PEDESTRIAN DETECTION AND TRACKING FOR SAFE ROBOT NAVIGATION IN URBAN ENVIRONMENTS

by

David Anthony Parham

A joint thesis

in cooperation between Capra Robotics ApS and the Faculty of the Technical University of Denmark in partial fulfillment of the requirements for the academic degree of

Master of Science

Danmarks Tekniske Universitet

Department of Electrical Engineering
Kongens Lyngby, Greater Copenhagen Region
September 21, 2022

THE INVOLVEMENT STRUCTURE OF THE FOLLOWING WORK

Dr. Lazaros Nalpantidis, First Examiner
Department of Electrical Engineering

M.Sc. Jonathan Binner Becktor, Second Examiner
Department of Electrical Engineering

Mads Bendt, First Advisor
Capra Robotics ApS, Chief Innovation Officer

Dr. Rui Pimentel de Figueiredo, Second Advisor
Capra Robotics ApS, AI Specialist

DECLARATION OF AUTHORSHIP

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such. It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Dedicated to my parents Fredrick Remond Parham and Michaela Wiebe-Parham
for their endless love, support and encouragement.

ACKNOWLEDGEMENTS

I feel I have learned a lot during my time at the Technical University of Denmark. Especially in the field of artificial intelligence, I have learned many new things, tried them out, and incorporated them to the best of my ability in this thesis. I am sure that I will always treasure and use these experiences in my private life and in my future professional career. First, I would like to express my sincere gratitude and appreciation for the collective support and guidance I received from my thesis advisors: Lazaros Nalpantidis, Jonathan Binner Becktor, Mads Bendt, and Rui Pimentel de Figueiredo. I could not have imagined better mentors and advisors for my dissertation, as each one of them was very diligent, instructive and challenging towards my ideas. Additionally, I would like to thank Capra Robotics ApS for providing me with all the needed resources and generally allowing me to work with them on such an exciting and challenging topic for my dissertation. Last but not least, I am deeply indebted to my family, the most important force in my life, for their support and strength. My parents' unconditional love, their upbringing and their encouragement to pursue my dreams have led me to follow this academic path. Without them, I would not have been able to overcome all the hardships I experienced throughout my education and ultimately in writing this thesis.

ABSTRACT

Data-driven methods for tracking people's poses and predicting their actions based on video streams have been successfully applied in numerous fields, including surveillance, service robotics, rehabilitation and healthcare physiotherapy, to name a few. However, the literature falls short of methods that attempt to apply these concepts in the context of autonomous outdoor navigation. More precisely, no one has yet attempted to combine these concepts into an end-to-end solution for real-time skeleton-based action recognition that supports the decision-making process for the task of autonomous outdoor navigation in urban areas. As such, this work aims to do just that, using the Lightweight OpenPose architecture for pose estimation and ST-GCN architecture for action classification and testing its real-time capabilities on a NVIDIA Jetson Xavier NX. In addition, the end-to-end solution will be extended with head pose estimation and proximity measurement algorithms to obtain more information about the outside world. This will provide the autonomous robots with a lot of information that will help them in the future with path planning and decision making *(Code available on GitHub)*.

CONTENTS

LIST OF FIGURES

LIST OF ACRONYMS

AI Artificial Intelligence

ANN Artificial Neural Network

CNN Convolutional Neural Network

CPU Central Processing Unit

DNN Deep Neural Network

FPGA Field Programmable Gate Arrays

FPS Frames Per Second

GCN Graph Convolutional Network

GDPR General Data Protection Regulation

GFLOP Giga Floating Point Operation

GNN Graph Neural Network

GPU Graphic Processing Unit

HPE Human Pose Estimation

LSTM Long Short-Term Memory

ML Machine Learning

NN Neural Network

PAF Part Affinity Field

PIF Part Intensity Field

RGB Red-Green-Blue

RNN Recurrent Neural Network

SBAR Skeleton Based Action Recognition

SBC Single Board Computer

SOTA State Of The Art

ST-GCN Spatial-Temporal Graph Convolutional Network

TNN Transformer Neural Network

1 INTRODUCTION

Before commencing with this thesis, it will be briefly discussed in this chapter why this topic needs to be investigated. In addition, the most important contributions and the structure of this scientific work will be presented.

1.1 Motivation

The use of autonomous robots and artificial intelligence in civic space is subject to strict regulations designed to ensure citizens' safety and privacy. Consequently, it is of the utmost importance that an autonomous robot can perceive the outside world and react appropriately to external influences. In order not to endanger citizens or violate any social norms (e.g., intrusion into the zones of interpersonal space) while performing its intended tasks. Human Pose Estimation (HPE) and Skeleton Based Action Recognition (SBAR) are two co-dependent methods that have already been widely adopted for related applications. Both methods have a long history of research that has gained renewed interest in recent years, due to advances in the development of depth, motion, and camera sensors, as well as powerful neural networks. What makes skeleton-based action recognition particularly interesting is the compact and lightweight nature of its input data and model sizes, while preventing sensitive information such as ethnicity, gender or age from being compromised, which is critical for General Data Protection Regulation (GDPR) compliance. However, no one has yet investigated the combined applicability of these approaches, within an end-to-end solution, to aid in the decision-making process for the task of autonomous outdoor navigation in urban areas, which is precisely the reason for this work.

1.2 Scientific contributions

Is a deep-learning-based end-to-end solution for autonomous outdoor navigation capable of real-time pedestrian detection, tracking, and action recognition on a hardware-constrained single-board computer? This is the scientific question to be answered with this work.

The proposed solution comprises two State Of The Art (SOTA) DNN-based models for pose estimation and skeletal action recognition, which combined can recognise navigation-related actions of multiple pedestrians in real-time. Supporting algo-

rithms then augmented these models for head pose estimation and measuring the distance between individuals. This generates a larger pool of information on which the capra robot can then base its action decisions on. As such, the main contribution of this work is to combine these components into an end-to-end solution and to further answer the scientific question formulated at the beginning of this section.

The main contributions in this work can therefore be summarized as follows:

- Relevant scientific literature review and selection of suitable algorithms and models for further use
- Implementation of the standalone components
- Assembling the individual components into a real-time capable (15-20 FPS) end-to-end solution
- Deployment and evaluation of the proposed solution on a suitable Single Board Computer (SBC) (e.g., NVIDIA Jetson Xavier NX)

1.2.1 Assumptions and limitations

Due to the COVID-19 pandemic, the still ongoing chip shortage as well as the resulting delivery difficulties of various manufacturers and distributors, it was not possible to deploy the end-to-end solution on an SBC and also to perform further tests in the time frame foreseen in this thesis. Therefore, assumptions are made based on experiments and other works to estimate the performance on such devices.

1.3 Systematic approach

The underlying thesis is structured as follows: The first chapter (2) discusses related work in the field of human pose estimation and skeleton-based action recognition and gives an insight into the current SOTA. The next chapter (3): 'Theoretical foundations' presents the elementary principles relevant to this thesis in order to facilitate the reader's access to the topic and the proposed solution. The main part of the thesis (Chapter 4) describes the definition of the domain-specific action space. Furthermore, all components and their functions of the proposed end-to-end solution are presented. The following chapter (5) presents the experiments conducted, and the results obtained. Chapter (6) discusses the significance and evaluation of the results. The thesis concludes in chapter (6), which provides a summary and conclusion of this thesis with regard to the objectives described in section 1.2.

1.4 Hardware specifications

The Capra Hircus robot (Fig. 1.1) is a manoeuvrable outdoor mobile robot based on Capra Robotics' patented chassis. Hircus is designed to easily handle different types of loads. Due to its flexible chassis and large motor power, it can be used for numerous tasks on uneven terrain as well as in urban areas. The robot itself is not used in the following project but is presented nevertheless, as the basic idea of the designed end-to-end solution is ultimately based on its actual integration of it onto the capra robot.

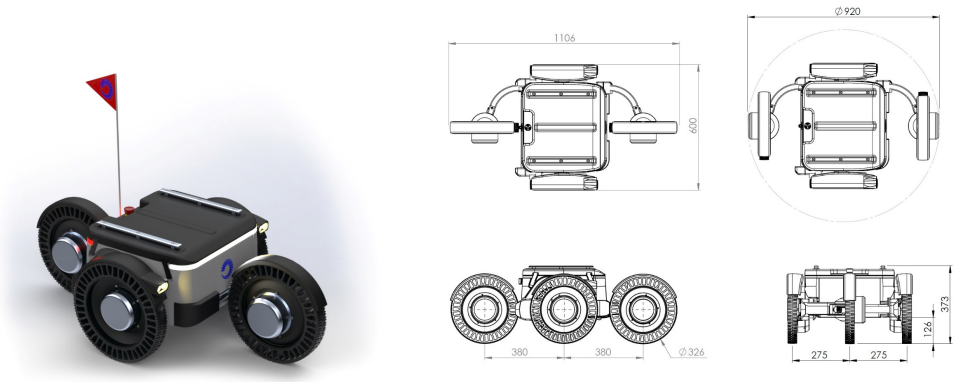


Figure 1.1: Left: Illustration of the barebone configuration of the Capra Hircus Robot, Right: Dimensions of the Capra Hircus Robot. A real-life demonstration can be found here.

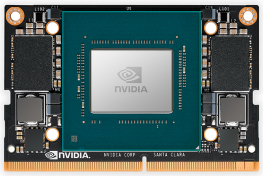
The full technical specification of the barebone configuration of the Capra Hircus Robot can be seen in Figure 1.2. The task addressed in this project, however, depends on a video stream and therefore requires two components to be added to the Hircus robot. Namely, a Red-Green-Blue (RGB) capabale mono camera and a powerful Single Board Computer, such as the NVIDIA Jetson Xavier NX. Which camera is used is not of great importance, which is why only the specifications of the NVIDIA Jetson Xavier NX are presented below.

1.4.1 Considerations

The hardware constraints imposed by the NVIDIA Jetson Xavier NX require certain factors to be considered in the decision-making process for the proposed solution. Such as model size. Because DNN often require the storage and access of numerous parameters for describing the model architectures. The size of these architectures therefore plays an essential role on the inference time (FPS). Therefore, it is advantageous to choose efficient architectures with a reduced number of parameters to achieve the best possible performance.

Dimensions and Weight	
External Dimensions (LxWxH)	1106x600x373 mm
Wheel Diameter	326 mm
Clearance Height	126 mm
Weight with 2 batteries	50 kg (with 2 batteries)
Speed and Performance	
Maximal Speed	6 km/h
Maximal Incline	30%
Curb climbing	140 mm
Turning Radius	465 mm
Chassis	4-wheel robot frame
Motor Type	4 pcs. hub motors of 250W
Payload Capacity (sum max 100 kg)	50 kg
Effect	
Battery Type	Li-NMC
Capacity	Up to 6 units of 25.9V, 20 Ah
Operating Time	17,5 Hours (with 6 batteries)
Range	100 km (with 6 batteries)*
Charging Time	3 Hours
Environment	
Operational Temperature Range	-20°C to +50 °C**
IP Rating	IP 65
Usage	Primarily Outdoors
Interface and Communication	
External Communication	Redundant 4G router with Dual-band WiFi and dual SIM
I/O	M12 Ethernet connector M12 Battery power (10A E-stop, 10A Continues)
Manual Steering	Remote Control and pull rod
Safety	
Collision Avoidance	2 Pcs. Ultrasound Sensors
Safety	4 Safety functions according to ISO 13849-1 - Emergency stop - Object detection - Brake validation - Detection of manually triggered brake

Figure 1.2: Technical Specifications of the Capra Hircus robot.



TECHNICAL SPECIFICATIONS

AI Performance	21 TOPS
GPU	384-core NVIDIA Volta™ GPU with 48 Tensor Cores
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3
Memory	8 GB 128-bit LPDDR4x 51.2GB/s
Power	10W|15W
Storage	16 GB eMMC 5.1
PCIe	1 x1 + 1 x4 (PCIe Gen3, Root Port & Endpoint)
CSI Camera	Up to 6 cameras (36 via virtual channels) 12 lanes MIPI CSI-2 D-PHY 1.2 (up to 30 Gbps)
Video Encode	2x4K4MP/sec 2x 4K @ 30 (HEVC) 6x 1080p @ 60 (HEVC)
Video Decode	2x690MP/sec 2x 4K @ 60 (HEVC) 12x 1080p @ 60 (HEVC) 32x 1080p @ 30 (HEVC)
Display	2 multi-mode DP 1.4/eDP 1.4/HDMI 2.0
DL Accelerator	2x NVDLA Engines
Networking	10/100/1000 BASE-T Ethernet
Mechanical	69.6 mm x 45 mm 260-pin SO-DIMM connector

Figure 1.3: Left: Illustration of the Nvidia Jetson Xavier NX, Right: Technical specifications.

2 RELATED WORK

Human pose estimation and skeleton-based action recognition have been intensively researched in academia for decades. As a result, many successful and efficient methods have been established over the years. HPE methods based on deep learning can generally be divided into several categories that are distinguished by their characteristics (Y. Chen et al., 2020). These methods include: generative (based on the human body), discriminative (without a body model), regression-based (direct mapping from the initial image to the position of the body joints), detection-based (generation of intermediate image patches or heat maps of the joint positions), one-stage (end-to-end training), multi-stage (stage-wise training), top-down (from high-level abstraction to low-pixel-level evidence), and bottom-up (from low-pixel-level evidence to high-level abstraction). Most papers, however, assign approaches in accordance with bottom-up and top-down divisions. As such, only the latter two methods will be discussed in more detail. In terms of skeleton-based action recognition methods, these can be divided into four categories and are also presented together with the two human pose estimation categories in the following sections.

2.1 2D-Human-Pose Estimation

- **Bottom-Up**-based methods predict all keypoints at first and then assemble them into full poses. DeepCut treats the problem of distinguishing multiple people in images as an integer linear problem and approaches it by partitioning partial recognition candidates into people clusters, followed by assigning the respective common labels to the people clusters (Pishchulin et al., 2016). Insafutdinov et al. (2016) further improved the performance of DeepCut in their work DeeperCut+ by using a deeper ResNet model and incorporating image-conditional pair-wise terms. Other works use a Part Intensity Field (PIF) to localize body parts and a Part Affinity Field (PAF) to associate body parts with each other or Convolutional Neural Network (CNN) models to detect individual keypoints and predict their relative displacements to form full human poses (Kreiss et al., 2019; Papandreou et al., 2018).
- **Top-Down**-based methods interpret the process of detecting keypoints as a two-stage pipeline, that is, run a person detector first and estimate body joints within the detected bounding boxes second. Works of Iqbal and Gall

(2016) use a person detection algorithm to select candidates in an image. These candidates are then cropped to their respective bounding box sizes and further grouped together. All these candidates are then fed into a CNN where the suppression of non-maxima and finally the joint labelling is performed. All top-down methods essentially follow this process. The apparent differences between any of these approaches lie primarily in the use of different person detectors, methods for forming or selecting the most ideal bounding boxes, and better algorithms for keypoints localization (Y. Chen et al., 2018; Fang et al., 2017).

