on the objective task (Aggarwal, 2018).

According to (Goodfellow *et al.,* 2016), the concept of transfer learning can be defined as a "Situation where what has been learned in one setting is exploited to improve generalization in another setting". In the context of machine learning, *generalization* refers to the model ability to perform a prediction in an expected way for any unseen sample obtained from the same distribution as the one used to train the machine learning model. Both concepts, transfer learning and generalization, represent two of the most important topics in machine learning research (Goodfellow *et al.,* 2016).

3.9 Monocular depth estimation

According to (Bhoi, 2019), monocular depth estimation is a pixel-wise continuous *regression problem* that can be formulated as follows:

Let I be the space of RGB images and $\mathcal{D}$ the codomain of real-valued depth maps. Given a training dataset

$$\mathcal{T} = \{(\mathbf{I}_i, \mathbf{D}_i)\}_{i=1}^{M}, \mathbf{I}_i \in I \text{ and } \mathbf{D}_i \in \mathcal{D}, \quad (4)$$

the objective is to find the non-linear transformation stated by (5):

$$\varphi: I \to D . \quad (5)$$

This formulation assumes the availability of a pixel-wise ground truth, i.e., a large training dataset of RGB-depth pair images. Finally, it should be noted that the latter is only applicable to the supervised learning approach, which is used in this thesis.

3.9.1 Benchmark datasets

As with any research topic in deep learning, the monocular depth estimation task requires large amounts of data to train a model. For this research topic, there are two standard and widely used datasets to train and evaluate the performance of a deep learning model under a specific learning strategy.

It is also important to consider that these datasets are divided according to an official split to consistently compare the different studies on this research topic. Both datasets are described as following.

- **NYU-Depth V2**

The NYU-Depth V2 dataset was proposed by (Silberman, Hoiem, Kohli, & Fergus, 2012). This dataset focuses on indoor scenes and was collected by means of an RGB-D camera (Microsoft Kinect sensor). It is the common benchmark for supervised monocular depth estimation.

This dataset contains 120,000 samples for training and 654 samples for validation with a resolution of 640 x 480 pixels. Due to the size of the training set, it is infeasible to perform the k-fold cross-validation technique. The next chapter describes this dataset in more detail.

- **KITTI**

The KITTI dataset was proposed by (Geiger, Lenz, Stiller, & Urtasun, 2013). This dataset only contains outdoor scenes and its ground truth images were collected using a LIDAR sensor (Velodyne 3D). In contrast to the NYU-Depth V2 dataset, this dataset is the common benchmark for unsupervised and semi-supervised monocular depth estimation.

An official split of this dataset is known as Eigen split (Eigen *et al.*, 2014), which contains 22,600 samples for training and 697 samples for validation with a resolution of 1224 x 368 pixels. Similar to the NYU-Depth V2, it is impractical to perform k-fold cross-validation due to the large size of the training set.

CHAPTER 4

Methodology

In this chapter, the implemented methodology for this research project is described in detail. Considering that the *base concept* of this project is the design and implementation of a low-latency fully convolutional network architecture for the pixel-wise continuous regression task known as monocular depth estimation, this chapter presents the development, training and validation of such an architecture. A simplified illustration of the implemented methodology is shown in Fig. 4.1.

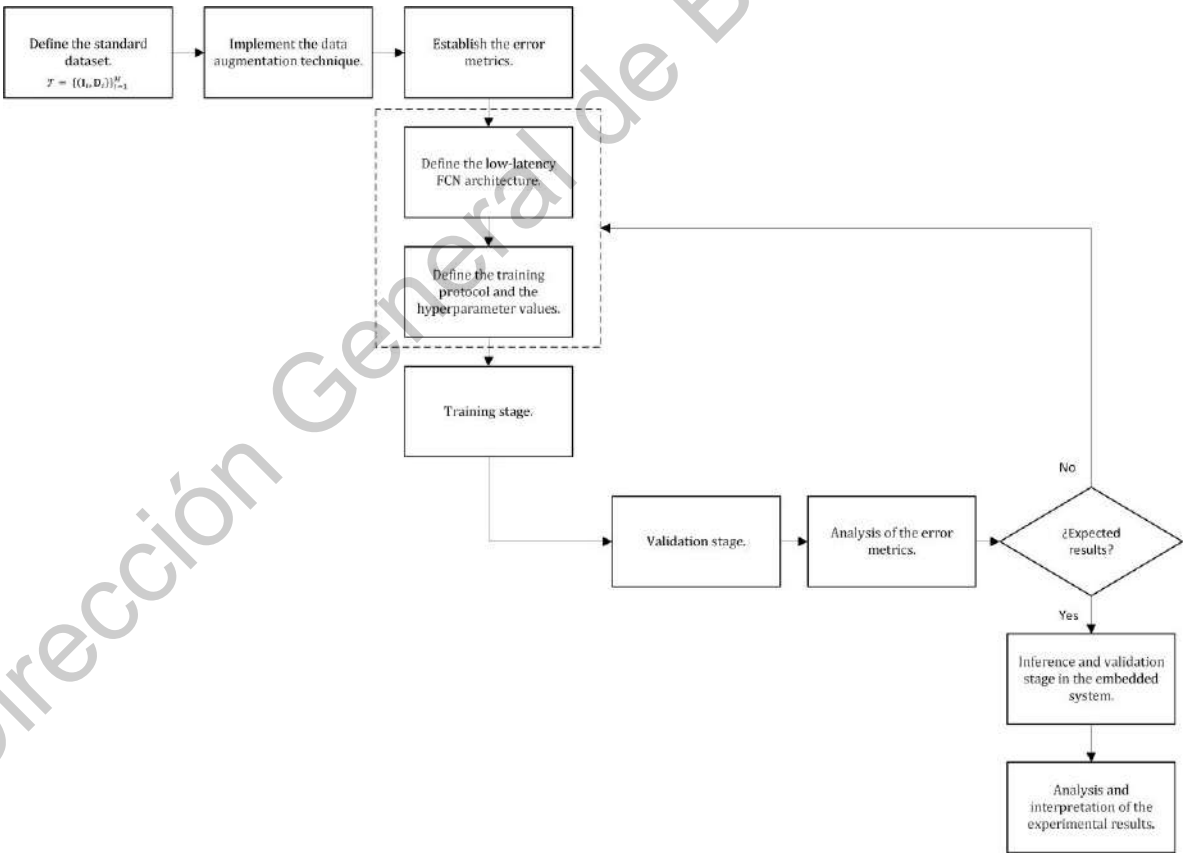


Fig. 4.1: Flow diagram of the implemented methodology.

4.1 Dataset definition

As mentioned in the previous chapter, the state-of-the-art architectures for monocular depth estimation are trained on standard datasets that are separated according to an official split (i.e., specific samples are for training and others for validation). The latter allows to compare the different studies on this research topic in a consistent manner.

In this sense, the selected dataset to train and validate the proposed low-latency FCN architecture is the standard **NYU-Depth V2** dataset. This dataset is publicly available since 2012 and contains RGB images of indoor scenes with their corresponding relative depth maps. It has been widely used in state-of-the-art studies related to supervised monocular depth estimation (Silberman *et al.*, 2012). Some samples of the NYU-Depth V2 dataset are shown in Fig. 4.2.

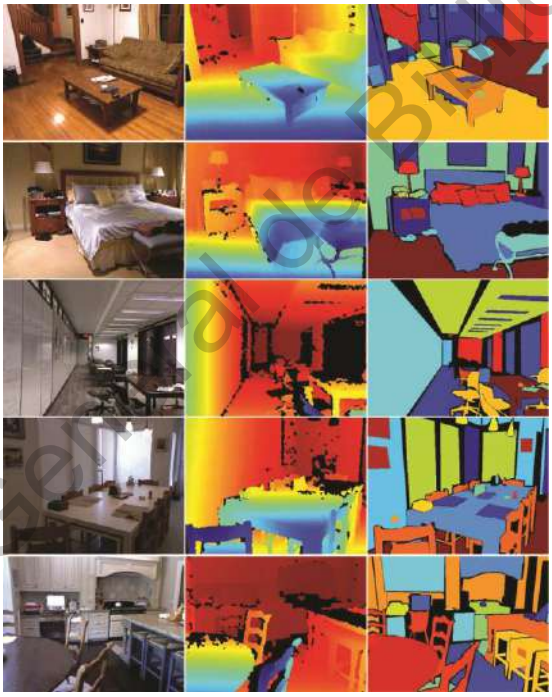


Fig. 4.2: Samples of the NYU-Depth V2 dataset (Silberman *et al.*, 2012).

Although the official dataset contains 120,000 training samples, many studies have shown that using a subset of 50,000 samples for training does not negatively impact the network performance but it can dramatically reduce the training time. On the other hand, the dataset contains 654 samples for validation. The original samples with resolution of 640 x 480 are downsampled to 224 x 224 for direct comparison purposes with a state-of-the-art work (Silberman *et al.*, 2012).

4.2 Data augmentation

As part of the implemented regularization techniques, data augmentation is performed before the training stage in order to artificially expand the number of training samples during the data loading process. Recall that data augmentation refers to (online) affine transformations applied to the training data (NYU-Depth V2). The data augmentation policy is similar to (Eigen *et al.*, 2012) and is described as following:

- **Scaling**. Input RGB images and target depth maps are scaled. Input images are scaled by a uniform random number $s \in [1, 1.5]$, while depth maps are divided by s.

- **Rotation**. Input RGB images and target depth maps are rotated by a uniform random number (degrees of rotation) $r \in [-5,5]$.

- **Translation.** Input RGB images and target depth maps are randomly cropped to 228 x 304.

- **Flips.** Input RGB images and target depth maps are both horizontally flipped with a probability of 0.5.

- **Color jittering.** Only for input RGB images: the brightness, contrast, saturation and hue are randomly changed.

- **Color normalization**. Input RGB images are standardized through the z-score normalization technique (the mean is subtracted to each value and the result is divided by the standard deviation).

Finally, the input RGB images and target depth maps are cropped to 224 x 224 in order to match the input layer size of the deep learning architecture.

4.3 Error metrics

In order to quantitatively evaluate a deep neural network for a pixel-wise continuous regression task, it is required to define the appropriate metrics for such task. For almost any research topic, the state-of-the-art methods usually indicate which error metrics have been found to be the most useful for evaluation and comparison purposes to related methods.

Taking into account the latter, this thesis project uses the following error metrics for model evaluation:

- Root mean squared error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{p=1}^{n}\left(y_p - \hat{y}_p\right)^2} \tag{6}$$

- Absolute relative error:

$$\text{Abs rel} = \frac{1}{n}\sum_{p=1}^{n}\frac{\left|y_p - \hat{y}_p\right|}{\hat{y}_p} \tag{7}$$

- Threshold accuracy (δ_1):

$$\delta_1 = \%\text{ of } y_p \text{ s.t.} \max\left(\frac{y_p}{\hat{y}_p}, \frac{\hat{y}_p}{y_p}\right) < 1.25 \tag{8}$$

  The threshold accuracy δ_1 should be understood as the percentage of pixels of the predicted depth map $\hat{y}$ for which the relative error is less than or equal to 25%.

For all the above-described continuous metric equations, y_p is a pixel of the target depth map y, $\hat{y}_p$ is a pixel in the predicted depth map $\hat{y}$ and n is the total number of pixels in the depth map $\hat{y}$. Furthermore, the proposed method is compared to a related state-of-the-art method through the of the average latency (in milliseconds) associated to the inference time.

It should be noted that RMSE is considered the main error metric for evaluating monocular depth estimation methods mainly because it has become the standard metric to validate regression models. A reason for its popularity lies in its ability to give a relatively high weight to large error values due to the fact that errors are squared first before averaging. According to Wofk (2020), RMSE provides a much more intuitive way to determine the performance of the monocular depth estimation method when compared to the threshold accuracy δ_1 simply because the former does not rely on a fixed relative error threshold. For the reasons stated above, throughout this thesis it is assumed that RMSE is the error metric with the highest precedence when evaluating the performance of any proposed low-latency fully convolutional network.

4.4 Baseline FCN architecture

As already stated, the fully convolutional network (FCN) architecture is currently considered the preferred approach for solving the monocular depth estimation task. In particular, the main focus of this thesis project is to develop a novel low-latency FCN architecture and perform its deployment on an embedded system.

Since the research topic associated to the design of low-latency deep learning-based methods for monocular depth estimation is still in its infancy and therefore may be considered a non-trivial task, it was decided to choose a baseline FCN state-of-the-art architecture for analysis in order to detect the possible areas of improvement. Throughout the following sections of this chapter, the motivation behind every proposed change is described in a detailed manner. The baseline FCN architecture is shown in Fig. 4.3.

