UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

WILLAMS DE LIMA COSTA

A comprehensive exploration of depthwise separable convolutions for real-time 3D hand pose estimation through RGB images

Recife

2020

WILLAMS DE LIMA COSTA

A comprehensive exploration of depthwise separable convolutions for real-time 3D hand pose estimation through RGB images

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Mídia e Interação

Orientador (a): Veronica Teichrieb

Coorientador (a): Lucas Silva Figueiredo

Recife

2020

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

C837c Costa, Willams de Lima
A comprehensive exploration of depthwise separable convolutions for real-time 3D hand pose estimation through RGB images/ Willams de Lima Costa. – 2020.
91 f.: il., fig., tab.

Orientadora: Verônica Teichrieb.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências e apêndice.

1. Mídia e interação. 2. Redes neurais. I. Teichrieb, Verônica (orientadora). II. Título.

006.7 CDD (23. ed.) UFPE - CCEN 2021 - 11

Willams de Lima Costa

“A comprehensive exploration of depthwise separable convolutions for real-time 3D hand pose estimation through RGB images”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 28 de fevereiro de 2020.

BANCA EXAMINADORA

_____

Prof. Dr. Lucas Silva Figueiredo
Centro de Informática / UFPE

_____

Prof. Dr. Cleber Zanchettin
Centro de Informática / UFPE

_____

Prof. Dr. Diego Thomas
Kyushu University/Department of Advanced information Technology

ACKNOWLEDGEMENTS

Ao SENHOR, Deus dos Exércitos, que me abençoou e me guiou por todo esse tempo. Pela sua graça, misericórdia e beneficiência que dura para todo o sempre. Louvado seja o nome do SENHOR!

À Isabel, pelo apoio imensurável e pela gigante motivação para a conclusão deste trabalho. Pela preocupação, cuidado, companhia, e por nunca ter estado longe, mesmo quando não podia estar perto fisicamente. A sua participação nesse trabalho é muito maior do que você consegue imaginar. À minha mãe e padrasto, que me ensinaram que a educação é a base, e por todo o apoio que me deram.

Agradeço profundamente aos meus amigos-irmãos que estiveram comigo por essa jornada. Que procuraram me entender nos momentos difíceis de cansaço. Que oraram e torceram por mim, procurando sempre me motivar. Pela compreensão dos sumiços e dos stress que apareciam por diversas razões. Nominalmente, gostaria de agradecer ao Gustavo e ao Diego, meus irmãos que me apoiaram grandiosamente e que nunca saíram do meu lado. Sou muito grato a Deus por ter vocês na minha vida.

Aos meus amados irmãos da ICM de Engenho do Meio, Setúbal, Olinda e tantas outras que me acompanharam em oração e me receberam de uma forma sensacional. Ao Bruno, que é muito mais que um professor pra mim, pelos conselhos e preocupação, e à Candinha, que perdeu o sono várias vezes por minha causa. À Adriana, Cláudia e tantas outras senhoras que me viam como filho e cuidaram de mim durante esse longo processo. Aos meus pastores que estiveram comigo, me aconselhando e orientando sobre as mais diversas coisas: Eduardo Britto, Carlos Melo, Lafayette, Joel e Myron. Atos 2:42.

Aos meus colegas de trabalho do Voxar, pelos bons momentos de descontração e pela força nos momentos em que eu precisava (haha lol), especialmente pra Ricardo, Zé e Arlindo. Ao professor Lucas Figueiredo, que foi muito além de um orientador, pela preocupação com a minha saúde e bem-estar, e também à VT, Joma, Rafael e tantos outros que me acompanharam e deram dicas valiosas no processo todo. Claro, agradecer também à Maria, sem ela estaríamos todos loucos.

No geral, agradeço a quem esteve ao meu lado durante essa caminhada, por quem entendeu as diversas situações que eu passei. Essa vitória é nossa.

ABSTRACT

Hand pose estimation is an important task in computer vision due to its various fields of application, but mainly for providing a natural interaction between humans and machines. There are significant challenges for solving this task, primarily due to the high degree of freedom that is present in the human hand, and the possibility of self-occlusion. We investigate the usage of depthwise separable convolutions, an optimized convolution operation, to speed-up the inference time for convolutional models trained for 3D hand pose estimation. We show that the execution time for this approach can be improved to be up to 34.28% faster, while maintaining the accuracy scores on the metrics proposed by the literature. Additionally, we performed an extensive exploration and analysis of the use of depthwise separable convolutions regarding common challenges in tracking such as blur and noise, aiming to understand better in which scenarios this type of convolution impacts on the tracker precision.

Keywords: Hand pose estimation. Human-computer interaction. Deep learning. Convolutional neural networks. Depthwise separable convolutions.

RESUMO

Estimação de pose de mãos é uma tarefa importante na visão computacional pelos seus vários campos de aplicação, mas principalmente por prover uma interação natural entre humanos e máquinas. Existem desafios significativos para resolver essa tarefa, principalmente devido ao alto grau de liberdade que está presente na mão humana e a possibilidade de auto-oclusão. Nós investigamos o uso de convoluções separáveis em profundidade, uma operação de convolução otimizada, para acelerar o tempo de inferência para modelos convolucionais treinados para estimação de pose 3D de mão. Nós mostramos que o tempo de execução para essa abordagem pode ser melhorado para ser até 34,28% mais rápido, mantendo os resultados de acurácia nas métricas propostas pela literatura. Adicionalmente, nós realizamos uma extensiva exploração e análise do uso de convoluções separáveis em profundidade em relação a desafios comuns em rastreamento, como borramento e ruído, procurando entender melhor em quais cenários esse tipo de convolução impacta na precisão do rastreador.

Palavras-chave: Estimação de pose de mão. Interação humano-computador. Aprendizado profundo. Redes neurais convolucionais. Convoluções separáveis em profundidade.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AuC	Area under the Curve
CNNs	Convolutional Neural Networks
EPE	End-point Error
FPS	Frames per Second
GANs	Generative Adversarial Networks
GPU	Graphical Processing Unit
HMD	Head-Mounted Display
ILSVRC	Imagenet Large Scale Visual Recognition Challenge
LSTM	Long short-term memory
MANO	hand Model with Articulated and Non-rigid defOrmations
MLP	Multilayer Perceptron
PAFs	Part Affinity Fields
PCK	Percentage of Correct Keypoints
RHD	Rendered Handpose Dataset
RNN	Recurrent Neural Network
S&P	Salt and Pepper
SMPL	Skinned Multi-Person Linear
STB	Stereo Benchmark Dataset
UX	User eXperience
VR	Virtual Reality
XR	eXtended Reality

CONTENTS

1 INTRODUCTION

The way people naturally communicate is through expressions, such as speech, movements, and gestures. We get to discover the world by looking around and manipulating physical objects. Natural interaction is the assumption that the users should be allowed to interact with technology as they are used to interact with the real world. The interfaces proposed by this approach follow paradigms that respect human perception, creating easy and intuitive interactions (VALLI, 2005). We focus research in this field on developing systems that understand these actions, allowing the users to interact with the environment naturally.

The success of systems that implemented this paradigm are measured by the influence on who is experiencing it. The user must have the illusion that they are dealing with a system that is alive and well-respondent to its inputs, that the system can understand the user. This illusion activates the cognitive dynamics that people commonly experience in real life, leading them to believe that they are not dealing with abstract media, but with physical objects, leading to a reduction of the cognitive load and increasing the attention on what we display to the user.

1.1 HUMAN-COMPUTER INTERFACES

Hardware and software involved in the interface between humans and computers must be able to deal with typical human behaviors. However, current input devices, such as keyboards and joysticks, do not resemble what the users have experienced before, and then users need to learn how to use each new device and interface. The ideal input device to interact with machines is a non-intrusive device that disappears in the environment, and an excellent example of such a device is the human body. That is why the hand is a suitable device for naturally interacting with systems since it is the primary tool that humans have used to interact with objects and the environment since birth. Since the user already knows how to use its hand, there will be no need for adaptation with the input device, and we define the interactions and gestures that will be recognized by the system through the human factors.

Detecting the hand and its position and orientation in various contexts is a difficult task. When used freely, the hand behaves such as a non-rigid deformable shape due to the high degree of freedom in its articulations and can perform fast and irregular movements. Besides that, it can be occluded by other objects and mainly by itself (self-occlusion) (SAATMANN; JOKINEN,

2015; SPRUYT; LEDDA; PHILIPS, 2014). This series of challenges, when tracking the hands, led to the usage of various aids, such as gloves powered with sensors or visual information, to quickly identify and track its position (WANG; POPOVIĆ, 2009; Ishiyama; Kurabayashi, 2016; HAN et al., 2018).

However, the need to wear a specific device is not natural. A natural interface allows rational relation between the persons' contact with the system, in which the user must be able to move freely inside and is not imposed to wear devices that they would not wear in normal conditions. The usage of gloves introduces unnatural mediation between people and the system when the tangibility factor is not being taken into consideration. For tangible interfaces, however, the usage of physical tools and objects can favor interaction, providing a mapping between such objects and the digital media or enabling tactile sense exploitation (VALLI, 2005; VALLI, 2008). A specific scenario in which the usage of hands may propose a better interaction is Virtual Reality (VR).

1.1.1 Virtual Reality

VR may be defined as a computer-generated digital environment that can be experienced and interacted with as if that environment were real. An ideal VR system would allow users to physically walk around objects and interact with them as if they are real. One of the ways that VR systems are implemented is through the usage of Head-Mounted Displays (HMDs). Well-implemented HMDs can give the user a profound sense of presence, creating a sensation that the user is immersed in the virtual environment (JERALD, 2015). Since it is natural and intuitive to reach out for objects using hands in real life, it is also natural and intuitive to do so in VR systems.

1.1.2 Tracking

An approach that allows the user to naturally interact with a system without intrusion is through the usage of computer vision. The usage of depth information simplifies the task of hand tracking without the need wearables. Depth sensors are informative and allow systems to better predict poses in the 3D spaces, especially when extracting near-field interactions such as hand movement near cameras or from egocentric views (ROGEZ et al., 2015). The usage of depth allows for the creation of techniques that are precise, robust to self-occlusion, and able

to run in real-time.

Even though consumer-grade HMDs for VR are shipped with cameras that are used for scanning rooms, record egocentric usage and integrating real-life components into the virtual world, most of these cameras are simple RGB color cameras, without depth information.

A vision-based approach that works on simple RGB images is then needed to allow a more natural interaction between users and virtual worlds on a broader range of devices. For this, approaches based on deep learning can be applied to extract useful features from images and estimate the pose of a human hand. However, the lack of depth information makes the process complex, demanding the need for a more robust architecture.

Although such a process can be easily executed by accelerating the network's calculations on high-end Graphical Processing Units (GPUs), approaches that can be executed in real-time and on consumer-grade GPUs, such as those who meet the minimum requirements for executing VR systems are still scarce. Interest grew on accelerating inference time from deep learning-based systems for the most different approaches, and these optimizations can be brought for the task at hand.

Finally, cameras present on HMDs are in constant motion due to the movement of the user. A robust approach that proposes estimating the pose of the user's hand must be able to deal with challenges that come from having a camera that is in constant motion, such as loss of focus and occlusion of the hand. There is a demand to validate the robustness of such algorithms, since constant problems with the tool may prejudice the user's interaction and experience with the system. Interaction with virtual systems may be expanded even further when we use information about position to propose the use of gestures.

1.1.3 Gestures

Gestures are a form of communication between humans. Gesticulation, as a form of interaction, is an attractive alternative to cumbersome interfaces for human-computer interaction. Since users already use hand gestures for expressions of feelings and desires, visual interpretation of hand gestures can help to achieve the naturalness desired for interactions between humans and computers (MURTHY; JADON, 2009).

The task of detecting hand gestures can be treated as an extension of hand tracking. By using Recurrent Neural Networks (RNNs), a class of neural networks that learns from the previous events, a system is able to take into account information from the previous step to

perform an action. By feeding hand poses into an RNN, it is possible to extend a hand pose estimation system into a hand gesture detection system, allowing usage on dynamic scenarios. As proposed by (Jain; Perla; Hebbalaguppe, 2017), we can apply the estimated hand pose into Long short-term memory (LSTM), a variant of the RNNs that adds the concept of memory, allowing for more complex gestures to be recognized.

The usage of gestures can be used as an attractive alternative to cumbersome interfaces for human-computer interaction, and, since it is a common form of communication between humans, it is also natural to interact with machines through gestures.

1.2 CONTRIBUTIONS

- Implementation of a state-of-the-art approach on 3D hand pose estimation with depth-wise separable convolutions, reaching a gain of 34.28% in performance without loss of accuracy;
- A detailed evaluation of the original technique and the same technique implemented with depthwise separable convolutions according to tracking challenges such as blur, noise and occlusion;
- A comparative analysis of four different configurations of the proposed approach.

1.3 OBJECTIVES

1.3.1 General objective

Perform an exploration of the applicability of depthwise separable convolutions to optimize deep learning models to the task of 3D hand pose estimation in real-time.

1.3.2 Specific objectives

- Implement a state-of-the-art approach for the task of 3D hand pose estimation through RGB images;
- Perform a benchmark on deep learning models, comparing accuracy between models with common (dense) convolutions against depthwise separable convolutions;

- Investigate the impact of tracking challenges, such as blur and noise, on models containing dense convolutions and depthwise separable convolutions.

1.4 DOCUMENT ORGANIZATION

The organization of this dissertation follows this order: in chapter 2 we explain basic deep learning and machine learning concepts, in chapter 3 we expose the results of a literature review detailing the approaches that are aligned the most with the proposal of this dissertation. We explain and detail the proposed approach in chapter 4, and use this implementation to evaluate various scenarios that simulate tracking problems, as we describe in chapter 5. Finally, chapter 6 presents our main conclusions about this work and the future works proposals.

2 DEEP LEARNING

In this chapter we will introduce the reader to basic deep learning concepts for a better understanding of the proposed approaches and experiments in this dissertation, focusing especially on convolutional neural networks. Firstly, we explain the difference between artificial intelligence, machine learning, and deep learning. We then begin to guide the reader through the earlier notions of machine learning, which we will refer to as shallow learning (in contrary to deep learning), such as machine learning tasks and methods to explore and learn from data. A brief introduction to data representation will introduce the tensors, a specific type of data used in the machine learning field. We proceed to investigate shallow learning approaches, focusing on neural networks, a beneficial model that we can use to learn from data. For specific tasks, we need to think of models that are deeper in levels, introducing us to the concept of deep learning, in which we will focus on the convolutional neural networks.

2.1 ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, AND DEEP LEARNING

Artificial Intelligence (AI) involves using methods based on the intelligent behavior of humans and other animals to solve complex problems (COPPIN, 2004). It is a field that was born soon after the Second World War when computer scientists began asking if computers could think. For thousands of years, philosophers and physicians have tried to understand how we think, and AI attempts not just to understand it but also to build intelligent entities. AI is a general field that includes machine learning and deep learning, and also includes approaches that are not learning-based. The literature commonly references these latter approaches as symbolic AI or GOFAI (Good Old Fashioned AI).

Symbolic approaches are tailored towards a small subset of what humans can do, with an exceptional performance on well-defined problems where all the information is available in principle, and the task for the system is to decide systematically through all the possibilities. In these scenarios, programmers hardcode a sufficiently large set of rules for the manipulation of knowledge. An excellent example of this problem is the game of chess, proved by the victory of IBM's Deep Blue system against the world chess champion Garry Kasparov in the year of 1997. The system's ability to search through all the possible future moves ahead of the human player at a much faster speed was a huge advantage in this competition (HARVEY, 2013; FRANCOIS,

2017).

However, symbolic AI carries a set of problems with its implementation and limitations in its execution. With a broad set of rules implemented, the programmers found it very difficult to revise the beliefs they encoded in the engines. The more rules that they add to the system, the more knowledge the system has about this specific problem, but it is not able to undo previous knowledge, making the flow to go in a single direction. Another issue is that symbolic AI works with static problems, such as given the current position of the pieces in the chessboard, what is the best next move? Humans and other biological creatures live in a dynamic world and have been crafted to behave in an adaptive form in it. Figuring out explicit rules for solving complex and fuzzy problems such as image classification is a difficult task when using symbolic AI, and a new approach arose to take its place, machine learning.

"Can a computer go beyond our set of instructions and learn to solve a task that the scientists are unable to teach it?" This question is what motivates the field of machine learning, to understand if computers can learn on their own how to perform specific tasks rather than a programmer having to craft the rules by hand. On the contrary to symbolic AI, in which the programmers were responsible for inputting these rules to the system and expecting an answer, as shown in Figure 1, in machine learning the answers are inputted to the system, and the rules are the output, as shown in Figure 2. These rules can be used to apply to new data and produce new original answers (FRANCOIS, 2017).

Figure 1 – The symbolic paradigm, in which the programmer inputs a set of rules and the data to be processed to the system, which produces an answer.

Source: Created by the author (2020)

Figure 2 – The machine learning paradigm, in which instead of inputting the rules to the system, the programmer inputs the known answers, and the system is responsible for generating (or learning) these sets of rules from the input data.

Source: Created by the author (2020)

Engineers adapt machine learning models to the data instead of explicitly programming them. The process of learning from data is named *training* by the literature. However, for some machine learning algorithms, this terminology is not valid, since the concept of training involves the idea of recurrently analyzing the data and changing the model's internal representations to best adapt to the data that is being used in this stage. Another suggested, and better-accepted term for machine learning is *fitting*, which involves adapting the model to the data, and can best represent a variety of algorithms. These systems are presented with many examples that are relevant to a task (commonly called datasets), and their task is to find representations of the input data and their generalization for use on unseen data (NAJAFABADI et al., 2015). The performance of these systems is directly related to the quality of the data that the engineers feed the system. Poor data representation is likely to reduce the performance of any machine learning algorithm.

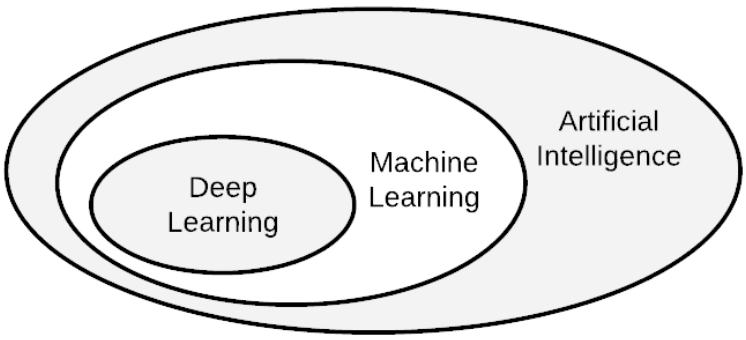
An approach that is capable of improving the capacity of generalization of the input data is to perform feature engineering. Feature engineering consists of designing a feature extractor that is able to transform the raw data into a suitable internal representation from which the machine learning algorithm could take advantage of. For some complex tasks, feature engineering is indispensable, since machine learning techniques have limited ability to process data in raw form (LECUN; BENGIO; HINTON, 2015).

An example of a complex task in which feature engineering is required is the Imagenet Large Scale Visual Recognition Challenge (ILSVRC), a benchmark in object category classification and detection on various object categories and millions of images. The goal of this competition is to estimate the content of photographs for automatic annotation of the contents of the image. For each image, algorithms produce a list of the top five object categories in descending order of confidence (RUSSAKOVSKY et al., 2015). Earlier approaches to solve this

challenge made usage of feature engineering in the format of descriptors to feed a machine learning algorithm, which would predict the contents of the image (LIN et al., 2010; PERRONNIN; SáNCHEZ; MENSINK, 2010).

The purpose of representation learning is to allow a machine learning algorithm to be fed with raw data and to discover the representations needed for solving the proposed task automatically. An approach from this paradigm is the set of techniques known as deep learning, which are models with multiple levels of representation that are obtained by composing simple linear and non-linear modules. These modules transform representations at one level into more abstract representations at subsequent levels. Deep learning is, then, a subset of machine learning algorithms that are able to extract representations in an automatic form from data (LECUN; BENGIO; HINTON, 2015; NAJAFABADI et al., 2015; BHATT; BHATT; PRAJAPATI, 2017). As shown in Figure 3, both machine learning, and deep learning are paradigms contained inside AI.

Figure 3 – Artificial intelligence, machine learning and deep learning.