2.2 Skeleton-based action recognition

- **Recurrent Neural Network (RNN)**-based methods process the skeletal data into sequences, which is why these signals can then be viewed as time series data with predefined traversal rules. This data is then fed into RNN-based models such as the Long Short-Term Memory (LSTM) to model the temporal information (movement of joints) across frames of the performed action (Shahroudy et al., 2016; Zaremba et al., 2014). Other research teams, such as Zhang et al. (2017), designed a view adaptation scheme for action modelling or constructed an adaptive tree-structured RNN (Wei et al., 2017), yet RNN approaches by their nature do not explicitly consider spatial dependencies between the different joints, which makes it difficult to improve these methods in such a way that their performance becomes competitive (L. Ke et al., 2022).

- **CNN**-based methods employ SOTA CNN techniques to extract the spatial and temporal features of the skeletal sequence. Each skeletal sequence is converted into a pseudo-image by reconfiguring the joint coordinates of the body into a 2D matrix in which the temporal dynamics of the sequence are encoded as changes in rows and the spatial structure of each image is encoded as columns. This data is then fed into a CNN for feature extraction and recognition (C. Cao et al., 2019; Q. Ke et al., 2017; Li et al., 2017; Liu et al., 2017; Soo Kim and Reiter, 2017; L. Wang et al., 2019). Such kinds of input, however, cannot exploit the locality nature of convolutional neural networks, which has made these kinds of approaches fade from the stage of frontier research. But thanks to the work of Soo Kim and Reiter (2017) CNN approaches have made their comeback in the form of 3D-Convolutional Neural Networks. From then on, many advanced 3D-CNN methods were proposed by the action recognition community, which outperformed I3D (Soo Kim and

Reiter, 2017) and other DNN methods. One such example is PoseC3D (Duan et al., 2021), which aggregates 2D heat maps of skeletal joints for different points in time by stacking them along the temporal dimension to form a 3D heat map, which is then fed into the 3D-CNN for action recognition.

- **Transformer Neural Network (TNN)**-based methods use self-attention mechanisms such as multi-head self-attention to enable the model to learn relationships between the individual joints of the skeletal input sequence (Kong et al., 2022; Plizzari et al., 2021; Qiu et al., 2022; Shi et al., 2020; Q. Wang et al., 2021). These Neural Networks (NNs), unlike Recurrent Neural Networks (RNNs) with LSTM, can also handle very long sequences and are not limited to hand-crafted traversal rules or graphs, making it more likely that these models can find correlations between distant body parts. However, transformers have only recently gained interest in the field of skeleton-based action recognition, which is why the current models are not only inferior in performance to current SOTA methods but also less efficient, as can be seen in the case of Shi et al. (2020) which tokenise each joint, causing the computation of self-attention to grow quadratically with the number of joint tokens.

- **Graph Convolutional Network (GCN)**-based methods have been a viable choice for skeleton action recognition since they were first introduced by the work of Yan et al. (2018). Their proposed method Spatial-Temporal Graph Convolutional Network (ST-GCN) and all derivatives essentially encode the skeleton data into a predefined spatio-temporal graph and model the joint movement patterns of an action with a Graph Convolutional Network. Other works have considered the angular properties of the involved joints (Qin et al., 2021), (symmetric) self-attention modules (Heidari and Iosifidis, 2021; L. Ke et al., 2022), or have reconfigured the GCN architecture into a dual-head GCN with two entangled branches (T. Chen et al., 2021), to better learn relevant joint pattern correlations for action recognition. Nevertheless, the computational cost of GCN-based methods can be pretty high, typically more than 15 GFLOPs for an action sample, which prompted members of the research community, such as Cheng et al. (2020) and Song et al. (2022), to develop efficient GCN models which can compete with SOTA methods.

2.3 Application Domains

The promising results of the presented architectures have led the research community and industry to test and apply their viability for real-time usage across various domains, including video surveillance (Poppe, 2010; Ziaeefard and Bergevin, 2015), service robotics (Munaro and Menegatti, 2014), healthcare (Mobini et al., 2014), augmented human assistance (Bellamy and Caleb-Solly, 2019; Pilarski et al., 2019), human-robot interaction (Aggarwal and Ryoo, 2011; Feichtenhofer et al., 2019) and autonomous driving (Xu et al., 2021), to name a few. What distinguishes the use of skeleton-based action recognition across these domains is mainly their action spaces and the insights gained from them. For example, in the case of video surveillance for threat detection, the aim is to detect and classify activities that indicate suspicious or aggressive behavior (Taha et al., 2015). While the action space in health rehabilitation involves movements that support the reconstruction of joints or movement injuries.

3 THEORETICAL FOUNDATIONS

This chapter describes some contextual information, basic terms and definitions from the field of artificial intelligence to facilitate the understanding of this thesis.

3.1 Biological Neural Networks

Many tasks that require the use of intelligence, such as pattern and object recognition, are difficult for a computer to replicate, while they are easy enough for animals and young children to perform naturally. For instance, how does a family dog distinguish between its owner and a stranger? Or how does a small child learn to differentiate between its parents' car and those of its neighbours? And how does the human brain unconsciously carry out complex pattern recognition tasks every day without the person noticing? The answer lies in biological brains. Almost all living entities have a true biological neural network connected to their nervous system. This network comprises numerous interconnected nerve cells called neurons that are responsible for coordinating and performing these tasks (Vadapalli, 2021).

Human beings possess the most impressive biological neuronal network of all living things. It consists of about 100 billion neurons, not including the neurons in the spinal cord or the pre-processing of external stimuli such as sight, hearing, smell and taste (Fridman, 2019). The cell body of a neuron is called the soma, while the inputs (dendrites) and outputs (axons) connect somas together, as seen in Figure 3.1. Each neuron receives electrochemical inputs (current pulses) from other neurons at its dendrites. When these electrical inputs are strong enough to overcome the neuron's internal resistances, they are transmitted further along the axons. Each axon may be connected to one or more dendrites at one of its 1,000 trillion intersections, called synapses. These neurons can also be activated to further propagate the process of transmitting current impulses in the network (Rosenbrock, 2018).

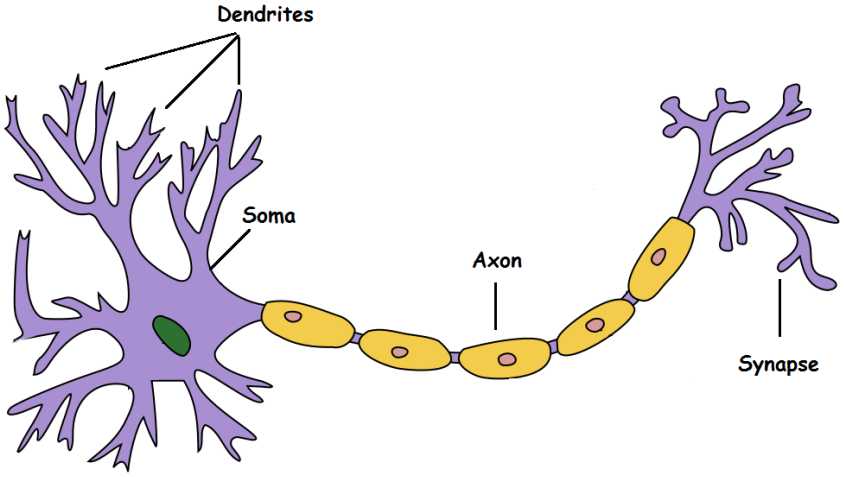


Figure 3.1: Anatomy of a biological neuron.

The key to success is that the "firing" of a neuron, which refers to the activation of that neuron, is a binary operation - the neuron either fires or does not. There are no other values. Put simply, a neuron is only triggered when the total signal received at the soma exceeds a certain threshold.

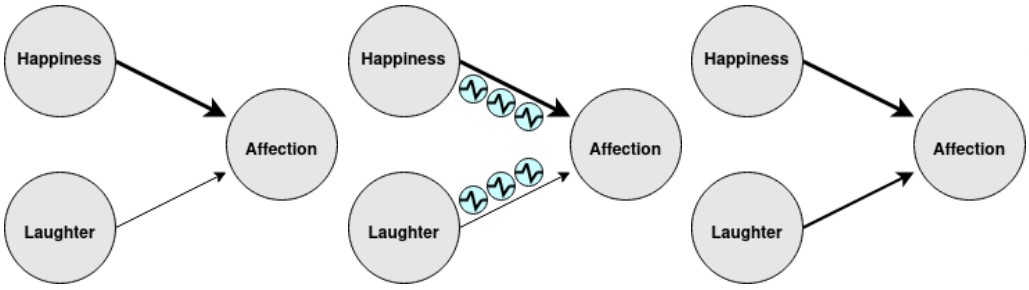
As one neuron transmits its impulses to many other neurons, the structure of a network eventually develops in which the neurons all influence each other. This means that the reaction to a stimulus (e.g. cold, warmth, hunger, comfort, etc.) depends largely on numerous different neurons all working together and influencing each other (Rupprecht, 2016). More on this in section 3.1.1.

Currently, there is insufficient knowledge about neuroscience and the deeper functions of the brain to properly model how the brain works. As such, the goal of Artificial Intelligence (AI) methods is not to mimic these functions of the human brain. Rather, to draw parallels from the parts that are understood and use them as inspiration to incorporate into one's own work (Rosenbrock, 2018). An example of this can be found in section 3.2.1, where the concepts of biological neural networks are applied to artificial intelligence.

3.1.1 Hebb's learning rule

Before the emergence of the first Artificial Neural Network (ANN), it was mainly psychologists and biologists who asked themselves how the human brain learns in detail. In 1949, the Canadian psychologist Donald O. Hebb put forward a hypothesis in "The organisation of behaviour" that later became known as Hebb's learning rule (Brown, 2020). His thesis: "cells that fire together, wire together" is based on the fact that whenever two neurons are active at the same time, the connection between them increases.

To illustrate this, consider the following example:



(a) Starting point **(b)** Firing jointly **(c)** End point

Figure 3.2: The amplification of the influence on the output neuron is represented by thicker lines. The thicker the line, the more influence the input neuron has on the output neuron.

Figure 3.2 shows three neurons, two input neurons and one output neuron. The current pulses of the input neurons have a value range from 0 to 1 and represent feelings of happiness and laughter. Initially, the output neuron responds exclusively to the happiness neuron. The input neuron laughter only minimally influences the development of affection. Applied, Hebb's learning rule states: The more often laughter and feelings of happiness are active together, the stronger the influence of laughter on the affection neuron will be. Due to the fact that the happiness neuron always causes the output neuron to fire, which in turn is connected to the laughter neuron (Rosenbrock, 2018).

3.2 Artificial Neural Networks

From the previous section 3.1, it is obvious that neural networks comprise many neurons which can solve highly complex tasks by concatenating them. This section deals with how an ANN learns the necessary knowledge to achieve this.

3.2.1 Feed-Forward

In order for a system to be classified as NN, it must follow a labelled, directed graph structure where each node in the graph performs simple computations. From graph theory, it is known that a directed graph consists of a set of nodes (i.e. vertices) and a set of links (i.e. edges) that connect nodes and edges. Figure 3.3 shows an example of such a NN graph. All nodes perform a simple calculation. Then, each connection transmits a signal (i.e. the output of the calculation) from one node to another, weighted by how much the signal is amplified or reduced. Some connections have large positive weights that intensify the signal, indicating that the signal is significant during classification. Whereas others have negative weights that reduce the strength of the signal, suggesting that the output of the node is less significant in the classification process. Finally, the previously computed result is run through an activation function that decides whether or not the neuron is triggered. Figure 3.4 illustrates the most known activation functions.

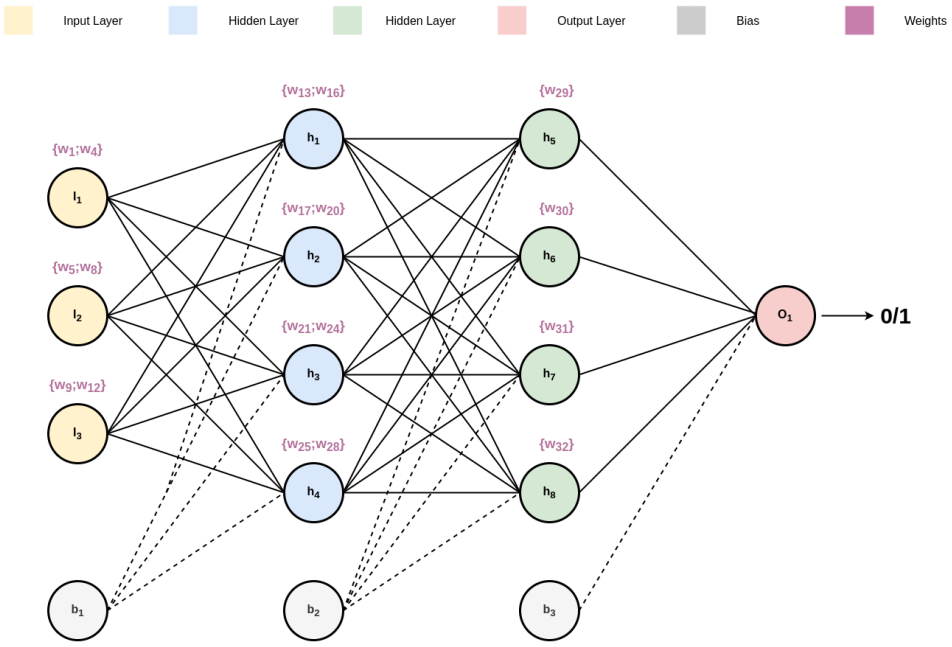


Figure 3.3: Structure of a simple neural network. Inputs are added to the network (yellow). Each neuron transmits a signal across its connections through the hidden layers (blue/green). A function calculates the output of the network (red)

Mathematically, the outputs of the neurons of the first hidden layer can be expressed by the following formula.

$$h(\theta) = w_\theta * i_1 + w_{\theta+4} * i_2 + w_{\theta+8} * i_3 * b_1, \quad where\ \theta\ \{1; 4\} \tag{3.1}$$

To calculate the remaining outputs of the neurons in this network, the formula only needs to be adjusted so that it now considers four input values instead of three.