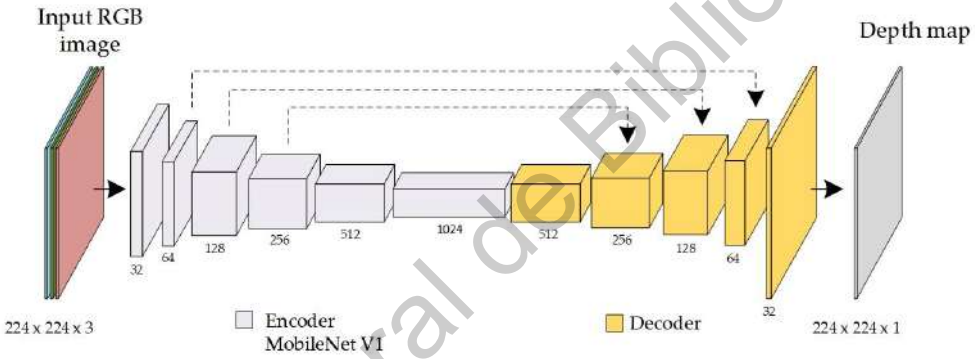


Fig. 4.3: Baseline FCN architecture. Adapted from (Wofk *et al.*, 2019).

As mentioned in the previous chapter, the FCN architecture is often referred to as an encoder-decoder architecture since the input image is initially fed to an encoder network and the resulting *latent space representation* is then fed to the decoder network that uses the extracted features to reconstruct the associated relative depth map. The selected baseline architecture is the one proposed in (Wofk *et al.*, 2019) that uses the MobileNet V1 feature extractor as the encoder network. This feature extractor network has been recognized as one of the first convolutional neural networks to address the latency problem on mobile devices for classification tasks on the computer vision domain (Howard *et al.*, 2017).

The MobileNet V1 network consists essentially of depthwise separable convolutions with depthwise and pointwise layers followed by batch normalization and ReLU layers. In this regard, the depthwise decomposition has become the common approach to significantly reduce the latency of a deep convolutional neural network. Recall that the depthwise decomposition enables the factorization of a conventional convolutional layer through the

use of n different depthwise kernels of size F_q x F_q x 1 (each one applied independently on each channel of the input tensor) followed by a 1 x 1 x d_q pointwise kernel (Howard *et al.*, 2017).

Although the MobileNet V1 with a depth multiplier equal to 1 (default size) is considered a milestone within the low-latency deep learning research, Howard *et al.* (2017) claimed this network could be further optimized by using low-dimensional tensors in order to reduce the number of learnable parameters, memory access cost and FLOPs. Nonetheless, applying a depthwise convolutional layer to low-dimensional tensors is limited in the amount of information it is able to extract. In this regard, there have been several proposals ranging from using expand/projection layers or instead, novel operators to replace certain convolution layers. On the one hand, the expand/projection layers allow to decompress the hidden representation before a depthwise separable convolution is applied to it. In this way, the feature extraction process is performed using the large tensor representation before a projection layer compresses the data again (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018).

The approach using expand/projection layers led to the next generation known as MobileNet V2. Although their proposed mechanism was proved to work as expected on computer systems with limited resources, Ma *et al.* (2018) found that the MobileNet V2 violates several design principles for low-latency deep learning architectures. In this regard, these design principles showed that some novel operators could be used together with low-latency convolution operations instead in order to replace certain layers in a network and thus, to reduce the number of learnable parameters. These guidelines were proposed in (Ma *et al.*, 2018) and state the following:

1. The number of channels at both ends of a layer should be kept the same to minimize the cost of memory access (MAC).
2. Overuse of convolution groups should be avoided as this increases the memory access cost.
3. It is suggested to minimize the use of fragmented operators to maintain a high degree of parallelism.
4. It is recommended to reduce the use of element-wise operations since its computational cost is not negligible.

As already described, the MobileNet V2 network uses an expand/projection layer which is also known as an inverted residual layer with linear bottleneck. According to (Ma *et al.*, 2018), this proposed layer violates the guideline number 1 (i.e., G1) as it does not preserve

the number of input and output channels to be equal. Moreover, the MobileNet V2 uses depthwise decomposition followed by ReLU activation functions on thick feature maps which violates the guideline number 4 (i.e., G4).

As a result of those guidelines, Ma *et al.* (2018) designed and proposed a novel low-latency feature extractor architecture for classification tasks named as ShuffleNet V2. On the one hand, the ShuffleNet V2 encoder can be considered one of the best low-latency feature extractors for classification tasks on mobile devices. According to the authors, this feature extractor was designed in such a way that the number of multiply-adds operations (FLOPs), memory access cost (MAC) and the number of learnable parameters were minimized. Table 4.1 shows a comparison between different low-latency encoders (Ma *et al.*, 2018).

Table 4.1: Complexity metrics for different encoders (depth mult. = 1.0x) with dense layers.

Architecture	FLOPs [G]	Weights [M]
MobileNet V1	0.569	4.2
MobileNet V2	0.300	3.4
ShuffleNet V2	0.146	2.3

Although the different encoders that have been described are considered to belong the low-latency category, Table 4.1 shows significant difference between the models for two of the most important complexity metrics (i.e., number of weights and FLOPs). On the one hand, FLOPs represent the number of multiply-adds operations that is carried out by a model. In contrast, the number of weights represent the number of learnable parameters a model has. In general, a higher value for each metric indicates a more complex and slower model.

However, it should be noted that Ma *et al.* (2018) also found that there are cases for which two different convolutional networks have similar FLOPs but have different speeds. Overall, they argued that is much better practice to use direct metrics (e.g., latency and MAC) instead of indirect metrics such as FLOPs for the design of low-latency convolutional neural network architectures. In particular, the ShuffleNet V2 efficient performance is mostly due to the incorporation of two novel operators named as *channel split* and *channel shuffle* which will be described in the next section in a detail manner.

All of the above represent and summarizes the motivation behind the selection of the ShuffleNet V2 feature extractor as the backbone for the monocular depth estimation task, on which it has not been tested yet as the encoder.

4.5 Selected encoders

This section describes in detail the two different low-latency feature extractors that were selected as the encoders for the monocular depth estimation task. Since none of these were designed specifically for the above-mentioned task, the ultimate purpose is to determine which of these contribute the most in terms of performance for such a pixel-level regression task.

4.5.1 ShuffleNet V2

As already stated, the ShuffleNet V2 feature extractor is selected as one of the encoders to be used as part of the proposed low-latency FCNs models. To the best knowledge of the author, this low-latency feature extractor has not been tested for the monocular depth estimation task. Overall, the ShuffleNet V2 employs two novel operators that can be used in conjunction with depthwise decomposition to further reduce the number of learnable parameters of a convolutional neural network (Ma *et al.*, 2018).

Building block

Similar to most feature extractors, the ShuffleNet V2 has a main building block from which the network is built upon. The main building block is shown in Fig. 4.4.

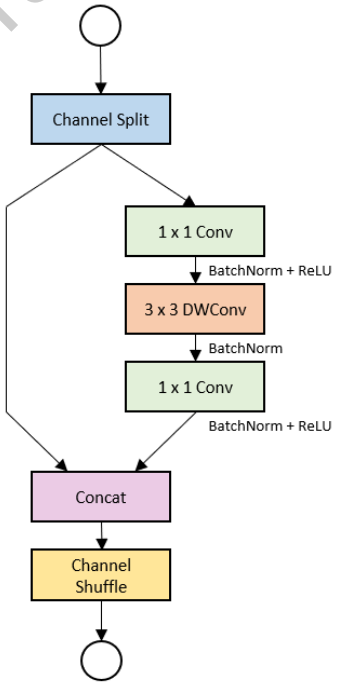


Fig. 4.4: Main building block of the ShuffleNet V2. Adapted from (Ma *et al.*, 2018).

On the one hand, the channel split operator was designed to replace the group convolutions to minimize memory access cost. As shown in Fig. 4.4, the channel split operator receives the input tensor (input feature maps) with n channels and divides it into two different branches with $n - n'$ and n' channels, respectively. While one of these branches performs the identity operation, the other one performs three convolutions with the same number of input and output feature maps in order to satisfy the guideline number one of the above-mentioned numbered list. In contrast with other approaches, the pointwise convolutions (i.e., 1x1) of the building block shown in Fig. 4.4 are not group-wise, a design decision that matches with the guideline number two. Despite the absence of a group-wise operation, it should be noted that the channel split operator produces itself two groups of feature maps which in turn allows to reduce the number of learnable parameters since only half of the input feature maps is processed (Ma *et al.*, 2018).

After the convolution operations, the feature maps are concatenated and finally shuffled by means of the channel shuffle operator. The main purpose of using the channel shuffle operator is to allow an information communication process between the two branches. The information flow between two or more channel groups is considered critical since it has been shown to have an impact in the performance of the network. As described in (Zhang, Zhou, Lin, & Sun, 2018), the channel shuffle operator allows to build a stronger hidden representation by relating the outputs of any channel group to the inputs of any other channel group. The latter represents an advantage that has led to several achievements within the low-latency convolutional network design (Ma *et al.*, 2018). An illustration of the channel shuffle operator is shown in Fig. 4.5.

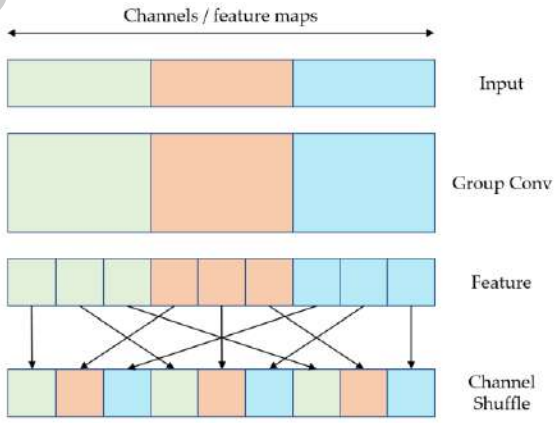


Fig. 4.5: Channel shuffle operator. Adapted from (Zhang *et al.*, 2018).

According to the authors, their approach consists on using the above-described channel operators together with depthwise convolutions to achieve an efficient design. As already explained before, the results obtained also allowed them to present four practical guidelines for hand-crafted design of low-latency architectures. Their main suggestion is that low-latency design should consider also the latency on specific devices (i.e., the impact of MAC in the execution time) and not only the number of FLOPs (Ma *et al.*, 2018).

The authors also found that their ShuffleNet V2 model is capable of outperforming other feature extractors especially under limited computational resources. For instance, they noted that MobileNet V2 has a lower performance due to the fact that it uses too few channels with the aim of reducing its complexity. In contrast, the efficient design of the ShuffleNet V2 model allows using a higher number of channels as the hidden representation and therefore, is capable of reusing and preserving the most important features of an input image (Ma *et al.*, 2018).

Generally speaking, the main building block shown in Fig. 4.4 is the basic structure that can be used repeatedly to build larger blocks known as 'stages' as illustrated in Table 4.2. Some other layers are exempt from using the main building block and instead, use standard convolutions or pooling operations (Ma *et al.*, 2018).

Table 4.2: Architecture details of the ShuffleNet V2 (1.0x model) (Ma *et al.*, 2018).

Layer	Output size	Output channels
Input	224 x 224	3
Conv1	112 x 112	24
MaxPool	56 x 56	24
Stage2	28 x 28	116
Stage3	14 x 14	232
Stage4	7 x 7	464
Conv5	7 x 7	1024
GlobalPool	1 x 1	-
FC	-	1000

Similar to other architectures, the ShuffleNet V2 was designed specifically for classification tasks and therefore has two distinctive layers at the end for that purpose. It is important to note that these layers should not be considered for a fully convolutional network.

4.5.2 MobileNet V3

In order to allow a thorough comparison, it was also determined to include an additional state-of-the-art feature extractor as part of the experimentation. In this sense and following the above discussion related to the MobileNet-based architectures, it was decided to use the next generation of the MobileNet family: the **MobileNet V3** architecture.

Even though the MobileNet V3 is the latest version of MobileNet, its design is mostly inspired on a low-latency feature extractor known as MnasNet-A1, the smaller version of the MnasNet architecture, and which in turn was inspired on the MobileNet V2. In other words, the MnasNet design approach was used as the baseline for the MobileNet V3. The obtained network was later improved by hand and refined using specialized algorithms such as NetAdapt. To provide some context, both the MnasNet and MobileNet V3 networks are low-latency architectures that were found through a sophisticated technique known as *neural architecture search* (NAS). The latter is an automated method for designing artificial neural networks on a specific task and dataset. Its main drawback is that it requires a fairly large amount of computing power. For instance, the engineers that developed MnasNet reported their automated search to last a total of 288 days of execution on highly specialized Google TPU processors (Howard *et al.*, 2019).