Source: Created by the author (2020)

2.2 LEARNING FROM DATA

Learning is the process of improving the performance of an agent on future tasks after making new observations about its surroundings. The essential task of machine learning (and consequently deep learning) algorithms is to extract useful representations of data, transforming the input into something that approaches the desired output. A representation is a way to look at the data that is proposed to represent it. Color images, for example, can be encoded

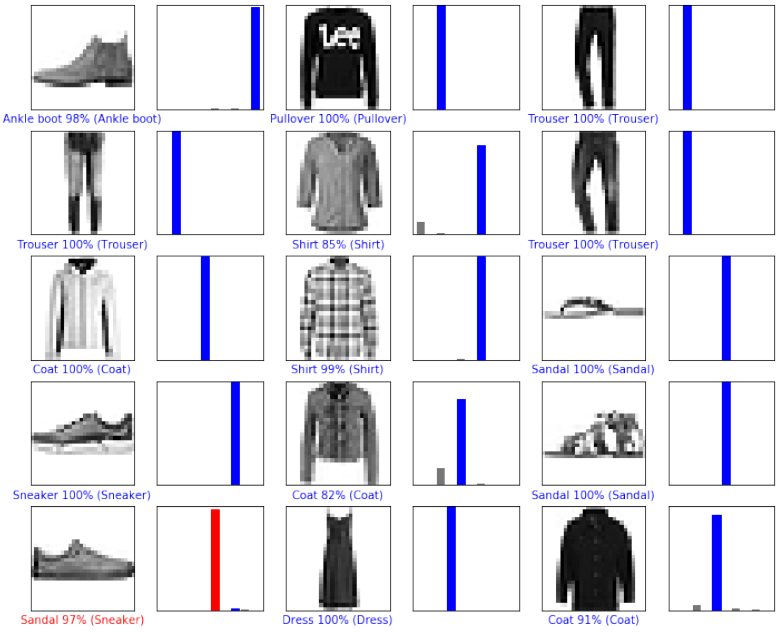
in RGB (red-green-blue) format, or in CMY (cyan-magenta-yellow) format. For humans, some of these representations are unusual. However, different representations can be useful to help shallow machine learning algorithms achieve a better understanding of the data. For example, selecting color pixels in formats such as RGB is considerably straightforward, while transform saturation is simpler in the HSV (hue-saturation-value) format. A good model is then a model capable of finding appropriate representations of the input data (FRANCOIS, 2017).

2.2.1 Supervised learning

Supervised learning is the most common case of learning. Given a training set of N paired input-output examples $(x_1, y_1), (x_2, y_2)...(x_n, y_n)$, each output is generated by an unknown function $y = f(x)$. The task of supervised learning is to discover a function h, which we call the hypothesis function, that approximates the function f. The hypothesis function is, then, a candidate of a model that correctly approximates our output. We measure the accuracy of the hypothesis on a test set containing examples that are distinct from the training set. This function generalizes well the data if it correctly predicts the value of y for examples that were not previously presented (RUSSELL; NORVIG, 2016).

In the paired examples, x and y can be of any type. We generalize this type by using a type of data commonly used for learning tasks, the Tensor, as we will explain in section 2.3. The type of y (the output) defines the task that we will solve. Given an y that is a part of a finite set of values, this task is called classification. In this task, the model is asked to specify which of k categories the input x belongs to. When $y = f(x)$, the model assigns the input x into a category identified by y. If y is restricted to two possible values, such as a 0 or 1, $true$ or $false$, the problem is a *binary classification* problem (or *two-class classification*), otherwise, it is a *multi-class classification* problem. In Figure 4, we show an example of multi-class classification.

Figure 4 – Example of a classification task. The objective is to correctly classify images of clothing, such as shoes and shirts. Correct classifications are colored blue, while incorrect are colored red. The labels expose the predicted label, the confidence and the groundtruth label.



Source: ABADI et al. (2015)

However, if y is a continuous value, such as time, the task is called **regression**. In regression, the model must predict a numerical value, given an input x by approximating a function $f : R^n \rightarrow R$. We expose an example of regression in Figure 5, an earlier result of the approach proposed in this dissertation.

Figure 5 – Example of a regression task. The objective is, given an image of a hand, to regress the 2D keypoints position in the image. We draw the estimated keypoints' positions in red and the groundtruth in green. This is an earlier result of the research in this master's dissertation.

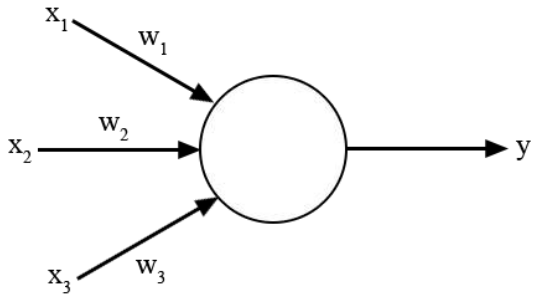
Source: Created by the author (2020)

2.2.2 Deep feedforward neural networks

An early artificial neuron model was proposed by (MCCULLOCH; PITTS, 1943) with the justification that neural events and the relations between neurons can be treated by means of propositional logic. This model simplifies the biological neuron by what was known at that time. This neuron is called the MCP Neuron (for McCulloch and Pitts), and served as inspiration for a more robust neuron model, the Perceptron (ROSENBLATT, 1962).

A Perceptron takes several real inputs and produces a single binary output, as we explain in Figure 6. One of the main improvements over the MCP Neuron was that the Perceptron has a simple rule to compute the output: the inputs $x_1, x_2, ..., x_n$ are weighted by real numbers $w_1, w_2, ..., w_n$ that represents the importance of the respective inputs to the output (NIELSEN, 2015). The neuron output is determined by whether a weighted sum is less than or greater than the threshold value, which is a parameter of the model. We explain the modeling of the output in Equation 2.1, in which t is the threshold parameter and $\sum_j w_j x_j$ is the weighted sum of the j_{th} neuron.

Figure 6 – The Perceptron neural model proposed by Rosenblatt (1962).



Source: Created by the author (2020)

$$y = \begin{cases} 0, & \sum_j w_j x_j \leq t \\ 1, & \sum_j w_j x_j > t \end{cases} \tag{2.1}$$

Although this notation works, the usage of a threshold may be simplified by the concept of bias. Bias is a measure of how easy it is to get the Perceptron to output 1. The bias is

modeled as $b = -t$, and the output is now modeled by Equation 2.2.

$$y = \begin{cases} 0, & \sum_j w_j x_j + b_j \leq 0 \\ 1, & \sum_j w_j x_j + b_j > 0 \end{cases} \tag{2.2}$$

Instead of using a weighted sum to calculate the neuron output, one may use a more robust function for this task. This concept is called the activation function, in which, following the same pipeline as the Perceptron, a specific neuron will activate if the output of this function is higher than a threshold t. Although there are many different functions available on the literature for different specific situations, the most commons activation functions used are the rectified linear unit (ReLU) and the sigmoid function.

Artificial neurons, although far from the capability of making a decision such as humans does, can weigh different evidence to make a simple decision. When combined into a network-like structure, the computational power of the model increases and may be able to solve high-complexity problems.

The most common architecture is the Feedforward Neural Networks, also called Multilayer Perceptrons (MLPs). This network defines a mapping $y = f(x; \theta)$ and learns the value of the parameters θ that results in the best $y = f(x)$ approximation. These learnable parameters are the weights, biases, and other parameters that are defined in the MLP model. The term feedforward is used because the information flows in a single direction, with no feedback connections in which the outputs of the model are fed back into itself. The architecture is called network since it is typically represented by composing together many different functions, forming a chain structure $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(f(x))))$, in which each of the $f^{(k)}$ function is referent to a layer of the network. Intermediate functions are called *hidden layers*, and by adding them, the network becomes able to extract high-order features from the data, acquiring a global perspective through the additional synapses added to the system.

A neural network contains, then, various artificial neurons connected and represented by weights and biases, which have to be tuned for a specific task. This process is called training, which is a stage that a training algorithm finds weights and biases that leads the output of the network to approximate $y = f(x)$. We quantify how well the algorithm performs this task by defining a cost function[1]. This cost function receives the weights and biases from the system, and calculates the difference between the expected result and the ground-truth annotation.

[1] Sometimes referred to loss or objective function.

The objective, then, of the training step is to minimize the cost function, finding a set of weights and biases which make the cost tends to zero.

The algorithm responsible for lowering the cost function is called the *optimizer* (NIELSEN, 2015). For each input pair $(x_1, y_1), (x_2, y_2)...(x_n, y_n)$, the optimizer will correct the weights and biases from the network with the objective of lowering the cost function. So, the network *learns* its parameters, fitting itself to a specific task.

2.3 CONVOLUTIONAL NEURAL NETWORKS

This class of networks has become one of the main algorithms in the computer vision field in recent years, proposing solutions that are more reliable and robust if compared to standard algorithms, as explained in section 2.1. Research on the design and development of models based on Convolutional Neural Networks (CNNs) has been a subject of considerable interest, mainly since there is no recipe for developing a model that can solve a variety of problems (GOODFELLOW; BENGIO; COURVILLE, 2016).

The presence of the term "convolutional" indicates that a mathematical operation called convolution is implemented on the network's architecture. This class receives this name only due to the usage of convolution in place of general matrix multiplication in at least one of their layers, and this layer is called the convolutional layer.

In this section, we will give a brief introduction to the subject, explaining operations such as convolution and pooling, the basic building blocks of the CNNs. The convolution operation employed in these kinds of architectures does not correspond precisely to the definition from other fields, such as engineering or mathematics, behaving as variants to the original meaning. We will also demonstrate how convolutions can be applied to many kinds of data.

Convolution is an operation that expresses the amount of overlap of a function g as it is shifted over another function f. The output of this operation quantifies how f is modified by g. The operation is defined by Equation 2.3, and is typically denoted by the asterisk product $s(\tau) = (f * g)(\tau)$.

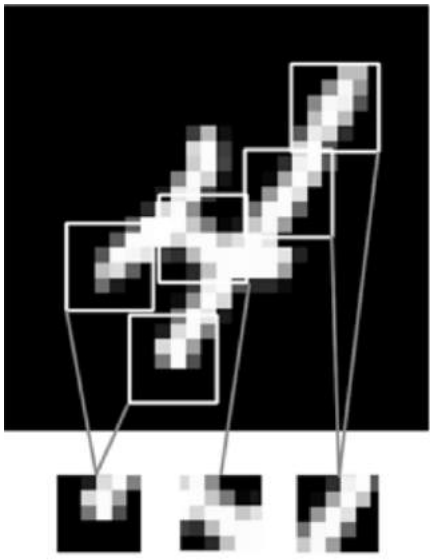
$$s(t) = \int f(\tau)g(t - \tau)d\tau \tag{2.3}$$

In the terminology of machine learning, the function f is referred to as the input. These classes of neural networks are specifically useful for processing data that has a grid-like topology. Examples include time-series data, which can be viewed as a 1-D grid of samples from regular time intervals, gray-scale image data, which can be visualized as a 2-D grid of pixels and color images, which can be visualized as a 3-D grid of pixels for a fixed number of channels. These inputs are generalized by machine learning systems and frameworks to a data structure with an arbitrary number of dimensions called tensor, which are structures that work as containers for (mostly numerical) data. A tensor containing a single scalar is a 0-dimensional tensor, while a vector is a 1-dimensional tensor, for example. When working with color images, the tensors are 3-dimensional (or third-order tensors), defined as $x \in R^{HxWxD}$, where H is the image height, W is the image width, and D is the image depth (WU, 2017).

The second function g is referred to as the kernel, a multidimensional array of parameters that are adapted by the learning algorithm during the training stage. This function outputs a feature map that can be used by another layer of the network, or else can be used as the actual output of the network.

The main motivation for using the convolution operation as an alternative to densely connected layers is that, while densely connected layers learn global patterns in their input feature space, layers that implement the convolution operation can learn local patterns. For images, these patterns are localized in 2D windows of the inputs, with its size being the kernel's window (FRANCOIS, 2017).

Figure 7 – Examples of local patterns learnt by convolutional layers.



Source: FRANCOIS (2017)

An important characteristic of these patterns is that they are translation equivariant[2], meaning that if the input changes, the output changes in the same way. A function $f(x)$ is equivariant to another function h if $f(h(x)) = h(f(x))$. When considering the convolution scope, if we define g as the function that corresponds to the translation of the local pattern $P' = g(P)$, the result is the same if we applied convolution to P' and then applied the transformation g to the output or if we apply the translation g to P and then apply convolution (GOODFELLOW; BENGIO; COURVILLE, 2016).

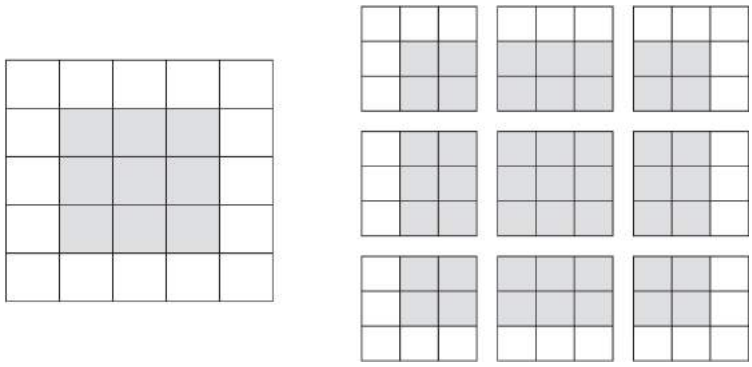
This means that, after learning a pattern in a specific region of the image, a CNN can recognize the same pattern anywhere. For densely connected networks, this pattern would have to be learned again in a new location. Since the visual world is translation invariant, this is an essential feature for processing images, as fewer samples are needed during the training phase. When stacking convolutional layers, the network can learn new patterns extracted from the already learned patterns, allowing these classes of networks to learn increasingly complex visual and perceptual concepts efficiently. We study the visualization of these patterns and the results of their activation in subsection 2.3.2.

Layers that implement the convolution operation are referred to as convolutional layers. There are two main parameters that act on these layers changing the output width and height: padding and strides. Besides these parameters, there is also the pooling layers that modifies the output feature map from convolutional layers and can act on its size.

Padding consists of adding rows and columns on each side of the input feature map (commonly the output of previous layers) to make it possible to center convolution windows around every input tile. Consider a $5x5$ feature map as the input. If the convolution operation is done with a $3x3$ kernel window, the output feature map will be $3x3$, since there are only nine tiles around which a $3x3$ window can be centered, as shown in Figure 8.

[2] Sometimes the term invariant and equivariant are used interchangeably in the literature, even having different meanings. Invariant means that there is no variance at all.

Figure 8 – All possible positions for window centering of a 3x3 kernel on a 5x5 input map.



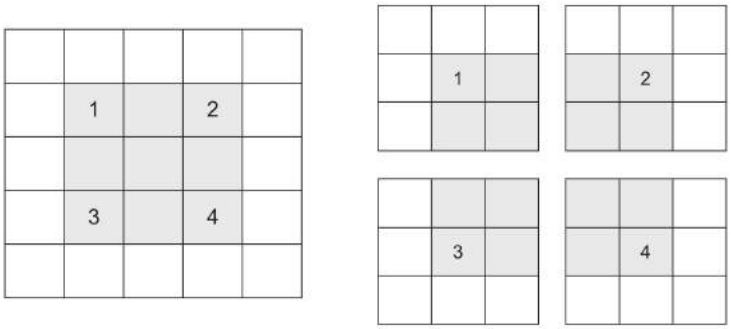
Source: FRANCOIS (2017)

When adding one column and one row for each side, the convolution can be centered around every input tile, and the output that would be reduced to $3x3$ is now $5x5$. When convolution layers have zero paddings, the size of the output maps is reduced after each convolutional layer, making the information alongside the borders be discarded quickly. These are called *valid convolutions*, which means that only valid window locations will be used, contrary to the *same convolutions*, which apply padding in such a way as to have an output with the same width and height as the input.

Convolutional layers extract features from inputs and output feature maps. However, these feature maps are sensitive to the position of the features in the input, and a way to approach this sensitivity is by downsampling the feature maps, helping to make the feature maps invariant to small translations of the input, since the values of most of the pooled outputs stays unchanged (GOODFELLOW; BENGIO; COURVILLE, 2016; JAIN et al., 2013). Then, usually applied together with the convolutional layers are the pooling layers, which replaces a region of the output feature map with a summary statistic of the nearby outputs. A common pooling layer is the max-pooling layer (ZHOU; CHELLAPPA, 1988), which reports the maximum output with a neighborhood.

Another factor that has influence in the output size of convolutional layers are the strides. This parameter controls the distance between two successive windows. In Figure 9, we can see how a convolutional operation with the stride parameter set to 2 happens.

Figure 9 – Patches that are extracted by a convolution with size $3x3$ and a stride of 2 over a $5x5$ input (FRANCOIS, 2017).



Source: FRANCOIS (2017)

Convolutions that are implemented with strides are called *strided convolutions*, and, such as pooling layers, are also used to downsample feature maps. In earlier literature, strided convolutions received little attention, but it gained focus for being computationally cheaper than applying a subsequent max-pooling layer and being able to achieve competitive results on ILSVRC (RUSSAKOVSKY et al., 2015). This led to the usage of such parameters in some of the CNNs models in the current literature (SPRINGENBERG et al., 2014; KUEN; KONG; WANG, 2016; HOWARD et al., 2017).

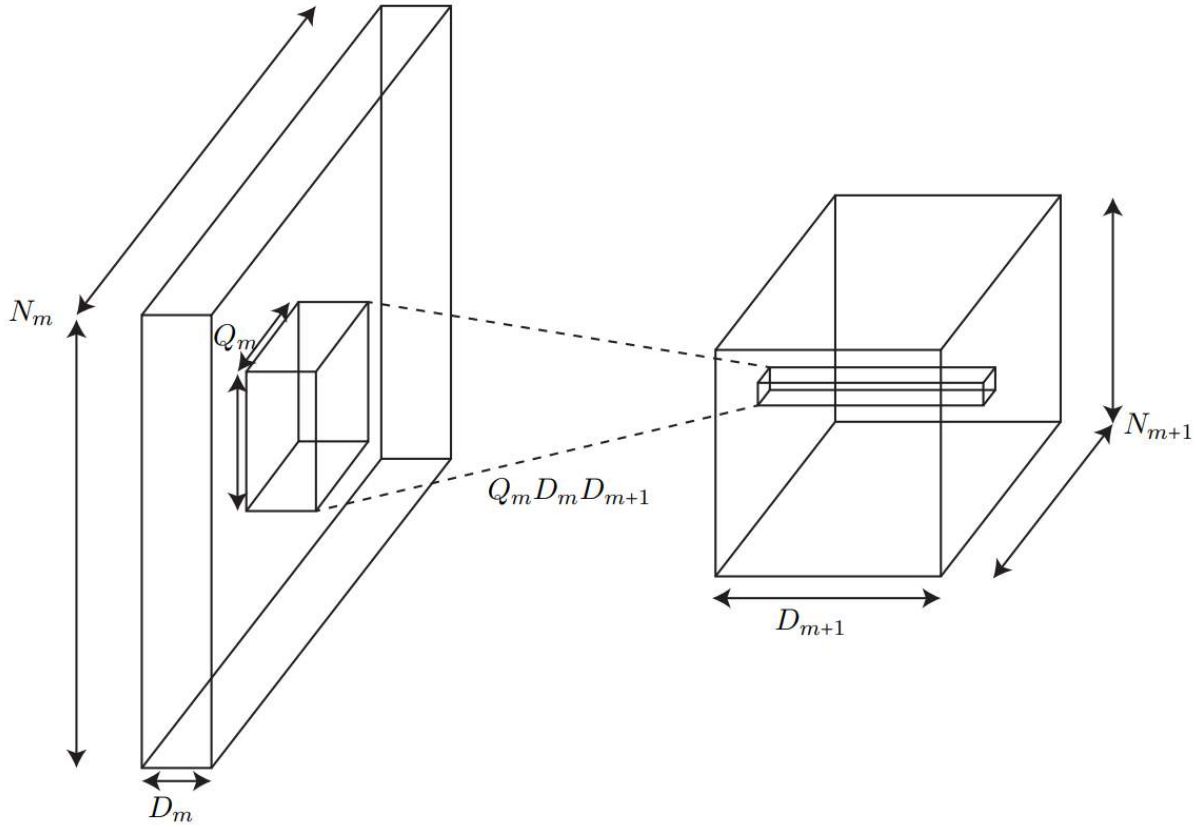
2.3.1 Optimizing the convolutional operation

Deep neural networks consist of a cascade of linear operators, which have learnable weights, and non-linear operators that transforms the data. CNNs take advantage of the image structure to limit the number of weights to be learned. However, the need to learn representation from little a priori information on the data is a difficult task, and creating deeper models to learn better these representations leads to more trainable weights. These networks involve relatively expensive convolutional operations, some of which lead to highly redundant output features (SIFRE; MALLAT, 2014). The depthwise separable convolutions proposes optimizations, reducing the cost of the convolutional operation by lowering the number of total multiplications done in the process, and reducing the redundancy in the output features.

Given an input of size (N_m, N_m, D_m), in which N_m is the input size and D_m its depth, a dense convolution would move a kernel of dimensions (Q_m, D_m), in which Q_m is the kernel size, around the image applying the convolution operation. The result of this operation is an output of size (N_m, N_{m+1}, D_{m+1}). The number of trainable parameters for this operation is

given by $Q_m \times D_m \times D_{m+1}$ and gives a vector of dimension D_{m+1} in the output layer. We illustrate the operation in Figure 10.