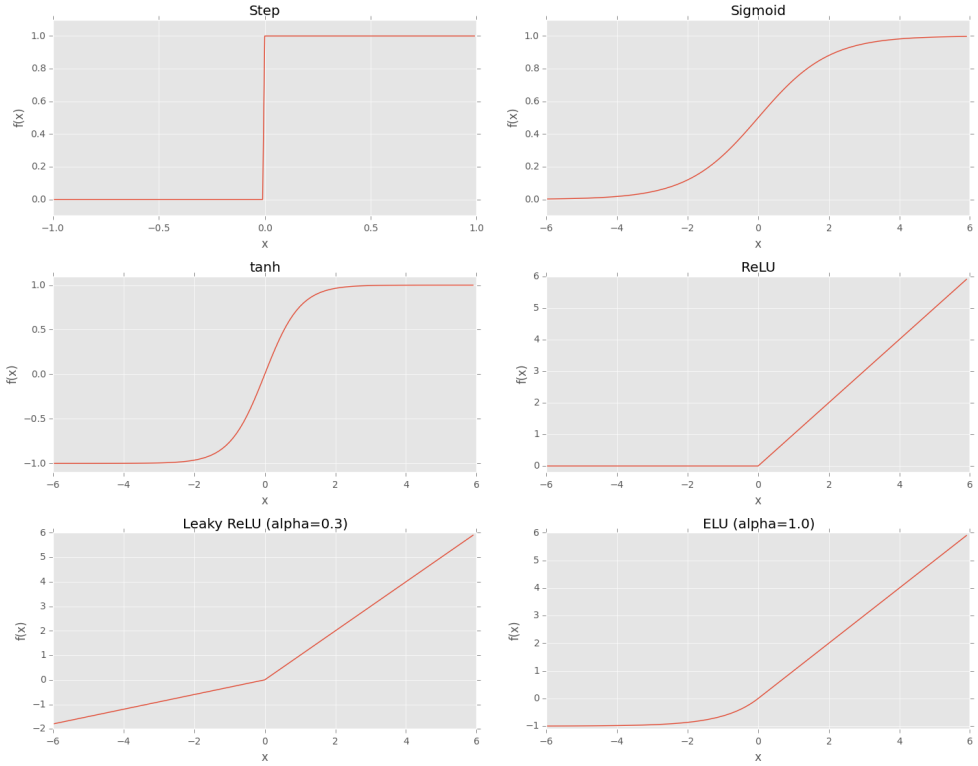


Figure 3.4: Left-top: Step function. Right-top: Sigmoid activation function. Left-centre: Hyperbolic tangent. Right-centre: Relu activation function. Left-bottom: Leaky ReLu (modified relu function that also takes negative values). Right-bottom: ELU (a variant of the ReLu function) (Rosenbrock, 2018).

3.2.2 Backpropagation

The term backpropagation, short for "backward propagation of errors", is an algorithm for supervised learning of artificial neural networks with gradient descent (More on the learning types of neural networks in sec. 3.2.3). The term was first introduced in the 1970s. Its importance, however, was not widely recognised until 1986, after a research paper by Rumelhart et al. (1986) was published. Since then, the backpropagation algorithm has become one of the most significant algorithms

in the history of neural networks. It was found that this algorithm was much more computationally efficient and thus faster than previous learning methods. This made it possible to investigate the feasibility of developing multilayer artificial neural networks, also referred to as DNN.

This algorithm was inspired from the central idea already introduced in section 3.1.1: "That whenever two neurons are active at the same time, the connections between them increase". The biological neural network of a human handles this process independently, whereas in a ANN this is taken over by the backpropagational algorithm. Here, one limits oneself to changing the bias of each neuron and the corresponding weights after each iteration. This means that there are exactly two parameters per neuron that need to be optimized in order to enable the network to learn how to correctly map arbitrary inputs to outputs - the biases and weights. In mathematical terms, this principle can be summarized in the form of the following cost function $J(\theta)$:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} LossFn(h_\theta(x^{(i)}), y^{(i)}) \tag{3.2}$$

This equation quantifies, in terms of the selected error function (i.e., sum of errors, squared errors, mean squared errors, etc.), the total difference between the predicted outcome $h_\theta(x^{(i)})$ and the actual outcome $y^{(i)}$, i.e., the total error value. It holds that y can only take discrete values $[0, 1]$, since this is a classification problem.

$$0 \geq h_\theta(x) \leq 1 \tag{3.3}$$

A crucial property of the forward pass is that it comprises several intermediate functions, where the previous output of a neuron in one layer is used as input to the neurons in the next layer. This property is important because it facilitates the backward propagation through the network using a method known as the chain rule (3.4). The chain rule determines how sensitive the parameters of a partial derivative are to changes of the opposing parameter, typically between the weight matrices W_i of a neuron (or the bias) and the total error.

$$\frac{\partial LossFn}{\partial W_{29}}(W_{29}) = \frac{\partial LossFn}{\partial O_{out}} \frac{\partial O_{out}}{\partial O_{in}} \frac{\partial O_{in}}{\partial W_{29}} \tag{3.4}$$

From differential calculus, it is known that the minimum of a linear equation can be determined by calculating its gradient. The determined gradient gives information about how steep the function is and which direction leads to a local or global minimum. Optimisation functions such as the gradient descent algorithm (3.5), which are involved in the backward pass, take advantage of this mathematical property to find the appropriate weights and or biases by which the total error rate decreases, i.e., is minimised.

$$W_i := W_i \alpha * \frac{\partial LossFn}{\partial W_i}(W_i) \tag{3.5}$$

An iteration of the backpropagation algorithm consists of three phases:

1. The forward pass, where our inputs are passed through the network. Then, based on the calculated results, output predictions are made.
2. The calculation of the total error, by adding up the individual neuron errors determined by an error function.
3. The backward pass, in which the gradient of the loss function is calculated in the last layer of the network. This is done in order to then recursively update the weights of the network so that they cause the actual output to be closer to the target output.

3.2.3 Types of learning

Supervised learning

In supervised learning, a Machine Learning (ML) algorithm is given a training data set with labelled data. In other words, this data set consists of (x, y) pairs, where x is the raw data, for example, an image matrix, and y is a description of what this data point x represents. Here, the task of the learning algorithm is to learn patterns that can automatically assign input data points to their correct class labels. Supervised learning is comparable to a teacher preparing his student (the Artificial Neural Network) for a maths exam. Given the Artificial Neural Networks' prior knowledge, it tries its best to find the correct answer to

the arithmetic problem: $4 \times (7 - 2)$. The value 26 could have been chosen as a possible answer. However, it is known that the correct solution is 20. When discussing the exam, the teacher draws the Artificial Neural Networks' attention to the wrong result and tries to bring it closer to the correct result in the next iteration by adjusting the Artificial Neural Networks' weights (Rupprecht, 2016). The training process continues until the desired stopping criterion, such as a low error rate or a minimum number of training iterations, is reached. Application examples of supervised learning problems include among others both classification and regression. Furthermore, logistic regression, random forest and support vector machines belong to the supervised learning algorithms (Rosenbrock, 2018).

Unsupervised learning

The second learning type of machine learning algorithms is unsupervised learning. Here, algorithms attempt to automatically recognise distinguishing features in the given data set without providing any clues to the input. To continue with the analogy introduced in this section, assume that a student is trying to determine distinctive features of cat breeds. The training dataset of the ANN consists of thousands of different cat photos of various breeds. After a certain runtime of unsupervised learning and independent adjustment of the weights and biases the network has learned to distinguish the cat breeds from each other. The NN, therefore, learns the characteristics of the different breeds without the teacher being there, to supervise the correctness of the results. If the ANN is then examined on the picture of a mixed-breed cat, the NN can determine to what percentage the mixed-breed cat corresponds to the previous breeds. This would also allow the network to find out which breeds were crossed in this cat based on its characteristics (Rupprecht, 2016). The main application of unsupervised learning includes analysis, clustering (=grouping) of data heaps and generative modelling. Examples of unsupervised learning algorithms are K-means, Keogh motif search, Principal Component Analysis (PCA), and Generative Adversarial Networks (Brownlee, 2019).

Semi-supervised learning

Semi-supervised learning solves classification problems where only a small proportion of data points have associated class labels. This process analyses the labelled data and attempts to use this information to label all unlabelled data points for use as additional training data. Through this process, the algorithm tries to learn the structure of the data to achieve "high" classification performance. Thus, this method aims to combine the strengths of the two learning algorithms described

above, in a hybrid form. Possible applications for semi-supervised learning are the dissemination of class labels, ladder networks and "co-learning / co-training" (Brownlee, 2019; Rupprecht, 2016).

Reinforcement learning

Reinforcement learning refers to a learning type in which an agent is trained to achieve its learning goal based on a series of decisions made in a given environment. The ANN learns by adjusting its connections based on rules that reward or punish the network according to its underlying knowledge of what is believed to be right or wrong. If the agent acts as intended, a reward will be given so that the network tweaks its weights so that this behaviour becomes "more likely". However, if the agent acts in an undesirable way that interferes with the learning goal, a punishment is given to reduce the chance of repeating that behaviour and increase the chance of all other behaviours. This variant of learning is suitable when the user does not know exactly what the outcome should be, but can at least confirm to the neural network that something worked well or not. An example of this would be an ANN that is supposed to play chess. In chess, it is difficult to measure the correctness of a move. This is because a single move opens up 35 different game situations; planning two or three moves in advance already allows 1225 (35^2) and 42875 (35^3) different game situations ("About predicting the future", n.d.). It is therefore not possible to present the neural network with a perfect result for every situation and thus teach the neural network how far it was from the optimal move. However, it is possible to judge the totality of all moves at the end of the game. If the neural network lost, then the moves it played were not that good. If it won, on the other hand, then the moves played were good. Real-world applications of reinforcement learning include natural language processing, robot motion control, and recommender systems. Common algorithms used include Markov Decision Process or Q Learning (Johnson, 2022).

3.3 Convolutional Neural Networks

With the advent of General Purpose Graphic Processing Units (GPUs), it has become feasible to train CNNs efficiently in a supervised manner. Therefore, they are considered a current state-of-the-art method for applications such as classification, object recognition, or language understanding (Rupprecht, 2016).

Almost all neural network architectures in the past followed a design with fully connected layers, where each input neuron is connected to each output neuron (Rupprecht, 2016). CNNs, on the other hand, operate mostly within small groups

of neurons (4s, 9s, etc.), which in turn are connected to a neuron in the next layer. Fully connected layers are used in CNNs only for the last layer in the network. CNNs can therefore be defined as a neural network architecture that uses a special convolutional layer instead of a fully connected layer for at least one layer in the network.

Convolutions are among the fundamental building blocks in the fields of machine vision and image processing (Rosenbrock, 2018). Their representation is limited to a matrix (filter mask), which is pushed pixel by pixel over another matrix (image). Because of this iterative process, new pixel values are calculated from the sum of the pair-wise multiplication of these two matrices (Rupprecht, 2016). In summary, a convolution can be defined as the application of a filter.

A possible example could look like this:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

(a) Mean filter mask of size 3 x 3

1	2	4	8
4	3	2	1
2	4	8	1
1	2	3	4

(b) Grey image of size 4 x 4

Figure 3.5: Depiction of a mean filter mask and a grey scale image (Rupprecht, 2016).

Figure 3.6 shows the step-by-step application of the (mean) filter mask, also called the convolution kernel, to the grey scale image shown in Figure 3.5.

The resulting image is automatically reduced by one pixel on all sides. This is because of the 3 x 3 filter mask. It is impossible to apply this mask completely at the 0,0 (pixel) coordinates, because it would extend beyond the image. However, this can be avoided by adding zero padding. This essentially extends the image with zero values by the number of pixels to which the image would be reduced to. At the same time, to avoid exceptions, all pixels that cannot use the entire mask are not recalculated. So only the pixels that can use the full filter mask and do not extend beyond the image are computed (Rupprecht, 2016). And even though there are several filter masks (sharpen, blur, edge detection, etc.), they are all applied to the images in the same way (Rosenbrock, 2018).

Each layer in a CNN applies a different set of these filters. Typically, the results from combinations of hundreds or thousands of filters are fed as output to the next

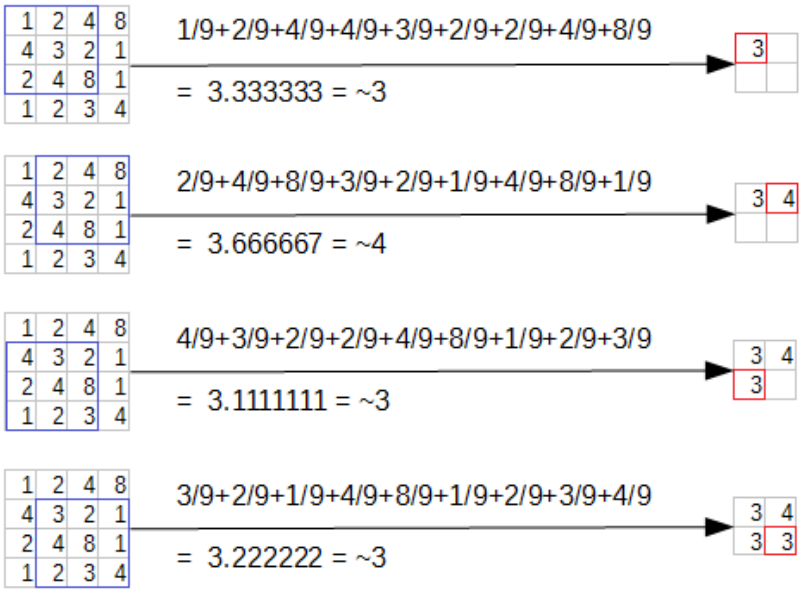


Figure 3.6: Step-by-step process of a convolution on a grey scale image (Rupprecht, 2016).

layer in the network (Rosenbrock, 2018).

In practice, CNNs provide two key benefits: local invariance and composition. The concept of local invariance allows an object to be classified within an image regardless of where that object is in the image. This local invariance is achieved by default through the use of "pooling layers", whose main task is to progressively reduce the spatial size (i.e. width and height) (Rosenbrock, 2018).

Alternatively, it is possible to achieve local invariance without pooling layers by using "strided convolutions". However, this is not recommended, as it can lead to unwanted image artifacts (Rosenbrock, 2018).

The second advantage is composition. Each filter converts a local patch of lower-level features into a higher level representation, much like a series of mathematical functions that build on the output of previous functions: $f(g(h(x)))$ - this composition allows the network to learn deeper features from the network. For example, the CNN can extract edges from multiple pixels, shapes from edges, and then highlight abstract objects from the shapes (see Fig. 3.7) - and this all happens naturally and automatically during the training process. The concept of the emergence of higher-level features from lower-level ones is why CNNs is such a powerful tool in the field of machine vision.

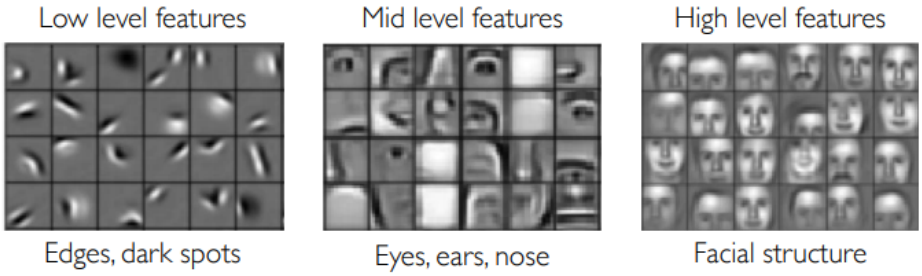


Figure 3.7: Elevation of the abstraction level through filter stacking (Rupprecht, 2016).

3.4 2D Human Pose Estimation

The estimation and tracking of 2D human poses is a computer vision task that involves the recognition, association, and tracking of the spatial location of semantic keypoints in images or video sequences. Examples of semantic keypoints in the context of human pose estimation would be body parts or joints such as "right shoulders", "left knees" or the "neck". Figure 3.8 illustrates this by portraying a skeleton with a full set of keypoints. It is also imperative to point out that pose motions are often driven by some specific human actions. Knowledge of a person's posture is therefore crucial for action recognition, which is discussed in more detail in 4.2.