Overall, the MobileNet V3 architecture can be considered as a hand-crafted improvement over MobileNet V2 in the following aspects:

- Some layers of the feature extractor are upgraded with a modified swish non-linearity known as *h-swish*. The original swish non-linearity is a relatively novel activation function that can be used as a drop-in replacement for ReLU. According to (Howard *et al.*, 2019), the swish non-linearity is capable of improving the accuracy of the model at the expense of a higher computational cost in embedded devices. To deal with that problem, they replaced the sigmoid function with a simpler equivalent function and only used the activation function in the deeper layers of the network where they found the computational cost to be reduced (Howard *et al.*, 2019).

- Lightweight attention modules based on the squeeze-and-excite block were included into the expand/projection layers that were first introduced in MobileNet V2. These modules implement an attention mechanism that allow the network to extract only the most relevant features for the task at hand (Howard *et al.*, 2019).

- In contrast to earlier versions of the MobileNet family, the MobileNet V3 is the first low-latency architecture to suggest the use of hardware-aware NAS algorithms together with hand-crafted network design principles as a way to harness the advantages from both approaches (Howard *et al.*, 2019).

Building block

Similar to the ShuffleNet V2, the MobileNet V3 has also a main building block from which the network is built upon. The main building block is shown in Fig. 4.6.

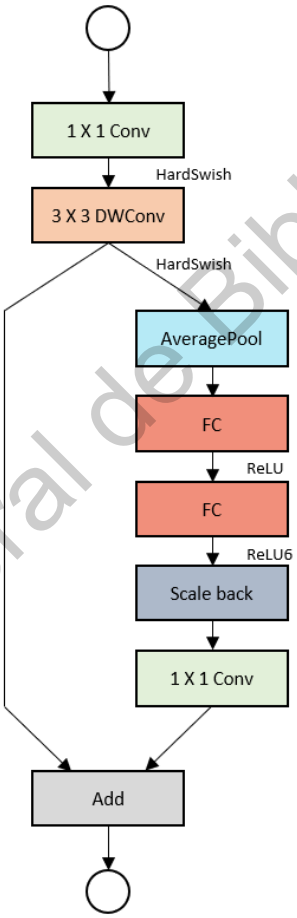


Fig. 4.6: Main building block of the MobileNet V3. Adapted from (Howard *et al.*, 2019).

As shown in Fig. 4.6, the MobileNet V3 begins with a 1x1 pointwise convolutional block that has 16 filters. In contrast to MobileNet V1 and V2, the authors in (Howard *et al.*, 2019) found out that using 32 filters for initial layers (e.g., input layer for a 224 x 224 x 3 tensor) is too expensive, despite having a small number of parameters. Their experiments showed

that the use of 16 filters is sufficient without affecting the performance of the model, but reduces the number of multiply-adds operations.

After the regular convolution, the non-linear *h-swish* activation function is applied to the obtained hidden representation. The h-swish activation function is given by the following expression:

$$\mathrm{h-swish}[x] = \; x\left[\frac{\mathrm{ReLU6}(x+3)}{6}\right] \tag{9}$$

where $\mathrm{ReLU6}(\cdot)$ is equal to $\min(\max(0, x), 6)$. The plot for this activation function is shown in Fig. 4.7.

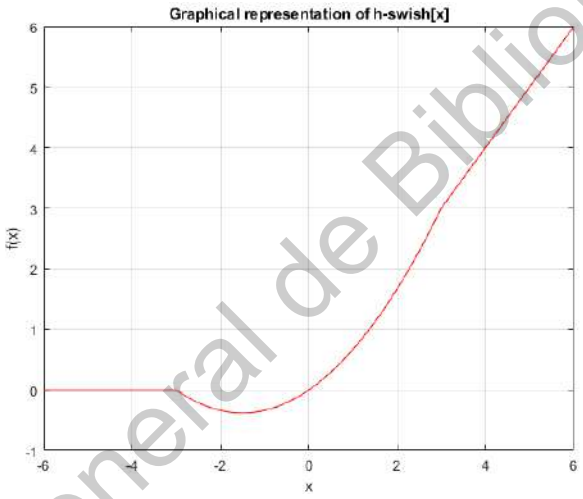


Fig. 4.7: Graphical representation of the h-swish activation function.

Following the structure in Fig. 4.6, the output is fed into a depthwise convolutional layer to further reduce its spatial resolution while extracting relevant features in an efficient manner and without changing the number of feature maps. The h-swish mapping is then applied to the output feature maps to introduce the required nonlinearities (Howard *et al.*, 2019).

As depicted in Fig. 4.6, the hidden representation is sent through two branches. While the left-hand side branch is maintained intact, an average pooling operation is applied to the right-hand side branch before processing the hidden representation with two consecutive dense layers (one that uses a simple ReLU and other that uses a ReLU6). The resultant hidden representation is scaled back before a regular convolutional layer is applied. Finally, the output from the two branches is added (Howard *et al.*, 2019).

Overall, the main building block shown in Fig. 4.6 serves as a predefined structure (known as bottleneck with residual or simply abbreviated as 'bneck') that is used repeatedly in order to construct the complex architecture that is given in Table 4.3. Note that this architecture has some layers that do not use the above-mentioned building block but implement a single specific operation (Howard *et al.*, 2019).

Table 4.3: Architecture details of the MobileNet V3 (1.0x large model) (Howard *et al.*, 2019).

Layer	Output size	Output channels
Input	224 x 224	3
conv2d	112 x 112	16
bneck, 3x3	112 x 112	16
bneck, 3x3	56 x 56	24
bneck, 3x3	56 x 56	24
bneck, 5x5	28 x 28	40
bneck, 5x5	28 x 28	40
bneck, 5x5	28 x 28	40
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	112
bneck, 3x3	14 x 14	112
bneck, 5x5	7 x 7	160
bneck, 5x5	7 x 7	160
bneck, 5x5	7 x 7	160
conv2d, 1x1	7 x 7	960
pool, 7x7	-	-
conv2d 1x1, NBN	-	-
conv2d 1x1, NBN	-	-

Similar to other state-of-the-art feature extractors, the MobileNet V3 uses a tunable parameter known as depth multiplier. By modifying this parameter, the number of filers used throughout the network can be increased or decreased and therefore, the size of the model varies accordingly. Overall, the latter means that this architecture defines a family of models and not just a single one (Howard *et al.*, 2019).

4.6 Decoder

The decoder is the final stage that allows the reconstruction of the relative depth map. In particular, it takes as input the *latent space representation* obtained through the encoding stage. From this perspective, the decoder is responsible for gradually increasing the spatial resolution and reducing the number of channels associated to the hidden representation. In contrast to encoders, there are no state-of-the-art low-latency decoders architectures from which to select for our experimentation purposes (Long *et al.*, 2015).

Since the design of low-latency decoders from scratch is out of the scope of this thesis, it was decided to use a depthwise separable convolution layer as the main building block, interleaved with a simple interpolation operator. This decoding approach is inspired by the low-latency decoder proposed in (Wofk *et al.*, 2019) and can be considered one of the simplest, yet most efficient upsampling methods to reconstruct a dense map, which in this case, is a relative depth map. The architecture details of a low-latency decoder are given in Table 4.4.

Table 4.4: Architecture details of a generic low-latency decoder.

Layer	Output size	Output channels
Latent space (input)	7 x 7	1024
Dec_dwconv1	7 x 7	464
Dec_dwconv2 + interpolat.	14 x 14	232
Dec_dwconv3 + interpolat.	28 x 28	116
Dec_dwconv4 + interpolat.	56 x 56	64
⋮	⋮	⋮
Dec_dwconv$n$ + interpolat.	224 x 224	c_{out}
Dec_pwconv	224 x 224	1

As shown in Table 4.4, the generic structure of a low-latency decoder is made up of n layers of *depthwise separable convolutions* followed by a single *pointwise convolution*. The depthwise separable convolutions layers uses a F_q x F_q x 1 kernel and may perform interpolation using the nearest neighbor method if required (e.g., the second layer of the decoder example does not perform interpolation). A single pointwise convolution is applied to the last hidden representation (224 x 224 x c_{out}) to reduce the number of channels and obtain the corresponding relative depth map. It is important to note that the depth and structure of the decoder needs to be defined based on the coupling requirements with the selected encoder (Wofk *et al.*, 2019).

In this sense, the decoder implementation should not only allow the ability to add and remove any number of layers, but also to define the number of input/output channels of the *k*-th layer. The latter allows the implementation of additive residual connections between the encoder and decoder. These residual connections allow to reduce the vanishing gradient problem and the subsequent loss of sharpness on the resultant depth map (Alhashim & Wonka, 2018).

4.7 Low-latency FCN architectures

After defining the encoding and decoding architectures separately as described in the previous section, the following step is to dock these architectures together to construct the low-latency FCN models suitable for monocular depth estimation.

It should be noted that before the coupling procedure, there is a critical action item related to the encoder architecture that needs to be resolved first. The latter refers to the fact that most of the state-of-the-art pretrained encoders are currently being used solely for *classification tasks* and not for *pixel-wise regression tasks*, and therefore, those architectures need to be modified for a proper docking with a decoder. More specifically, some layers (i.e., global pooling and fully connected) need to be removed in order to keep the latent space representation intact and ready to be fed into the decoding stage.

4.7.1 ShuffleNet-based model

After modifying the ShuffleNet V2 architecture shown in Table 4.2 and coupling it with a customized decoder based on the structure given in Table 4.4, the low-latency FCN architecture shown in Fig. 4.8 was obtained.

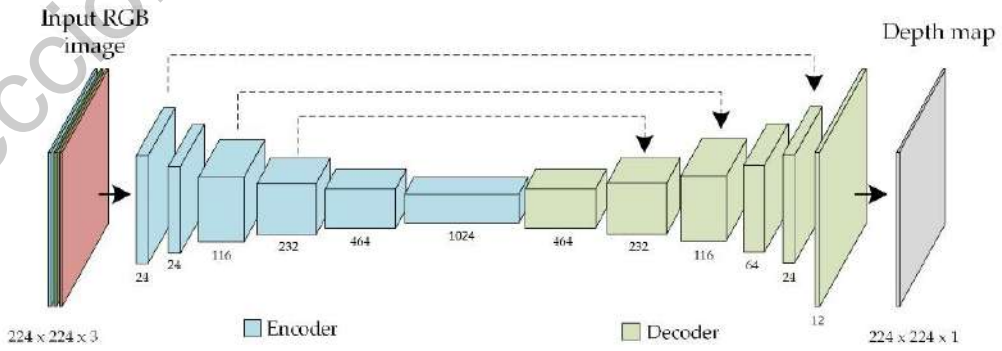


Fig. 4.8: Proposed low-latency ShuffleNet-based model (*Shuff-dw-res-1.0*).

As shown in Fig. 4.8, three residual additive connections between the encoder and decoder were added, which are illustrated as dotted arrows. The architecture details are shown in Table 4.5.

Table 4.5: Architecture details of the proposed low-latency ShuffleNet-based model.

Layer	Output size	Output channels
Input	224 x 224	3
Conv1	112 x 112	24
MaxPool	56 x 56	24
Stage2	28 x 28	116
Stage3	14 x 14	232
Stage4	7 x 7	464
Conv5	7 x 7	1024
Dec_dwconv1	7 x 7	464
Dec_dwconv2 + interpolat.	14 x 14	232
Dec_dwconv3 + interpolat.	28 x 28	116
Dec_dwconv4 + interpolat.	56 x 56	64
Dec_dwconv5 + interpolat.	112 x 112	24
Dec_dwconv6 + interpolat.	224 x 224	12
Dec_pwconv	224 x 224	1

4.7.2 MobileNet-based model

After removing the last layers of the MobileNet V3 architecture, it was possible to couple its initial layers with a symmetrical decoder based on the structure given in Table 4.4. The resultant low-latency FCN architecture is shown in Fig. 4.9.