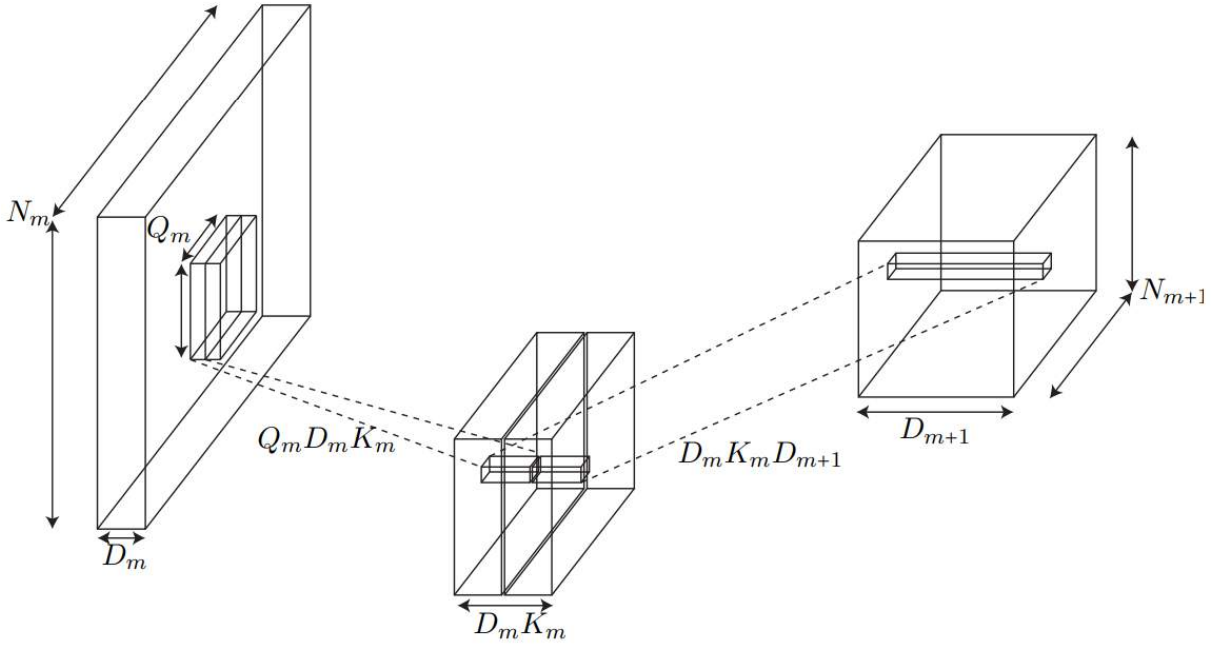
Figure 10 – Dense convolution in which a block of $Q_m \times D_m$ values are multiplied by a matrix with $Q_m \times D_m \times D_{m+1}$ weights to generate an output with dimension D_{m+1}.



Fonte: SIFRE; MALLAT (2014)

The depthwise separable convolutions proposed by Sifre & Mallat (2014) factorizes multi-dimensional convolutions. First, we apply a *depthwise convolution*, in which each input depth is filtered with K_m filters, a variable called depth multiplier that controls the number of free parameters and can be chosen arbitrarily without changing the input and output depth, of a kernel size Q_m. For this stage, we have $Q_m \times D_m \times K_m$ trainable parameters. The result of this operation is an intermediate layer of depth $D_m \times K_m$, and, at each position of the layer, the vector consisting of all depths of this intermediate layer is retransformed with a matrix multiplication, the *pointwise convolution* stage, resulting in $D_m \times K_m \times D_{m+1}$ and an output layer of depth D_{m+1}. We exhibit the operation in Figure 11.

Figure 11 – The depthwise separable convolutions proposed by Sifre & Mallat (2014). Each depth channel in the input is filtered with K_m (a variable that controls the free parameters quantity) filters of Q_m size, resulting in an intermediate layer of depth $D_m \times K_m$. At each position of this intermediate layer, the vector consisting of all depths is retransformed and positioned in the output layer of depth D_{m+1} (SIFRE; MALLAT, 2014).



Source: SIFRE; MALLAT (2014)

Therefore, a depthwise separable convolution layer can have the same output depth of a dense convolution layer, displaying the possibility of a direct replacement with less redundancy. A depthwise separable convolution layer has a total of $K_m \times D_m \times (Q_m + D_{m+1})$ trainable parameters, against $Q_m \times D_m \times D_{m+1}$ trainable weights for a dense convolution layer.

Sifre & Mallat (2014) conducted an experiment in which the first layers of the network proposed by Zeiler & Fergus (2014) are implemented with dense convolutions or depthwise separable convolutions. The results of the experiment are exposed in Table 1, and we can notice that an architecture powered with depthwise separable convolutions has fewer parameters and therefore requires less computational resources. In the experiment, the authors trained the network on the ImageNet dataset, and the architecture powered by the depthwise separable convolutions required 20% fewer steps to converge to an accuracy similar or better than the model with dense convolutions.

Table 1 – Results of the experiment conducted by Sifre & Mallat (2014). The first layers in the network of Zeiler & Fergus (2014) are swapped with a depthwise separable convolutional layer. m is the layer index, D_m is the input depth, D_{m+1} is the output depth, Q_m is the kernel size and K_m is the depth multiplier.

m	D_m	D_{m+1}	Q_m	K_m	**# of dense parameters**	**# of separable parameters**	**ratio**
1	3	96	7x7	4	14112	1740	87%
2	96	256	5x5	4	614k	107k	82%

Source: SIFRE; MALLAT (2014)

2.3.2 Understanding what CNNs learn

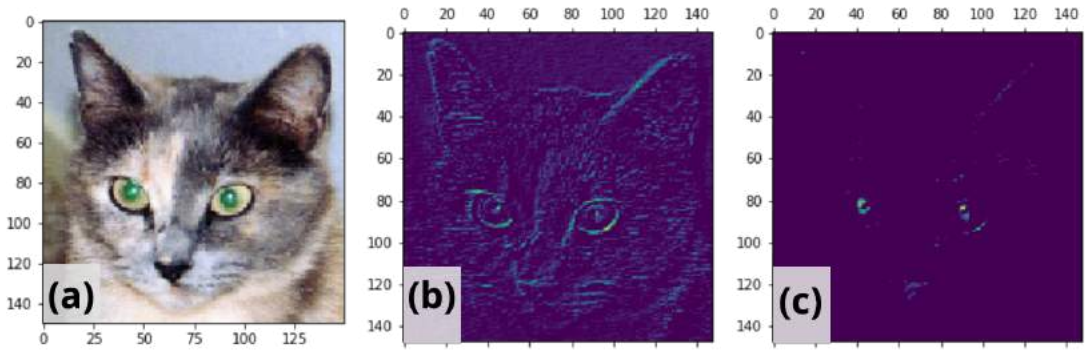
Interpretability is a demand for supervised learning that grows when a mismatch between the output of a model and what was expected happens. To trust a model, one must be able to understand why the model is performing such outputs, and the possibility to do so is still unclear for deep learning systems. Typically, there is a trade-off between accuracy and interpretability (LIPTON, 2016). Rule-based approaches such as symbolic artificial intelligence and a few shallow machine learning techniques, such as *random forests* are highly interpretable. However, as explained in section 2.1, shallow approaches such as this one fail to solve more complex tasks.

By using deep learning approaches, we trade interpretability for models that are uninterpretable and achieve greater performance. Deeper architectures, such as ResNets (HE et al., 2016), achieve state-of-the-art performance in several challenging tasks, but their complexity makes these models hard to interpret (SELVARAJU et al., 2017).

Filters are learned through the process of learning. A common way of understanding what the CNNs generalized from the training data is to visualize the output of the intermediate convolutional layers. This gives a view of how the input is decomposed throughout the network. In Figure 12, we exemplify this procedure. First, we created a model trained on the Cats and Dogs dataset[3], and then we saved the output of the layer activation. The activation of the 3rd channel (Figure 12(b)), has a high response to diagonal edges. The 30th channel (Figure 12(c)), have a higher activation on the eyes of the cat. This visualization makes a road for understanding what the CNNss are learning from data. However, it is still not sufficient to trust a prediction for most scenarios.

[3] The Cats and Dogs dataset is available in <https://www.kaggle.com/c/dogs-vs-cats>

Figure 12 – Visualization of intermediary activations of a generic convolutional model. (a) Input image from the Cats and Dogs dataset, (b) Visualization of the 3rd channel from the first layer of the proposed model. (c) Visualization of the 30th channel from the same layer.



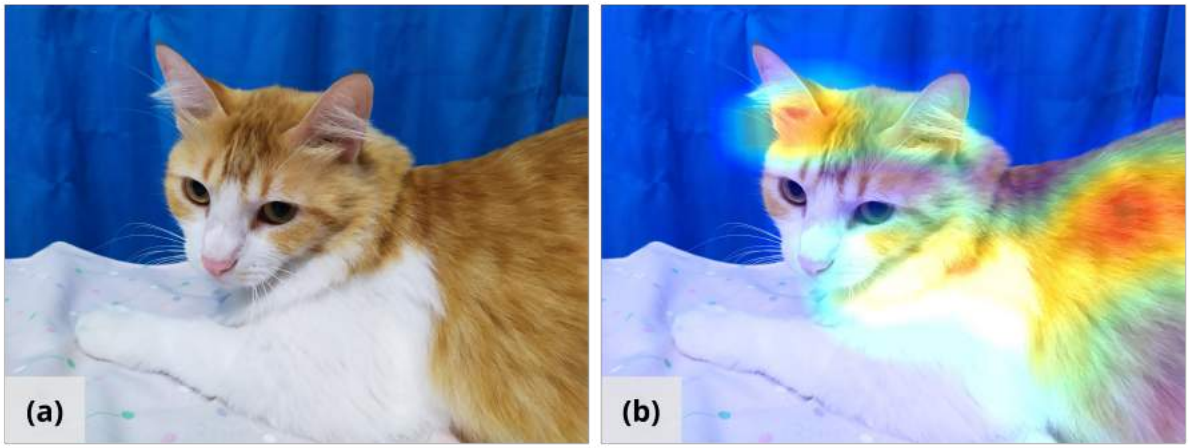
Source: FRANCOIS (2017)

A more robust visualization technique is the Grad-CAM (SELVARAJU et al., 2017). Grad-CAM is a class-discriminative localization technique that is able to generate visual explanations from a network without requiring changes in architecture or re-training. We compute the gradient of the score for the desired class c with respect to the feature maps of a convolutional layer $A^{(k)}$. The gradients are multiplied by $\frac{1}{Z}$, which is the global average pooling, to obtain the neuron importance weights α_k^c, which is the importance of the feature map k for a target class c, as explained in Equation 2.4 (SELVARAJU et al., 2017).

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{2.4}$$

Given an input image, the output of the technique is a heatmap the same size as the selected convolutional layer, which is resized to match the input image to allow the user to understand the classification. We exemplify this output when we select the last layer of the $VGG-16$ architecture, as shown in Figure 13.

Figure 13 – Output from the Grad-CAM (SELVARAJU et al., 2017) visualization technique, given the last layer of a $VGG-16$ architecture. **(a)** Input image fed to the neural network. **(b)** Resized activation map for the class "tiger cat".



Source: Created by the author (2020)

For the tasks of body pose estimation, many approaches in the literature use the output from the last convolutional layer to estimate pose, as we explain further in subsection 3.2.1. For this task, the filters from the last convolutional layer are the direct estimated location of the joints in the image instead of serving as the input for another layer responsible for the prediction.

3 LITERATURE REVIEW

This chapter surveys the literature on human pose estimation and depthwise separable convolutions and it is structured as follows: firstly, we perform an overview of the state-of-the-art in human pose estimation techniques, dividing between body pose and hand pose estimation. For hand pose estimation, which is the focus of this work, we also evaluated different proposals to solve this task efficiently, such as estimation from different input devices or approaches with a mathematical foundation. Lastly, we evaluate works on depthwise separable convolutions, from the technique to applications of the technique.

3.1 REVIEW METHODOLOGY

To review the state-of-the-art methods in both fields, we conducted an extensive search on significant computer vision and deep learning conferences and journals. From the results of this search, we selected the papers that were most aligned with the research proposed in this master's dissertation. The selected papers according to the following criteria: **(1)** since we are interested only in approaches that are able to extract features automatically in a robust way, they should use convolutional neural networks as part of its main procedure; **(2)** they must tackle the prediction and estimation of hand poses; **(3)** and at last, they should propose a solution that can be executed in real-time. For works in hand pose estimation, we also selected the papers based on the results of the metrics commonly used in the literature, such as end-point error and percentage of correct keypoints. Once the first set of papers was selected, we conducted an upwards snowball technique (WOHLIN, 2014) to access and select newer works that could represent enhancements from state of the art, and then evaluated all papers that were selected by this process.

3.2 HUMAN POSE ESTIMATION

Human pose estimation is a fundamental task in computer vision that impacts many key vision tasks, such as image understanding, animation, gaming, and activity recognition. To estimate a pose is to localize human joints in 2D images. This task can become challenging for images in which the person is occluded or has small, barely visible joints.

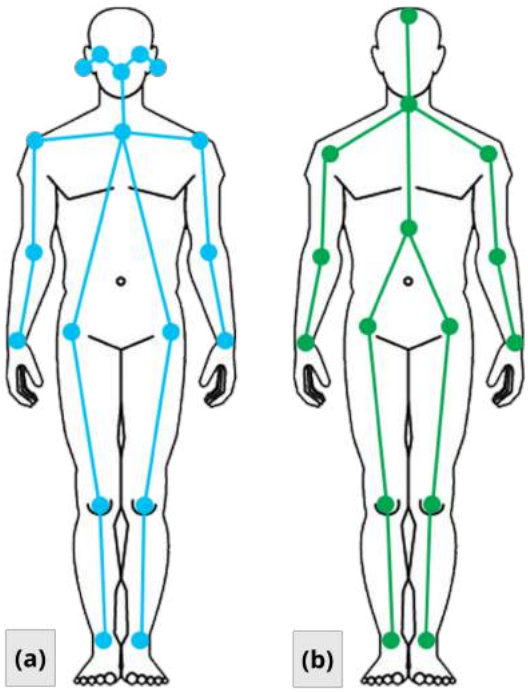
Following the literature, we define in this dissertation the problem of human pose estimation as a combination of two separate problems: the estimation of the joints from the body and the estimation of the joints from the hand.

3.2.1 Body pose estimation

Body pose estimation is the task of localizing human joints in a 2D image containing a human body. The answer to this task is a set of coordinates that can be connected to describe the pose of a person. We define as joints each of the parts that are present in the inference, and a valid connection between two joints is known as a pair.

Although different works propose different approaches and solutions for the task, all of the listed approaches require datasets that allow the model to learn from it. Therefore, the capability of generalizing the pose information depends directly on which joints are present on the dataset. In Figure 14, we exemplify the joints that are present in two of the commonly used datasets for this task: the COCO Keypoints Challenge (LIN et al., 2014) and the MPII Human Pose Database (ANDRILUKA et al., 2014). A few approaches propose to infer mathematically the location of some of the joints to allow training from datasets with a discrepancy in joints.

Figure 14 – Human joints available in two of the most common datasets for the task. Annotations from the **(a)** COCO Keypoints Challenge and **(b)** from the MPII Human Pose Database.



Source: Created by the author (2020)

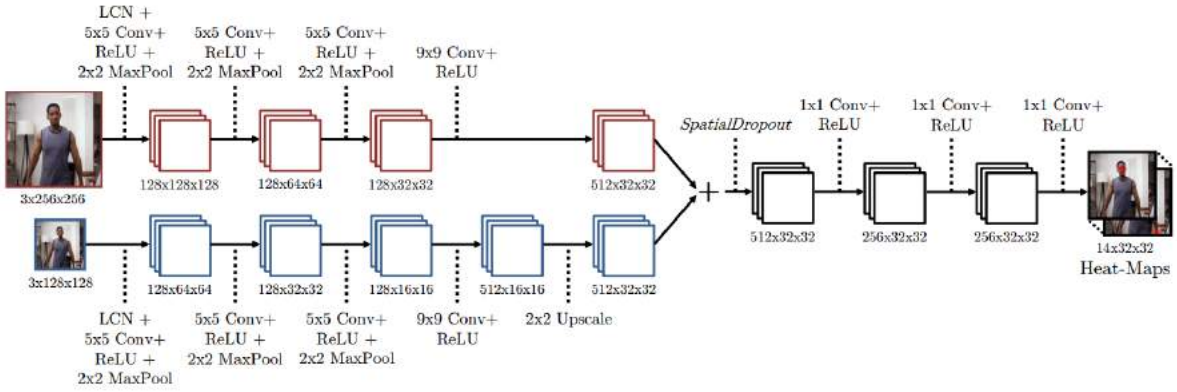
Earlier approaches treated this challenge as a regression task. Toshev & Szegedy (2014) proposed an approach that is divided in at least two stages: in the initial stage, a simple CNN is used to extract features from the image and feed them to simple fully-connected layers to regress the keypoint position x_i, y_i in the image, in which i represents the ID of the joint. This initial pose is then refined at subsequent stages in which additional regressors are trained to predict a displacement of the joint locations from the previous estimation to its actual location. Each subsequent stage is treated as a refinement to the currently predicted pose. The image is cropped around the predicted joint location, and the pose regressors can learn features for finer scales, leading to high precision. In Figure 15, we show the pipeline proposed by the authors.

Figure 15 – Architecture proposed by Toshev & Szegedy (2014). In the initial stage, the model predicts the keypoint locations x_i, y_i for the i_{th} keypoint. At the following stages, the input image is cropped around the previously detected keypoint, and a refiner module refines the pose location.

Source: TOSHEV; SZEGEDY (2014)

Tompson et al. (2015) proposed a more robust approach that generates heatmaps as outputs, which predicts the probability of the joint being in the given pixel. This approach showed to be successful, and it is continually used in the state-of-the-art literature. The proposed architecture is a multi-resolution CNN, in which the image is fed in various scales to various branches of the network. The motivation for this approach is to recover the spatial precision lost due to the usage of pooling layers. We show an overview of this architecture in Figure 16.

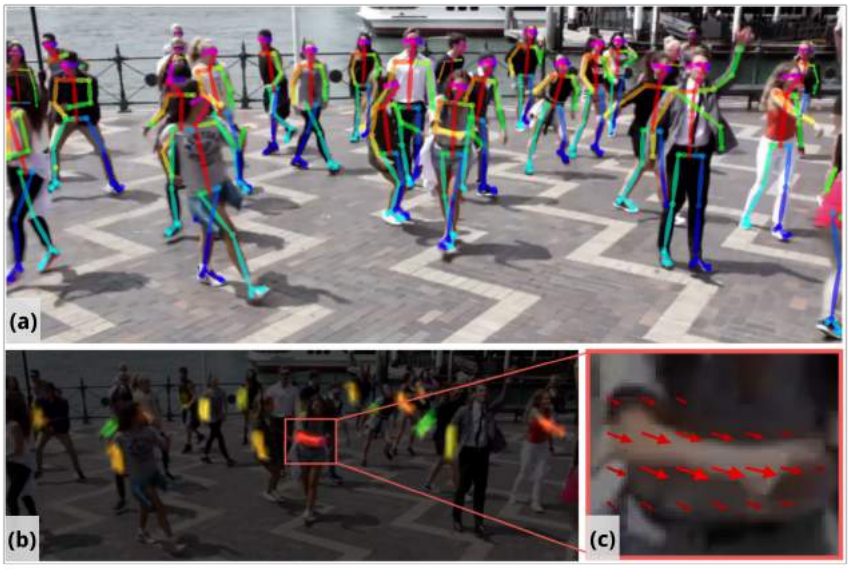
Figure 16 – Multi-resolution CNN proposed by Tompson et al. (2015). The model uses a pyramid of 3 levels (only 2 levels are shown for brevity). The output is a heatmap for each joint describing the per-pixel likelihood for the joint occurrence in each spatial location.



Source: TOMPSON et al. (2015)

Following the proposal of generating heatmaps as outputs, Wei et al. (2016) extends the *Pose Machine* (RAMAKRISHNA et al., 2014), a sequential pose estimation algorithm that predicts a confidence level for each body part and iteratively improves this estimation on various stages of the algorithm, into a *Convolutional Pose Machine*, which follows the same strategy but implementing the architecture with convolutional layers. The usage of convolutional layers in the implementation allows the model to learn feature representations for images and spatial context directly from data. The *Convolutional Pose Machine* is improved in Cao et al. (2018) to allow the association of predicted joints through an algorithm referred to as Part Affinity Fields (PAFs), which are 2D vectors fields that encode the location and orientation of the limbs over the image domain. The usage of PAFs addresses the problem of false associations when inferring pose from images with more than one person present. We show an output of this algorithm in Figure 17.