The performance of semantic keypoint tracking in live video sequences requires high computational resources, which limited the accuracy of pose estimation in the past. With recent advances, new applications with real-time requirements such as self-driving cars and last-mile delivery robots are becoming feasible. However, the biggest challenge to date is the handling of motion blur, video defocus, pose occlusions, and the inability to capture temporal dependency among video frames for multi-person pose estimation.

One of the prevailing cornerstones for solutions are CNNs. As already pointed out in Section 3.3 those NNs are currently the base for the most powerful image processing models. Which is why current state-of-the-art methods for estimating human posture are usually based on the development of a CNN architecture specifically tailored for that use case.

Keypoint		
ID	**Name**	**Abbrev.**
0	nose	nose
1	neck	neck
2	right shoulder	r_sho
3	right elbow	r_elb
4	right wrist	r_wri
5	left shoulder	l_sho
6	left elbow	l_elb
7	left wrist	l_wri
8	right hip	r_hip
9	right knee	r_knee
10	right angkle	r_ank
11	left hip	l_hip
12	left knee	l_knee
13	left ankle	l_ankel
14	right eye	r_eye
15	left eye	l_eye
16	right ear	r_ear
17	left ear	l_ear

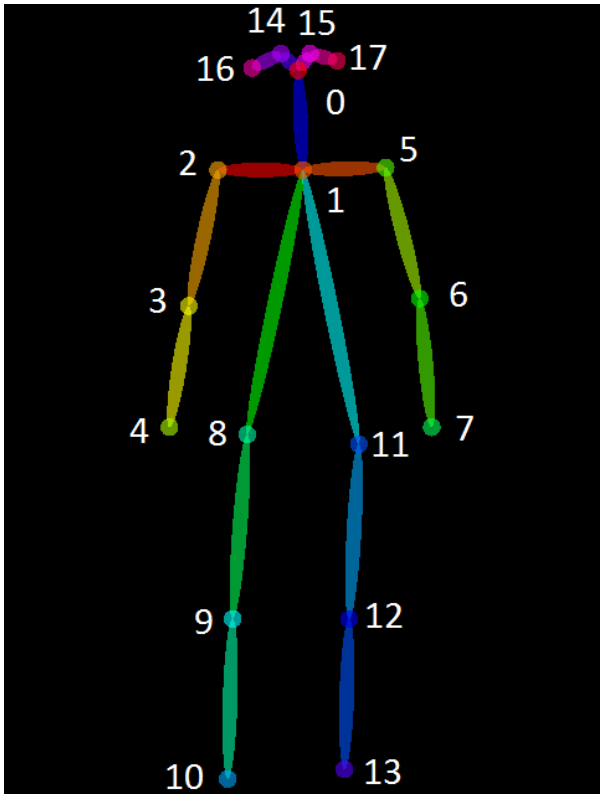


Figure 3.8: Keypoints detected by OpenPose (Z. Cao et al., 2021) on the Coco Dataset (Lin et al., 2014).

3.5 Graph Neural Networks

NNs have traditionally been used for processing fixed-size and/or regular-structure inputs (such as sentences, images and videos) that can be easily represented in tensors. This makes them less ideal for processing graph-structured data that is not as easily represented, as seen in Figure 3.9.

However, graphs are a powerful and general form of representation into which virtually any other form of data can be converted. Even images can be modelled as graphs. And although counterintuitive, one can learn more about the symmetries and structure of images by viewing them as graphs and develop an intuition that helps in understanding other, less grid-like graph data. As such, section 3.5.1 demonstrates the standard and graph-structured representation of images.

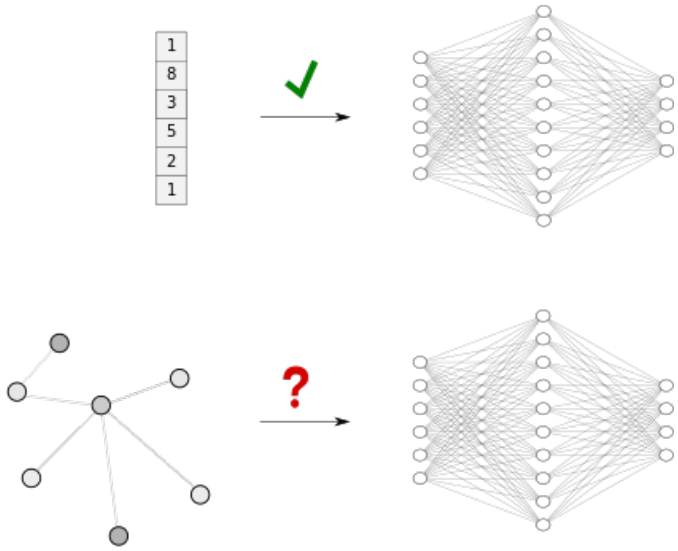


Figure 3.9: Data representation conflict for graph-structured data through standardised handling of matrix and tensor inputs of NNs (Daigavane et al., 2021).

3.5.1 Images as graphs

Traditionally, an image can be conceived as a rectangular grid of image channels represented as arrays (e.g. 244 x 244 x 3 floats), as shown in Figure 3.10.

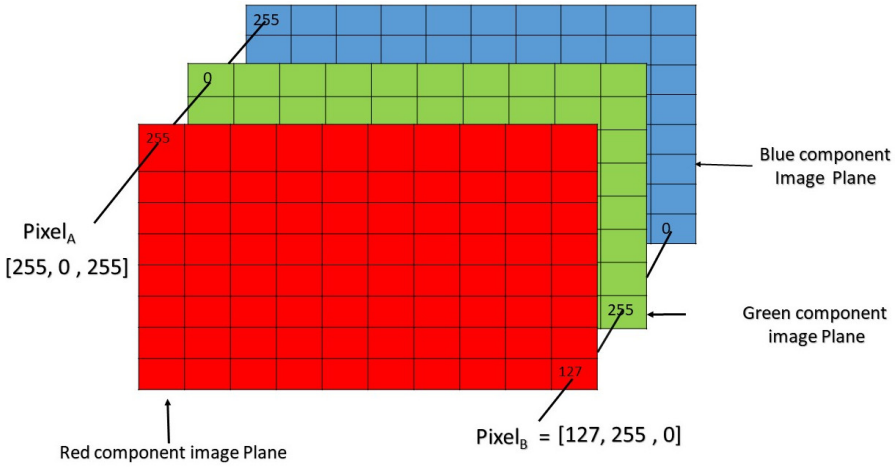


Figure 3.10: Pixels of an RGB image are formed from the corresponding pixel of the three component images.

However, as mentioned above, images can also be represented as graphs with a regular structure, where each pixel represents a node and is connected to the neighbouring pixels by an edge. Each pixel without an edge has 8 neighbours, and the information stored at each node is a three-dimensional vector representing the

RGB values of the pixel. This notion allows images to be represented in different ways, see 3.11.

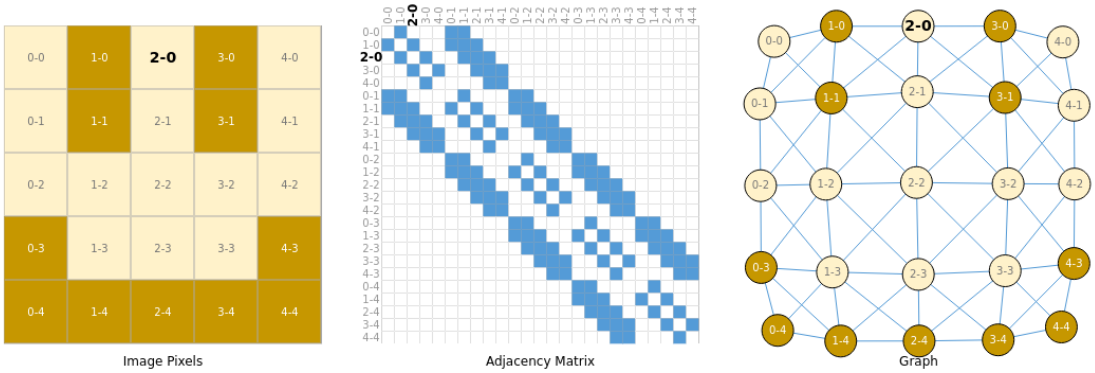


Figure 3.11: Different representation of the same 5x5 image.

The adjacency matrix can visualise the connectivity of the graph. For this, the nodes, in this case all 25 pixels of the image, are sorted. They are then filled with the values of the nodes that are connected by an edge.

3.5.2 Graph Convolutional Neural Networks

But as already mentioned in the beginning of the section, standard NNs aren't suitable for handling graph-structured data. That's where Graph Neural Networks (GNNs) come into play. GNNs are a family of neural networks that can operate naturally on graph-structured data. And especially GCN as part of that family have shown great results for the task of skeleton-based action recognition.

3.5.3 Extending Convolutions to Graphs

The concept of convolution has already been introduced in section 3.3. However, due to the distinct way in which images are treated when represented as graphs, modifications are required to perform convolutions. In particular, because ordinary convolutions are not node order invariant, as they depend on the absolute positions of the pixels. For this reason, GNN models and their derivatives must employ methods to learn a mapping from individual nodes in the graph to fixed-size vectors with real values (also called representations or embeddings). This is achieved by using polynomial filters, which serve as a substitute for traditional kernel filters.

To illustrate the operation of the polynomial filters, an arbitrary order of the n nodes of a given graph G is specified. Denoting the 0 - 1 adjacency matrix of G by A, allows one to construct the diagonal degree matrix D of G as follows:

$$D_v = \sum_u A_{uv} \tag{3.6}$$

where A_{uv} corresponds to the entry in the matrix A determined by row position v and the column position u.

Then, the graph Laplacian L is the square n x n matrix defined as:

$$L = D - A \tag{3.7}$$

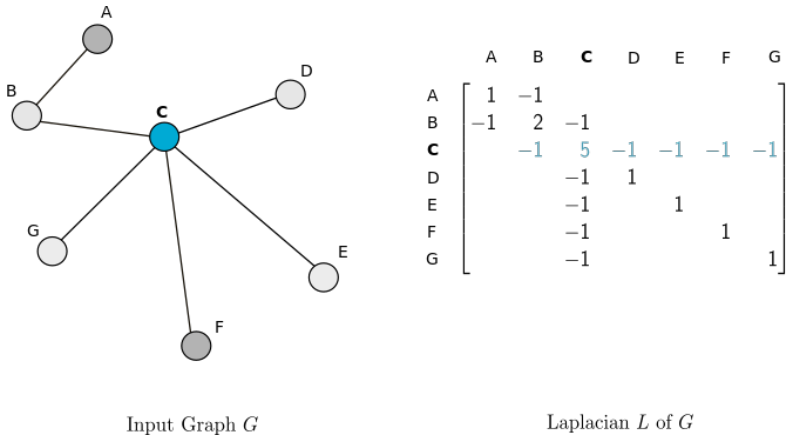


Figure 3.12: The Laplacian L for an undirected graph G with the row corresponding to node C highlighted (Daigavane et al., 2021).

This notion allows to construct polynomials of the form:

$$p_w(L) = w_0 I_n + w_1 L + w_2 L^2 + \cdots + w_d L^d = \sum_{i=0}^{d} w_i L^i \tag{3.8}$$

where each of these polynomials that follow the same structure can also be expressed by its vector coefficients $w = [w_0, \ldots, w_d]$. It should be noted that every $w, p_w(L)$ is an n x n matrix, just like L. This makes it possible to perceive polynomials as the equivalent of 'filters' in CNNs, and the coefficients w as the weights of the 'filters'.

While maintaining the previously selected order of all nodes, it is possible to stack all node features x_v (i.e., RGB pixel values) to get a vector $x \in \mathbb{R}^n$.

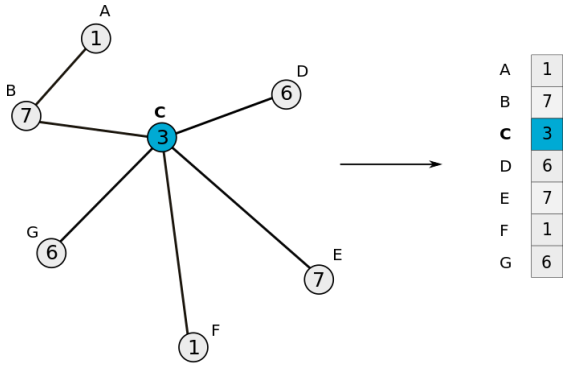


Figure 3.13: Mapping of the individual node n of the graph G to a fixed-size feature vector x.

With this terminology, the mathematical representation of a convolution x' of a feature vector x can ultimately be defined as follows:

$$x' = p_w(L)x = \sum_{i=0}^{d} w_i L^i = w_0 I_n x = x \tag{3.9}$$

4 METHODS

This chapter presents and explains the workings of all the deep learning models and algorithms used, which make up the designed end-to-end solution.

4.1 Data

Data availability and quality are the most crucial determinants of a well-performing DNN model. And depending on the application of the NN, it is possible to use pre-trained models that have been trained for a specific purpose. As long as the data of these models are compatible with the own use case.

Figure 4.1 shows a flow diagram of the designed end-to-end solution. Two DNN models (more in sec. 4.2 & 4.3) come into play in this end-end solution. The first one, OpenPose, is pre-trained, as training from scratch wouldn't have brought any change in performance, given that the end result would have been the same. ST-GCN, on the other hand, needed to be trained from scratch, to recognise the domain-specific action space. More in section 4.1.1.

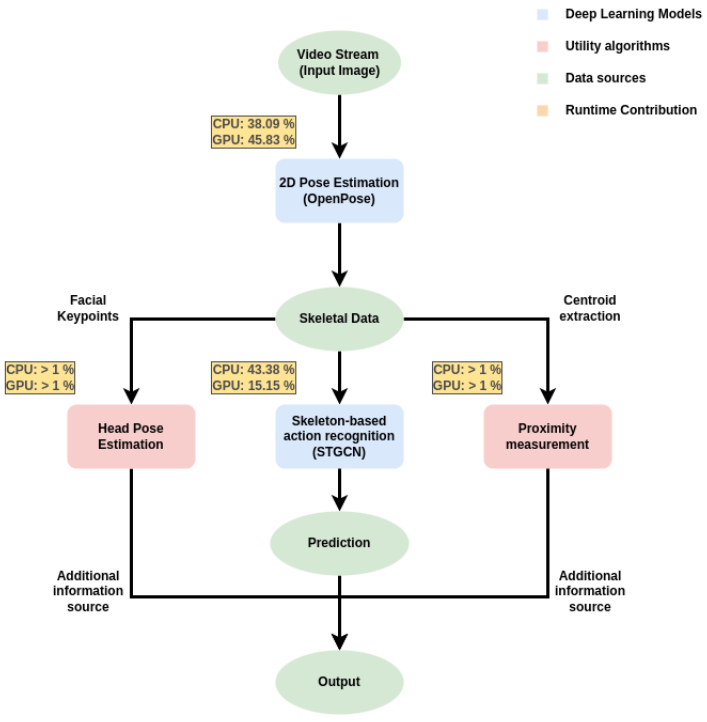


Figure 4.1: Diagram of the proposed end-to-end solution.