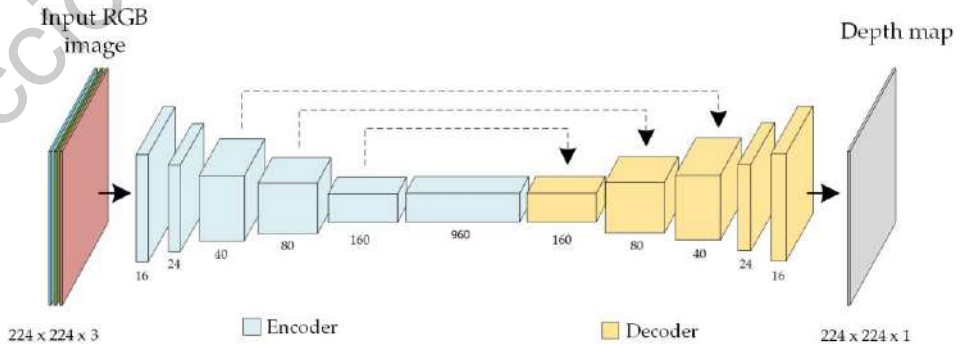


Fig. 4.9: Proposed low-latency MobileNet-based model (*Mob-dw-res-1.0*).

Similar to the ShuffleNet-based model, three residual additive connections between the encoder and decoder were added, which are depicted as dotted arrows in Fig. 4.9. The architecture details are shown in Table 4.6.

Table 4.6: Architecture details of the proposed low-latency MobileNet-based model.

Layer	Output size	Output channels
Input	224 x 224	3
conv2d	112 x 112	16
bneck, 3x3	112 x 112	16
bneck, 3x3	56 x 56	24
bneck, 3x3	56 x 56	24
bneck, 5x5	28 x 28	40
bneck, 5x5	28 x 28	40
bneck, 5x5	28 x 28	40
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	80
bneck, 3x3	14 x 14	112
bneck, 3x3	14 x 14	112
bneck, 5x5	7 x 7	160
bneck, 5x5	7 x 7	160
bneck, 5x5	7 x 7	160
conv2d, 1x1	7 x 7	960
Dec_dwconv1	7 x 7	160
Dec_dwconv2 + interpolat.	14 x 14	80
Dec_dwconv3 + interpolat.	28 x 28	40
Dec_dwconv4 + interpolat.	56 x 56	24
Dec_dwconv5 + interpolat.	112 x 112	16
Dec_pwconv + interpolat.	224 x 224	1

4.8 Materials

This section briefly describes both the hardware configuration and the software development tools used for the project implementation. Finally, the training protocol is explained in detail considering the *supervised training strategy* as the selected one.

4.8.1 Hardware setup

The implementation, training and host validation was done using Python 3.7 using the PyTorch 1.4.0 framework installed on an ASUS X556U 8GB RAM laptop with a single NVIDIA GeForce 930MX GPU with 2GB VRAM running a Linux Ubuntu environment.

With respect to the validation in target, the selected embedded system is the NVIDIA Jetson Nano which is shown in Fig. 4.10.

Fig. 4.10: NVIDIA Jetson Nano development kit.

Some technical specifications of the embedded system are indicated in Table 4.7.

Table 4.7: Technical specifications of the NVIDIA Jetson Nano.

Feature	Specification
CPU	Quad-core ARM A57 @1.43 GHz
GPU	128-core Maxwell architecture
Memory	4GB 64-bit LPDDR4
USB	4x USB 3.0, USB 2.0 Micro-B
Power supply	5V DC / 2A
Board dimensions	100mm x 80mm x 29mm

4.8.2 Training protocol

The training protocol includes some implementation details and the selected hyperparameter values. Note that this section only applies to the initial benchmark between the ShuffleNet-based models and the MobileNet-based models, where the training protocol remains fixed while the architecture details vary.

Regarding the initialization of the learnable parameters, the encoders are pretrained on ImageNet (Deng, Dong, Socher, Li, Li, & Fei-Fei, 2009). On the other hand, the decoders' filters are randomly initialized using a Gaussian distribution. The Stochastic Gradient Descent (SGD) is used as the optimizer with a momentum of 0.90 and a weight decay enabled with a value of 0.0001. The learning rate is initialized in 0.01 and is configured to decrease its value by a factor of 3 every 5 elapsed epochs. Finally, the batch size for the training stage is set to 16 and the maximum number of epochs is set to 20. Some of these hyperparameter values were taken from (Ma & Karaman, 2018).

Since the monocular depth estimation is a pixel-level regression task, it requires a regression loss function. Under this context, the regression loss functions consider the difference between the target depth map y and the prediction of the deep convolutional neural network $\hat{y}$. In this sense, the Mean Absolute Error (MAE) was selected as the loss function for the training stage (Carvalho, Saux, Trouvé-Peloux, Almansa, & Champagnat, 2018):

$$L_1(y, \hat{y}) = \frac{1}{n}\sum_{p=1}^{n}|y_p - \hat{y}_p|, \tag{10}$$

where y_p is a pixel of the target depth map y, $\hat{y}_p$ is a pixel in the predicted depth map $\hat{y}$ and n is the total number of pixels in the depth map.

CHAPTER 5

Results

In this chapter, the quantitative and qualitative results obtained after evaluating the proposed models on the official NYU-Depth V2 validation dataset are initially presented. This initial benchmark allowed to determine the best low-latency FCN from those proposed. After the benchmark, an *ablation study* is performed to determine the appropriate training protocol and hyperparameter values. Finally, the performance of the best FCN model is presented and compared against a related state-of-the-art method on the selected *embedded system*.

5.1 Proposed FCNs

As stated in Chapter 4, eight different low-latency FCN architectures are proposed with the main intention of generating a diversified search space of low-latency models. All proposed networks differ from each other from an architectural standpoint and therefore, the number of learnable parameters (weights) and number of FLOPs are also quite different. Despite of the latter, all of them can be considered as low-latency FCN architectures. To the best knowledge of the author, these architectural configurations have not been tested yet for the monocular depth estimation task. The architecture details for each configuration are shown in Table 5.1.

Table 5.1: Architecture details of the proposed low-latency fully convolutional networks.

Network name	Feature extractor	Depth multiplier	Residual connections	Upsampling blocks	Weights [M]	FLOPs [G]
Shuff-dw-0.5	ShuffleNet V2	0.5x	No	7 w/(3x3)	0.58	0.07
Shuff-dw-1.0	ShuffleNet V2	1.0x	No	7 w/(3x3)	1.90	0.21
Shuff-dw-res-0.5	ShuffleNet V2	0.5x	Yes	7 w/(5x5)	0.60	0.08
Shuff-dw-res-1.0	ShuffleNet V2	1.0x	Yes	7 w/(5x5)	1.93	0.22
Mob-dw-0.75	MobileNet V3	0.75x (Large)	No	6 w/(3x3)	1.90	0.17
Mob-dw-1.0	MobileNet V3	1.0x (Large)	No	6 w/(3x3)	3.16	0.24
Mob-dw-res-0.75	MobileNet V3	0.75x (Large)	Yes	6 w/(5x5)	1.91	0.17
Mob-dw-res-1.0	MobileNet V3	1.0x (Large)	Yes	6 w/(5x5)	3.18	0.24

5.2 Initial benchmark

In order to evaluate the performance of the proposed FCN architectures (see Table 5.1) under the same conditions, a specific training protocol and hyperparameter values were defined. The purpose of this initial benchmark is to rank the architectures according to their performance, including the inference time achieved in a personal computer (see specifications below). Furthermore, this analysis also made it possible to select the best baseline architecture for the ablation study. The settings for the training protocol are shown in Table 5.2.

Table 5.2: Training protocol and the hyperparameter values.

Setting	Value
Batch size	16
Optimizer	Stochastic Gradient Descent Momentum = 0.90 Weight decay = 0.0001
Loss function	MAE
Transfer learning	Yes
Learning rate	Initialization = 0.01 Decay factor = 3 Periodicity = 5 epochs
Data augmentation	Yes
Total epochs	20

The hardware configuration consists of an ASUS X556U 8GB RAM laptop with a single NVIDIA GeForce 930MX GPU with 2GB VRAM.

5.2.1 Quantitative results

In this section, the quantitative results obtained after evaluating the proposed low-latency FCN architectures against the official NYU-Depth V2 validation dataset are presented. Table 5.3 shows a comparison of the different proposed low-latency models using the standard metrics used in prior works (Eigen *et al.*, 2014). Similar to the state-of-the-art studies, each metric value in Table 5.3 represents the average on the official NYU-Depth V2 validation dataset. Note that the ↑ symbol means that a higher value is better and the ↓ symbol means that a lower value is better.

Table 5.3: Quantitative results obtained on the official NYU-Depth V2 validation dataset.

Network name	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
Shuff-dw-0.5	0.670	0.705	0.502	0.192	13	77
Shuff-dw-1.0	0.615	0.748	0.453	0.169	14	71
Shuff-dw-res-0.5	0.661	0.707	0.494	0.192	13	77
Shuff-dw-res-1.0	0.601	0.757	0.445	0.168	15	66
Mob-dw-0.75	0.596	0.757	0.442	0.172	14	71
Mob-dw-1.0	0.595	0.760	0.441	0.174	16	62
Mob-dw-res-0.75	0.613	0.739	0.459	0.175	14	71
Mob-dw-res-1.0	0.605	0.749	0.452	0.177	16	62

As shown in Table 5.3, the first four columns after the 'Network name' column show the obtained average error metrics for each of the proposed low-latency FCN models. The last two columns indicate the associated latency in two different formats. At a first glance, the results suggest that the Mobilenet-based architectures are capable of achieving a slightly better performance than their counterpart ShuffleNet-based architectures.

The smaller FCN architectures from Table 5.1 (i.e., *Shuff-dw-1.0* and *Shuff-dw-res-0.5*) achieved the worst performance. The latter is expected since it is well known that as the number of learnable parameters of a network decreases, its performance also decreases. Despite the previous fact, the RMSE performance of these lightweight networks is within 0.670 meters, which is 23% smaller than that obtained in (Eigen *et al.*, 2014), the first paper to propose a convolutional neural network to solve the monocular depth estimation problem. Both networks are able to achieve an average execution time of 13 milliseconds per RGB image on the above described hardware configuration.

Eventhough the other six FCN architectures achieved similar quantitative results among them, the MobileNet-based models without skip additive connections achieved the best performance overall. Similar to state-of-the-art studies, the qualitative results are also required for a complete comparison. To gain some context, it should be also noted that the proposed networks can be considered as very lightweight FCNs due to their much more compact size compared to other similar studies. Table 5.4 indicates that even the largest proposed model (i.e., *Mob-dw-res-1.0*) is smaller than the low-latency FCN model proposed in (Wofk *et al*., 2019).

Table 5.4: Size comparison between different low-latency FCN models.

Network name	Weights [M]	FLOPs [G]
Wofk *et al.*	3.93	0.74
Shuff-dw-res-1.0	1.93	0.22
Mob-dw-res-1.0	3.18	0.24

As part of the quantitative results, Fig. 5.1 shows how the average training loss value for each low-latency FCN at each epoch during the training stage. As also shown in Fig. 5.1, only five out of the eight FCN architectures are capable of reaching a loss value less than 0.30, while no model could achieve a loss value less than 0.25.

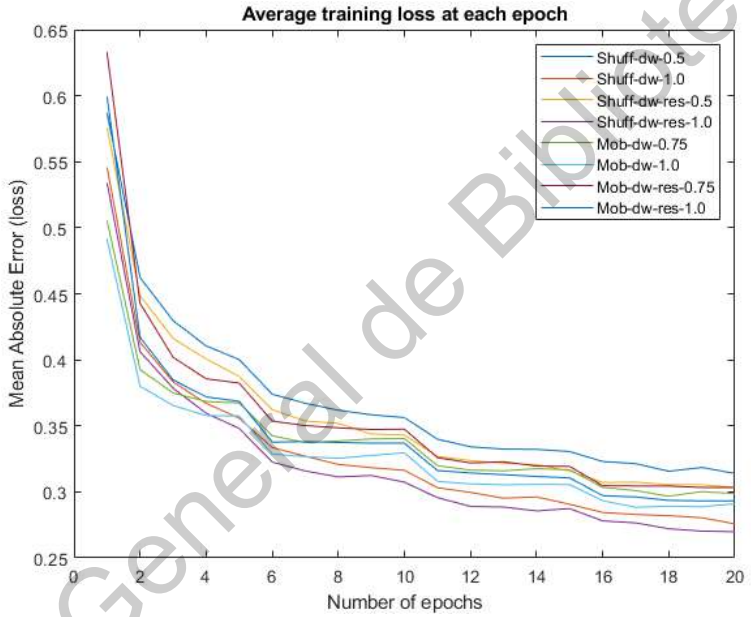


Fig. 5.1: Average training loss of the proposed low-latency FCN architectures.