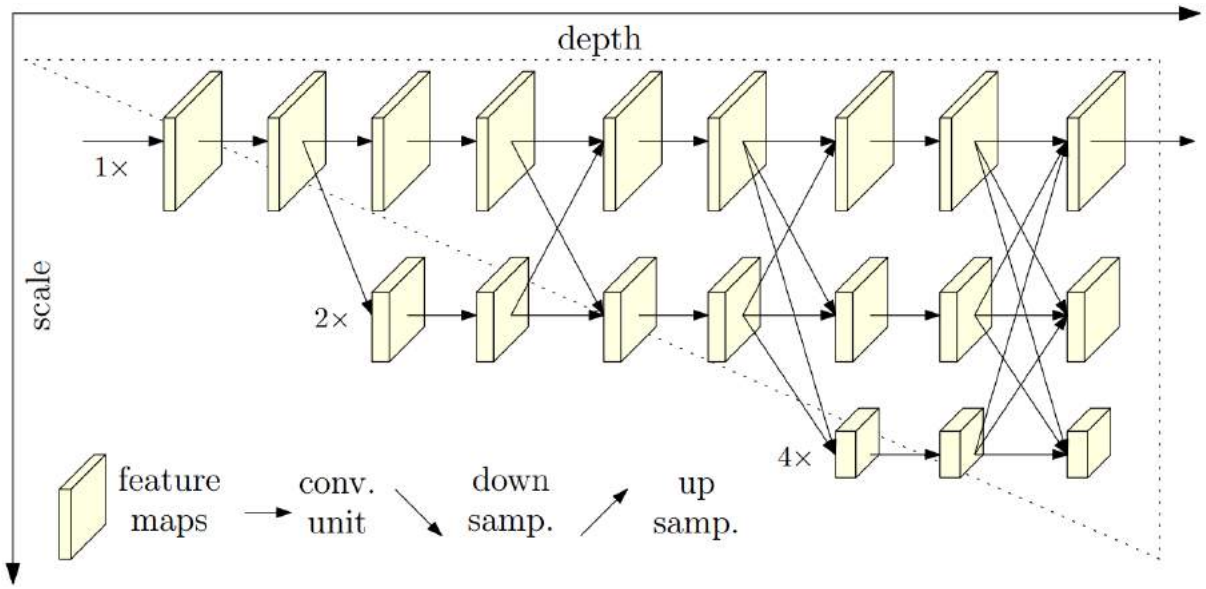
Figure 17 – Results from Cao et al. (2018). **(a)** Multi-person pose estimation with the body parts correctly associated through PAFs. **(b)** PAFs corresponding to the limb connecting the right elbow and wrist. **(c)** A 2D vector of every PAFs encoding the position and orientation of the limbs.



Source: CAO et al. (2018)

A recent approach proposed by Sun et al. (2019) uses high-resolution representation throughout the whole architecture to estimate human pose. Starting from a high-resolution subnetwork as the first stage, the authors propose to gradually add high-to-low resolution subnetworks to form more stages and connect the multi-resolution networks in parallel. Then, the exchange blocks aggregate information from all of the subnetworks and are used to estimate the heatmap. This work was able to improve the state-of-the-art due to this approach that maintains high-resolution representations of the input image. We show an overview of this architecture in Figure 18.

Figure 18 – Architecture proposed by Sun et al. (2019), consisting of parallel high-to-low resolution subnetworks with repeated information exchange blocks across the subnetworks.



Source: SUN et al. (2019)

3.2.2 Hand pose estimation

The work on hand pose estimation follows the literature of human pose estimation. In this subsection, we will introduce the key differences of the approaches to solve these two tasks.

Hand pose estimation is the task of localizing hand joints in a 2D image containing the image of a human hand. As it is with body pose estimation, the answer to this task is a set of coordinates that can be connected to describe the pose of a hand. Approaches that are proposed to solve the task of body pose estimation serves as an inspiration to various of the works present in the literature. However, the usage of the same direct approaches usually results in poor results.

3.2.2.1 RGB-D and RGB

Earlier approaches demonstrated satisfactory results for estimating poses when leveraging depth information. Depth sensors are particularly informative when extracting near-field interactions such as hand movement near HMDs or from egocentric views (ROGEZ et al., 2015), and helps to disambiguate the fingers.

Datasets containing depth information have been widely available over the academic com-

munity and has been since 2014. The NYU Hand Pose Dataset (TOMPSON et al., 2014) is known as of earlier motivators for the research, and each frame contains RGB-D data from 3 Kinects[1]: a frontal and two side views. Other datasets who contributed to the early development of the research were the ICVL dataset (TANG et al., 2014), and the MSRA dataset (SUN et al., 2015).

The availability of consumer-grade depth cameras also motivates the usage of depth information. Sharp et al. (2015) proposes the usage of the Kinect to track the user's hand and fits a detailed 3D hand model to the received depth information by using Particle Swarm Optimization (PSO).

However, methods based on optimizers, such as the one described previously, depends directly on a careful initialization of the tracker and information from the previous frames. Such tracking-methods have difficulty in handling drastic changes in the tracked hand, which are common as the hands tend to move fast (OBERWEGER; WOHLHART; LEPETIT, 2015).

Over the last years the interest of the academic community grew on applying deep learning to do such a task. In one of the earlier approaches, Oberweger, Wohlhart & Lepetit (2015) proposes a two-stage architecture, in which the first stage makes use of CNNs that are responsible for estimating a low-level representation of the hand pose[2]. The authors achieve this representation by a bottleneck, defining a smaller amount of neurons than one needed to solve this task. The low-level representation is then fed to the second stage, which is a network responsible for estimating the pose.

Following the notion of implementing CNNs to estimate pose from depth information, Ge et al. (2016) proposes a method that makes use of computer vision principles to add robusticity. Given a depth map as input, this depth map is converted to a set of 3D points in the world coordinate system by using the depth camera's intrinsic parameters. With the 3D points, the authors generate multi-view projections of the points and use this information to feed a multi-resolution neural network that outputs the heatmap for the joints.

Although research on pose estimation from RGB-D continues today, a shift in interest occurred around 2017 to allow accurate and efficient 3D hand tracking using conventional RGB cameras that exist everywhere, such as smartphones, tablets, and webcams. Another motivation for RGB cameras is that several RGB-D sensors do not provide reliable information

[1] Kinect is a motion sensing input device produced by Microsoft. More information on the Kinect sensor is available at <https://developer.microsoft.com/en-us/windows/kinect>

[2] Usually referred to as *prior* in the literature

in out-door environments.

Earlier approaches proposed the estimation of pose through various computer vision concepts (EROL et al., 2007). Segen & Kumar (1999) proposes estimating the pose through the shadow created by the hand in an environment with controlled luminosity, and Sato, Saito & Koike (2001), O'Hagan, Zelinsky & Rougeaux (2002) proposes estimating the pose from segmentation information. Besides requiring a controlled environment and calibration from cameras and software, the definition of pose estimation is different from the current definition from the literature, such definition being *"to estimate the location, orientation, and shape of the hand in an image."*.

From current literature, Panteleris & Argyros (2017) proposes to estimate pose from stereo RGB using optimization techniques. Given the stereo pair input, a distinctiveness map is calculated and cropped around the hand position. Using PSO, the hand pose is scored against a set of poses, and the best match is superimposed on the input image. However, approaches based on optimization are tricky and susceptible to loss in tracking due to the fast movements of the hand, resulting in interest for learning-based approaches that rely on datasets to learn from. For 3D hand pose estimation from RGB images, two primary datasets are available in the literature: The RHD (ZIMMERMANN; BROX, 2017) and the STB (ZHANG et al., 2016). We introduce these datasets in section 4.3.

Zimmermann & Brox (2017) proposes the first known approach to learn full 3D hand pose estimation from single color images, without the need of depth information. Due to the lack of well-annotated datasets for the task of 3D hand pose estimation, a significant contribution of this work is the proposal and validation of a strategy of using a synthetic dataset and then fine-tuning the network with a dataset containing real human hands. This proposal was adopted on several works in the literature.

Even though articulated hands have a significant number of degrees of freedom, only a restricted number of poses are possible when applied the biomechanical constraints. Therefore, there is a subspace of valid poses, and dimensionality reduction techniques may be applied to improve the accuracy of the approach. Spurr et al. (2018) proposes the usage of *variational autoencoders*, a variation of the known CNN process that generates a low-level representation referred to as latent space, which aggregates the possible poses. These representations work as a statistical hand model and are more robust than the approaches previously proposed.

The lack of well-annotated datasets is a problem for almost all deep learning tasks. To tackle this problem, Mueller et al. (2018) proposes the usage of Generative Adversarial Networks

(GANs) to perform image-to-image translation from synthetic images to real images. This approach learns mappings from synthetic images to real images and from real images to synthetic images, allowing the generation of more massive datasets. With this new dataset as input, the authors propose a CNN with a residual backbone that serves as feature extractors for 3D pose estimation.

Finally, Cai et al. (2018) proposes a weakly-supervised method to improve the training pipeline. When training on the RHD dataset, the network estimates the depth map from the 3D hand pose that is inferred from the network and uses this depth map to calculate a loss cost between the ground-truth. The loss from the inferred depth map is used during the training, improving the accuracy of the model.

3.2.2.2 Model-based and geometric approaches

Various methods propose the recovery of body mesh from an RGB image by finding parameters of a Skinned Multi-Person Linear (SMPL) model (LOPER et al., 2015). This model learns pose-dependent blend shapes from a large number of scans, producing realistic results. However, full-body models such as SMPL do not contain hands information, assuming an open, rigid hand.

Romero, Tzionas & Black (2017) collected a dataset of detailed hand scans from both the left and right hands of men and women and used this data to build a statistical hand model similar to the SMPL body model (LOPER et al., 2015). This model is called *hand Model with Articulated and Non-rigid defOrmations (MANO)*, and its availability allowed for hand pose estimation techniques to proceed to recover hand mesh, and implementations of this model as a differentiable layer (HASSON et al., 2019) allowed for models that relied on regressing its parameters (BOUKHAYMA; BEM; TORR, 2019; HASSON et al., 2019; ZHANG et al., 2019; BAEK; KIM; KIM, 2019; HASSON et al., 2019). However, it is discussed in the literature that the number of vertices in MANO is not sufficient to realistically model shape deformations of the hand (KULON et al., 2019).

An approach proposed by Kulon et al. (2019) introduces the first hand model learned from a collection of meshes. The learned model contains approximately 8.000 vertices and is capable of generating realistic hand shapes by using a mesh autoencoder. This model was achieved by using geometric deep learning, in which the convolutions are generalized to perform on graphs, allowing for models to generalize mesh information.

3.3 DEPTHWISE SEPARABLE CONVOLUTIONS

Sifre & Mallat (2014) proposed the depthwise separable convolutions during an internship at Google Brain, in 2013, inspired by prior work on transformation-invariant scattering (SIFRE; MALLAT, 2013; SIFRE; MALLAT, 2014). Laurent Sifre applied this optimized convolution on the AlexNet architecture and obtained small gains in accuracy while significant gains in convergence speed during training, as well as a meaningful reduction in model size.

Later, an efficient model for mobile and devices with low computational power was proposed by Howard et al. (2017), called Mobilenets, based on the depthwise separable convolutions.

Following the implementations of the Inception V1 and V2 modules, Chollet (2017) proposes an "extreme" inception architecture, called Xception. Standard inception modules are replaced by extreme versions, which implement depthwise separable convolutions in the reverse order: first the pointwise convolution, then the depthwise convolution.

These proposed architectures show that depthwise separable convolutions are a good fit for solving tasks such as object recognition. Other works in the literature point to the most diverse applications for this new convolution operation.

For agriculture, depthwise separable convolutions are present on architectures used for classifying diseases in plants (KAMAL et al., 2019). For medicine, Qi et al. (2019) proposes the usage of this layer for stroke lesion segmentation in brains, and Girish et al. (2019) for intra-renal cyst segmentation. For electronics, Yoo, Choi & Choi (2018) proposes the usage of architectures composed of depthwise separable convolutions to allow for large models be used on embedded systems. Other applications include action recognition (ZHOU et al., 2019), geology (ZHU et al., 2018) and many more on the current literature.

4 REAL-TIME 3D HAND POSE ESTIMATION THROUGH RGB IMAGES

In chapter 1 we discussed how natural interaction impacts systems and applications. The hand is the primary tool that humans have used to interact with objects and the environment since birth, and, for this reason, we learn how we use it from start, becoming a valuable input device for naturally interacting with systems.

Therefore, hand pose estimation can be a powerful tool to improve User eXperience (UX) in various types of applications, including eXtended Reality (XR) scenarios. A common approach for estimating 3D hand pose is to feed RGB images with depth information (RGB-D) on CNNs trained for this task. Although consumer-grade depth cameras have made these approaches popular, the need for additional hardware makes this approach unappealing, mainly because HMDs usually are shipped with simple color cameras. Besides the usage on HMDs, approaches that are based on RGB allow for a broader range of compatible devices, such as security cameras, webcams and smartphones. Thus, interest in estimating pose from RGB images grew, and many approaches were proposed to solve this problem.

We follow the pipeline proposed by Zimmermann & Brox (2017), which is composed by CNNs trained on a dataset containing synthetic hands and then fine-tuned for estimating human hands. We selected this work as our reference work considering the following motivations:

- The approach performs well on the metrics proposed by the literature and on benchmarks for this task;
- Further investigation of the architecture exhibited the possibility of improvement of the technique by the usage of depthwise separable convolutions to speed-up inference time;
- The available implementation[1] is based on Tensorflow (ABADI et al., 2015), a framework that makes deployment of models to production easy by using Tensorflow Serving.

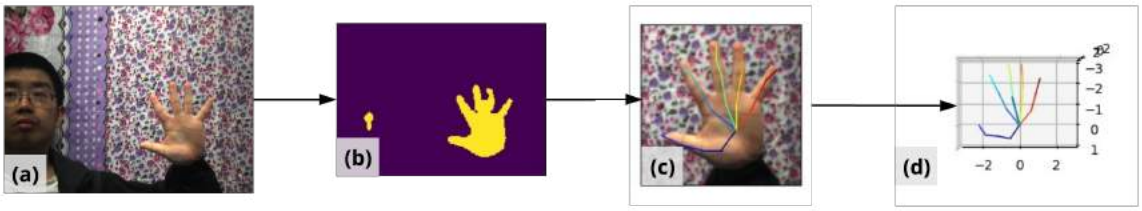
4.1 IMPLEMENTATION

Depth sensors are particularly informative when extracting near-field interactions such as hand movement near HMDs or from egocentric views (ROGEZ et al., 2015), and the lack of this information has made the task a complex one. Then, for the task of estimating pose from single

[1] The authors made available code for evaluating and training only the first step of the pipeline. The code is available at <https://github.com/lmb-freiburg/hand3d>.

RGB images, a robust architecture was proposed by the authors. The technique is composed of three stages: **1. HandSegNet (Hand segmentation)**: On the first stage, the hand is segmented using the convolutional model described in Wei et al. (2016). We use the output map to estimate the most likely location of the center of the hand, and use this position for a fixed size crop of $256x256$px. **2. PoseNet (2D keypoints estimation)**: With the cropped images, we perform the 2D keypoints estimation with a CNN trained to predict J score maps, where each map contains information about the likelihood that a certain keypoint is present at a spatial location. Predicting score maps is the conventional approach for estimating 2D keypoints, as proposed in Tompson et al. (2015), Wei et al. (2016), Newell, Yang & Deng (2016), Xiao, Wu & Wei (2018), Sun et al. (2019). **3. PosePrior (3D pose lifting)**: With the 2D coordinates in hand, we can lift the 2D keypoints to a 3D space, by using a network that learned to predict relative, normalized 3D coordinates conditioned on potentially incomplete or noisy score maps. The network is trained to predict coordinates within a canonical frame and estimate the transformation from the image world to this canonical frame. We show an example of the full pipeline in Figure 19.

Figure 19 – Intermediate and final results of the full pipeline for 3D hand pose estimation as proposed by Zimmermann & Brox (2017). **(a)** RGB image given as input to the network. **(b)** The hand is segmented by **HandSegNet** and we use the segmentation map to crop the image around the detected hand. **(c)** The cropped image is fed to **PoseNet** which estimates the 2D pose of the hand. **(d)** Finally, **PosePrior** lifts the detected pose from 2D to 3D coordinates.



Source: Created by the author (2020)

To compensate for the lack of depth information, the model must be able to extract more representative features, making the process computationally costly and highly dependent on high-end GPUs for real-time execution. Aiming to optimize the results obtained by this pipeline performing experiments to analyze if depthwise separable convolutions will succeed in speeding up such a specific task as hand pose estimation and if this approach will result in accuracy loss.

4.2 ENVIRONMENT

We conducted the experiments listed in this dissertation on a virtual environment created with the Anaconda distribution. Virtual environments serve to help manage dependencies and to isolate projects from the operating system, meaning that updates and changes to the system's settings will not affect the environment configuration. The virtual environment uses the Python programming language in the version 3.7.4 in an Ubuntu GNU/Linux 16.04 x64 host operating system.

We configured the virtual environment to use CUDA®, a parallel computing platform and programming model developed by NVIDIA® to accelerate the execution of applications through the usage of GPUs. For our setup, while we execute part of the workload in CPU (processing and filtering images, for example), the core of the operations are performed on GPU (tensor operations, for example). The development environment for the usage of CUDA® is the CUDA® Toolkit, and the version used for the experiments is the 10.0.130.

Alongside with CUDA®, we also configured our environment to use NVIDIA® cuDNN, the Deep Neural Network library, which is a GPU-accelerated library of primitives for deep neural networks. These implementations have highly optimized standard routines such as convolutions, pooling, and activation. Using cuDNN allows us to focus on the architectural problems and to train the models instead of spending time on low-level GPU performance tuning. During the experiments, the cuDNN version used was the 7.6.0.

The main framework used for training and evaluating was Tensorflow (ABADI et al., 2015), an open-source software library for machine learning. This framework was chosen mainly due to the optimizations found in the depthwise separable convolution operation. We list the main packages installed in the environment related to the applications' performance in Table 2.

We executed the codes for training and evaluating all of the proposed models in a desktop computer powered with an Intel® Core i7-4790K CPU, with 32 gigabytes of RAM. We executed the CUDA® operations on an NVIDIA RTX 2080 Ti, with 12 gigabytes of RAM using the driver version 410.93.

Table 2 – Overview of the packages installed in the virtual environment that are performance-related.

Package name	Version
cudatoolkit	10.0.130
cudnn	7.6.0
keras-gpu	2.2.4
numpy	1.16.4
opencv-python	4.1.1.26
python	3.7.4
scipy	1.3.1
tensorflow	1.13.1

Source: Created by the author (2020)

4.3 EXPERIMENTS

The first step is to prepare the datasets to be used in this experiment. As suggested by Zimmermann & Brox (2017), we use two datasets in the process: the first is the Rendered Handpose Dataset (RHD) (ZIMMERMANN; BROX, 2017)[2], which contains approximately 44 thousand images. Each sample on the dataset provides the RGB images annotations for:

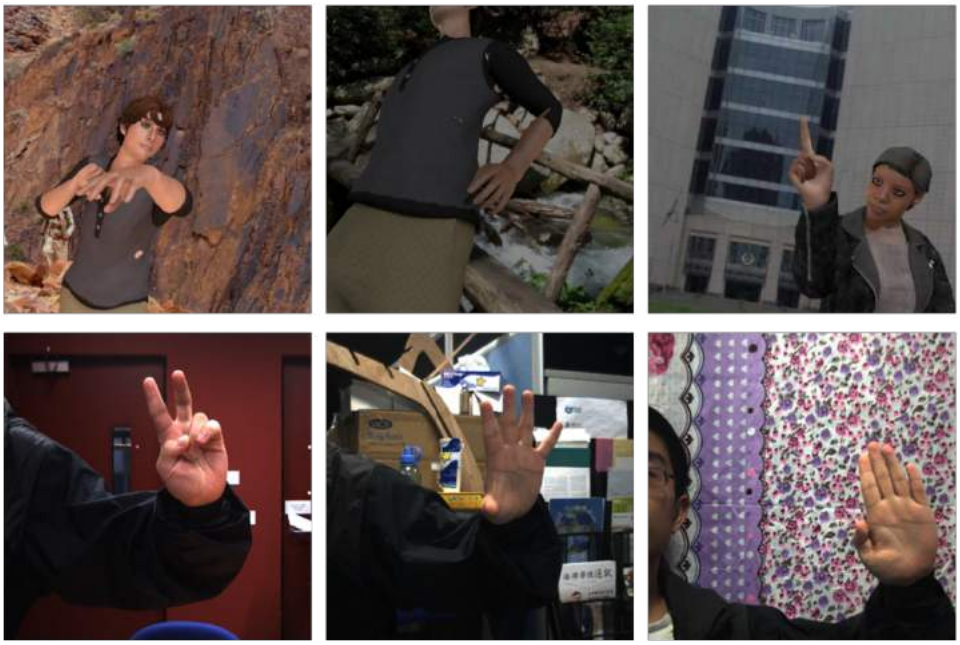
- A corresponding depth map;
- Segmentation masks for background, person and fingers;
- 21 keypoints for each hand with their *uv* and *xyz* coordinates in the world frame;
- Intrinsic Camera Matrix K.

The second dataset, containing the real hand images, is the Stereo Benchmark Dataset (STB) (ZHANG et al., 2016)[3], with six different sequences of a person performing different hand poses. Each sequence contains 1.500 frames, adding to a total of 18.000 frames in the dataset. The process of training with synthetic images and then training with real images is called *fine-tuning*, a common approach in the literature to deal with cases in which well-annotated publicly available datasets are scarce. We show examples of images from both datasets in Figure 20.