4.1.1 Defining the action space

The action space defines a range of well-chosen actions that are specific to the domain of autonomous outdoor navigation and that the proposed model can recognize. To create such an action space, it is first necessary to consider which movements are actually relevant for this application. Considering that the robot will be used in civic spaces, it can be assumed that the human performed actions are limited to dynamic locomotion and static poses.

In the initial phase, several popular benchmark datasets, such as *NTU_RGB_D*, *JHMDB* and *G3D* (Bloom et al., 2012; Jhuang et al., 2013; Shahroudy et al., 2016) were analysed to find actions and poses suitable for the application at hand. The result was the first draft of an action space:

Objects and Animals			
Cycle rickshaw	Conference bike	Recumbent bike	Cargo bike
City-roller	E-scooter	Cars	Kettcar
E-board	Segway	Motorized wheelchair	Rollator
Walking frame	Crutches	Dog	Bike
Bags	Shopping cart	Pushing pram	Broom
Skateboard	Bobby-car	Blind/walking cane	Grocery bags
Wheelchair	Box		

Postures			
Drunken	Gait abnormalities	Physical disabilities	Normal
Only on left leg	Only on right leg		

Directions			
Forward	Backward	Sideways left	Sideways right
Turn around left	Turn around right	On the spot	

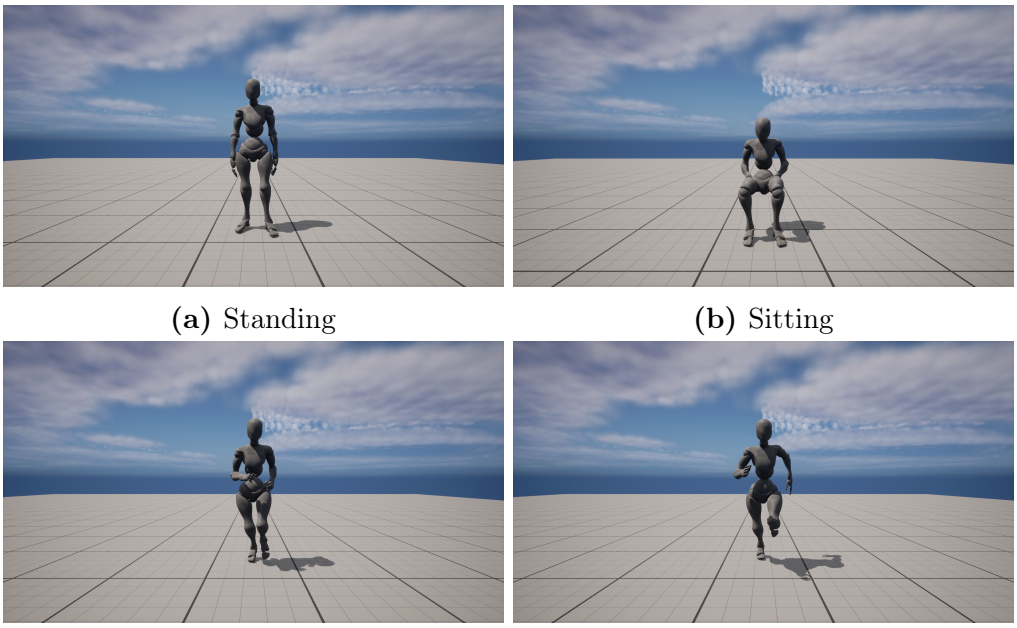
Actions			
Walking w, w/o object	Crawling	Running	Jogging w, w/o object
Standing w, w/o object	Roller skating	Biking	Skateboarding
Playing frisbee	Pushing cart	Pushing pram	Sweeping floor
Dribbling basketball	Passing football	Open/close door	Sitting
Jump	Drop something	Pick up something	Bump into someone
Walk side by side	Lend an arm to support someone	Walk hand in hand	

Figure 4.2: Initial draft of the designed action space.

However, to save time, this action space was reduced to 6 actions (i.e. standing, sitting, walking, jogging, running, skateboarding), as the results can easily be extrapolated to other actions if needed. Figure 4.3 illustrates some of these actions.

4.2 OpenPose

OpenPose, developed by researchers at Carnegie Mellon University, is a bottom-up approach for performing real-time pose estimation (Z. Cao et al., 2021). It takes a CNN-based approach and tackles the problem with a multi-stage classifier, where each stage improves the results of the previous one. This method is well suited



(a) Standing **(b)** Sitting

(c) Jogging **(d)** Running

Figure 4.3: Visualization of samples from the defined action space (created with UE5 (Epic Games, 2022) and Mixamo (Adobe Systems, 2022)). Top row: samples of static poses. Bottom row: sample of dynamic locomotion.

for this project, as it has already been tested for accuracy and performance in both academia and industry. More importantly, it can process pose estimates for multiple people with only a "small" drop in inference and increased computational costs, which is critical for hardware-constrained edge devices where real-time processing is desired (Osokin, 2019).

We based our work on the popular bottom-up method OpenPose, it has almost invariant to number of people inference time.

For this work, an OpenPose derivative called Lightweight OpenPose (Osokin, 2019) model was used, pre-trained on the Coco dataset. (Lin et al., 2014). *Note: OpenPose and Lightweight OpenPose are used interchangeably.*

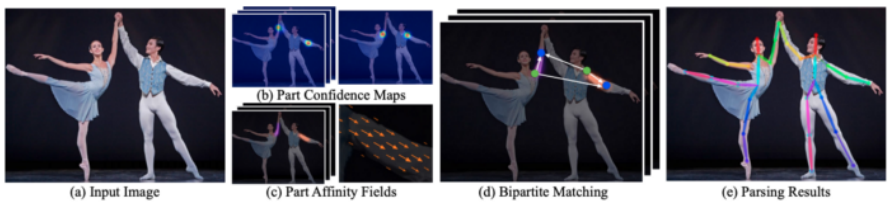


Figure 4.4: OpenPose Pipeline: Image taken from Z. Cao et al., 2021.

4.2.1 Procedure

In the pre-processing step the entire RGB input image is fed into a CNN baseline network to extract the corresponding feature maps F. In the original work, the authors used the first 10 layers of the VGG-19 architecture for this purpose. Osokin (2019) however, used the more efficient dilated MobileNet v1 as their feature extractor for Lightweight OpenPose. These feature maps are then forwarded to the multi-stage classifier. Multi-stage simply means that the classifier output of the stages are stacked on top of another in every stage of the process, as seen in Figure 4.5.

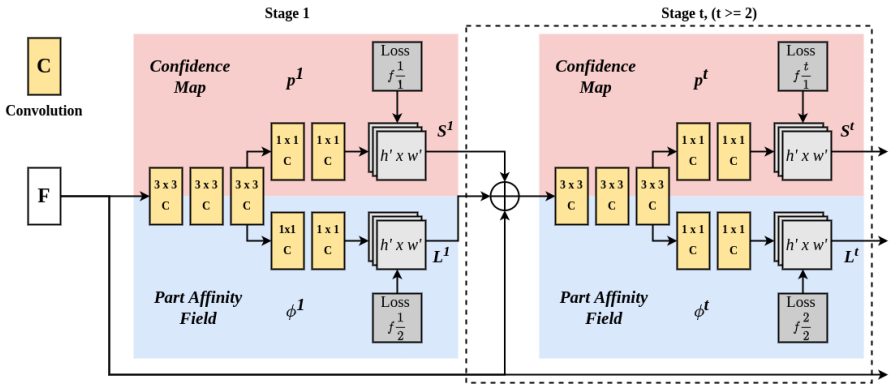


Figure 4.5: Lightweight OpenPose Architecture: All layers are shared except those that are directly responsible for producing the confidence score (Fig. 4.4) and the part affinity fields (Fig. 4.7).

In the first stage (left half of Figure. 4.5), the network produces an initial set of detection confidence maps S and a set of Part Affinity Field L. The confidence map can be understood as a 2D representation of the belief that a particular body part can be located in any given pixel. While PAFs encode the position and orientation of limbs that connect certain joints.

Confidence maps can be expressed mathematically as follows:

$$\begin{aligned} S = S_1, S_2, S_3, \ldots, S_j \\ S_j \in \mathbb{R}^{w\ x\ h} \\ j \in \{1; J\} \end{aligned} \tag{4.1}$$

where J represents the total number of body parts. See Figure. 3.8 for an overview. S refers to the score that has been given on

Part affinity fields maps can mathematically be expressed as follows:

$$
\begin{aligned}
L = L_1, L_2, L_3, \ldots, L_c \\
L_c \in \mathbb{R}^{w\ x\ h\ x\ 2} \\
c \in \{1; C\}
\end{aligned}
\tag{4.2}
$$

where C represents the total number of limbs. The paper refers to part pairs as limbs, which essentially represent a array containting the limb pairs that are connected by joints.

Then, in each subsequent stages (right half of Figure 4.5), the predictions from both branches in the previous stage, along with the original image features F, are concatenated and used to produce more refined predictions. as seen in Figure 4.6. In the OpenPose implementation, the final stage t is chosen to be 6.

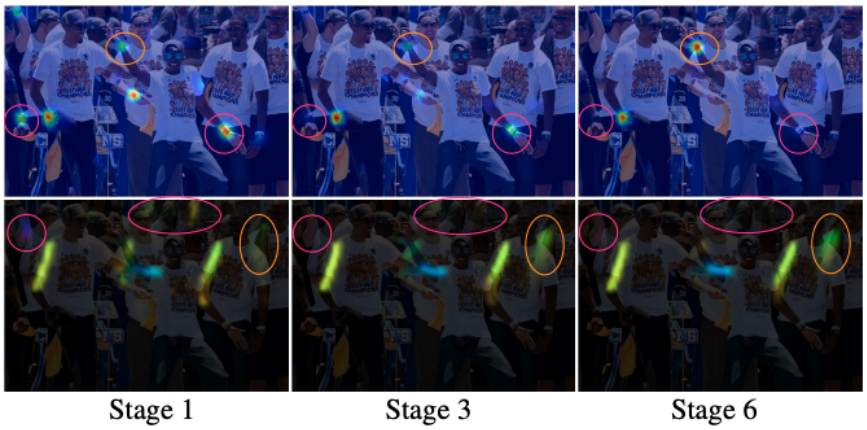


Figure 4.6: Outcome of a multi-stage network. Top-row: Shows the network predicting confidence maps of the right wrist. Bottom-row: Shows the network predicting the PAF of the right forearm (right shoulder - right wrist) across different stages.

At the final stage, a greedy inference algorithm is used to process the confidence maps and affinity fields (Figure 4.4d). In this process, all false keypoint associations are eliminated, which leaves only the "correct" 2D keypoints for all people (Figure 4.4e). The selection for elimination is based on the part-pair array. This is an array that contains all pairs of limbs that are connected by joints. This information is then used to determine the most natural connection for a skeleton by iteratively eliminating keypoint candidates.

Figure 4.7: A zoomed-in view of the PAF. Left: PAF corresponding to the limb connection right elbow and wrist. The colour encodes the orientation. Right: A 2D vector in each pixel of each PAF encodes the position and orientation of the limb.

4.3 ST-GCN

ST-GCN (Yan et al., 2018) is a supervised machine learning model that classifies human actions based on graph-structured skeletal information obtained from human posture estimation algorithms such as OpenPose. Moreover, this architecture provides a systematic way to do convolution on a graph that can exploit the locality of graph convolution together with temporal dynamics and has been trained on the custom data introduced in section 4.1.

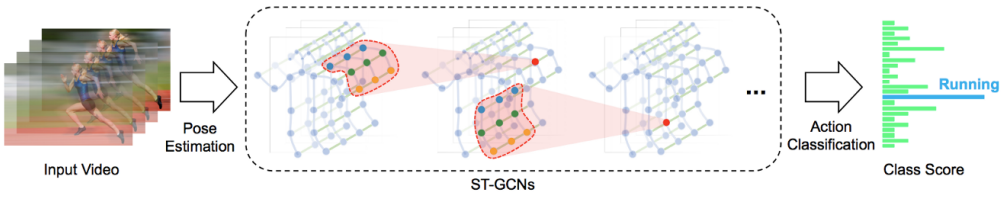


Figure 4.8: Action classification pipeline: The action frames provided by OpenPose represent the input. The graph-structured skeleton data then pass through several spatio-temporal graph convolution blocks and are finally classified according to their action.

4.3.1 Procedure

Chapter 3.5.2 already explained how GCN-based models perform convolutions on graph-like structures in the spatial domain. ST-GCN builds on this approach and introduces some partitioning rules (e.g. that a kernel only acts on the neighbour directly connected to the node) to find correlations between neighbouring keypoints (joints) in the temporal domain when actions are performed.

The definitions for the three examined partitioning strategies from the original work are presented below (See Fig. 4.9 for visual illustration):

- **Uni-labelling**: Places all neighbourhood nodes with their respective root nodes in the same subset
- **Distance partioning**: Places the root node and its neighbouring nodes into two different subsets (based on a specified distance threshold)
- **Spatial configuration partioning**: If a joint is further away from the centroid than the average distance between them, it is placed in the centrifugal group. Otherwise, it is placed in the centripetal group. The root node itself is placed in the third group.

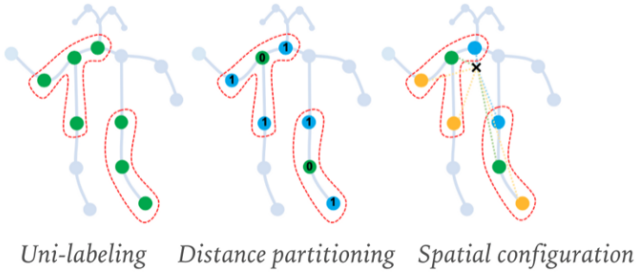


Figure 4.9: Partitioning strategies analysed in the original paper. *Note: Spatial configuration partioning produced the best result.*

Figure 4.10 shows the network architecture of ST-GCN. The network consists of several spatio-temporal graph convolution blocks. Each of these blocks consists of four components. The first component involves temporal convolution at each node to obtain the temporal feature vectors. The second component involves the selection of a subset based on the selected partitioning strategy. The third component involves the spatial convolution of this subset and the last component represents another temporal convolution to obtain optimised results.

