



Universidade do Minho
Escola de Engenharia

Filipe Barbosa Fernandes

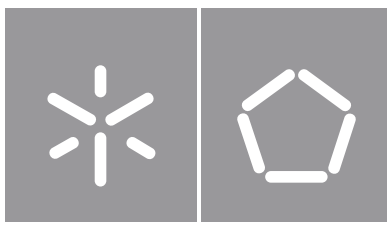
Skeleton-based Action Recognition in Industrial Settings

**Skeleton-based action recognition
in Industrial Settings**

Filipe Fernandes

UMinho | 2022

março de 2022



Universidade do Minho

Escola de Engenharia

Filipe Barbosa Fernandes

Skeleton-based Action Recognition in Industrial Settings

Dissertação de Mestrado

Engenharia Eletrónica Industrial e Computadores

Controlo, Automação e Robótica

Trabalho efetuado sob a orientação do

Professor Luís Filipe Castro Freitas Louro

Doutor Carlos André de Oliveira Faria

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



**Atribuição-NãoComercial-SemDerivações
CC BY-NC-ND**

<https://creativecommons.org/licenses/by-nc-nd/4.0/>

Acknowledgments

This dissertation marks the end of my cycle of studies concerning the master's degree and I would like to use this opportunity to thank and show my appreciation for everyone who accompanied and encouraged me during this journey.

First of all, I would like to thank my supervisor Professor Luís Louro and Doctor Carlos Faria for all the advice, support and guidance throughout this year.

I would like to express a very special thanks to Professor Estela Bicho and Professor Sérgio Monteiro for being present in the meetings and helping with the important decisions that dictated the direction of this dissertation.

A huge thanks to Adriano Pinto from DTx for assisting me with all the machine learning insight and pointing in the right direction to solve most of the deep learning implementation problems.

To all my dearest and closest friends, a special thanks for helping me with all the struggles throughout these five years, but especially to João Cunha, João Pereira and Duarte Lima that were there every time I asked for counselling during the development of this masters dissertation.

And last, but not least to my parents, brother and godfathers, a big thanks for making this academic journey possible, for supporting me every time I needed them and for all their care and attention.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Resumo

Reconhecimento de ações em ambiente industrial com base no esqueleto humano

Atualmente, com a evolução da indústria, surgiram os robôs colaborativos, que são mais pequenos, seguros e capazes de partilhar o espaço de trabalho com operadores humanos, no entanto a interação humano-robô é limitada.

Esta dissertação propõe um sistema de fácil integração num robô colaborativo capaz de fazer o robô identificar, com baixa latência, quase em tempo real, que ações o humano está a desempenhar. A implementação de tal sistema considera um sistema de visão, o qual está orientado para o humano, aliado a um modelo de *deep learning* capaz de identificar com precisão, no menor tempo possível, a ação executada pelo ser humano. Foi usada uma câmara *Kinect v2* da Microsoft, aliada ao *Openpose*, que é capaz de estimar a posição do esqueleto humano em tempo real.

A utilização de um sistema capaz de extrair informação do esqueleto humano em tempo real em conjunto com o algoritmo de *deep learning*, permite extrair características de alto nível modeladas para o problema de deteção do movimento humano, enquanto é mantida uma classificação muito precisa num curto espaço de tempo.

Recentemente, a classificação da pose humana tem sido alvo de bastante investigação e aplicação. Contudo, a disponibilidade dos dados é escassa, especialmente em ambiente industrial. Para tal, no contexto desta proposta, foi necessário criar um conjunto de dados baseado numa célula de trabalho para treinar a rede neural.

As principais contribuições desta dissertação são: uma lista de vários métodos do atual estado da arte para o reconhecimento das ações com base no esqueleto humano para a deteção offline e online, um dataset baseado numa célula de trabalho para o reconhecimento das ações em ambiente industrial, um visualizador de esqueleto 3D para melhor compreensão do movimento do esqueleto e uma rede neuronal treinada para o reconhecimento das ações que classifica a ação em menos de 200 milissegundos, com alta precisão.

Palavras-chave: Esqueleto Humano; Reconhecimento de Ações; Robótica Colaborativa; Ambiente Industrial

Abstract

Skeleton-based Action Recognition in Industrial Settings

Nowadays, with the evolution of industry, collaborative robots emerged, which are smaller, safer and able to share the workspace with human operators, however the human-robot interaction is limited.

This dissertation proposes an easily integrated system in a collaborative robot capable of making the robot perceive, with low latency, which actions the human is performing.

The implementation of such a system considers a vision system, which is oriented towards the human, powered with a *deep learning* model capable of accurately identifying, in the shortest possible time, the action performed by the human. In this dissertation a Kinect v2 camera from Microsoft was used, coupled with *Openpose*, which provides an estimation of the human skeleton pose in real-time.

The use of a system capable of extracting information from the human skeleton in real time in conjunction with the *deep learning* algorithm, allows the extracting of high-level features of human movement. This discriminative information allows a highly accurate classification, robust throughout a short period of time.