[2] The Rendered Handpose Dataset is available in <https://lmb.informatik.uni-freiburg.de/resources/datasets/RenderedHandposeDataset.en.html>.

[3] The Stereo Benchmark Dataset is available in <https://sites.google.com/site/zhjw1988/>.

Figure 20 – Sample images from RHD (ZIMMERMANN; BROX, 2017) (top row) and from STB (ZHANG et al., 2016) (bottom row).



Source: Created by the author (2020)

We split both datasets into two sets: the RHD is divided into ***R-train***, containing 41.258 images and ***R-val***, containing 2.728 images. The STB is divided into ***S-train***, containing 30.000 images and ***S-val*** containing 6.000 images. Following the procedure specified in the article, we trained *HandSegNet* on ***R-train*** whereas we trained both *PoseNet* and *PosePrior* on ***R-train*** and ***S-train***.

With the models trained and evaluated, we continued to analyze which steps of the pipeline we could optimize with the depthwise separable convolutions. We focused on the *PoseNet* stage due to the high quantity of convolution layers in this model.

We refer to the architecture proposed by Zimmermann & Brox (2017) for *PoseNet* as **original**. The authors suggest that the first 17 convolutional layers have their weights pre-loaded from the *Convolutional Pose Machines* model (WEI et al., 2016), a network trained to extract features and detect 2D keypoints of the human body. Although the task is different, these weights help the network to begin the training phase with some level of generalization of the task, performing the fine-tuning on these weights for the task at hand. Our proposed **mixed** architecture swaps convolutional layers for depthwise separable convolutional layers, except for the first 17 convolutional layers on this model, since we would lose the pre-loaded weights. We

describe the architectures for both **original** and **mixed** configurations in subsection 4.3.1.

We proceeded to train the **mixed** version using the same procedure as the **original**, aiming to compare the difference in accuracy from the layers' swap, and fairly evaluate the results. Besides the **mixed** and the **original** configurations, we experimented on a third configuration, which we refer to as **lr**, with a new learning rate schedule for the **original** configuration with the objective of achieving betters results on the proposed metrics. The results for the 2D pose estimation from *PoseNet* are described on section 4.4.

4.3.1 Training

In this subsection we detail the training procedures for each module, describing loss and training hyperparameters such as learning rate and optimizers. For all configurations except **lr**, we follow the training procedure described in Zimmermann & Brox (2017).

HandSegNet training: We train *HandSegNet* for hand segmentation on ***R-train*** for a total of 40000 iterations on the dataset, using a standard softmax cross-entropy loss and a batch size of 8. We have used the Adam optimizer with a learning rate scheduler, beginning with 10^{-5} for the first 20000 iterations on the dataset, and decaying by a scale of 10 each 10000 iterations. The architecture proposed for *HandSegNet* is exposed in Table 3.

Table 3 – Summary of the *HandSegNet* architecture, which receives a $256x256$ RGB image and outputs the correspondent hand mask.

#	Layer type	Layer shape
-	Input image	(256, 256, 3)
1	Convolution + ReLU	(256, 256, 64)
2	Convolution + ReLU	(256, 256, 64)
3	Maxpool	(128, 128, 64)
4	Convolution + ReLU	(128, 128, 128)
5	Convolution + ReLU	(128, 128, 128)
6	Maxpool	(64, 64, 128)
7	Convolution + ReLU	(64, 64, 256)
8	Convolution + ReLU	(64, 64, 256)
9	Convolution + ReLU	(64, 64, 256)
10	Convolution + ReLU	(64, 64, 256)
11	Maxpool	(32, 32, 256)
12	Convolution + ReLU	(32, 32, 512)
13	Convolution + ReLU	(32, 32, 512)
14	Convolution + ReLU	(32, 32, 512)
15	Convolution + ReLU	(32, 32, 512)
16	Convolution + ReLU	(32, 32, 512)
17	Convolution	(32, 32, 2)
18	Upsampling	(256, 256, 2)
19	Argmax	(256, 256, 1)
-	Output hand mask	(256, 256, 1)

Source: Created by the author (2020)

PoseNet training: We initialize the first 17 layers with the weights of the Convolutional Pose Machines model (WEI et al., 2016), and initialize all other layers randomly. We begin by training *PoseNet* on ***R-train*** for 30000 iterations using a batch size of 8 and a L_2 loss. Again, we use the Adam optimizer with a learning rate scheduler, beginning with 10^{-4} for the first 10000 iterations, 10^{-5} for the next 10000 iterations and 10^{-6} until the end. We then proceed to train *PoseNet* on ***S-train***, following the same hyperparameters as training in ***R-train***, except for **lr** which uses a different learning rate schedule: beginning with 10^{-5} for the first 10000 iterations and decreasing by a scale of 10 for each 1000 subsequent iterations, with a final learning rate of 10^{-8} from the iteration 25000 until the end. The architecture for *PoseNet* is exposed on Table 4.

PosePrior training: We begin by training *PosePrior* on ***R-train*** for 80000 iterations on the dataset with a batch size of 8, and using the Adam optimizer with a learning rate schedule that begins with 10^{-5} and drops to 10^{-6} at iteration 60000, using again a L_2 loss. We then proceed to train on ***S-train*** following the same hyperparameters of ***R-train***. The architecture for *PosePrior* is exposed on Table 5.

Table 4 – Summary of the original PoseNet architecture (a) and the proposed PoseNet architecture (b). The layers 1 from 17 have their weights pre-loaded from (WEI et al., 2016) and were not swapped for separable convolutions. We highlight the layers that were changed in this experiment.

	original (a)		**mixed (b)**	
#	**Layer type**	**Output shape**	**Layer type**	**Output shape**
-	Input from HandSegNet	(256, 256, 3)	Input from HandSegNet	(256, 256, 3)
1	Convolution + ReLU	(256, 256, 64)	Convolution + ReLU	(256, 256, 64)
2	Convolution + ReLU	(256, 256, 64)	Convolution + ReLU	(256, 256, 64)
3	MaxPooling2D	(128, 128, 64)	MaxPooling2D	(128, 128, 64)
4	Convolution + ReLU	(128, 128, 128)	Convolution + ReLU	(128, 128, 128)
5	Convolution + ReLU	(128, 128, 128)	Convolution + ReLU	(128, 128, 128)
6	MaxPooling2D	(64, 64, 128)	MaxPooling2D	(64, 64, 128)
7	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
8	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
9	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
10	Convolution + ReLU	(64, 64, 256)	Convolution + ReLU	(64, 64, 256)
11	MaxPooling2D	(32, 32, 256)	MaxPooling2D	(32, 32, 256)
12	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
13	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
14	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
15	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
16	Convolution + ReLU	(32, 32, 512)	Convolution + ReLU	(32, 32, 512)
17	Convolution	(32, 32, 21)	Convolution	(32, 32, 21)
18	Concat(16, 17)	(32, 32, 533)	Concat(16, 17)	(32, 32, 533)
19	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
20	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
21	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
22	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
23	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
24	Convolution	(32, 32, 21)	Separable Convolution	(32, 32, 21)
25	Concat(16, 17, 24)	(32, 32, 554)	Concat(16, 17, 24)	(32, 32, 554)
26	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
27	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
28	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
29	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
30	Convolution + ReLU	(32, 32, 128)	Sep. Conv. + ReLU	(32, 32, 128)
31	Convolution	(32, 32, 21)	Separable Convolution	(32, 32, 21)

Source: Created by the author (2020)

Table 5 – Summary of the *PosePrior* architecture, which receives a $32x32x21$ heatmap of the joints and outputs the 3D joints.

#	Layer type	Layer shape
-	Input from PoseNet	(32, 32, 21)
1	Convolution + ReLU	(32, 32, 32)
2	Convolution + ReLU	(16, 16, 32)
3	Convolution + ReLU	(16, 16, 64)
4	Convolution + ReLU	(8, 8, 64)
5	Convolution + ReLU	(8, 8, 128)
6	Convolution + ReLU	(4, 4, 128)
7	Reshape	130
8	FC + ReLU + Dropout (0.2)	512
9	FC + ReLU + Dropout (0.2)	512
10	Fully connected	512
-	Output	63

Source: Created by the author (2020)

4.4 RESULTS

We evaluated the PoseNet configurations using the metrics proposed by Zimmermann & Brox (2017). The End-point Error (EPE) metric quantifies the difference in pixels between the ground-truth and the predicted results. For this metric, we evaluated the mean and the median of all 6.000 images from ***S-val***. We also applied a metric called Area under the Curve (AuC), in which, varying a threshold in pixels, we calculate the Percentage of Correct Keypoints (PCK) and then calculate the area under this specific curve. Finally, we calculate the average and maximum inference time for each image in ***S-val***, transforming this metric into Frames per Second (FPS). For the **original** and **lr** configurations, which share the same architecture, the average frame rate was 33 FPS, achieving a maximum frame rate of 35 FPS. For **mixed**, which uses depthwise separable convolutions, the average frame rate achieved was of 38 FPS, a 15.15% decrease in inference time, and a maximum frame rate of 47 FPS, a 34.28% decrease in inference time, when compared to both **original** and **lr** configurations. We expose the quantitative results in Table 6. It is relevant to notice that even though the EPE metrics are slightly better than the original results from the authors, the AuC metric is still worse. We were not able to reproduce the exact results from the authors since there are a few gaps in information on the paper. Besides this gap, it is also significant to notice that training is not a deterministic process, leading to different results in each run.

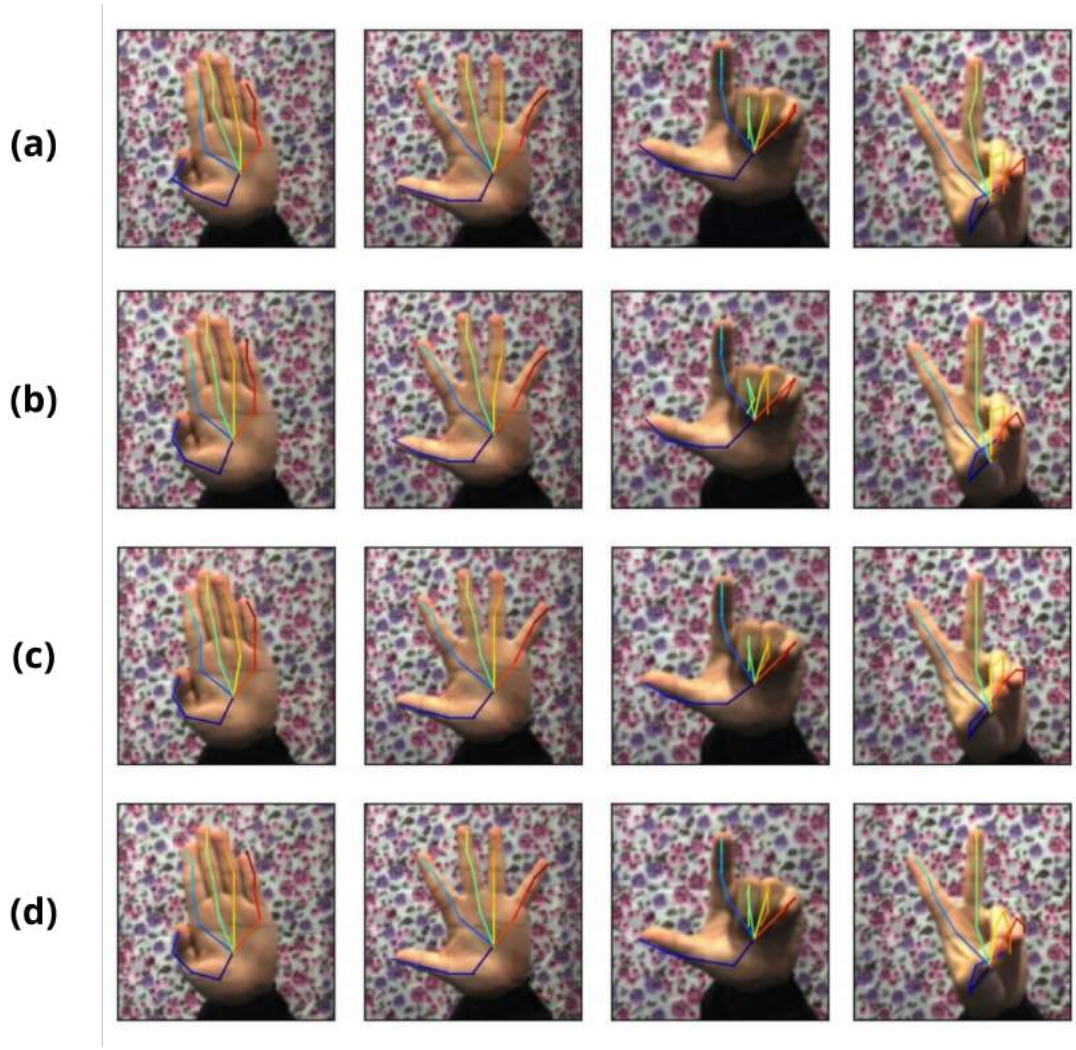
Table 6 – Quantitative evaluation of the *PoseNet* stage on ***S-val*** using the metrics proposed on Zimmermann & Brox (2017).

Metric	Zimmermann and Brox (2017)	Ours (original)	Ours (lr)	Ours (mixed)
Average mean EPE	18.52 px	16.95 px	**15.71 px**	17.60 px
Average median EPE	5.53 px	6.09 px	**5.47 px**	6.31 px
AuC	**0.762**	0.733	0.757	0.723
Average framerate	33 fps (29.5 ms)			**38 fps (26.4 ms)**
Maximum framerate	35 fps (28.2 ms)			**47 fps (21.16 ms)**

Source: Created by the author (2020)

We also proceeded to qualitatively evaluate the usage of depthwise separable convolutions by comparing the visual results from **original**, **lr** and **mixed** configurations on ***S-val***. We show the results from the qualitative analysis in Figure 21.

Figure 21 – Qualitative analysis of the proposed configurations. **(a)** shows the results from Zimmermann & Brox (2017), **(b)** from the **original** configurations, **(c)** from the **mixed** configuration and **(d)** from the **lr** configuration.



Source: Created by the author (2020)

After evaluating the proposed changes on *PoseNet*, we proceeded to evaluate quantitatively the full pose estimation pipeline on ***S-val*** to investigate if the changes in the 2D pose estimation module would have negative impacts in the final result.

We created a single model attaching each of the separate modules to estimate the pose on a single run. With the weights loaded to the model, we proceeded to evaluate the full pipeline on the proposed quality metrics and execution frame rate.

For the models containing the **original** or **lr** as *PoseNet* configurations, which share the same architecture, the average frame rate of the full pose estimation pipeline was 34 FPS, and the maximum frame rate was of 36 FPS. For the model using **mixed** as the 2D pose estimation module, the average frame rate was 39 FPS, and the maximum frame rate was 41

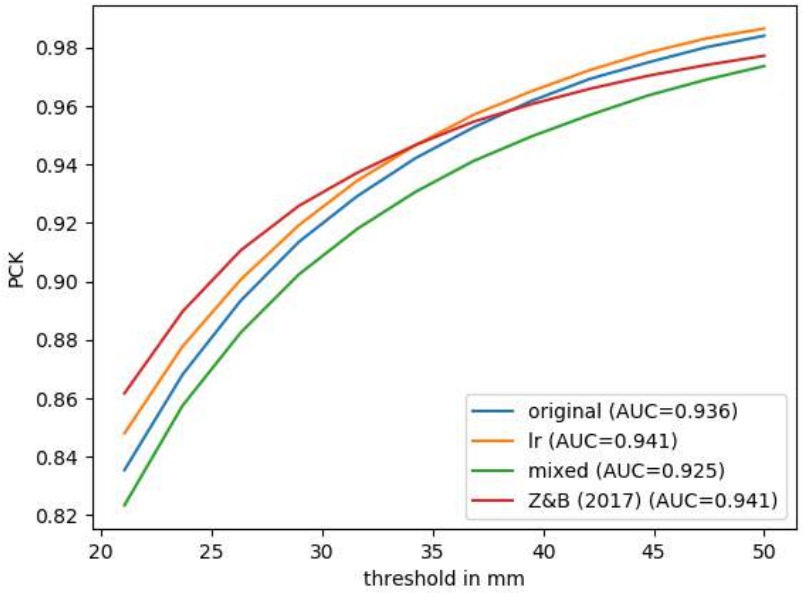
Table 7 – Quantitative evaluation of the full pipeline on ***S-val*** using the metrics proposed on Zimmermann & Brox (2017).

Metric	Zimmermann and Brox (2017)	Ours (original)	Ours (lr)	Ours (mixed)
Average mean EPE	**12.21 px**	13.01 px	12.54 px	13.80 px
Average median EPE	**9.40 px**	10.35 px	9.98 px	10.95 px
AuC	**0.941**	0.936	**0.941**	0.925
Average framerate	34 fps (28.7 ms)			**36 fps (27.8 ms)**
Maximum framerate	36 fps (27.6 ms)			**41 fps (24.5 ms)**

Source: Created by the author (2020)

FPS. We expose the quantitative results in Table 7, and in Figure 22 we visually compare the percentage of correct keypoint metric against the threshold in millimeters.

Figure 22 – The variation of the percentage of correct keypoints against a threshold in millimeters.



Source: Created by the author (2020)

4.5 ANALYSIS

We can observe from the proposed experiment that the usage of depthwise separable convolutions leads to a significant gain in inference time. The average frame rate for the **mixed** configuration was 15.15% higher than the average frame rate for **original** and **lr**. We have noticed an even higher gain for the maximum frame rate achieved, which was 34.28%

higher than the maximum achieved by **original** and **lr**.

Analyzing the results qualitatively, we can see that there is little difference between the predictions of the three configurations. Figure 21 indicates that the **mixed** configuration can be executed faster than the other two proposed configurations, and at the same time, the accuracy loss is not significant. We confirm the accuracy stability by the quantitative analysis exposed in Table 6, which shows that the average mean error for the **mixed** configuration is under 3 pixels from the other proposed configurations.

When evaluating the full pipeline, we noticed an increase in inference time for all three configurations due to internal framework optimizations. However, the **mixed** configuration continues to be faster, and there is still gain in the frame rate. For the full pipeline, the average frame rate for the **mixed** configuration was 14.71% higher than the average frame rate for **original** and **lr** configurations, and the maximum frame rate is 13.89% higher.

5 BENCHMARKING DEPTHWISE SEPARABLE CONVOLUTIONS IN MULTIPLE TRACKING PROBLEMS

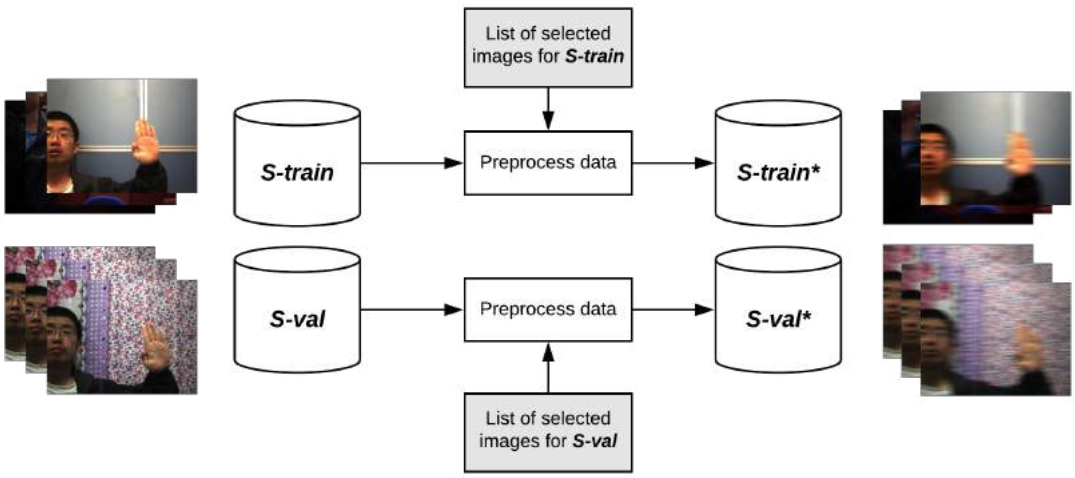
In this chapter we will expose the results for a series of experiments conducted to validate the usage of depthwise separable convolutions. Due to the slightly difference in accuracy between the **original** and **mixed** configurations, we deepened the experiments about the usage of such techniques in challenging tracking scenarios, aiming to understand if there are particular cases that the depthwise separable convolutions are better or worse than dense convolutions. First, we introduce the reader to the environment used, which we fixed throughout the experiments, then we introduce each of the experiments, explaining its structure afterward. Finally, the results of the experiments are exposed.

5.1 EXPERIMENTS

In this section we describe experiments regarding the usage of depthwise separable convolutions in images containing typical issues that make tracking difficult. The **original** and **mixed** configurations that we proposed in the previous chapter are evaluated on a new test set, in which we used standard image processing techniques to simulate some artifacts that are challenging for tracking tasks. We performed all experiments listed in this chapter on the environment described in section 4.2.