Similar to many interesting optimization problems, the MAE loss function associated to each proposed FCN is generally non-convex. The latter means that there are potentially many local minima, which makes it difficult to achieve convergence during traning without overfitting. As it will be discussed in Section 5.3, an ablation study is important to determine the appropriate training protocol and hyperparameter values for a specific architecture given a training dataset. As already stated, this initial benchmark used the training protocol stated in Table 5.2.

Since the ultimate goal is to find the best coefficients for the non-linear transformation between the space of RGB images and the codomain of real-valued depth maps, it is important to identify whether a best model has been obtained after a specific training epoch ends. For that reason, the model validation step is required to be done right after a training epoch has ended. The plot of the average validation loss at each epoch is illustrated in Fig. 5.2. Note that since the validation loss at each epoch is computed using a batch size equal to one, it is considered normal for it to oscillate.

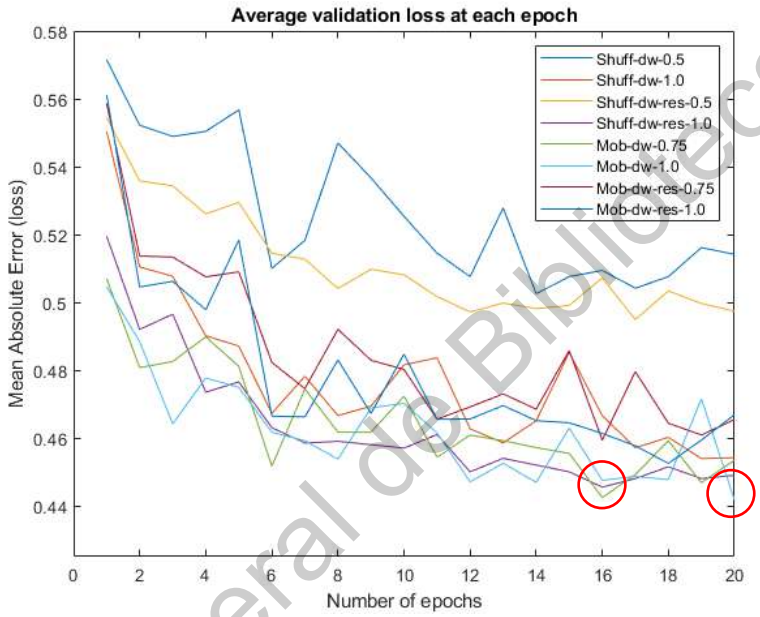


Fig. 5.2: Average validation loss of the proposed low-latency FCN architectures. The left-hand red circle indicates that *Shuff-dw-res-1.0* and *Mob-dw-0.75* achieved their lowest loss value in epoch 16. The right-hand red circle shows that *Mob-dw-1.0* achieved its lowest point in epoch 20.

For instance, Fig. 5.2 shows circled in red the number of epoch at which the lowest MAE loss values were obtained for the best performing low-latency FCN models. On the one hand, both *Shuff-dw-res-1.0* and *Mob-dw-0.75* achieved their best performance in epoch 16. On the other hand, the best model for *Mob-dw-1.0* was obtained until epoch 20. It should be noted that the networks were not trained further epochs mainly to avoid a negative effect known as network overfitting. Up to approximately the epoch 20, the performance of the low-latency FCNs were observed to improve on the official NYU-Depth V2 validation dataset (unseen data). However, beyond that point, the improvement of the model on the training data comes at the expense of a decrease in its *generalization capability*.

5.2.2 Qualitative results

The predicted relative depth maps obtained on the official NYU-Depth V2 validation dataset for each of the proposed low-latency FCN architectures in Table 5.1 are presented in Fig. 5.3 and Fig. 5.4. On the one hand, Fig. 5.3 shows the comparison for the low-latency FCN architectures that use the ShuffleNet V2 as the feature extractor.

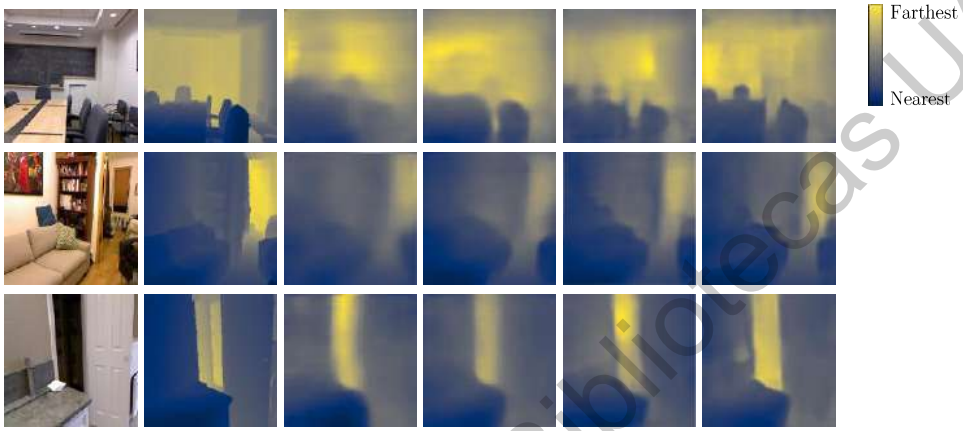


Fig. 5.3: Comparison of qualitative results on some samples of the NYU-Depth V2 validation dataset when using the ShuffleNet V2 encoder. From left to right: (a) Input RGB image; (b) Ground-truth; (c) Shuff-dw-0.5; (d) Shuff-dw-1.0; (e) Shuff-dw-res-0.5; (f) Shuff-dw-res-1.0.

Similarly, Fig. 5.4 shows the comparison for the low-latency FCN architectures that use the MobileNet V3 as the encoder.

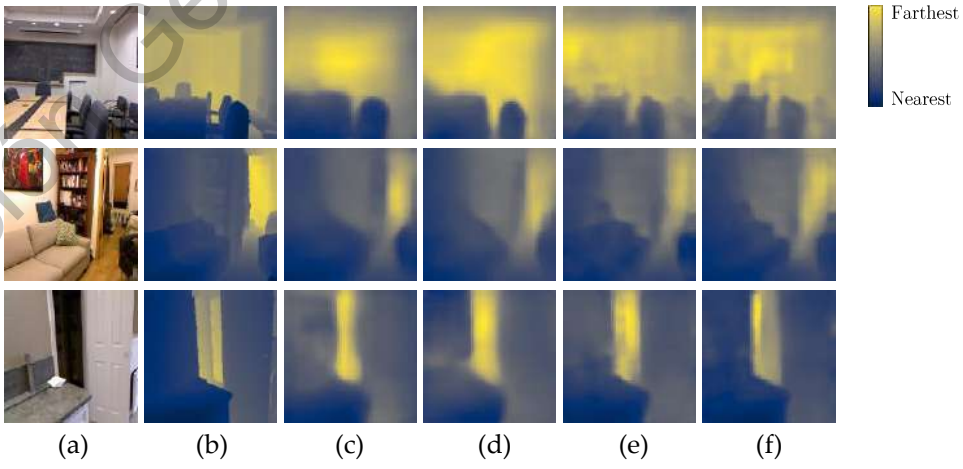


Fig. 5.4: Comparison of qualitative results on some samples of the NYU-Depth V2 validation dataset when using the MobileNet V3 encoder. From left to right: (a) Input RGB image; (b) Ground-truth; (c) Mob-dw-0.75; (d) Mob-dw-1.0; (e) Mob-dw-res-0.75; (f) Mob-dw-res-1.0.

In order to select the most appropriate low-latency FCN architecture for the ablation study (see section 5.3), the following points were taken into account:

1. The expected model performance given the number of learnable parameters.
2. The proportion between the size of the model and its associated execution time on the official validation dataset.
3. The sharpness of the qualitative results compared to the ground truth depth maps.
4. The allocated VRAM for a given model during the training stage (considering the limited resources an embedded system has).

The assessment consisted on evaluating the pros and cons of the proposed models considering them mainly as two different subsets depending on their feature extractor. On the one hand, the MobileNet-based models, specifically the *Mob-dw-1.0*, achieved the best performance among the proposed models. Overall, it was actually expected for this model to achieve the lowest RMSE since it is the largest model (i.e., it has 3.16 M learnable parameters) compared to the rest of the proposed architectures (see Table 5.3). Because of its size, the allocated VRAM during training (~1.8 GB with batch size equal to 16) was considerably larger than that of the ShuffleNet-based largest model (~1.3 GB with batch size equal to 16). The major drawback of the MobileNet-based models is their low sharpness on the qualitative results, which is due to the fact that *Mob-dw-1.0* and *Mob-dw-0.75* do not use additive skip connections.

On the other hand, the ShuffleNet-based models achieved a RMSE performance slightly lower on the given training protocol and hyperparameter values. Nonetheless and as indicated in Table 5.3, the average relative error (Rel ↓) for *Shuff-dw-res-1.0* and *Shuff-dw-1.0* is in fact lower than that obtained by any of the MobileNet-based models. The latter indicates there is no significant performance gap between the two different approaches. Considering only the ShuffleNet-based architectures, the *Shuff-dw-res-1.0* stands as the model with the best quantitative results. In regard to the execution time, this model is capable of processing a 224 x 224 RGB image in 15 milliseconds using the above-mentioned hardware configuration. If we also consider the qualitative results, the depth images obtained by *Shuff-dw-res-1.0* are clearly the sharpest from the eight proposed low-latency FCN architectures (see column (f) in Fig. 5.3).

Given the previous analysis, it is clear that *Shuff-dw-res-1.0* is the most promising model from those that were proposed. Therefore, this specific ShuffleNet-based architecture is selected as the model on which an ablation study would be performed.

5.3 Ablation studies

After selecting the *Shuff-dw-res-1.0* model as the best performing model, it was decided to further analyze its performance against variations of the hyperparameter values. Each of the hyperparameters is analyzed in isolation, i.e., only the hyperparameter of interest is varied while the others remain fixed. However, if a hyperparameter setting improves the performance of the model, it is considered as the new baseline for the next test. The only hyperparameter that was initially modified from those of Table 5.2 was the number of total epochs which was set to 25.

5.3.1 Batch size

According to some authors such as (Goodfellow *et al.*, 2016), it is recommended to use a large value for batch size (e.g., 8, 16, 32, etc.) since it helps the optimization algorithm to obtain a better estimate of the gradient. However, considering that the batch size is limited by the 2GB GPU VRAM capacity, it was only possible to test with values of 8 and 16. Fig. 5.5 shows the average validation loss for the above-mentioned batch sizes.

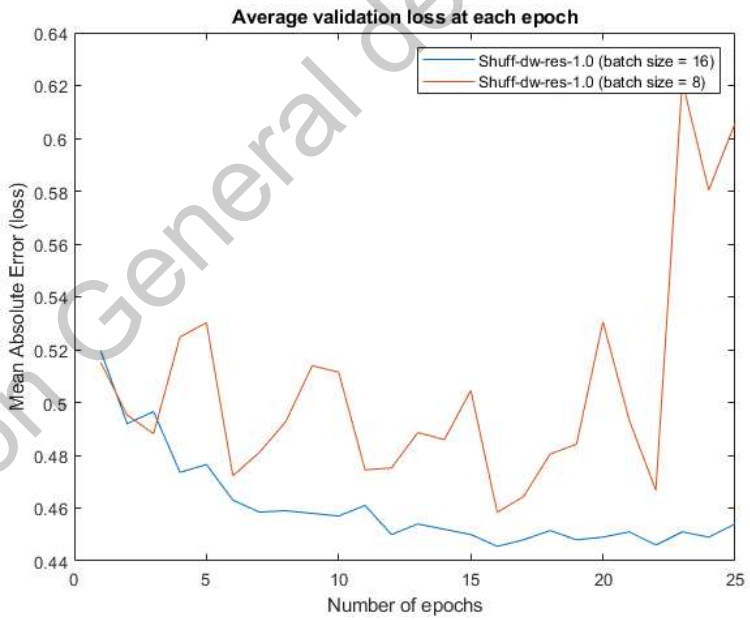


Fig. 5.5: Average validation loss for different batch sizes on the NYU-Depth V2 dataset.

Fig. 5.5 indicates that a batch size equal to 8 produces bigger fluctuations in the loss value before it abruptly diverges in the last epochs. In contrast, it is observed that using 16

samples to update the model learnable parameters allows to find a better local minimum. The quantitative results are shown in Table 5.5.

Table 5.5: Quantitative results for different batch sizes on the NYU-Depth V2 validation dataset.