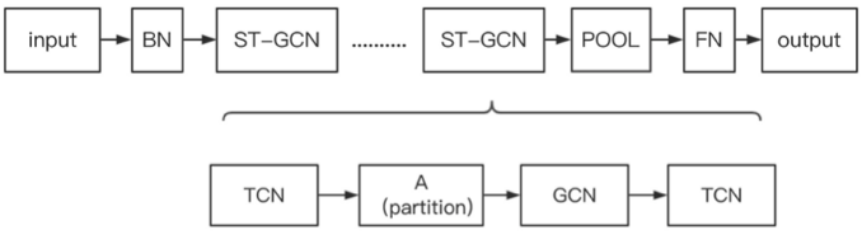


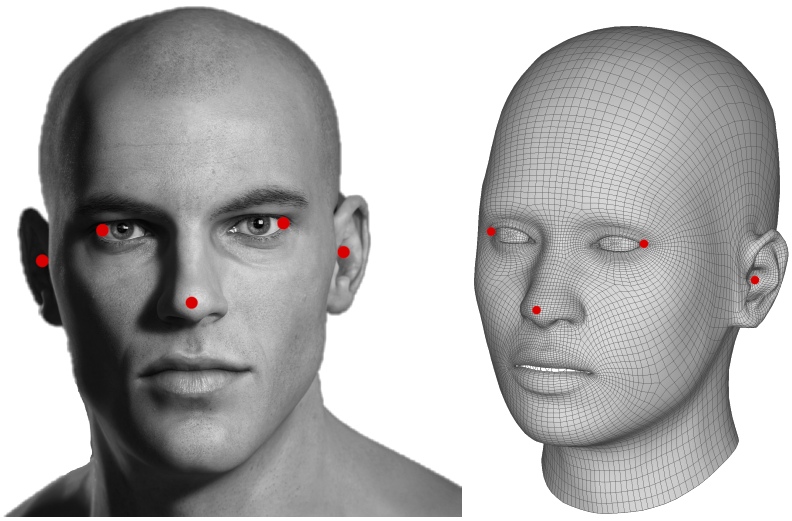
Figure 4.10: ST-GCN model architecture.

4.4 Head Pose Estimation

Head pose estimation can be referred to as Perspective-n-Point problem or PNP in computer vision jargon that focuses on a specific area of the body - the head. The aim is to determine the tilt of the head relative to the camera and use this information to help the robot determine in which direction this person is looking. This information is valuable because it can be used later in the robot's decision-making process, providing information about group formations or forecasting that a pedestrian is ready to cross the street after tilting his or her head left and right to check for incoming traffic.

4.4.1 Procedure

The head pose estimation is inspired by Mallick (n.d.), and has a few base requirements to work properly: 1. 2D Coordinates of a few facial landmarks, 2. Generic 3D coordinates of the same facial landmarks in an arbitrary reference frame and the rotation matrix and translation vector between them. Figure 4.11 illustrates the selection of the used keypoints (i.e., left ear, left eye, nose, right eye, and right ear).



(a) 2D **(b)** 3D

Figure 4.11: Arbitrary selection of common facial landmarks in 2D and 3D space.

Figure 4.12 portrays three coordinate systems that come into play for capturing the head pose estimation. The 3D coordinates of the selected facial features are in world coordinates. It should be well known from the field of robotics that a pose can be defined by a rotation matrix R and a translation vector t. If this information

is available, it is possible to infer from the 3D points in world coordinates to the 3D points in camera coordinates. These points can then be further projected onto the 2D image plane using the intrinsic parameters of the camera.

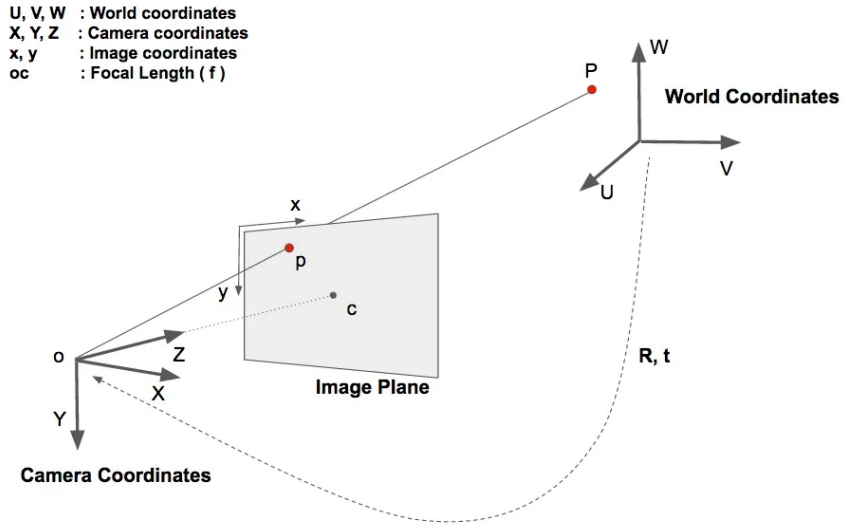


Figure 4.12: Reverse projection from 2D to 3D. Image taken from Mallick, n.d.

This process can be represented mathematically by the following equations:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R \begin{bmatrix} U \\ V \\ W \end{bmatrix} + t$$

$$\Rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [R \mid t] \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

In extended form, the above equation looks like this:

$$\Rightarrow \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

and equates to a complex linear system that can be partially solved for the unknowns r_{ij} and (t_x, t_y, t_z) based on the number of point correspondences (i.e.

(U, V, W)) and (X, Y, Z).

4.4.2 solvePnP

The OpenCv library (Bradski, 2000) provides two functions called solvePnP and solvePnPRansac ("Perspective-n-Point (PnP) pose computation", n.d.), respectively, that can be used to estimate 3D poses for PnP problems. Both functions do fundamentally the same thing, with the only difference being that solvePnPRansac uses RANSAC (Fischler and Bolles, 1981) to estimate 3D poses robustly in the presence of noisy data points. Both functions offer several algorithms for 3D pose estimation, which can be selected using the parameter *flag*. To find a suitable algorithm, one has to consider the number of observable facial landmarks, since some algorithms have minimum requirements. Given the limited number of facial landmarks provided by OpenPose under the 18-point keypoints scheme. This project used solvePnP with the SOLVEPNP_SQPNP flag, as it showed the best results.

4.5 Proximity measurement

Proximity measurement is about estimating the distance between people. This information is valuable for the robot's decision-making process, as there are social rules that must be followed in order not to disturb the general state of comfort. One of these rules could be to push through a group of people. The head pose estimation in combination with the distance measurement is a good indicator if a social group has formed and if there is enough space for the robot to pass though it.

4.5.1 Procedure

The proximity measurement procedure is a two-step process. First, a person's centroid is calculated by using the neck, left hip, and right hip keypoints, and then computing the euclidean distance between these centroids. It is important to note that this approach is only an approximation given by the limitation of 2D data.

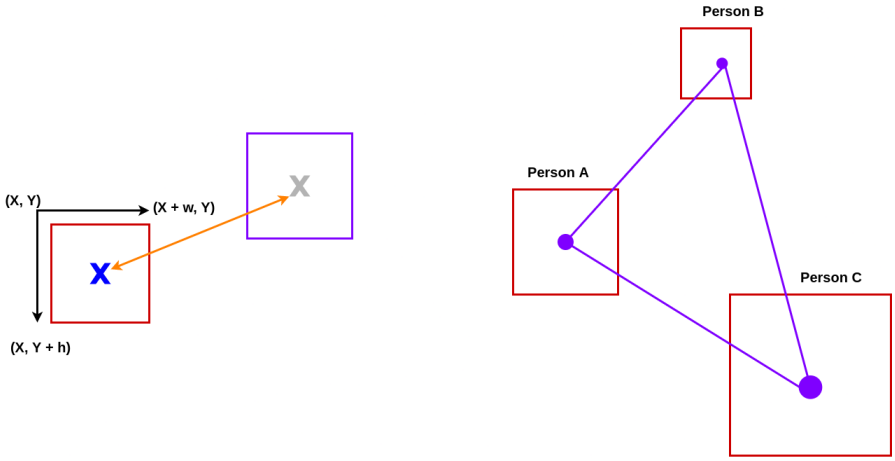


Figure 4.13: Left) Calculating distances between centroids of bounding boxes. Right) Calculating different permutations between centroids.

5 RESULTS

The following chapter describes the experiments conducted and evaluates the individual components of the proposed end-to-end solution as well as the entire end-to-end solution on the basis of the results obtained.

5.1 Pose Estimation

Figure 5.1 illustrates the inference of an extract from the video sequence of an example from the custom dataset. It clearly shows the accuracy of the extracted skeleton (highlighted in yellow) obtained by the Lightweight OpenPose architecture.

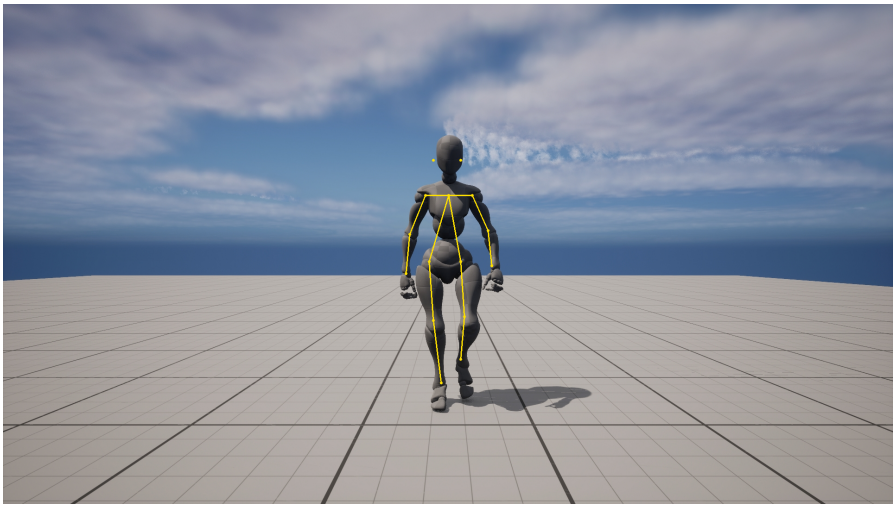


Figure 5.1: OpenPose inference on a sample image.

Given that in the application area of this project, however, the skeletons of an arbitrary number of persons have to be extracted, it is important to find out to what extent the performance changes with increasing numbers of persons. To this end, an experiment was conducted to analyse the change in FPS between 1, 5 and 9 people in the frame. Figure 5.2 shows the results of this experiment.

The result of this experiment makes it clear that there is a negative correlation between performance and the number of people in the image, as the FPS value drops from 60 to 40 when using a GPU and the number of people in the image

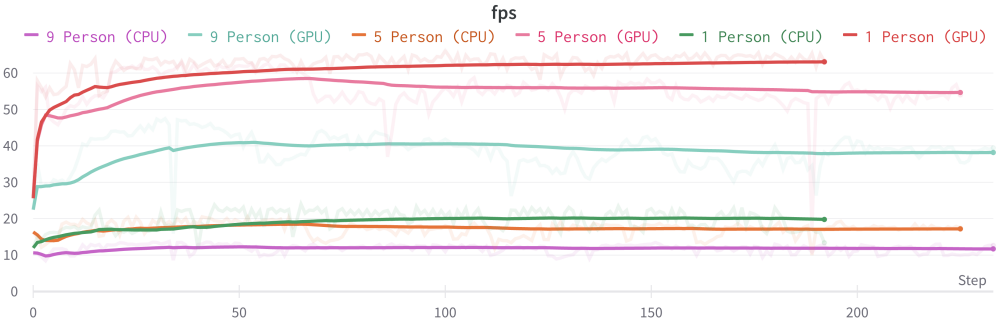


Figure 5.2: Performance experiment results indicate a negative correlation between FPS and the number of people in the frame. A demonstration of the experiment can be found here.

increases from 1 to 9. The same trend can be observed when using the Central Processing Unit (CPU).

5.1.1 Optimizations

Figure 7.2 shows that the OpenPose model is one of the most computationally intensive components of the designed end-to-end solution. Thus, it is also one of the most critical touch points in terms of optimisation potential. In chapter 4.2, the architecture of the present human pose estimation algorithm was already discussed in detail. Therefore, it should be known at this point that the model passes through a certain number n of refinement stages. As such, the first suggestion for optimisation is to reduce the default value of 6 to 0, so that we only rely on the initial stage (no refinement stages). And the next suggestion is to addresses the arithmetic computational cost. By reducing the precision from 32-bit to 16-bit (half precision), usually, non-destructive optimisations can be performed that do not negatively affect performance.

Performing those optimisations led to a FPS increase of 73% (standalone), and a 11.59% decrease of computational cost as can seen in Figure 7.3.

5.2 Action recognition

As the action recognition model was trained from scratch, it is relevant to evaluate the performance of the model not only visually but also statistically. For this purpose, the loss and accuracy metrics of the training and validation runs are used in the following. Figure 5.3 shows a progression of the respective metrics.

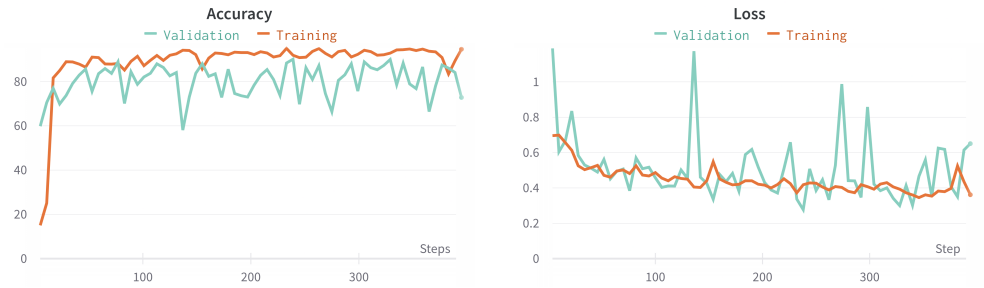


Figure 5.3: Evaluation plots. The left image contains the curve of accuracy for both training and validation and shows results of 92% and 81% respectively. The right image shows the curve of the loss for both training and validation and shows results of 0.4 and 0.6 respectively.

The metrics suggest a successful and robust progression (no under or overfitting), demonstrating that the action recognition model was able to correctly classify the actions from the custom dataset 81% of the time.

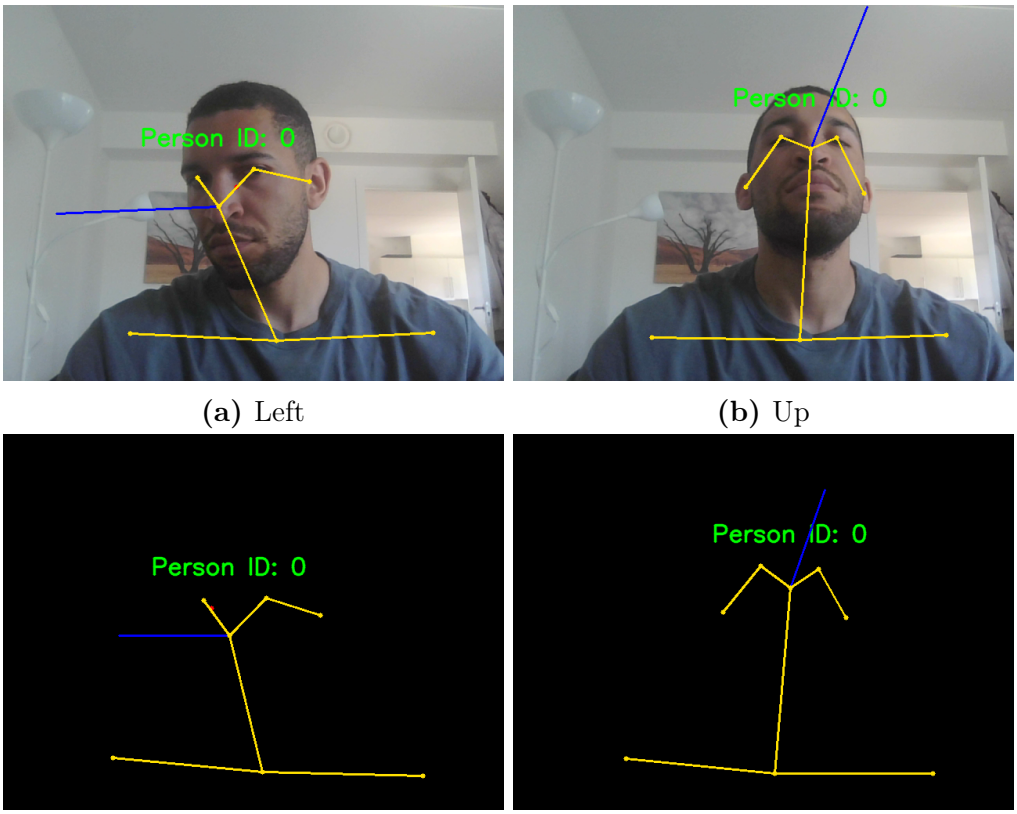
5.3 Head Pose Estimation

Figure 5.4 shows sample images of the viewing direction in both RGB and skeleton viewing modes and gives a first visual impression of the performance of the implemented algorithm. However, in order to evaluate the performance of the algorithm numerically, a study was conducted. In this study, 10 participants annotated 4 images taken from the Head Pose Database (Gourier et al., 2004) dataset by drawing a simple line in the direction of head tilt. The vector values obtained by these lines were then averaged and used as ground truth against which the results could be numerically measured (See Fig. 5.5).