The test and training set for the proposed experiments are based on ***S-val*** and ***S-train***, respectively. For each set, we selected 20% of the images and processed them adding challenging artifacts from the proposed scenario. The same list of images is used for each of the proposed experiments, allowing for a direct comparison between the results. We refer to the processed datasets as ***S-val**** and ***S-train****. We explain the process of generating these sets in Figure 23.

Figure 23 – Generation of ***S-train**** and ***S-val**** sets.



Source: Created by the author (2020)

5.1.1 Blurred images

In this subsection we will describe the experiments on the usage of depthwise separable convolutions on images that were artificially blurred. We processed the images using *OpenCV* (BRADSKI, 2000), a computer vision library. First, we will expose our results on images that we smoothened using Gaussian blur, simulating a defocused lens. Later, we will expose our results on images that were blurred using motion blur, simulating shakiness on a photo.

5.1.1.1 Gaussian blur

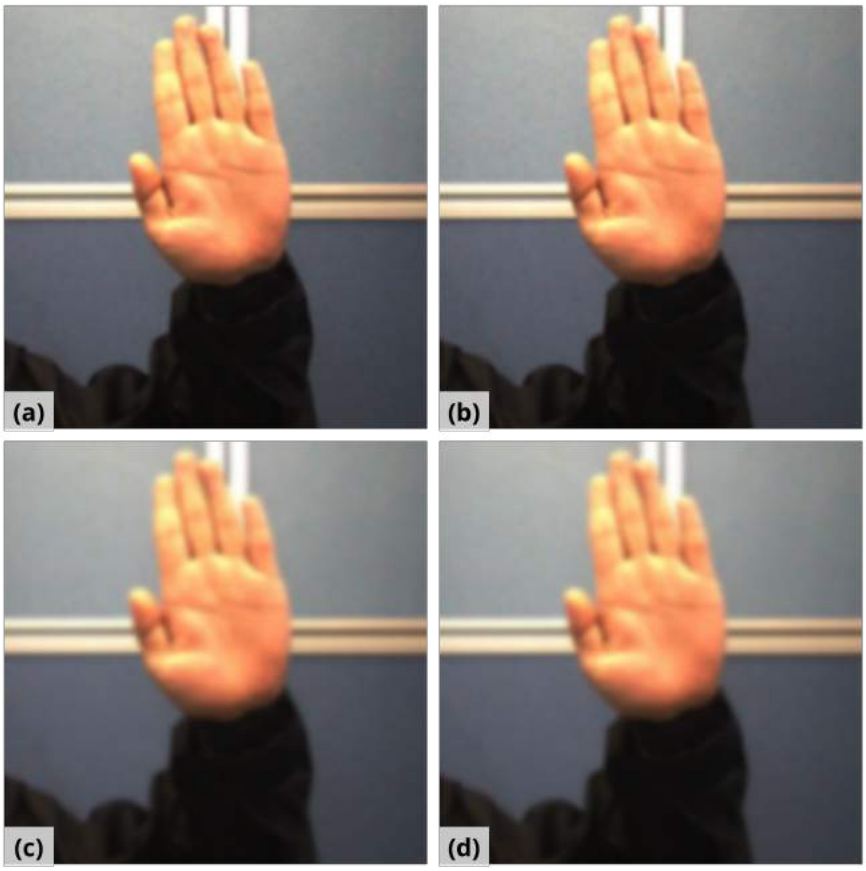
Defocused lens are a degradation in which an image is out of focus, causing a blurry effect. This degradation is challenging for computer vision algorithms since useful features that could be used for tracking or detection are lost. We used Gaussian blur to simulate a camera that is not correctly focused, and this specific type of blur was chosen since it is already applied to model different unfocused lenses better than averaging filters (GONZALEZ; WOODS, 2018).

Spatial filters, such as Gaussian blur, modify an image by replacing each pixel value by a function of the pixel and its neighbors. When applying a low-pass filter on an image, the result is a blurred (or smoothed) image. A filter such as the Gaussian filter is commonly applied to noise removal tasks, reducing irrelevant details in an image.

When blurring an image using the Gaussian filter, each input pixel is weighted by the distance of the center pixel $I[x_c, y_c]$ according to equation 5.1, where $d = \sqrt{(x - x_c)^2 + (y - y_c)^2}$ is the distance of the current pixel $[x, y]$ from the center pixel $[x_c, y_c]$ of the image in which the filter is being applied. We use these equations to generate a kernel in which the size can be arbitrarily chosen, and convolve this kernel with the image that is being blurred (STOCKMAN; SHAPIRO, 2001). For this experiment, we used three different kernel sizes: $7x7$, $13x13$ and $21x21$, each producing different outputs, as illustrated in Figure 24.

$$g(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-d^2}{2\sigma^2}} \tag{5.1}$$

Figure 24 – An image from the ***S-val*** set that was selected and processed using Gaussian blur. **(a)** Original image, without processing. **(b)** Original image blurred with Gaussian blur with kernel size of $7x7$, **(c)** $13x13$ and **(d)** $21x21$.



Source: Created by the author (2020)

5.1.1.2 Motion blur

Computer vision algorithms, such as tracking algorithms, tend to suffer due to motion blur in images. This primary cause of this problem is due to the movement of an object's position in the scene during the exposure time, which is the time that the camera shutter remains open (POTMESIL; CHAKRAVARTY, 1983). Usually, when the camera is posed on moving vehicles, such as automobiles or planes, or even when human hands hold the camera, the captured images can suffer from motion blur (YITZHAKY; KOPEIKA, 1997).

Even though many approaches propose the identification and removal of motion blur in images (YITZHAKY; KOPEIKA, 1997; CHO; MATSUSHITA; LEE, 2007; SOREL; FLUSSER, 2008; SUN et al., 2015), in this experiment we aim to understand how robust **original** and **mixed** configurations are to estimate pose in images degraded by horizontal motion blur. A motion blur kernel averages the pixel values in a particular direction, such as a directional low pass filter. We convolve the filter with the image, blurring it in a horizontal direction. The amount of blur will depend on the size of the kernel, and for this experiment, we evaluated three different kernel sizes: $15x15$, $30x30$, and $45x45$, as exemplified in Figure 25. We give an example of a horizontal $7x7$ kernel on Equation 5.2.

Figure 25 – An image from the ***S-val*** set that was selected and processed using horizontal motion blur. **(a)** Original image, without processing. **(b)** Original image blurred with horizontal motion blur with kernel size of $15x15$, **(c)** $30x30$ and **(d)** $45x45$.

Source: Created by the author (2020)

$$\frac{1}{7}\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

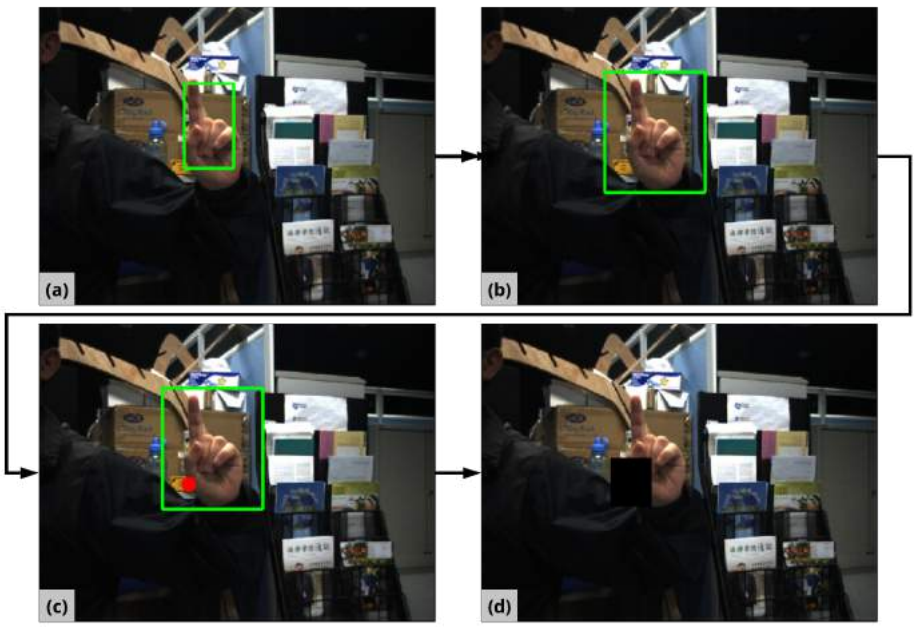
5.1.2 Occlusion

Occlusion happens when a part of the object that we want to track is covered by another object, having its features covered. Due to the loss of features, occlusion is a challenging task for trackers and other computer vision algorithms.

In this experiment, we simulate partial occlusion of the hand by covering different parts with a black square. Besides self-occlusion, which is a task-specific challenge, object occlusion is also typical when interacting with objects.

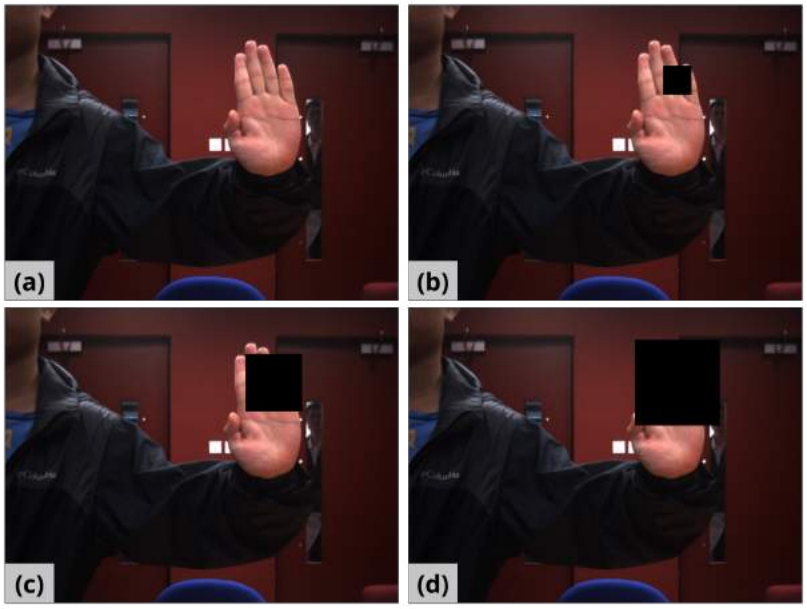
Using the ground-truth annotations, we estimate a bounding-box around the area of the hand by calculating the minimum and maximum positions for all the keypoints. We add a margin of 15% to this bounding box to better allocate the hand inside of it. Using the positions of the bounding box, we randomly select a point x, y that we use as the center of the occlusion square. We use the same point for each set, allowing for the direct comparison of the results. We then proceed to draw the square over the image, using different ratios to increase the total area of the square. We exemplify the pipeline on Figure 26 and the results for each of the ratios in Figure 27.

Figure 26 – Full pipeline of the occlusion process. **(a)** The bounding-box is detected using ground-truth annotations. **(b)** We expand the bounding-box with a 15% margin. **(c)** We estimate a random point inside the bounding-box and use it to **(d)** draw a square over the image.



Source: Created by the author (2020)

Figure 27 – An image from the ***S-val*** set that was selected and processed adding occlusion. **(a)** Original image, without occlusion. **(b)** Square with the 0.1, **(c)** 0.2 and **(d)** 0.3 parameters.



Source: Created by the author (2020)

5.1.3 Noise

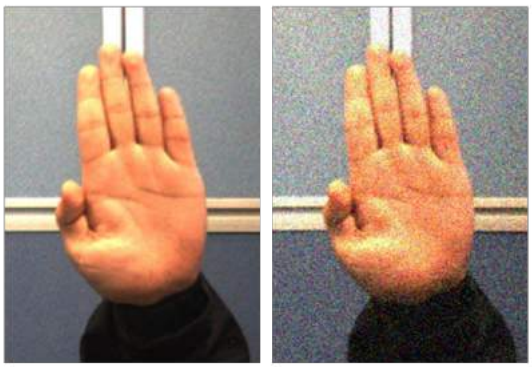
Noise is an unwanted component of the image, which occurs in images for various reasons, mainly to faulty camera sensors or transmission in a noisy channel (CHARLES, 2005; CHAN; HO; NIKOLOVA, 2005). In this subsection, we experiment on two common types of noise in images. We processed the images for the sets using *scikit-image*[1] (WALT et al., 2014) library, a collection of algorithms for image processing.

5.1.3.1 Gaussian noise

Gaussian noise is probably the most common noise, resulting from poor illumination from the scene or high temperature from the sensors (CHARLES, 2005). This noise is an additive noise, and we implement it by adding to the image random noise samples from a normal distribution with mean $\mu = 0$ and standard deviation $\sigma^2 = 0.01$. The *scikit-image* library already contains a module that allows for the direct application of Gaussian noise. We show a result of the implementation on Figure 28.

[1] Available at <https://scikit-image.org/>

Figure 28 – Original image from the training set against its augmentation with Gaussian noise.



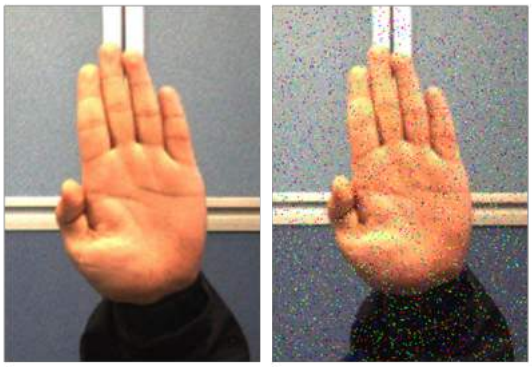
Source: Created by the author (2020)

5.1.3.2 Salt and pepper

Salt and Pepper (S&P) noise generalizes a variety of processes that results in the same image degradation in which only a few pixels are noisy, but they are very noisy (CHARLES, 2005). Errors in image acquisition are what generally causes degradation of images by S&P (TOH; ISA, 2009).

Setting a probability $r = 0.05$ that a pixel is corrupted, we apply the S&P noise by setting $\frac{r}{2}$ randomly selected pixels to *salt* and another $\frac{r}{2}$ randomly selected pixels to *pepper*. The *scikit-image* library already contains a module that allows for the direct application of S&P noise. Since we are working on color images, instead of having black-and-white pixels on the image, the noisy pixel are red, green, or blue. We show a comparison of the original image against the S&P noise in Figure 29.

Figure 29 – Original image from the training set against its augmentation with S&P noise.

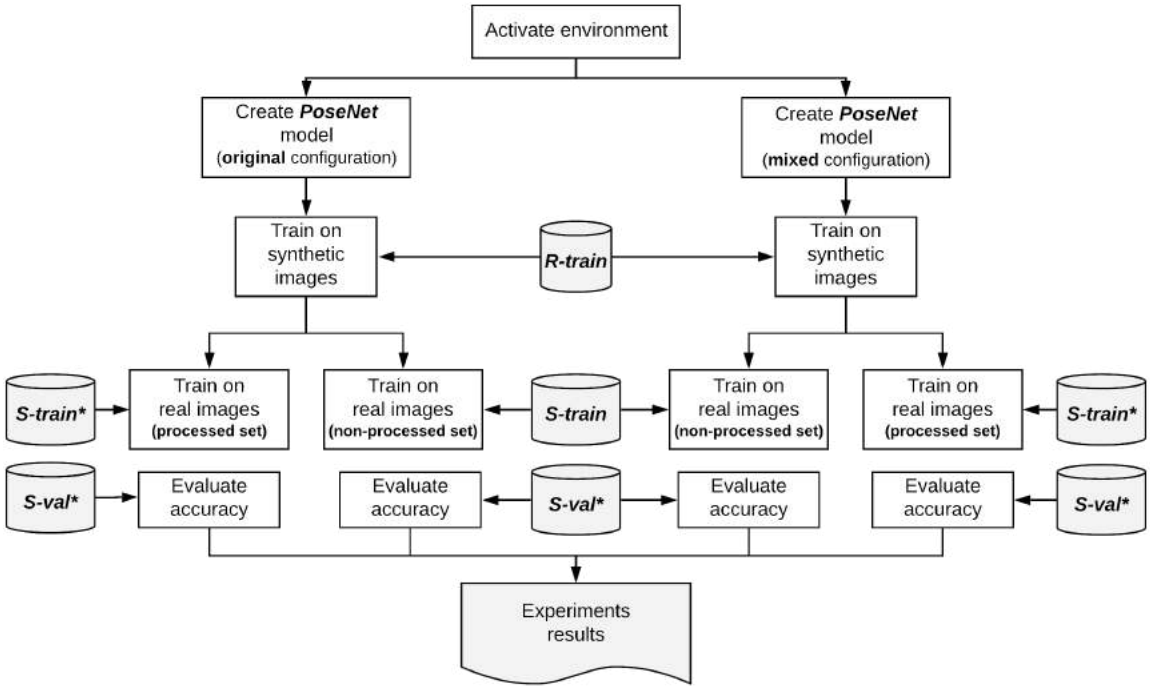


Source: Created by the author (2020)

5.2 RESULTS

In this section, we expose the results from the experiments proposed in chapter 5. For each of the proposed configurations, we apply a pipeline to train and evaluate accordingly to each of the proposed tracking challenges. We also evaluate two different scenarios: both scenarios consist of evaluating on ***S-val****, however, in the first scenario, we only train the configurations on ***S-train***, evaluating the capability of each configuration to deal with unseen challenges, while training on ***S-train**** in the second scenario. We explain the full pipeline on Figure 30.

Figure 30 – Evaluation process for the proposed experiments.



Source: Created by the author (2020)

Following the same evaluation metrics explained in section 4.4, we evaluate the models using EPE and AuC metrics on all of 6.000 images of ***S-val****. For the experiments involving blur, we expose the results for the Gaussian blur task in Table 8 and for the motion blur task in Table 9. In these experiments, we tested different kernel sizes to simulate various intensities of the proposed challenges, which are the rows of the tables. We also show a few qualitative results in Figure 31 and Figure 32. For each of the image pairs from the qualitative results in this section, the first image is the complete image that we use as input to the network, with the estimated 2D hand pose drawn over the image. The second image on the pair is zoomed

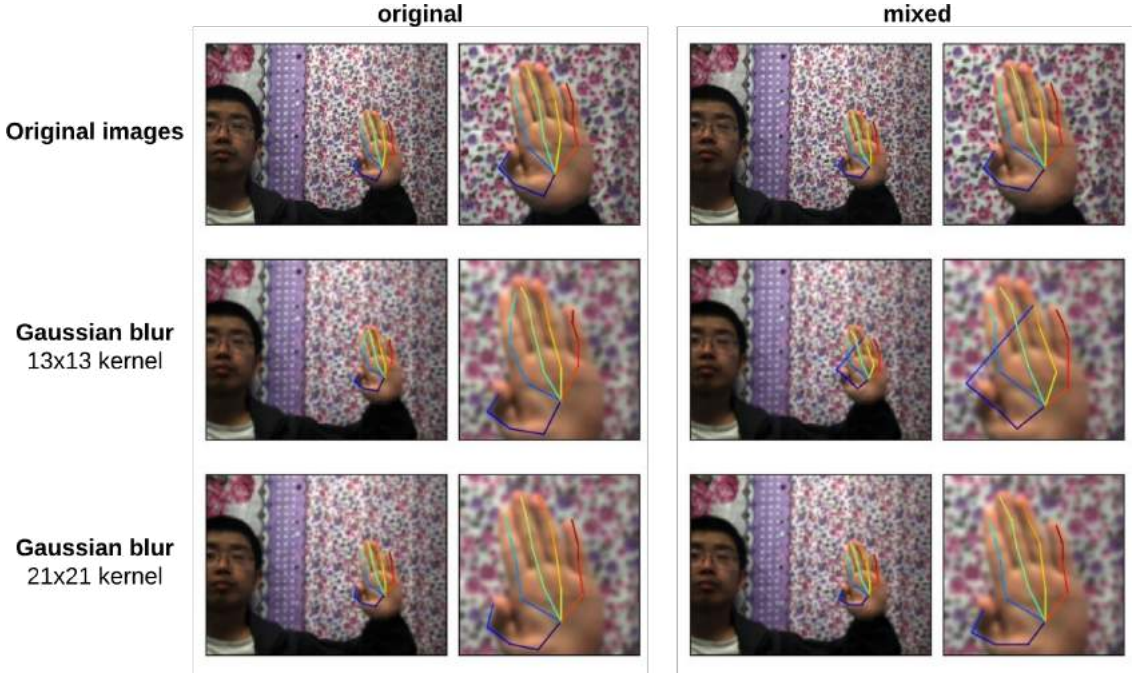
on the hand that is segmented by *HandSegNet*, and is the image fed to *PoseNet* for pose estimation.

Table 8 – Quantitative evaluation of **mixed** and **original** architectures on ***S-val**** processed with Gaussian blur, following the metrics proposed by Zimmermann & Brox (2017).