Batch size	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
8	0.634	0.736	0.472	0.182	15	66
16	0.601	0.757	0.445	0.168	15	66

5.3.2 Optimizer

An important ablation study was to determine the best optimizer (i.e., optimization algorithm) for the low-latency FCN on the monocular depth estimation task. According to (Wilson, Roelofs, Stern, Srebro, & Recht, 2017), there are currently two main trends when it comes to the loss function optimization. Specifically, there is the standard method known as Stochastic Gradient Descent (SGD) and adaptive gradient optimization methods (e.g., Adam, RMSProp, AdaGrad, etc.). In particular, the state-of-the-art methods for monocular depth estimation use either SGD with momentum or Adam (Wofk *et al.*, 2019; Alhashim *et al.*, 2019). Due to the above, it was decided to train and validate the model using both optimizers. The learning rate for Adam optimizer was set to 0.0001 as in (Alhashim *et al.*, 2019). The results are shown in Fig. 5.6.

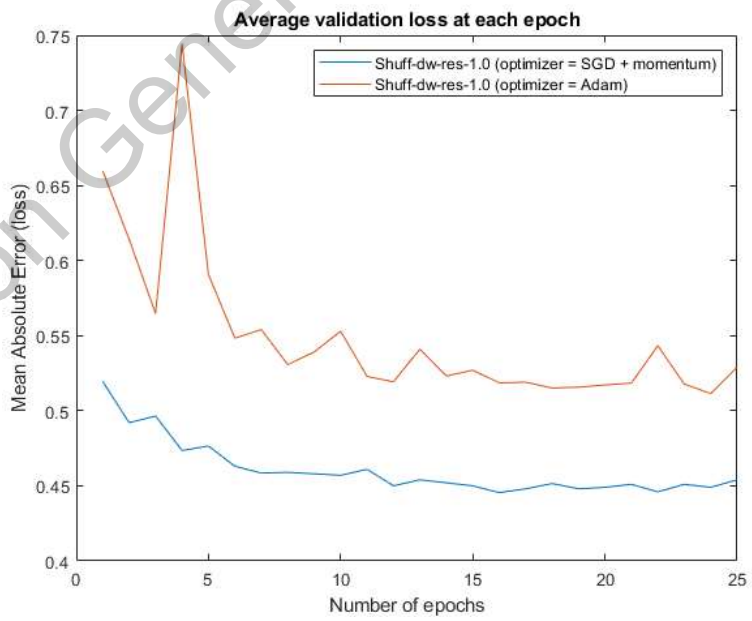


Fig. 5.6: Average validation loss for different optimizers on the NYU-Depth V2 dataset.

The quantitative results are shown in Table 5.6.

Table 5.6: Quantitative results for different optimizers on the NYU-Depth V2 validation dataset.

Optimizer	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
SGD + momentum	0.601	0.757	0.445	0.168	15	66
Adam	0.673	0.685	0.511	0.198	15	66

Similar to the findings in (Wilson *et al.*, 2017), the obtained quantitative results indicate that SGD with momentum significantly outperforms the adaptive gradient method (Adam). In other words, it can be said that the solution (local minimum) found by the Adam algorithm generalizes poorly on the official validation dataset.

5.3.3 Learning rate

The next logical step was to further analyze the impact of the learning rate on the model performance. Undoubtedly, the learning rate initialization is regarded as a very important hyperparameter, as it controls how quickly the search space is explored. Fig. 5.7 shows the average validation loss for three different learning rates.

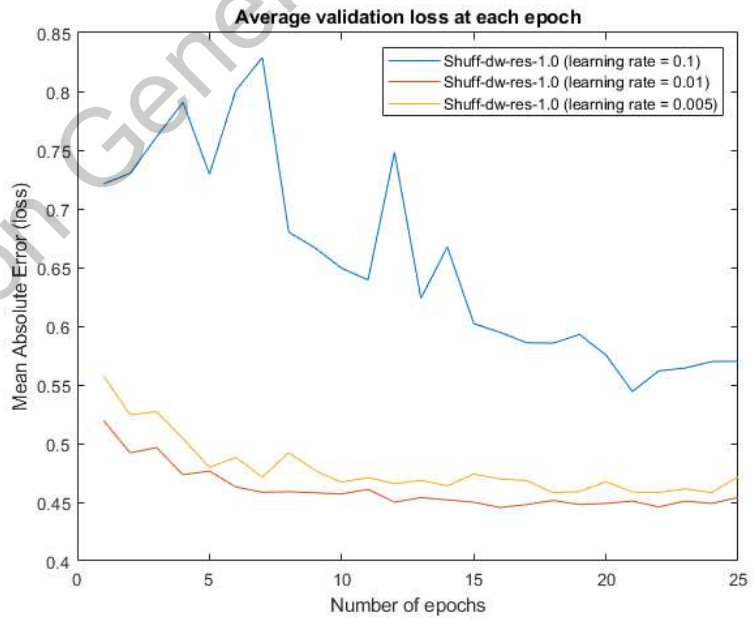


Fig. 5.7: Average validation loss for different learning rates on the NYU-Depth V2 dataset.

As shown in Fig. 5.7, a learning rate that is too large (i.e., 0.1) causes the low-latency FCN model to reach to a suboptimal solution. In the other hand, a very small learning rate (i.e., 0.005) is not enough to trigger significant changes to the weights at each update step, which in turn caused the learning process to stagnate. The quantitative results are illustrated in Table 5.7.

Table 5.7: Quantitative results for different learning rates on the NYU-Depth V2 validation dataset.

Learning rate	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
0.1	0.715	0.653	0.544	0.207	15	66
0.01	0.601	0.757	0.445	0.168	15	66
0.005	0.612	0.740	0.458	0.174	15	66

The obtained results in Table 5.7 clearly indicate that using 0.01 is a good learning rate for this pixel-level continuous regression task.

Decay factor

Overall, the learning rate decay is a technique that consists on gradually decreasing the learning rate by a factor during the training stage. In this regard, it is important to determine the optimal value $\in \mathbb{N}$ to use as the decay factor. Fig. 5.8 shows the average validation loss for different decay factors considering a learning rate initialized as 0.01.

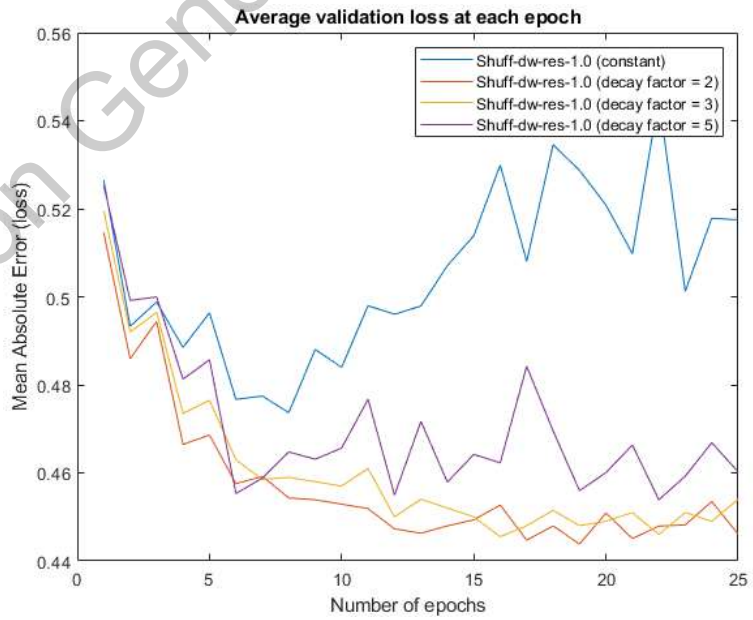


Fig. 5.8: Average validation loss for different decay factors on the NYU-Depth V2 dataset.

It is interesting to note in Fig. 5.8 that a decay factor equal to 1 (i.e., constant learning rate) has the worst performance overall. Furthermore, this ablation study allowed to observe that a decay value of 2 is capable of improving the generalization capability of the low-latency FCN model on new data. The quantitative results are shown in Table 5.8.

Table 5.8: Quantitative results for different factors on the NYU-Depth V2 validation dataset.

Decay factors	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
1	0.634	0.729	0.473	0.174	15	66
2	0.600	0.759	0.443	0.166	15	66
3	0.601	0.757	0.445	0.168	15	66
5	0.608	0.749	0.454	0.175	15	66

Weight decay

Another important optimizer hyperparameter (related to SGD) is known as *weight decay* or *L2 regularization*. As already described in a previous chapter, the weight decay is simply a method to penalize large weight values in order to avoid an overfitted model. Fig. 5.9 shows the average validation loss for two different weight decay values.

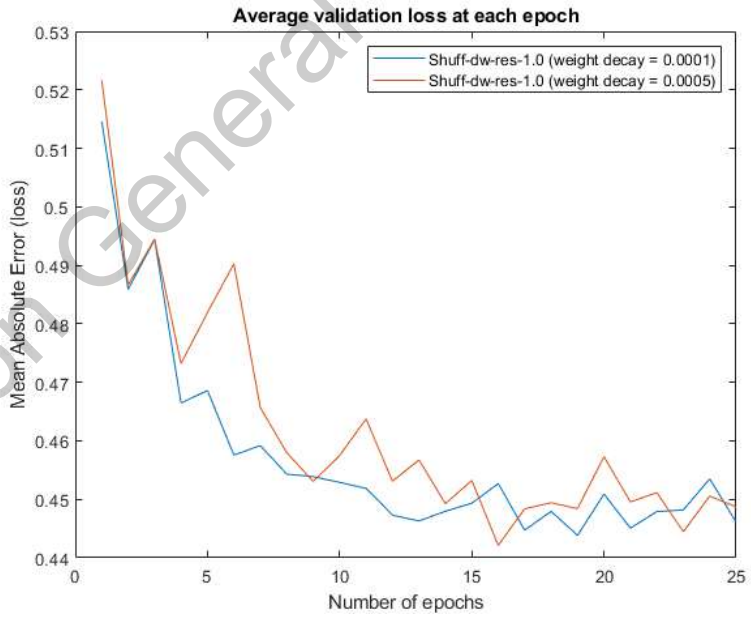


Fig. 5.9: Average validation loss for different weight decay values on the NYU-Depth V2 dataset.

It can be said that a larger penalization value improves the validation performance of the model as shown in Table 5.9.

Table 5.9: Quantitative results for different L2 values on the NYU-Depth V2 validation dataset.

Weight decay	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
0.0001	0.600	0.759	0.443	0.166	15	66
0.0005	0.596	0.758	0.442	0.166	15	66

5.3.4 Transfer learning

One of the main factors that positively impact the performance of the proposed low-latency FCN is the usage of the state-of-the-art technique known as transfer learning. Fig. 5.10 shows that when transfer learning is used, it enables significant improvement in performance for the *Shuff-dw-res-1.0* architecture. Note that all three models in Fig. 5.10 were trained using the same hyperparameter values.

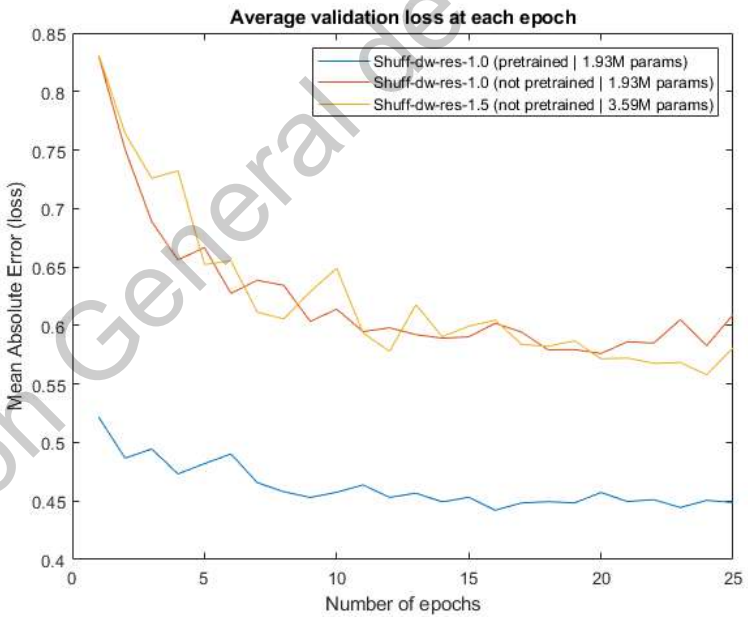


Fig. 5.10: Impact of encoder pre-training on the average validation loss.

After analyzing the Fig. 5.10, it is quite clear that the *Shuff-dw-res-1.5* model could have outperformed the *Shuff-dw-res-1.0* model if only the pretrained model had been used. However, it was not possible to get neither the pretrained model nor the required ImageNet

dataset to generate it. After several attempts, the ImageNet dataset administrators did not respond to any of the requests made by the research team.