The obtained values, have been calculated by the mean absolute error loss function. Which essentially takes the average of the absolute errors between the two vectors. The algorithm achieved an error rate of 7.06, and after using Figure 5.6 as a guide to unambiguously interpret this result, one can clearly state that the performance is very accurate.

5.4 Proximity measurement

In order to evaluate the performance of the proximity measurement, an experiment was conducted. In this experiment, 2-3 people were placed at different positions in the camera's frame (the positions were marked with tape). The distance between these people was measured with a tape measure and serves as a benchmark for



(a) Left **(b)** Up

(c) Left **(d)** Left

Figure 5.4: Image pairs of full RGB images (top row) and the corresponding abstracted version containing the keypoints (bottom row) with the viewing direction enabled. A demonstration can be found here.

the evaluation. This reference value is then compared with the values determined by the predicted values. This process was repeated 3 times to obtain an average result.

Figure 5.7 shows the results of the experiment. The proposed algorithm achieves a performance of 64.4% accuracy when simply averaged and produces a mae score of 68.11, indicating that the proposed approximation is currently not reliable enough to rely on for the decision-making process. See Figure 5.6 for interpretation of the mae score.

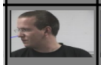
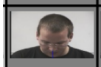
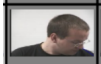
Ground truth	Prediction	MAE
		4.0
		1.5
		20.25
		2.5
		7.06

Figure 5.5: Results of the conducted survey. The mean absolute error (MAE) loss was used to calculate the error between the two vectors, with values closer to 0 indicating good performance. See 7.1 for the complete survey information.

Values	Interpretation
< 10	Highly accurate
11 – 20	Good
21 – 50	Reasonable
51+	Inaccurate

Figure 5.6: MAE interpretation table, taken from Zirebwa et al., 2015.

	AB			AC			BC			Acc.
	GT	P	E	GT	P	E	GT	P	E	
Video 1	1.74m	1.58m	90.8%	1.98m	1.67m	84.3%	2.56m	3.24m	73.4%	
Video 2	2.00m	1.60m	80.0%	1.98m	2.32m	82.8%	2.47m	0.78m	31.6%	
Video 3	1.74m	0.78m	44.5%	2.47m	1.50m	35,3%	1.42m	0.80m	56.3%	
			71.8%			**67.5%**			**53.8%**	**64.4%**

Figure 5.7: Results of the proximity benchmark. The true distance values between 3 people were compared with the predicted values of the respective videos. All error rates were then averaged to give an error rate of 64.4%.

5.5 End-to-end solution

Extending the FPS experiment already conducted in section 5.1, the inference of action recognition is now included in this evaluation in a follow-up experiment. The objective of this experiment is to analyse the change in FPS between 1, 5 and 9 people in the frame for the entire end-to-end solution. Figure 5.8 shows the obtained results.

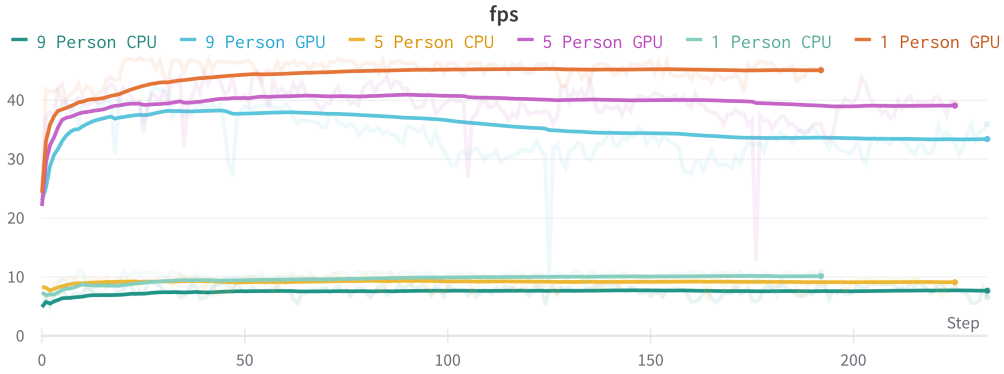


Figure 5.8: Performance experiment results indicate a negative correlation between FPS and the number of people in the frame. (*Note: The optimization steps introduced in section 5.1 have already been in cooperated.*)

In this experiment, a negative correlation between the number of people and the performance can also be observed. As the performance in FPS drops from 45 to 35, when increasing the people in the frame from 1 to 9. A demonstration of the inference of the proposed solution can be found here.

(*Note: The action recognition predictions are printed in the terminal so as not to clutter the output frame. Additionally, it is imperative to emphasise that the visualisation is only needed for human understanding, as an autonomous robot will only need the generated information. See Fig. 5.9*).

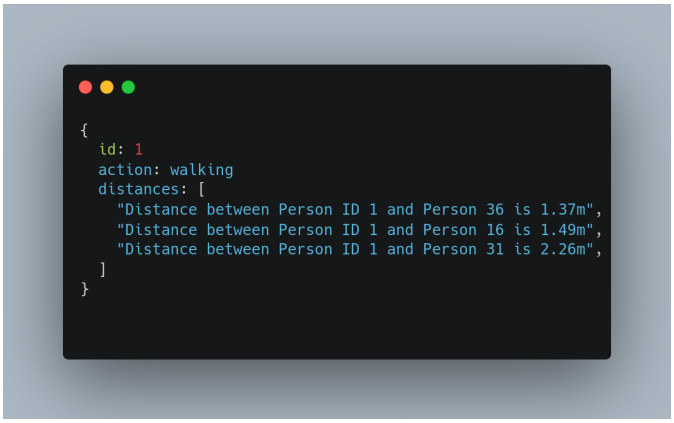


Figure 5.9: Snippet illustrating the console output of the action classification of a pedestrian in JSON format.

6 DISCUSSION

The results from the previous chapter clearly show that this thesis has succeeded in combining several components into an end-to-end solution for skeleton-based action recognition. In particular, the head pose estimation has proven to be very reliable, as indicated by the mae score of 7.06. The optimisation approaches on the OpenPose model also proved to be effective, as they improved the runtime speed. Figure 7.3 also confirms this result, as the system load of the model has dropped from the original 45.84% to 32.34%. One statement that is not so easy to answer, however, is whether the solution provided can meet the real-time requirements on an NVIDIA Jetson Xavier NX. For this reason, it is necessary to make assumptions based on related work. The OpenDR team (Passalis et al., 2022), whose code base laid the foundation for this project, provides such metrics.

Method	CPU	Jetson TX2	Jetson Xavier NX	RTX 2070 Ti
OpenPose	47.4	17.1	18.8	98.2
ST-GCN	13.26	4.89	15.27	63.32

Table 6.1: Obtained marks.

They established in their tests, that the standalone OpenPose algorithm is capable to perform with up to 18.8 FPS on a NVIDIA Jetson Xavier NX. The standalone action recognition was able to run at 15.27 FPS on the Jetson Xavier NX. Unfortunately, they did not provide a joint test. But if one takes into consideration that the performance in terms of FPS drops from 68 FPS (OpenPose) to 22 FPS (OpenPose + ST-GCN) can observe a reduction in performance of 0.65%. If this value roughly holds across different hardware then we can assume that the to expected performance would be around 6.5 FPS, which would indicate that the proposed solution doesn't fullfill it's use case. This however, cannot be verified as of now and needs to be answered in follow up works.

The results have also revealed another shortcoming of the proposed end-to-end solution. Namely, contrary to earlier assumptions, ST-GCN is not able to predict individual actions. The experiments have shown that this model architecture considers actions holistically. This means that the model does not classify actions in a recurrent way for all the skeletons provided by OpenPose, but focuses only on a small subset of skeletons in the frame that might be involved in a joint action. Unfortunately, discussions with some relevant people in the field, as well as the

documentation of the OpenDR project and especially insufficient attention in the literature review phase are responsible for this misunderstanding. In particular, the corresponding function parameter of this architecture: num_person is described in the documentation as follows: "Specifies the number of body skeletons in each frame", and thus leaves room for misinterpretation.

Note: One workaround could be to inference over each skeleton in the frame individually, but that comes with the cost of a increase in computational cost of $O(n^2)$, which might become a reasonable solution under the assumption that the number of pedestrians is limited and a action recognition is found that performs inferences efficiently.

Nevertheless, these findings are relevant and insightful because they suggest that an end-to-end solution that meets the requirements of this work is feasible with some minor adjustments. And these insights are important because an autonomous robot needs to access this kind of information in real-time to make action decisions. Otherwise, it is not possible to use these kinds of robots in civic spaces, as they could harm citizens. It is also important to note that the data modality of the skeletal information has proven to be appropriate for the intended use case, which is also worth highlighting in terms of compliance with the GDPR.

7 CONCLUSION

Due to the unavailability of the intended SBC, it is unclear whether the final results presented in Chapter 5 for the developed end-to-end solution for skeleton-based action recognition (15-20 FPS on an NVIDIA Jetson Xavier NX) will be met. However, it was clearly demonstrated that the combination of the individual components into an end-to-end solution was achieved. What follows next for this project is to substitute ST-GCN for a different model architecture and deploy the proposed solution on an NVIDIA Jetson Xavier NX, evaluate the performance and ultimately the integration onto the Capra Hircus robot. Moving forward, it will also be necessary to combine action recognition with path planning and to design, and integrate the robot's response strategy to react to the pedestrian's predicted actions. Other interesting areas of research could focus on further improving the performance of the proposed solution. This could be done by reducing model sizes with novel quantization or pruning techniques, as shown in the work of J. Chen and Ran (2019) and Han et al. (2016), or by using other models that are more efficient and powerful. Allowing them to ultimately achieve comparable results on less powerful and less expensive hardware such as the NVIDIA Jetson Nano. Another continuation of the work could be to compare 3D skeletal action recognition, as it contains valuable information in the extra dimension that could help in the task of head pose estimation and spatial positioning of multiple people in the scene. Flamis et al. (2021) introduce another possible area for continuation as they claim in their paper that GPUs, although beneficial for training, consume a lot of energy during inference. Therefore, other hardware accelerators such as Field Programmable Gate Arrayss could be explored as they are claimed to be much more energy efficient and therefore very suitable for the task of ML inference on edge devices.

APPENDIX

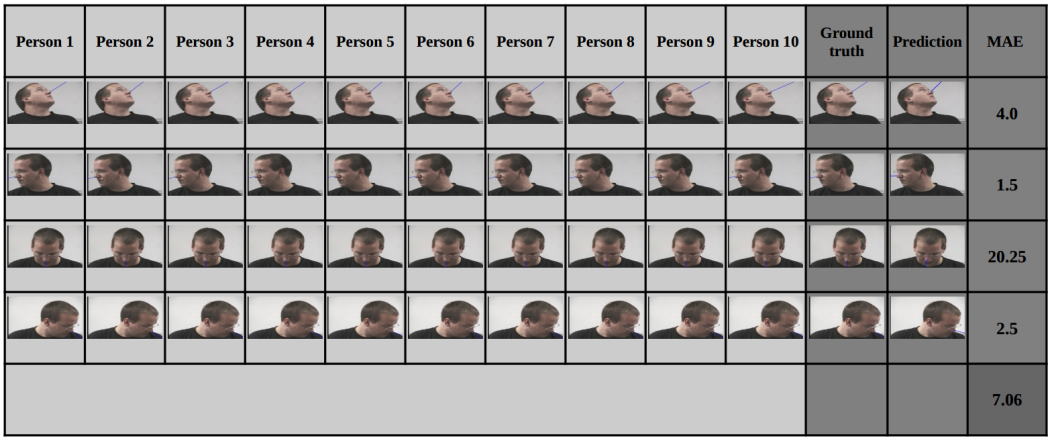


Figure 7.1: Description of the experiment: 10 participants annotated pictures based on the direction of gaze. The drawn vectors were then averaged to obtain a ground truth vector against which the results of the gaze direction estimation algorithm could be measured. The mean absolute error (MAE) loss was used to calculate the error between the two vectors, with values closer to 0 indicating good performance. See Figure 5.6 as interpretation guideline.

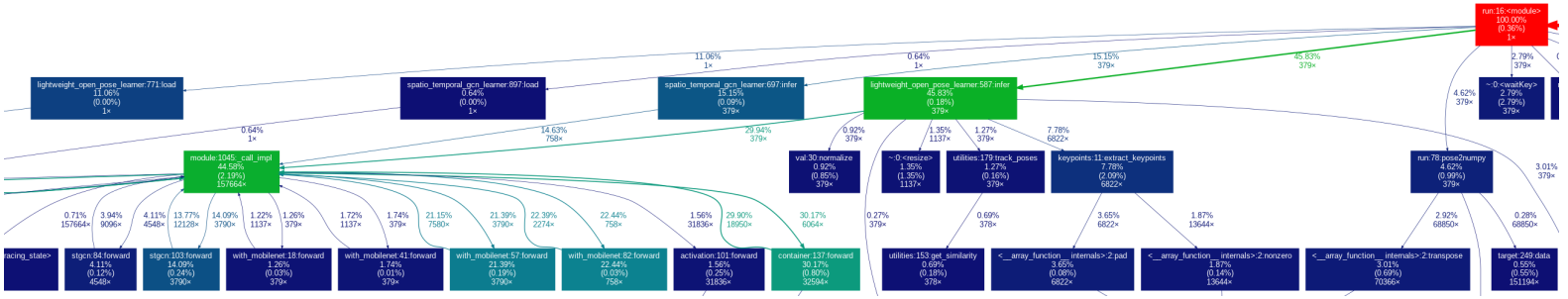


Figure 7.2: This image illustrates a portion of the call stack at runtime of the end-to-end solution when analysed with the cProfile function and plotted with gprof2dot. The potential bottlenecks are colour-coded, and essentially boil down to the inference of both deep learning models, with a runtime contribution of 45.83 % GPU (38.09 % CPU) for OpenPose and 15.15 % GPU (43.38 % CPU) for ST-GCN.