Train set	Kernel size	original			mixed		
		Average mean EPE	Average med. EPE	AuC	Average mean EPE	Average med. EPE	AuC
Standard	-	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	7x7	**22.745**	**6.577**	**0.681**	23.939	6.785	0.670
	13x13	**26.251**	**7.121**	**0.628**	27.923	7.313	0.623
	21x21	25.554	**6.995**	0.641	**25.402**	7.117	**0.642**
Trained on S-train*	7x7	23.153	**6.798**	**0.671**	**24.266**	7.062	0.659
	13x13	26.784	**7.387**	**0.626**	**26.120**	7.487	**0.626**
	21x21	25.629	**6.998**	0.641	**25.390**	7.119	**0.642**

Source: Created by the author (2020)

Figure 31 – Qualitative results for the Gaussian blur experiment. The first row displays the original results, without the usage of Gaussian blur. The second and third row displays results of images with Gaussian blur, using $13x13$ and $21x21$ kernels, respectively.



Source: Created by the author (2020)

Table 9 – Quantitative evaluation of **mixed** and **original** architectures on ***S-val**** processed with motion blur, following the metrics proposed by Zimmermann & Brox (2017).

Train set	Kernel size	original			mixed		
		Average mean EPE	Average med. EPE	AuC	Average mean EPE	Average med. EPE	AuC
Standard	-	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	15x15	**26.331**	**7.122**	**0.627**	27.861	7.298	0.624
	30x30	31.039	7.331	0.599	**27.914**	**7.73**	**0.609**
	45x45	**32.475**	**7.33**	**0.597**	35.906	7.592	0.589
Trained on S-train*	15x15	**24.761**	**7.059**	**0.644**	26.965	7.279	0.634
	30x30	28.027	8.345	0.585	**27.930**	**7.737**	**0.609**
	45x45	**28.422**	**7.894**	**0.609**	31.142	8.111	0.593

Source: Created by the author (2020)

Figure 32 – Qualitative results for the motion blur experiment. The first row displays the original results, without the usage of motion blur. The second and third row displays results of images with motion blur, using $30x30$ and $45x45$ kernels, respectively.

Source: Created by the author (2020)

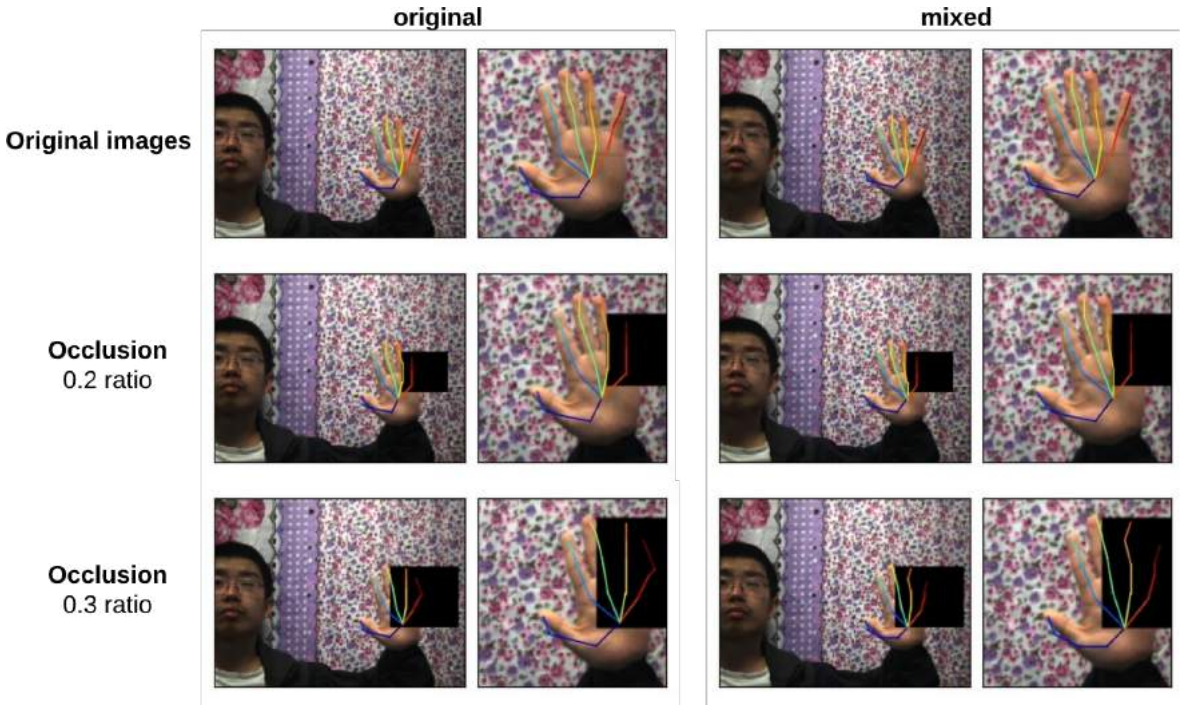
Continuing with the proposed experiments, we evaluate the results on occlusion. For this experiment, we vary the parameter that is responsible for controlling the square's size. We expose the results for this challenge in Table 10, in which each row is relative to a different square size, that is controlled by a ratio. We also expose a few qualitative results in Figure 33.

Table 10 – Quantitative evaluation of **mixed** and **original** architectures on ***S-val**** processed with occlusion, following the metrics proposed by Zimmermann & Brox (2017).

Train set	Ratio	original			mixed		
		Average mean EPE	Average med. EPE	AuC	Average mean EPE	Average med. EPE	AuC
Standard	-	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	0.1	**18.372**	**6.207**	**0.722**	19.135	6.427	0.711
	0.2	**22.085**	**6.412**	**0.697**	22.902	6.663	0.686
	0.3	**28.583**	**6.731**	**0.662**	29.854	6.980	0.651
Trained on S-train*	0.1	**18.725**	**6.426**	**0.717**	18.939	6.663	0.710
	0.2	21.444	6.497	0.696	**21.222**	**6.563**	**0.699**
	0.3	27.783	6.843	0.667	**27.717**	**6.800**	**0.670**

Source: Created by the author (2020)

Figure 33 – Qualitative results for the occlusion experiment. The first row displays the original results, from images without the occlusion challenge. The second and third row displays results of images in which we occluded the area of the hand, using a ratio of 0.2 and 0.3, respectively.



Source: Created by the author (2020)

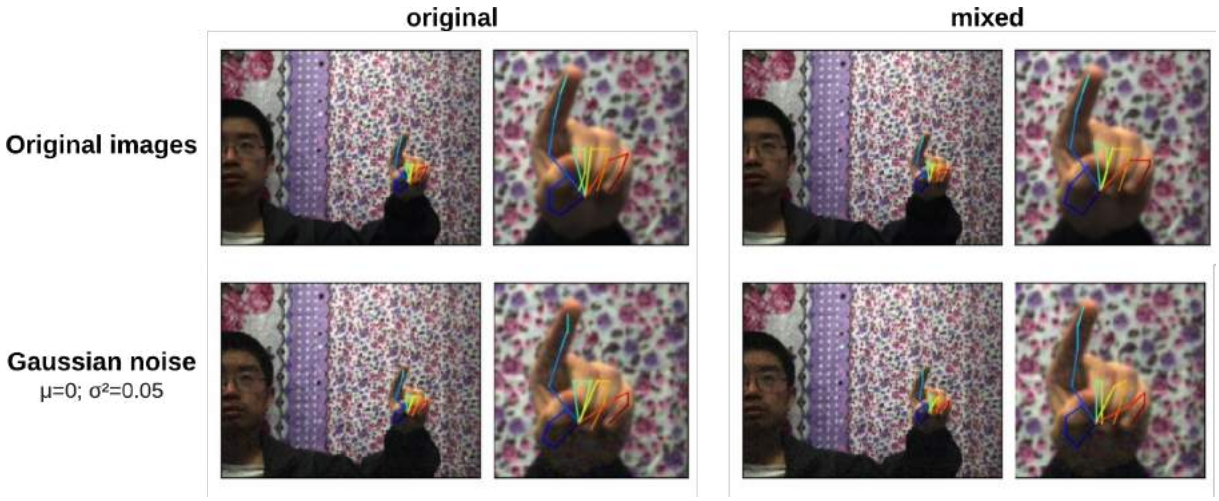
Lastly, we proceed to evaluate the models on noisy images. In this experiment, we inserted noise following the default parameters provided by *scikit-image*, which are common parameters found in the literature. We expose the results for this challenge in Table 11 for Gaussian noise and Table 12 for S&P. We also show a few examples of qualitative results in Figure 34 and Figure 35.

Table 11 – Quantitative evaluation of **mixed** and **original** architectures on ***S-val**** processed with Gaussian noise, following the metrics proposed by Zimmermann & Brox (2017).

Train set	original			mixed		
	Average mean EPE	**Average median EPE**	**AuC**	**Average mean EPE**	**Average median EPE**	**AuC**
Standard	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	**38.845**	**7.326**	**0.597**	38.997	7.588	0.587
Trained on S-train*	**26.109**	**7.062**	0.636	25.61	7.238	**0.637**

Source: Created by the author (2020)

Figure 34 – Qualitative results for the Gaussian noise experiment. The first row displays the original results, from images with no added noise. The second row displays results of images in which we added noise using the *scikit-image* library, with $\mu = 0$ and $\sigma^2 = 0.01$ as the parameters from the normal distribution.



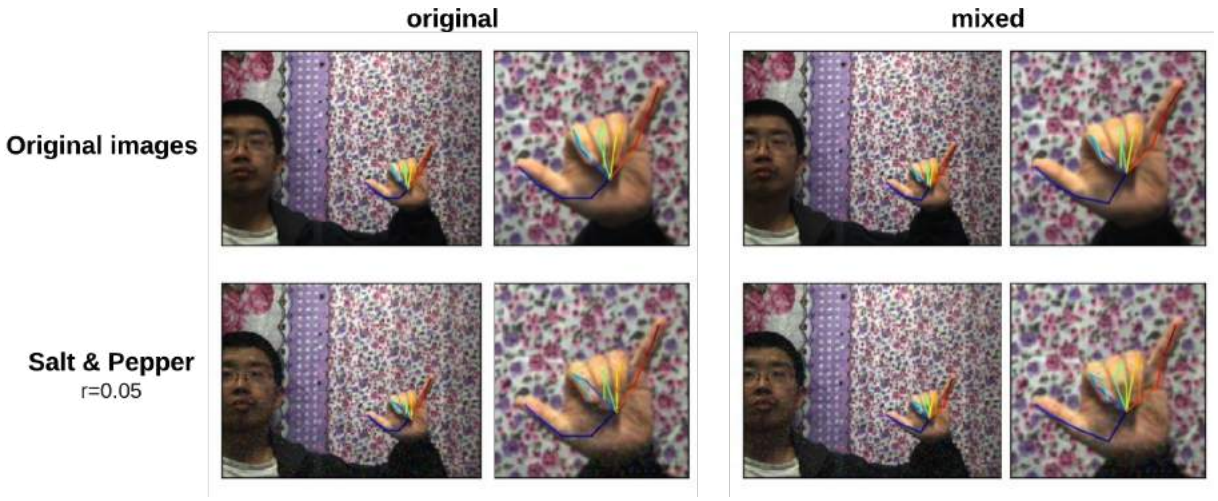
Source: Created by the author (2020)

Table 12 – Quantitative evaluation of **mixed** and **original** architectures on ***S-val**** processed with S&P noise, following the metrics proposed by Zimmermann & Brox (2017).

Train set	original			mixed		
	Average mean EPE	**Average median EPE**	**AuC**	**Average mean EPE**	**Average median EPE**	**AuC**
Standard	**16.958**	**6.095**	**0.733**	17.608	6.313	0.723
Trained on S-train	**32.361**	**7.323**	**0.598**	35.895	7.599	0.588
Trained on S-train*	30.161	8.075	0.586	**28.826**	**7.939**	**0.602**

Source: Created by the author (2020)

Figure 35 – Qualitative results for the S&P noise experiment. The first row displays the original results, from images with no added noise. The second row displays results of images in which we added noise using the *scikit-image* library, with $r = 0.05$ as the probability that a pixel is corrupted.



Source: Created by the author (2020)

5.3 DISCUSSION

From the experiments' observed results, we aimed understand how the usage of depthwise separable convolutions can impact on a deep learning solution that needs to be robust to common tracking challenges.

For the challenge of blurry images, on the experiment involving Gaussian blur, which aims to simulate the loss of focus from a camera, we can see from the quantitative analysis proposed on Table 8 that the **original** configuration performs slightly better on ***S-train***. This performance difference means that this configuration can perform well even when not trained on such a

challenge. A possible reason for this characteristic is due to the number of trainable parameters available in this configuration, which is higher than the quantity in **mixed**, as we explained in subsection 2.3.1. This pattern is present in all of the results from the proposed experiment. However, when trained on ***S-train****, the **mixed** configuration outperforms **original**, yielding lower average mean EPEs for all kernel sizes.

For the average median EPEs, the **original** configuration outperforms **mixed** in all observed scenarios, with a maximum difference of 0.264 pixels. The difference between the average mean EPE metric and the average median EPE metric is substantial and way higher than the standard metric results. A possible reason for this difference is that the Gaussian blur may be leading the network to drastic errors, generating high outliers, and prejudicing the metric. It is also possible to observe that the 21×21 kernel is less prejudicial to the task than the 13×13 kernel, as we can also see in Figure 31, in which the **mixed** configuration is unable to correctly estimate a joint on the image with a 13×13 kernel applied but performs well when the 21×21 kernel is applied. Currently, we have no conclusions regarding this curious behavior.

When observing the results from the motion blur experiments, we notice that the difference between the average mean and median EPE is also present and that all configurations tend to produce higher errors on the average mean EPE metric. Again, this challenge leads the network to create a higher quantity of outliers, and this severely prejudices this metric. The **original** configuration performs better in almost all of the proposed scenarios, except for when we apply a 30×30 kernel, in which the **mixed** configuration performs better for all of the train sets. This observation points to the impression that the usage of depthwise separable convolutions yields higher accuracy for intense blurring. However, this supposition needs further investigation and more tests on more challenging scenarios. There is also little difference between the results from 15×15 and 30×30 kernels, indicating that both configurations maintain consistent predictions until the image is severely blurred. The accuracy gain for the first two kernel sizes evaluated on **mixed** when the train set is changed is not meaningful. A possible explanation for this small accuracy gain is that since **mixed** has a smaller quantity of trainable parameters, the network needed to select better representations, leading to better results on unseen data and better training. There are a few examples in the literature of occasions in which networks powered by depthwise separable convolutions converge faster for this same reason (CHOLLET, 2017). For these experiments, we aimed to test stressful scenarios, and surprisingly, the configurations could extract robust poses in these scenarios. This exciting result is why we also consider as future work to extend the stressful scenarios aiming to understand where the precision starts

to decline.

From the results of the occlusion experiment, we can see that there is some level of robustness to small occlusion. In Table 10, the maximum difference between the standard metric results and the results from the 0.1 experiment is 2.177 pixels for the average mean EPE, relative to the results from the **mixed** configuration trained on ***S-train***. In this train set, the **original** configuration outperforms the **mixed** configuration for a small difference in each metric. The opposite happens in ***S-train****, in which **mixed** is better, but also by a small difference. In Figure 33, we can see that both configurations are robust enough to estimate pose with some level with correctitude even when a large part of the hand is occluded.

From the noisy images, there is a massive difference between the standard average mean EPE metric and the experiments' results. Qualitatively, the results have a reasonable level of correctitude; however, in the cases in which the network fails, the error is abnormal. Again, the difference between the average median EPE metric is small, meaning that the general error is equivalent to the standard, such as noticeable in Figure 34 and Figure 35.

We have noticed from the quantitative results that, in most of the times, there is a decrease in the average mean EPE metric when training on ***S-train****. Especially for the Gaussian noise experiment, in which both the **original** and **mixed** configurations had a reduction of approximately 13 pixels in this metric, as described in Table 11. For a few cases, training on ***S-train**** yielded worse results in the metric, but all below 1 pixel of difference.

6 CONCLUSION AND FUTURE WORKS

Hand pose estimation is a powerful tool that can help users to interact naturally with computational systems. In this dissertation, we performed an exploration of the usage of depthwise separable convolutions for real-time hand pose estimation in RGB images, allowing a broader range of applicability.

Extending an approach in the current literature proposed by Zimmermann & Brox (2017), we compared the authors' implementation using dense convolutions with our implementation using depthwise separable convolutions. From this experiment, we noticed that the usage of this optimization could lead to improvements in inference time with gains up to 34.28% and of 15.15% on average when compared with the model with dense convolutions while maintaining similar accuracy scores on the metrics proposed by the literature. This speedup in inference time facilitates the use of deep learning models to be executed in real-time on systems with mid-end GPUs, allowing for natural interaction and various other technologies to be in range for a wider public through computers and smartphones with lower computational power.

To compare the robustness to challenges that difficult the task of tracking, we proposed an evaluation of different scenarios such as blur and noise by evaluating both models on an augmented testing set. In this comparison, the model powered by depthwise separable convolutions continued to maintain similar accuracy scores on the metrics proposed by the literature, with the maximum difference between the average error of both models being around $\pm$ 3 pixels, which can be lowered by training on the challenging scenario. This difference leads us to believe that the robustness of those models for tracking challenges is comparable, and tends to be even more robust in some scenarios, such as intense blurring. From the results of the proposed experiments, we could understand that the usage of depthwise separable convolutions are a good fit, even when the proposed architecture needs to be robust for challenges in tracking tasks.

For future works, we will investigate further optimizations, such as pruning, to decrease inference time for models training on the task of real-time 3D hand pose estimation. Besides that, we will also perform different validations on new scenarios that propose different tracking challenges, such as illumination change and evaluations that are non-synthetic, adding the human factor on real-life experiments. Finally, we intend to propose a robust, well-optimized tool for real-time inference powered by depthwise separable convolutions, and compatible with

popular game engines such as Unity and Unreal Engine.

REFERENCES

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

ANDRILUKA, M.; PISHCHULIN, L.; GEHLER, P.; SCHIELE, B. 2d human pose estimation: New benchmark and state of the art analysis. In: *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*. [S.l.: s.n.], 2014. p. 3686–3693.

BAEK, S.; KIM, K. I.; KIM, T.-K. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 1067–1076.

BHATT, N.; BHATT, N.; PRAJAPATI, P. Deep learning: A new perspective. *International Journal of Latest Technology in Engineering, Management & Applied Science*, v. 6, p. 136–140, 2017.

BOUKHAYMA, A.; BEM, R. d.; TORR, P. H. 3d hand shape and pose from images in the wild. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 10843–10852.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

CAI, Y.; GE, L.; CAI, J.; YUAN, J. Weakly-supervised 3d hand pose estimation from monocular rgb images. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 666–682.

CAO, Z.; HIDALGO, G.; SIMON, T.; WEI, S.-E.; SHEIKH, Y. Openpose: realtime multi-person 2d pose estimation using part affinity fields. *arXiv preprint arXiv:1812.08008*, 2018.

CHAN, R. H.; HO, C.-W.; NIKOLOVA, M. Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. *IEEE Transactions on image processing*, IEEE, v. 14, n. 10, p. 1479–1485, 2005.

CHARLES, B. 4.5 - image noise models. In: BOVIK, A. (Ed.). *Handbook of Image and Video Processing (Second Edition)*. Second edition. Burlington: Academic Press, 2005, (Communications, Networking and Multimedia). p. 397 – 409. ISBN 978-0-12-119792-6. Disponível em: <http://www.sciencedirect.com/science/article/pii/B9780121197926500875>.

CHO, S.; MATSUSHITA, Y.; LEE, S. Removing non-uniform motion blur from images. In: CITESEER. *2007 IEEE 11th International Conference on Computer Vision*. [S.l.], 2007. p. 1–8.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2017. p. 1251–1258.

COPPIN, B. *Artificial intelligence illuminated*. [S.l.]: Jones & Bartlett Learning, 2004.

EROL, A.; BEBIS, G.; NICOLESCU, M.; BOYLE, R. D.; TWOMBLY, X. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding*, v. 108, n. 1, p. 52–73, 2007. ISSN 1077-3142. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1077314206002281>.

FRANCOIS, C. *Deep learning with Python*. [S.l.]: Manning Publications Company, 2017.

GE, L.; LIANG, H.; YUAN, J.; THALMANN, D. Robust 3d hand pose estimation in single depth images: from single-view cnn to multi-view cnns. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 3593–3601.

GIRISH, G.; SAIKUMAR, B.; ROYCHOWDHURY, S.; KOTHARI, A. R.; RAJAN, J. Depthwise separable convolutional neural network model for intra-retinal cyst segmentation. In: IEEE. *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.], 2019. p. 2027–2031.

GONZALEZ, R. C.; WOODS, R. E. *Digital image processing*. 4. ed. [S.l.]: Prentice hall Upper Saddle River, 2018.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016.

HAN, S.; LIU, B.; WANG, R.; YE, Y.; TWIGG, C. D.; KIN, K. Online optical marker-based hand tracking with deep labels. *ACM Trans. Graph.*, ACM, New York, NY, USA, v. 37, n. 4, p. 166:1–166:10, jul. 2018. ISSN 0730-0301. Disponível em: <http://doi.acm.org/10.1145/3197517.3201399>.