The quantitative results for the two different encoder weights initialization approaches (i.e., pretrained and not pretrained) are shown in Table 5.10 for different depth multipliers.

Table 5.10: Quantitative comparison between the two different encoders initialization strategies.

Model	RMSE [m] ↓	δ_1 ↑	MAE ↓	Rel ↓	GPU [ms] ↓	fps ↑
Shuff-dw-res-1.0 (pretrained)	0.596	0.758	0.442	0.166	15	66
Shuff-dw-res-1.0 (not pretrained)	0.755	0.634	0.576	0.219	15	66
Shuff-dw-res-1.5 (not pretrained)	0.733	0.659	0.557	0.214	16	62

The results in Table 5.10 confirm that transfer learning allows to reduce the RMSE metric by 21.06% for the *Shuff-dw-res-1.0* model.

5.4 Comparison with prior work

Once the *Shuff-dw-res-1.0* best model was obtained, it was directly compared to the state-of-the-art low-latency fully convolutional network proposed by Wofk *et al.* (2019) on the official NYU-Depth V2 validation dataset.

The quantitative comparison was performed using both devices: the host computer and the selected target device. Furthermore, a qualitative comparison between the proposed method and the state-of-the-art method is also considered at the end of this section.

5.4.1 Quantitative comparison

This section shows a quantitative comparison between results of the related state-of-the-art method and the proposed method on the above-mentioned validation dataset. Note that the initials NP for some of the following result tables stand for the method from Wofk *et al.* (2019) with additional post-processing based on a network pruning technique. It should be noted that the ↑ symbol means that higher is better and the ↓ symbol means that lower is better.

Comparison in host

Table 5.11 shows the quantitative comparison between the results of the proposed method against those from the related state-of-the-art- method.

Table 5.11: Comparison of quantitative results on the official NYU-Depth V2 validation dataset in the host personal computer (ASUS X556U 8GB RAM with a single NVIDIA GeForce 930MX GPU with 2GB VRAM).

Model	RMSE [m] ↓	δ_1 ↑	GPU [ms] ↓	fps ↑	Weights [M]	FLOPs [G]
Wofk *et al.*	0.599	0.775	24	41	3.93	0.74
Wofk *et al.*+NP	0.604	0.771	17	58	1.34	0.37
Proposed model	0.596	0.758	15	66	1.93	0.22

As shown in Table 5.11, the RMSE value for the proposed method is lower when compared to the related method (without and with compression) by 0.5% and 1.16% respectively. The latter demonstrates that a more compact deep learning model than the state-of-the-art method is indeed capable of achieving less error.

Additionally, the execution time in milliseconds achieved by the proposed method is lower than that obtained by the related method (without compression) by 37.5% (i.e., 1.6x faster). Furthermore, the execution time obtained by the proposed method is even less than that achieved using the compressed model of the related method by 11.7% (i.e., 1.1x faster). In general, it can be said that the proposed FCN is significantly more lightweight, while achieving a noticeable improvement in the RMSE metric.

Comparison in Jetson Nano

The successful deployment of deep learning models in embedded systems depends on several factors that are explained in detail in the following chapter. Once the installation dependencies were solved, the results of Table 5.12 were obtained.

Table 5.12 Comparison of quantitative results on the official NYU-Depth V2 validation dataset in the target device (NVIDIA Jetson Nano with 4GB VRAM – 10W power mode).

Model	RMSE [m] ↓	δ_1 ↑	GPU [ms] ↓	fps ↑	Weights [M]	FLOPs [G]
Wofk *et al.*	0.599	0.775	59	17	3.93	0.74
Wofk *et al.*+NP	0.604	0.771	62	16	1.34	0.37
Proposed model	0.596	0.758	68	15	1.93	0.22

5.4.2 Qualitative comparison

Once the low-latency FCN model was already deployed in the embedded system, the next step was to perform a qualitative comparison using some validation samples between the proposed method and the related work is shown in Fig. 5.11.

It should be noted that the relative depth on each sample scene increases as the color of the pixels in the depth map becomes lighter and vice versa. Furthermore, an error map between each model predictions and the ground-truth depth maps are also presented. For the sake of clarity, both colormaps are attached to Fig 5.11.

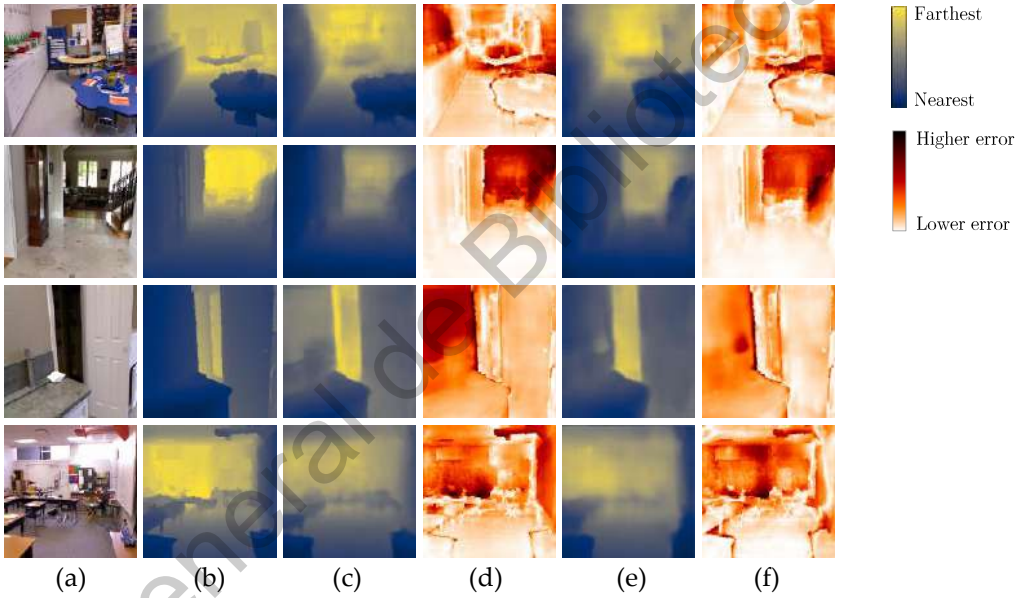


Fig. 5.11: Comparison of qualitative results on some samples of the NYU-Depth V2 validation dataset. From left to right: (a) Input RGB image; (b) Ground-truth; (c) Wofk *et al.* (2019); (d) Error map between Wofk *et al.* (2019) and ground-truth; (e) Proposed method (*Shuff-dw-res-1.0*); (f) Error map between proposed method and ground-truth.

The corresponding depth maps obtained by the proposed method are shown in column (e) of Fig. 5.11, while the qualitative results of the related work are shown in column (c). Although the related method shows slightly better sharpness in some regions, note how the proposed method achieves a more accurate relative depth estimation for the closest objects in almost every image shown in Fig. 5.11. The latter observation can be confirmed by looking at the error maps that are shown in column (f) for the proposed method and in column (d) for the related work. Note how there are fewer red areas for the proposed method.

Finally, the model predictions are compared to each other using a 3D representation as shown in Fig. 5.12.

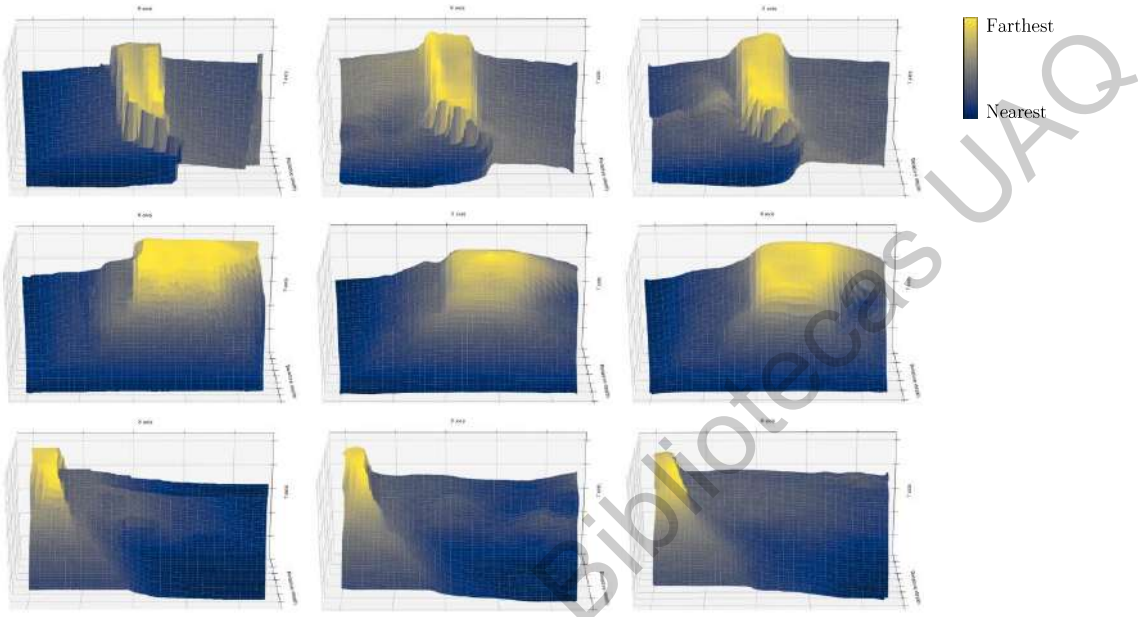


Fig. 5.12: Comparison of 3D representations of model predictions against their ground-truth depth map on some samples of the NYU-Depth V2 validation dataset. From left to right: (a) Ground-truth; (b) Wofk *et al.* (2019); (c) Proposed method (*Shuff-dw-res-1.0*).

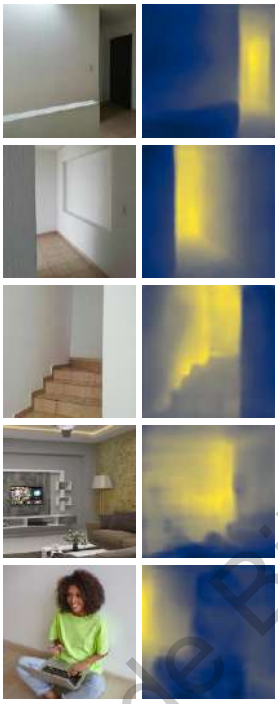
Overall, it is clear that the proposed method achieves a comparable or better qualitative performance to that of the state-of-the-art method. On the other hand, it is also fair to say that the qualitative performance of low-latency FCNs for monocular depth estimation is still limited in terms of resolution.

5.5 Inference phase results

The inference phase refers to the stage when the machine learning model is used to obtain predictions on unseen input data that does not belong to the training nor the validation dataset used to train/validate the model.

It should be noted that there is no ground truth available for this phase so it cannot be quantitatively evaluated. In this sense, the qualitative results can be used to estimate the correctness of the predictions (see Fig. 5.13). Overall, it is observed that the *Shuff-dw-res-1.0*

model has a very good generalization capability on unseen data. However, it is also observed that adding objects with fine details (i.e., furniture) or people is still challenging.



(a) (b)

Fig. 5.13: Qualitative results in indoor scenes that do not belong to the NYU-Depth V2 validation dataset. From left to right: (a) Input RGB image; (b) Prediction of the proposed method (*Shuff-dw-res-1.0*).

Although the obtained model is intended to be used exclusively to predict the relative depth of indoor scenes, it was tested on some outdoor scenes as shown in Fig. 5.14. Although performance at the local level is poor, less error is observed at a global structure level.

(a) (b)

Fig. 5.14: Qualitative results in outdoor scenes. From left to right: (a) Input RGB image; (b) Prediction of the proposed method (*Shuff-dw-res-1.0*).

5.6 Discussion

This section presents a discussion of the obtained quantitative and qualitative performance of the proposed method.

5.6.1 Model performance

On the one hand, the quantitative results in Table 5.11 and Table 5.12 related to error metrics (i.e., RMSE) indicate that the proposed method outperforms the low-latency state-of-the-art method. The latter observation is considered notable given the fact that the encoder network used in the related method (without compression) has 3.9x more FLOPs and 1.8x more parameters than the one used as part of the proposed method. As a result, the mean latency of the proposed method is 37.5% less than that obtained with the related method (without compression). Furthermore, the proposed method is even faster than the compressed model of the related method in about 11%.