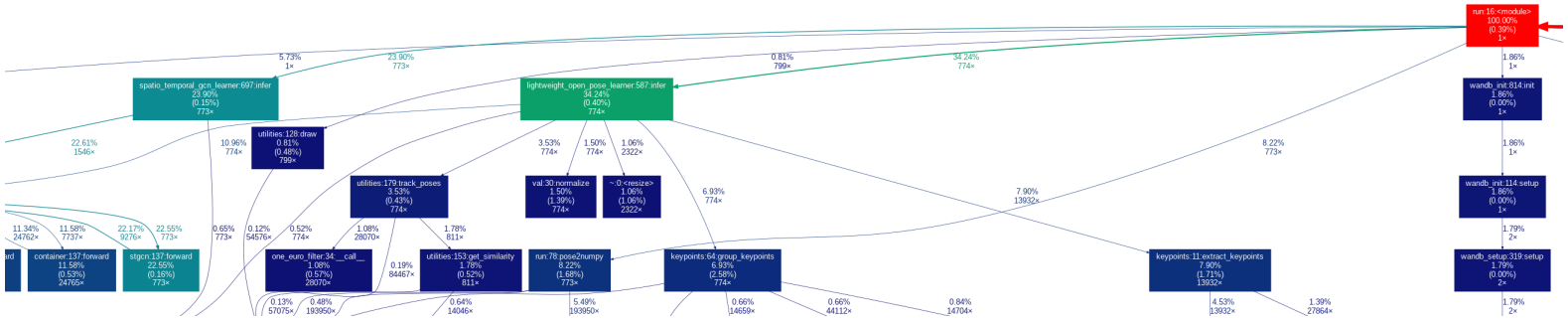


Figure 7.3: This image illustrates a portion of the call stack at runtime of the optimized end-to-end solution when analysed with the cProfile function and plotted with gprof2dot. The potential bottlenecks are colour-coded, and essentially boil down to the inference of both deep learning models, with a runtime contribution of 34.24 % GPU (17.84 % CPU) for OpenPose and 23.90 % GPU (63.44 % CPU) for ST-GCN.

REFERENCES

About predicting the future. (n.d.). https://course.elementsofai.com/. (Cit. on p. 18)

Adobe Systems. (2022, June 1). *Mixamo* (Version 5.1.1). https://www.mixamo.com/. (Cit. on p. 29)

Aggarwal, J., & Ryoo, M. (2011). Human activity analysis: A review. *ACM Comput. Surv.*, *43*(3) (cit. on p. 9).

Bellamy, D. D., & Caleb-Solly, P. (2019). Collaborative hri and machine learning for constructing personalised physical exercise databases. *TAROS* (cit. on p. 9).

Bloom, V., Makris, D., & Argyriou, V. (2012). G3d: A gaming action dataset and real time action recognition evaluation framework. *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 7–12 (cit. on p. 28).

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (cit. on p. 36).

Brown, R. E. (2020). Donald o. hebb and the organization of behavior: 17 years in the writing. *Molecular Brain*, *13* (cit. on p. 12).

Brownlee, J. (2019). *Generative adversarial networks with python*. (Cit. on pp. 17, 18).

Cao, C., Lan, C., Zhang, Y., Zeng, W., Lu, H., & Zhang, Y. (2019). Skeleton-based action recognition with gated convolutional neural networks. *IEEE Trans. Cir. and Sys. for Video Technol.*, *29*(11), 3247–3257 (cit. on p. 7).

Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E., & Sheikh, Y. (2021). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*, 172–186 (cit. on pp. 22, 28, 29).

Chen, J., & Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE*, *107*, 1655–1674 (cit. on p. 47).

Chen, T., Zhou, D., Wang, J., Wang, S., Guan, Y., He, X., & Ding, E. (2021). Learning multi-granular spatio-temporal graph network for skeleton-based action recognition. *Proceedings of the 29th ACM International Conference on Multimedia* (cit. on p. 8).

Chen, Y., Wang, Z., Peng, Y., Zhang, Z., Yu, G., & Sun, J. (2018). Cascaded pyramid network for multi-person pose estimation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 7103–7112 (cit. on p. 7).

Chen, Y., Tian, Y., & He, M. (2020). Monocular human pose estimation: A survey of deep learning-based methods. *Comput. Vis. Image Underst.*, *192*, 102897 (cit. on p. 6).

Cheng, K., Zhang, Y., He, X., Chen, W., Cheng, J., & Lu, H. (2020). Skeleton-based action recognition with shift graph convolutional network. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 180–189 (cit. on p. 8).

Daigavane, A., Ravindran, B., & Aggarwal, G. (2021, September 2). *Understanding convolutions on graphs*. https://distill.pub/2021/understanding-gnns/. (Cit. on pp. 23, 25)

Duan, H., Zhao, Y., Chen, K., Shao, D., Lin, D., & Dai, B. (2021). Revisiting skeleton-based action recognition. *ArXiv*, *abs/2104.13586* (cit. on p. 8).

Epic Games. (2022, June 1). *Unreal engine* (Version 5.1.1). https://www.unrealengine.com. (Cit. on p. 29)

Fang, H., Xie, S., Tai, Y.-W., & Lu, C. (2017). Rmpe: Regional multi-person pose estimation. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2353–2362 (cit. on p. 7).

Feichtenhofer, C., Fan, H., Malik, J., & He, K. (2019). Slowfast networks for video recognition. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 6201–6210 (cit. on p. 9).

Fischler, M. A., & Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, *24*, 381–395 (cit. on p. 36).

Flamis, G., Kalapothas, S., & Kitsos, P. (2021). Best practices for the deployment of edge inference: The conclusions to start designing. *Electronics* (cit. on p. 47).

Fridman, L. (2019). Deep learning basics [Accessed: 2022–05-29]. (Cit. on p. 10).

Gourier, N., Hall, D., & Crowley, J. L. (2004). Estimating face orientation from robust detection of salient facial structures (cit. on p. 40).

Han, S., Mao, H., & Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv: Computer Vision and Pattern Recognition* (cit. on p. 47).

Heidari, N., & Iosifidis, A. (2021). On the spatial attention in spatio-temporal graph convolutional networks for skeleton-based human action recognition. *2021 International Joint Conference on Neural Networks (IJCNN)*, 1–7 (cit. on p. 8).

Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M., & Schiele, B. (2016). Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *ECCV* (cit. on p. 6).

Iqbal, U., & Gall, J. (2016). Multi-person pose estimation with local joint-to-person associations. *ArXiv*, *abs/1608.08526* (cit. on p. 6).

Jhuang, H., Gall, J., Zuffi, S., Schmid, C., & Black, M. J. (2013). Towards understanding action recognition. *International Conf. on Computer Vision (ICCV)*, 3192–3199 (cit. on p. 28).

Johnson, D. (2022, July 16). *Reinforcement learning: What is, algorithms, types & examples*. https://www.guru99.com/reinforcement-learning-tutorial.html. (Cit. on p. 18)

Ke, L., Peng, K.-C., & Lyu, S. (2022). Towards to-a-t spatio-temporal focus for skeleton-based action recognition. *ArXiv*, *abs/2202.02314* (cit. on pp. 7, 8).

Ke, Q., Bennamoun, M., An, S., Sohel, F., & Boussaid, F. (2017). A new representation of skeleton sequences for 3d action recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 7).

Kong, J., Bian, Y., & Jiang, M. (2022). Mtt: Multi-scale temporal transformer for skeleton-based action recognition. *IEEE Signal Processing Letters*, *29*, 528–532 (cit. on p. 8).

Kreiss, S., Bertoni, L., & Alahi, A. (2019). Pifpaf: Composite fields for human pose estimation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 11969–11978 (cit. on p. 6).

Li, C., Zhong, Q., Xie, D., & Pu, S. (2017). Skeleton-based action recognition with convolutional neural networks. *2017 IEEE International Conference on Multimedia & Expo Workshops, ICME Workshops, Hong Kong, China, July 10-14, 2017*, 597–600 (cit. on p. 7).

Lin, T.-Y., Maire, M., Belongie, S. J., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. *ECCV* (cit. on pp. 22, 29).

Liu, H., Tu, J., & Liu, M. (2017). Two-stream 3d convolutional neural network for skeleton-based action recognition. *ArXiv*, *abs/1705.08106* (cit. on p. 7).

Mallick, S. (n.d.). *Head pose estimation using opencv and dlib*. https://learnopencv.com/wp-content/uploads/2016/09/pose-estimation-requirements-opencv.jpg. (Cit. on pp. 34, 35)

Mobini, A., Behzadipour, S., & Foumani, M. S. (2014). Accuracy of kinect's skeleton tracking for upper body rehabilitation applications. *Disability and Rehabilitation: Assistive Technology*, *9*, 344–352 (cit. on p. 9).

Munaro, M., & Menegatti, E. (2014). Fast rgb-d people tracking for service robots. *Autonomous Robots*, *37*, 227–242 (cit. on p. 9).

Osokin, D. (2019). Real-time 2d multi-person pose estimation on cpu: Lightweight openpose. *ICPRAM* (cit. on pp. 29, 30).

Papandreou, G., Zhu, T. L., Chen, L.-C., Gidaris, S., Tompson, J., & Murphy, K. P. (2018). Personlab: Person pose estimation and instance segmentation with a bottom-up, part-based, geometric embedding model. *ECCV* (cit. on p. 6).

Passalis, N., Pedrazzi, S., Babuska, R., Burgard, W., Dias, D., Ferro, F., Gabbouj, M., Green, O., Iosifidis, A., Kayacan, E., Kober, J., Michel, O., Nikolaidis,

N., Nousi, P., Pieters, R., Tzelepi, M., Valada, A., & Tefas, A. (2022). Opendr: An open toolkit for enabling high performance, low footprint deep learning for robotics. *arXiv preprint arXiv:2203.00403* (cit. on p. 45).

Perspective-n-point (pnp) pose computation. (n.d.). https://docs.opencv.org/5.x/d5/d1f/calib3d_solvePnP.html. (Cit. on p. 36)

Pilarski, P. M., Butcher, A., Johanson, M. B., Botvinick, M. M., Bolt, A., & Parker, A. S. R. (2019). Learned human-agent decision-making, communication and joint action in a virtual reality environment. *ArXiv*, *abs/1905.02691* (cit. on p. 9).

Pishchulin, L., Insafutdinov, E., Tang, S., Andres, B., Andriluka, M., Gehler, P. V., & Schiele, B. (2016). Deepcut: Joint subset partition and labeling for multi person pose estimation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4929–4937 (cit. on p. 6).

Plizzari, C., Cannici, M., & Matteucci, M. (2021). Skeleton-based action recognition via spatial and temporal transformer networks. *Comput. Vis. Image Underst.*, *208-209*, 103219 (cit. on p. 8).

Poppe, R. (2010). A survey on vision-based human action recognition. *Image Vis. Comput.*, *28*, 976–990 (cit. on p. 9).

Qin, Z., Liu, Y., Ji, P., Kim, D., Wang, L., McKay, B., Anwar, S., & Gedeon, T. (2021). Fusing higher-order features in graph neural networks for skeleton-based action recognition (cit. on p. 8).

Qiu, H., Hou, B., Ren, B., & Zhang, X. (2022). Spatio-temporal tuples transformer for skeleton-based action recognition. *ArXiv*, *abs/2201.02849* (cit. on p. 8).

Rosenbrock, A. (2018). *Deep learning for computer vision with python*. PyImageSearch. (Cit. on pp. 10–12, 14, 17, 19, 20).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536 (cit. on p. 14).

Rupprecht, M. (2016). *Praxisbuch der mustererkennung*. (Cit. on pp. 11, 17–21).

Shahroudy, A., Liu, J., Ng, T.-T., & Wang, G. (2016). Ntu rgb+d: A large scale dataset for 3d human activity analysis. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1010–1019 (cit. on pp. 7, 28).

Shi, L., Zhang, Y., Cheng, J., & Lu, H. (2020). Decoupled spatial-temporal attention network for skeleton-based action recognition. *ArXiv*, *abs/2007.03263* (cit. on p. 8).

Song, Y., Zhang, Z., Shan, C., & Wang, L. (2022). Constructing stronger and faster baselines for skeleton-based action recognition. *IEEE transactions on pattern analysis and machine intelligence*, *PP* (cit. on p. 8).

Soo Kim, T., & Reiter, A. (2017). Interpretable 3d human action analysis with temporal convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (cit. on p. 7).

Taha, A., Zayed, H. H., Khalifa, M. E., & El-Horbaty, E.-S. M. (2015). Skeleton-based human activity recognition for video surveillance. *International journal of scientific and engineering research*, *6*, 993–1004 (cit. on p. 9).

Vadapalli, P. (2021, February). *Biological neural network: Importance, components & comparison*. https://www.upgrad.com/blog/biological-neural-network/. (Cit. on p. 10)

Wang, L., Koniusz, P., & Huynh, D. Q. (2019). Hallucinating idt descriptors and i3d optical flow features for action recognition with cnns. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 8697–8707 (cit. on p. 7).

Wang, Q., Peng, J., Shi, S., Liu, T., He, J., & Weng, R. (2021). Iip-transformer: Intra-inter-part transformer for skeleton-based action recognition. *ArXiv*, *abs/2110.13385* (cit. on p. 8).

Wei, S., Song, Y., & Zhang, Y. (2017). Human skeleton tree recurrent neural network with joint relative motion feature for skeleton based action recognition. *2017 IEEE International Conference on Image Processing (ICIP)*, 91–95 (cit. on p. 7).

Xu, F., Xu, F., Xie, J., Pun, C.-M., Lu, H., & Gao, H. (2021). Action recognition framework in traffic scene for autonomous driving system. *IEEE Transactions on Intelligent Transportation Systems*, 1–11 (cit. on p. 9).

Yan, S., Xiong, Y., & Lin, D. (2018). *Spatial temporal graph convolutional networks for skeleton-based action recognition* (Research work). The Chinese University of Hong Kong. (Cit. on pp. 8, 32).

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). *Recurrent neural network regularization* (Research work). New York University and Google Brain. (Cit. on p. 7).

Zhang, P., Lan, C., Xing, J., Zeng, W., Xue, J., & Zheng, N. (2017). View adaptive recurrent neural networks for high performance human action recognition from skeleton data. *2017 IEEE International Conference on Computer Vision (ICCV)*, 2136–2145 (cit. on p. 7).

Ziaeefard, M., & Bergevin, R. (2015). Semantic human activity recognition: A literature review. *Pattern Recognit.*, *48*, 2329–2345 (cit. on p. 9).

Zirebwa, F., Kapenzi, Makuvaro, V., Sammie, B., & Madanzi, T. (2015). An evaluation of the performances and the subsequent calibration of three solar radiation estimation models for semi-arid climates in midlands zimbabwe. *Midlands State University Journal of Science, Agriculture and Technology*, *5* (cit. on p. 42).