HARVEY, I. Perspectives on artificial intelligence: Three ways to be smart. In: HARVEY, I.; CAVOUKIAN, A.; TOMKO, G.; BORRETT, D.; KWAN, H.; HATZINAKOS, D. (Ed.). *SmartData*. New York, NY: Springer New York, 2013. p. 27–38. ISBN 978-1-4614-6409-9.

HASSON, Y.; VAROL, G.; TZIONAS, D.; KALEVATYKH, I.; BLACK, M. J.; LAPTEV, I.; SCHMID, C. Learning joint reconstruction of hands and manipulated objects. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2019. p. 11807–11816.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778.

HOWARD, A. G.; ZHU, M.; CHEN, B.; KALENICHENKO, D.; WANG, W.; WEYAND, T.; ANDREETTO, M.; ADAM, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Ishiyama, H.; Kurabayashi, S. Monochrome glove: A robust real-time hand gesture recognition method by using a fabric glove with design of structured markers. In: *2016 IEEE Virtual Reality (VR)*. [S.l.: s.n.], 2016. p. 187–188.

JAIN, A.; TOMPSON, J.; ANDRILUKA, M.; TAYLOR, G. W.; BREGLER, C. Learning human pose estimation features with convolutional networks. *arXiv preprint arXiv:1312.7302*, 2013.

Jain, V.; Perla, R.; Hebbalaguppe, R. [poster] airgestar: Leveraging deep learning for complex hand gestural interaction with frugal ar devices. In: *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct)*. [S.l.: s.n.], 2017. p. 235–239. ISSN null.

JERALD, J. *The VR book: Human-centered design for virtual reality*. [S.l.]: Morgan & Claypool, 2015.

KAMAL, K.; YIN, Z.; WU, M.; WU, Z. Depthwise separable convolution architectures for plant disease classification. *Computers and Electronics in Agriculture*, Elsevier, v. 165, p. 104948, 2019.

KRIZHEVSKY, A.; HINTON, G. et al. *Learning multiple layers of features from tiny images*. [S.l.], 2009.

KUEN, J.; KONG, X.; WANG, G. Delugenets: Deep networks with massive and flexible cross-layer information inflows. *CoRR*, abs/1611.05552, 2016. Disponível em: <http://arxiv.org/abs/1611.05552>.

KULON, D.; WANG, H.; GÜLER, R. A.; BRONSTEIN, M.; ZAFEIRIOU, S. Single image 3d hand reconstruction with mesh convolutions. *arXiv preprint arXiv:1905.01326*, 2019.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436–444, 2015.

LIN, T.-Y.; MAIRE, M.; BELONGIE, S.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLLÁR, P.; ZITNICK, C. L. Microsoft coco: Common objects in context. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 740–755.

LIN, Y.; LV, F.; ZHU, S.; YANG, M.; COUR, T.; YU, K.; CAO, L.; LI, Z.; TSAI, M.; ZHOU, X. et al. Imagenet classification: fast descriptor coding and large-scale svm training. *Large scale visual recognition challenge*, 2010.

LIPTON, Z. C. *The Mythos of Model Interpretability*. 2016.

LOPER, M.; MAHMOOD, N.; ROMERO, J.; PONS-MOLL, G.; BLACK, M. J. Smpl: A skinned multi-person linear model. *ACM transactions on graphics (TOG)*, ACM, v. 34, n. 6, p. 248, 2015.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, Springer, v. 5, n. 4, p. 115–133, 1943.

MUELLER, F.; BERNARD, F.; SOTNYCHENKO, O.; MEHTA, D.; SRIDHAR, S.; CASAS, D.; THEOBALT, C. Ganerated hands for real-time 3d hand tracking from monocular rgb. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 49–59.

MURTHY, G.; JADON, R. A review of vision based hand gestures recognition. *International Journal of Information Technology and Knowledge Management*, v. 2, n. 2, p. 405–410, 2009.

NAJAFABADI, M. M.; VILLANUSTRE, F.; KHOSHGOFTAAR, T. M.; SELIYA, N.; WALD, R.; MUHAREMAGIC, E. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, SpringerOpen, v. 2, n. 1, p. 1, 2015.

NEWELL, A.; YANG, K.; DENG, J. Stacked hourglass networks for human pose estimation. In: SPRINGER. *European conference on computer vision*. [S.l.], 2016. p. 483–499.

NIELSEN, M. A. *Neural networks and deep learning*. [S.l.]: Determination press San Francisco, CA, USA:, 2015. v. 25.

OBERWEGER, M.; WOHLHART, P.; LEPETIT, V. Hands deep in deep learning for hand pose estimation. *arXiv preprint arXiv:1502.06807*, 2015.

O'HAGAN, R.; ZELINSKY, A.; ROUGEAUX, S. Visual gesture interfaces for virtual environments. *Interacting with Computers*, OUP, v. 14, n. 3, p. 231–250, 2002.

PANTELERIS, P.; ARGYROS, A. Back to rgb: 3d tracking of hands and hand-object interactions based on short-baseline stereo. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2017. p. 575–584.

PERRONNIN, F.; SáNCHEZ, J.; MENSINK, T. Improving the fisher kernel for large-scale image classification. In: . [S.l.: s.n.], 2010. v. 6314, p. 143–156.

POTMESIL, M.; CHAKRAVARTY, I. Modeling motion blur in computer-generated images. In: *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: Association for Computing Machinery, 1983. (SIGGRAPH '83), p. 389–399. ISBN 0897911091. Disponível em: <https://doi.org/10.1145/800059.801169>.

QI, K.; YANG, H.; LI, C.; LIU, Z.; WANG, M.; LIU, Q.; WANG, S. X-net: Brain stroke lesion segmentation based on depthwise separable convolution and long-range dependencies. In: SPRINGER. *International Conference on Medical Image Computing and Computer-Assisted Intervention*. [S.l.], 2019. p. 247–255.

RAMAKRISHNA, V.; MUNOZ, D.; HEBERT, M.; BAGNELL, J. A.; SHEIKH, Y. Pose machines: Articulated pose estimation via inference machines. In: SPRINGER. *European Conference on Computer Vision*. [S.l.], 2014. p. 33–47.

ROGEZ, G.; KHADEMI, M.; Supančič III, J. S.; MONTIEL, J. M. M.; RAMANAN, D. 3D Hand Pose Detection in Egocentric RGB-D Images. In: AGAPITO, L.; BRONSTEIN, M. M.; ROTHER, C. (Ed.). *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, 2015. p. 356–371. ISBN 978-3-319-16178-5.

ROMERO, J.; TZIONAS, D.; BLACK, M. J. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics (TOG)*, ACM, v. 36, n. 6, p. 245, 2017.

ROSENBLATT, F. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Spartan Books, 1962. (Report (Cornell Aeronautical Laboratory)). Disponível em: <https://books.google.ca/books?id=7FhRAAAAMAAJ>.

RUSSAKOVSKY, O.; DENG, J.; SU, H.; KRAUSE, J.; SATHEESH, S.; MA, S.; HUANG, Z.; KARPATHY, A.; KHOSLA, A.; BERNSTEIN, M.; BERG, A. C.; FEI-FEI, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, v. 115, n. 3, p. 211–252, 2015.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SAATMANN, P.; JOKINEN, K. Experiments with hand-tracking algorithm in video conversations. In: LINKÖPING UNIVERSITY ELECTRONIC PRESS. *Proceedings of the 2nd European and the 5th Nordic Symposium on Multimodal Communication, August 6-8, 2014, Tartu, Estonia*. [S.l.], 2015. p. 81–86.

SATO, Y.; SAITO, M.; KOIKE, H. Real-time input of 3d pose and gestures of a user's hand and its applications for hci. In: IEEE. *Proceedings IEEE Virtual Reality 2001*. [S.l.], 2001. p. 79–86.

SEGEN, J.; KUMAR, S. Shadow gestures: 3d hand pose estimation using a single camera. In: IEEE. *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*. [S.l.], 1999. v. 1, p. 479–485.

SELVARAJU, R. R.; COGSWELL, M.; DAS, A.; VEDANTAM, R.; PARIKH, D.; BATRA, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In: *The IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017.

SHARP, T.; KESKIN, C.; ROBERTSON, D.; TAYLOR, J.; SHOTTON, J.; KIM, D.; RHEMANN, C.; LEICHTER, I.; VINNIKOV, A.; WEI, Y. et al. Accurate, robust, and flexible real-time hand tracking. In: ACM. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. [S.l.], 2015. p. 3633–3642.

SIFRE, L.; MALLAT, S. Rotation, scaling and deformation invariant scattering for texture discrimination. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2013. p. 1233–1240.

SIFRE, L.; MALLAT, S. Rigid-motion scattering for image classification. *Ph. D. dissertation*, Citeseer, 2014.

SOREL, M.; FLUSSER, J. Space-variant restoration of images degraded by camera motion blur. *IEEE Transactions on Image Processing*, IEEE, v. 17, n. 2, p. 105–116, 2008.

SPRINGENBERG, J. T.; DOSOVITSKIY, A.; BROX, T.; RIEDMILLER, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

SPRUYT, V.; LEDDA, A.; PHILIPS, W. Robust arm and hand tracking by unsupervised context learning. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 14, n. 7, p. 12023–12058, 2014.

SPURR, A.; SONG, J.; PARK, S.; HILLIGES, O. Cross-modal deep variational hand pose estimation. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 89–98.

STOCKMAN, G.; SHAPIRO, L. G. *Computer Vision*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN 0130307963.

SUN, J.; CAO, W.; XU, Z.; PONCE, J. Learning a convolutional neural network for non-uniform motion blur removal. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015.

SUN, K.; XIAO, B.; LIU, D.; WANG, J. Deep high-resolution representation learning for human pose estimation. *arXiv preprint arXiv:1902.09212*, 2019.

SUN, X.; WEI, Y.; LIANG, S.; TANG, X.; SUN, J. Cascaded hand pose regression. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2015. p. 824–832.

TANG, D.; CHANG, H. J.; TEJANI, A.; KIM, T.-K. Latent regression forest: Structured estimation of 3d articulated hand posture. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 3786–3793.

TOH, K. K. V.; ISA, N. A. M. Noise adaptive fuzzy switching median filter for salt-and-pepper noise reduction. *IEEE signal processing letters*, IEEE, v. 17, n. 3, p. 281–284, 2009.

TOMPSON, J.; GOROSHIN, R.; JAIN, A.; LECUN, Y.; BREGLER, C. Efficient object localization using convolutional networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2015. p. 648–656.

TOMPSON, J.; STEIN, M.; LECUN, Y.; PERLIN, K. Real-time continuous pose recovery of human hands using convolutional networks. *ACM Transactions on Graphics (ToG)*, ACM, v. 33, n. 5, p. 169, 2014.

TOSHEV, A.; SZEGEDY, C. Deeppose: Human pose estimation via deep neural networks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2014. p. 1653–1660.

VALLI, A. Notes on natural interaction. v. 5, n. 2012, p. 80, 2005.

VALLI, A. The design of natural interaction. *Multimedia Tools and Applications*, v. 38, n. 3, p. 295–305, Jul 2008. ISSN 1573-7721. Disponível em: <https://doi.org/10.1007/s11042-007-0190-z>.

WALT, S. van der; SCHÖNBERGER, J. L.; Nunez-Iglesias, J.; BOULOGNE, F.; WARNER, J. D.; YAGER, N.; GOUILLART, E.; YU, T.; CONTRIBUTORS the scikit-image. scikit-image: image processing in Python. *PeerJ*, v. 2, p. e453, 6 2014. ISSN 2167-8359. Disponível em: <https://doi.org/10.7717/peerj.453>.

WANG, R. Y.; POPOVIĆ, J. Real-time hand-tracking with a color glove. *ACM transactions on graphics (TOG)*, ACM, v. 28, n. 3, p. 63, 2009.

WEI, S.-E.; RAMAKRISHNA, V.; KANADE, T.; SHEIKH, Y. Convolutional pose machines. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 4724–4732.

WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. [S.l.: s.n.], 2014. p. 1–10.

WU, J. Introduction to convolutional neural networks. 2017.

XIAO, B.; WU, H.; WEI, Y. Simple baselines for human pose estimation and tracking. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. [S.l.: s.n.], 2018. p. 466–481.

YITZHAKY, Y.; KOPEIKA, N. Identification of blur parameters from motion blurred images. *Graphical Models and Image Processing*, v. 59, n. 5, p. 310 – 320, 1997. ISSN 1077-3169. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1077316997904354>.

YOO, B.; CHOI, Y.; CHOI, H. Fast depthwise separable convolution for embedded systems. In: SPRINGER. *International Conference on Neural Information Processing*. [S.l.], 2018. p. 656–665.

ZEILER, M. D.; FERGUS, R. Visualizing and understanding convolutional networks. In: SPRINGER. *European conference on computer vision*. [S.l.], 2014. p. 818–833.

ZHANG, J.; JIAO, J.; CHEN, M.; QU, L.; XU, X.; YANG, Q. 3d hand pose tracking and estimation using stereo matching. *CoRR*, abs/1610.07214, 2016. Disponível em: <http://arxiv.org/abs/1610.07214>.

ZHANG, X.; LI, Q.; MO, H.; ZHANG, W.; ZHENG, W. End-to-end hand mesh recovery from a monocular rgb image. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.: s.n.], 2019. p. 2354–2364.

ZHOU, S.; BAI, L.; YANG, Y.; WANG, H.; FU, K. A depthwise separable network for action recognition. *DEStech Transactions on Computer Science and Engineering*, n. cisnrc, 2019.

ZHOU, Y.-T.; CHELLAPPA, R. Computation of optical flow using a neural network. In: *IEEE International Conference on Neural Networks*. [S.l.: s.n.], 1988. v. 1998, p. 71–78.

ZHU, Y.; BAI, L.; PENG, W.; ZHANG, X.; LUO, X. Depthwise separable convolution feature learning for ihomogeneous rock image classification. In: SPRINGER. *International Conference on Cognitive Systems and Signal Processing*. [S.l.], 2018. p. 165–176.

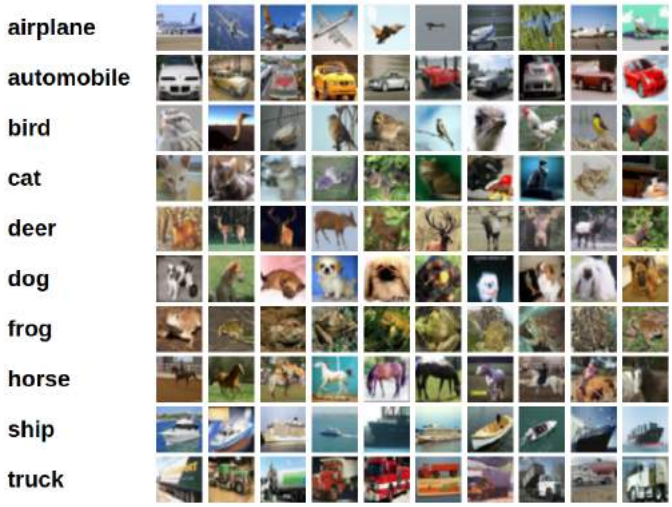
ZIMMERMANN, C.; BROX, T. Learning to Estimate 3D Hand Pose from Single RGB Images. *Proceedings of the IEEE International Conference on Computer Vision*, v. 2017-October, p. 4913–4921, 2017. ISSN 15505499.

APPENDIX A – EARLY EXPERIMENTS

In this appendix, we discuss early experiments on depthwise separable convolutions that led us to the interest of analyzing such an approach for the context of 3D hand tracking in real-time through RGB images. We have conducted these experiments using the same environment as exposed in section 4.2.

A.1 EXPERIMENTS ON CIFAR-10

Figure 36 – The CIFAR-10 dataset.



Source: KRIZHEVSKY; HINTON et al. (2009)

The CIFAR-10[1] dataset (KRIZHEVSKY; HINTON et al., 2009), named after the Canadian Institute for Advanced Research (which funded the project) is a small dataset of tiny images labeled after 10 classes: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* and *truck*, which class containing 6000 $32x32$ images. We chose to experiment on this dataset to evaluate how depthwise separable convolutions perform on small images and on a more generic task such as object recognition.

The dataset was created from a pre-labeled subset, and the annotation process was designed to filter out mislabeled images. The labelers received a class and were asked to examine all the images that were present in that class. The criteria for deciding if an image belongs to a class were as follows:

[1] <https://www.cs.toronto.edu/~kriz/cifar.html>

- The answer to the question "what is in this picture?" should be a class name on the list of likely answers;
- The image should be photo-realistic;
- There should be only one prominent instance of the object to which the class refers since there is no bounding-box annotation;
- The object may be partially occluded or seen from an unusual viewpoint, as long as its identity is still clear.

The classes are entirely mutually exclusive, meaning that there is no overlap between *automobiles* and *trucks*, for example. In this experiment, we evaluate how the usage of depthwise separable convolutions can speed-up the inference of a simple model trained to an object recognition dataset.

We define the model using simple convolutional blocks and fully-connected layers, a typical architecture for such problems. We trained the model described in Table 13 for 10 epochs on the CIFAR-10 (KRIZHEVSKY; HINTON et al., 2009) dataset, using a batch size of 64 and a learning rate of 0.001 using the $Adam$ optimizer. A second model with the same architecture is created, except for the convolutional layers, which are replaced by depthwise separable convolutions. The model receives the same configuration and parameters from the previous model. The architecture of both models can be seen and compared in Table 13.

A.1.1 Results

We have evaluated both models using top-5 accuracy on the testing set, in which we accept the prediction as correct if the correct label is one of the top 5 predictions of the system. The models achieved similar results, close to 72% of accuracy. However, we could notice a 63.3% decrease in the average inference time, decreasing from 24.13 ms (or 414 fps) to 14.7 ms (or 676 fps) when performing on the same set.

A.1.2 Discussion

For this specific experiment, the difference between the inference time of both models is higher than the difference between the models proposed in chapter 4. A possible explanation for

Table 13 – Summary of the original architecture used (a) and the proposed changes (b). We highlight the layers that were changed in this experiment.

	Original (a)		Separable (b)	
#	Layer type	Output shape	Layer type	Output shape
1	Convolution	(?, 32, 32, 64)	Separable convolution	(?, 32, 32, 64)
2	ReLU	(?, 32, 32, 64)	ReLU	(?, 32, 32, 64)
3	MaxPooling2D	(?, 16, 16, 64)	MaxPooling2D	(?, 16, 16, 64)
4	BatchNormalization	(?, 16, 16, 64)	BatchNormalization	(?, 16, 16, 64)
5	Convolution	(?, 16, 16, 128)	Separable convolution	(?, 16, 16, 128)
6	ReLU	(?, 16, 16, 128)	ReLU	(?, 16, 16, 128)
7	MaxPooling2D	(?, 8, 8, 128)	MaxPooling2D	(?, 8, 8, 128)
8	BatchNormalization	(?, 8, 8, 128)	BatchNormalization	(?, 8, 8, 128)
9	Convolution	(?, 8, 8, 256)	Separable convolution	(?, 8, 8, 256)
10	ReLU	(?, 8, 8, 256)	ReLU	(?, 8, 8, 256)
11	MaxPooling2D	(?, 4, 4, 256)	MaxPooling2D	(?, 4, 4, 256)
12	BatchNormalization	(?, 4, 4, 256)	BatchNormalization	(?, 4, 4, 256)
13	Convolution	(?, 4, 4, 512)	Separable convolution	(?, 4, 4, 512)
14	ReLU	(?, 4, 4, 512)	ReLU	(?, 4, 4, 512)
15	MaxPooling2D	(?, 2, 2, 512)	MaxPooling2D	(?, 2, 2, 512)
16	BatchNormalization	(?, 2, 2, 512)	BatchNormalization	(?, 2, 2, 512)
17	Flatten (reshape)	(?, 2048)	Flatten (reshape)	(?, 2048)
18	Dense	(?, 128)	Dense	(?, 128)
19	Dropout	(?, 128)	Dropout	(?, 128)
20	BatchNormalization	(?, 128)	BatchNormalization	(?, 128)
21	Dense	(?, 256)	Dense	(?, 256)
22	Dropout	(?, 256)	Dropout	(?, 256)
23	BatchNormalization	(?, 256)	BatchNormalization	(?, 256)
24	Dense	(?, 512)	Dense	(?, 512)
25	Dropout	(?, 512)	Dropout	(?, 512)
26	BatchNormalization	(?, 512)	BatchNormalization	(?, 512)
27	Dense	(?, 1024)	Dense	(?, 1024)
28	Dropout	(?, 1024)	Dropout	(?, 1024)
29	BatchNormalization	(?, 1024)	BatchNormalization	(?, 1024)
30	Dense	(?, 10)	Dense	(?, 10)

this behavior is that, for this proposed experiment, the model containing depthwise separable convolutions does not contain dense convolutions, which we could not remove in the previous experiments since it would make inviable to load the previously trained weights from the

Convolutional Pose Machines (WEI et al., 2016) model. However, the similarity in the accuracy metric leads us to believe that the usage of depthwise separable convolutions are a good fit for tasks such as object recognition, and it is what motivated us to explore the applicability of such approaches for a more specific task such as hand pose estimation.