The results imply that using low-latency convolutional layers (i.e., depthwise separable convolutions) together with channel shuffle and split operators can also be seriously considered for pixel-wise continuous regression tasks where computational resources are very restricted. In general, the proposed approach shows that the selected feature extractor (i.e., ShuffleNet V2) is capable of achieving an appropriate latent space representation of the input with fewer computational cost than the related method. Despite the fact that the proposed method uses a smaller FCN model than Wofk *et al.* (2019), it is noticeable in both quantitative and qualitative results that the proposed method achieves predictions with less error. It is also observed in Fig. 5.11 column (e) that most of the nearest objects in the predicted depth maps appear to be more visually consistent with the ground truth maps on the NYU-Depth V2.

5.6.2 Transfer learning

The quantitative results that are shown in Table 5.10 indicate that the performance of a low-latency fully convolutional network on the NYU-Depth V2 dataset is strongly correlated to the initialization of the weights. In this sense, it was observed that the best performance for the proposed FCN architectures is obtained when the transfer learning technique is used, i.e., when the learnable parameters of the encoder are initialized with a pretrained model on the ImageNet dataset.

It should be noted that the availability of these pretrained models was found to be quite limited, not only because the training process would necessarily require a high-performance

computer with a high-end GPU, but mainly because it was not possible to gain access to the ImageNet dataset. For this reason, it can be said that the lack of a pre-trained model to initialize the encoder of a low-latency FCN could severely limit its performance.

5.6.3 Framework maturity

As stated by Wofk *et al.* (2019), most of the deep learning frameworks (e.g., PyTorch, Tensorflow, etc.) have not yet reached a production-ready maturity level to fully support lightweight models. The latter means that most of the implementations of the deep learning operators (e.g., depthwise decomposition, channel shuffle, etc.) are hardware-specific (e.g., x64 and ARM) and could vary between the several official frameworks' releases. Moreover, since low-latency convolutional architectures are part of a quite recent research topic, it is also expected that some of these operators are not yet fully optimized in the abovementioned deep learning frameworks.

In addition, the deep learning frameworks are usually tailored by the embedded system supplier (e.g., NVIDIA) to enable their installation on their specific embedded operating system (e.g., Linux4Tegra). Overall, all of these customizations and lack of optimizations are responsible for the discrepancies related to the obtained model performance in terms of the execution time on the embedded system.

CHAPTER 6

Conclusion and future work

This chapter presents the conclusion of the research in terms of the contributions and provides the future work that may further improve the methodology.

6.1 Conclusion

Due to their enhanced performance, deep learning-based methods are increasingly being used to solve specific problems in almost any scientific and technological domain. As described in Chapter 4, the monocular depth estimation task is no exception to this emerging trend. While the latter is the overall picture of the state-of-the-art methods for this research topic, most of these studies are mostly focusing on improving the accuracy at the cost of complex and large deep learning architectures that cannot be deployed on embedded systems.

It has been said that a very promising direction in this research area is the development of smaller and lightweight networks since these will have greater implications for practical use cases (Zhao *et al.*, 2020). This thesis contributes to this goal by proposing a very low-latency fully convolutional network whose performance is comparable to state-of-the-art methods with a lower execution time. In particular, this work proposes a low-latency FCN architecture for monocular depth estimation in systems with limited computational resources. The experimental results of the proposed approach show that the low-latency convolution layers (depthwise separable convolutions) and the operators (channel shuffle and channel split) that are used as part of the ShuffleNet V2 encoder can help lay the foundation on the design of very lightweight FCNs architectures for real-time monocular depth estimation.

To the best of the authors' knowledge, the ShuffleNet V2 feature extractor had not been tested in the monocular depth estimation task, until now. It should also be noted that, without the additional use of compression techniques, the obtained model is capable of achieving a shorter inference time than the related state-of-the-art method (Wofk *et al.*, 2019),

while also reducing the RMSE metric by 1.16%. In particular, the proposed fully convolutional network achieves a reduction of 37.5% in inference time with respect to the related state-of-the-art method. As a consequence, it should be also noted that the reduced size of the proposed architecture allows the training and validation stages to be carried out on a typical laptop with technical specifications well below from those required by state-of-the-art methods that are not considered efficient.

Overall, it should be noted that low-latency FCNs for monocular depth estimation is a relatively new research topic that was started just a couple of years ago and therefore it can be said that it has a great future ahead of it. According to (Zhao *et al.*, 2020), the ability to execute high-performance deep learning models for monocular depth estimation on embedded devices could have disruptive applications in the not-too-distant future, which makes this research topic so worthwhile.

6.2 Recommendations for future work

The following are research ideas that could be explored as future work:

- To validate the proposed method in the outdoor scene's dataset known as KITTI (Geiger *et al.*, 2013).

- To include both the channel shuffle and channel split novel operators as part of a new decoder design.

- To investigate if there is another public classification dataset that can be used to generate pretrained models capable of obtaining similar performance as those from ImageNet.

- To explore hyperparameter optimization methods such as grid search or random search to determine optimal hyperparameter values. For this pixel-level regression task, it would be necessary a very high-performance computer to achieve this.

- To include the *Akaike's information* criterion as a method to compute the so-called goodness-of-fit test statistics of the proposed model.

References

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning: A Textbook* (1st ed.). Cham, Switzerland: Springer Nature. https://doi.org/10.1007/978-3-319-94463-0

Alhashim, I., & Wonka, P. (2018). High quality monocular depth estimation via transfer learning. Retrieved from https://arxiv.org/abs/1812.11941

Bhoi, A. (2019). Monocular Depth Estimation: A Survey. Retrieved from http://arxiv.org/abs/1901.09402

Carvalho, M., Le Saux, B., Trouvé-Peloux, P., Almansa, A., & Champagnat, F. (2018). On regression losses for deep depth estimation. In *2018 25th IEEE International Conference on Image Processing*, ICIP 2018, (pp. 2915–2919).

Cassimatis, N. L., Trafton, J. G., Bugajska, M. D., & Schultz, A. C. (2004). Integrating cognition, perception and action through mental simulation in robots. Robotics and Autonomous Systems, 49 (1-2 SPEC. ISS.), 13–23. https://doi.org/10.1016/j.robot.2004.07.014

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2009, (pp. 248–255).

Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, (pp. 2366–2374).

Fu, H., Gong, M., Wang, C., Batmanghelich, K., & Tao, D. (2018). Deep Ordinal Regression Network for Monocular Depth Estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (pp. 2002–2011).

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research, 32(11), 1231–1237. https://doi.org/10.1177/0278364913491297

Giancola, S., Valenti, M., & Sala, R. (2018). A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies (SpringerBriefs in Computer Science) (1st ed. 2018 ed.). Cham, Switzerland: Springer. https://doi.org/10.1007/978-3-319-91761-0

Godard, C., Mac Aodha, O., & Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2017, 2017-Janua, (pp. 6602–6611). https://doi.org/10.1109/CVPR.2017.699

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning (Adaptive Computation and Machine Learning series) (1st ed.). Cambridge, MA: The MIT Press.

Gurram, A., Urfalioglu, O., Halfaoui, I., Bouzaraa, F., & López, A. M. (2018). Monocular Depth Estimation by Learning from Heterogeneous Datasets. In *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June, (pp. 2176–2181). https://doi.org/10.1109/IVS.2018.8500683

Hang, H., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *4th International Conference on Learning Representations*.

Haykin, S. (2008). Neural Networks and Learning Machines (3rd ed.). New Jersey, USA: Pearson Education; (November 18, 2008).

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. Retrieved from https://arxiv.org/abs/1704.04861.

Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for Mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, ICCV 2019, (pp. 1314–1324). https://doi.org/10.1109/ICCV.2019.00140.

Kaehler, A., & Bradski, G. (2017). Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library (1st ed.). Sebastopol, CA: O'Reilly Media.

Karsch, K., Liu, C., & Kang, S. B. (2012). Depth extraction from video using non-parametric sampling. In *European conference on computer vision*, ECCV 2012, (pp. 775–788). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33715-4_56.

Khan, F., Salahuddin, S., & Javidnia, H. (2020). Deep learning-based monocular depth estimation methods—A state-of-the-art review. Sensors, 20(8), 2272. https://doi.org/10.3390/s20082272.

Kumari, S., Jha, R. R., Bhavsar, A., & Nigam, A. (2019). Autodepth: Single image depth map estimation via residual CNN encoder-decoder and stacked hourglass. In *2019 IEEE International Conference on Image Processing*, ICIP 2019, (pp. 340–344).

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision*, 3DV 2016, (pp. 239–248). https://doi.org/10.1109/3DV.2016.32

Larsson, S. (2019). Monocular depth estimation using deep convolutional neural networks. (Master's dissertation, Linköping University, Linköping, Sweden).

Liu, F., Shen, C., Lin, G., & Reid, I. (2016). Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(10), 2024–2039. https://doi.org/10.1109/TPAMI.2015.2505283

Liu, Y., Stiles, N. R., & Meister, M. (2018). Augmented Reality Powers a Cognitive Prosthesis for the Blind. Retrieved from https://www.biorxiv.org/content/10.1101/321265v1.full

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2015, (pp. 3431–3440).

Ma, F., & Karaman, S. (2018). Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE International Conference on Robotics and Automation*, ICRA 2018, (pp. 4796–4803).

Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). Shufflenet V2: Practical guidelines for efficient CNN architecture design. In *Proceedings of the European conference on computer vision*, ECCV 2018, (pp. 116–131). Springer, Cham, Switzerland. https://doi.org/10.1007/978-3-030-01264-9_8

Mitchell, T. M. (1997). Machine Learning (1st ed.). Burr Ridge, IL: McGraw-Hill Education.

Molleda, J. (2008). Técnicas de Visión por Computador para la Reconstrucción en Tiempo Real de la Forma 3D de Productos Lamina-dos (Doctoral dissertation. Informatics Department, Universidad de Oviedo, Gijón España).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 32, (pp. 8026–8037).

Pilzer, A., Xu, D., Puscas, M., Ricci, E., & Sebe, N. (2018). Unsupervised adversarial depth estimation using cycled generative networks. In *Proceedings - 2018 International Conference on 3D Vision*, 3DV 2018, (pp. 587–595). https://doi.org/10.1109/3DV.2018.00073

Poggi, M., Aleotti, F., Tosi, F., & Mattoccia, S. (2018). Towards Real-Time Unsupervised Monocular Depth Estimation on CPU. In *IEEE International Conference on Intelligent Robots and Systems*, (pp. 5848–5854). https://doi.org/10.1109/IROS.2018.8593814

Rocchini, C., Cignoni, P., Montani, C., Pingi, P., & Scopigno, R. (2001). A low cost 3D scanner based on structured light. Computer Graphics Forum, 20(3), 299–308. Boston, MA: Blackwell Publishers Ltd.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobilenetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 4510–4520).

Saxena, A., Chung, S. H., & Ng, A. Y. (2006). Learning Depth from Single Monocular Images. In *Proceedings of the 18th International Conference on Neural Information Processing Systems*. MIT Press, Cambridge, MA, (pp. 1161–1168). https://doi.org/10.1007/s11263-007-0071-y

Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from RGBD images. In *Proceedings of the 12th European Conference on Computer Vision*. Springer, Berlin, Heidelberg, (pp. 746–760). https://doi.org/10.1007/978-3-642-33715-4_54

Wang, L., Famouri, M., & Wong, A. (2020). DepthNet Nano: A Highly Compact Self-Normalizing Neural Network for Monocular Depth Estimation. Retrieved from https://arxiv.org/abs/2004.08008

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., & Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NeurIPS 2017, (pp. 4148–4158).

Wofk, D. (2020). Fast and energy-efficient monocular depth estimation on embedded systems (Master's dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA).

Wofk, D., Ma, F., Yang, T. J., Karaman, S., & Sze, V. (2019). FastDepth: Fast monocular depth estimation on embedded systems. In *2019 International Conference on Robotics and Automation*, ICRA 2019, (pp. 6101–6108).

Yang, A. (2017). Relative depth estimation from single monocular images with deep convolutional network (Doctoral dissertation, University of Missouri, Columbia, USA).

You, Y., Wang, Y., Chao, W. L., Garg, D., Pleiss, G., Hariharan, B., ... & Weinberger, K. Q. (2019). Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. In *Proceedings of the International Conference on Learning Representations*, ICLR 2020.

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. Retrieved from https://arxiv.org/abs/2106.11342

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 6848–6856).

Zhao, C., Sun, Q., Zhang, C., Tang, Y., & Qian, F. (2020). Monocular depth estimation based on deep learning: An overview. Science China Technological Sciences, 63, 1612–1627 (2020). Cham, Switzerland: Springer. https://doi.org/10.1007/s11431-020-1582-8