Recently, human pose classification has been the subject of research and application. However, data availability is scarce, especially in industrial environment. Therefore, in the context of this proposal, it was necessary to create a dataset based on a work-cell to train the neural network.

The main contributions of this dissertation are: a comprehensive list of several state-of-the-art methods for the skeleton-based human action recognition problem for offline and online detection, a dataset settled on a work-cell for action recognition in industrial environment, a 3D skeleton viewer for better understanding of the skeleton motion, and a neural network trained for action recognition that classifies the action in less than 200 milliseconds with high accuracy.

Keywords: Skeleton-based, Action Recognition; Collaborative Robotics; Industrial Environment

Contents

1	Introduction	16
1.1	Contextualization	16
1.2	Motivation and Challenges	17
1.2.1	Examples of Real-World Applications	17
1.2.2	Research Challenges	19
1.3	Objectives	20
1.4	Dissertation Structure	21
2	Theoretical Foundations	22
2.1	Deep Neural Networks	22
2.2	Convolutional Neural Networks	24
2.2.1	Convolutional Layer	26
2.2.2	Pooling Layer	27
2.2.3	Fully-connected Layer	28
2.3	Convolutional Neural Network Architectures	28
3	State of the Art	35
3.1	Action Representation	35
3.1.1	Global Representations	35
3.1.2	Local Representations	36
3.1.3	Depth-Based Representations	37
3.2	Related Work	38
3.3	Datasets for Human Action Recognition	45
4	Design and Implementation	50
4.1	Data Acquisition	50

4.1.1	Hardware	50
4.1.2	Software	52
4.1.3	Implementation	52
4.2	Data Preparation for Training	55
4.2.1	Dataset	55
4.2.2	Segmentation	56
4.2.3	3D Skeleton Visualizer	56
4.3	Deep Learning Model	58
4.3.1	VA-CNN	59
4.3.2	Fine-tuning the Model	63
4.4	Online Classification	64
5	Tests and Results	67
5.1	Sliding Window Size Tests	67
5.2	Different Training Methods	69
5.3	Online Action Recognition	70
5.4	Action Recognition from Live Data Stream	76
6	Conclusion	78
6.1	Discussion	78
6.2	Future Work	79
	Bibliography	80
	Appendices	85
A	Activation Functions	86
B	NTU-RGBD 120 Actions	87
C	Kinect v2 ROS Topic List	88
D	Frame ROS Topic from Ros_Openpose	89
E	Final Model VA-CNN	94

List of Abbreviations

2D Two-Dimensional.

3D Three-Dimensional.

CNN Convolutional Neural Networks.

DMM Depth Motion Maps.

DNN Deep Neural Networks.

FPS Frames Per Second.

ILSVRC ImageNet Large Scale Video Recognition Competition.

IR Infrared.

LSTM Long short-term memory.

ReLU Rectified Linear Units.

ROS Robot Operating System.

SDK Software Development Kit.

ToF Time-of-Flight.

VA View Adaptation.

List of Figures

1.1	Visual Surveillance by Tracking Human Attention (Benfold and Reid, 2009)	17
2.1	Typical architectures of artificial neural networks (ANN) and deep neural networks (DNN), (Aslam et al., 2019).	23
2.2	Neuron internal architecture ¹	23
2.3	The receptive field of each convolution layer with a 3×3 kernel. The green area marks the receptive field of one pixel in Layer 2, and the yellow area marks the receptive field of one pixel in Layer 3, (Lin et al., 2017)	24
2.4	Example where a 5×5 local receptive field is connected to a neuron of the hidden layer, (Nielsen, 2015)	25
2.5	Sliding the receptive field over the entire layer, (Nielsen, 2015)	26
2.6	Convolutional layer with 3 filters and a local receptive field of 5, (Nielsen, 2015)	26
2.7	2×2 max-pooling layer, (Nielsen, 2015)	27
2.8	Dimensions of the max-pooling layer, (Nielsen, 2015)	28
2.9	Architecture of LeNet-5, a Convolutional Neural Network, for digits recognition, (LeCun et al., 1998)	29
2.10	Architecture of <i>AlexNet</i> , (Tsang, 2018)	30
2.11	<i>VGG</i> multiple configurations developed by Simonyan and Zisserman (2015)	31
2.12	Inception module with dimensionality reduction, (Szegedy et al., 2015)	32
2.13	A simplified illustration of the <i>GoogLeNet</i> architecture, (Shin et al., 2016)	33
2.14	Multiple architectures of <i>ResNet</i> developed for ImageNet, (He et al., 2016)	33
2.15	Residual learning (residual block), (He et al., 2016)	34
3.1	(a) Reference coordinates of HOJ3D. (b) Modified spherical coordinate system for joint location. (Xia et al., 2012)	38
3.2	An illustrative sketch of the hierarchical recurrent neural network, (Du et al., 2015)	40

3.3	Example of online detection, where the action threshold $\theta_c = 7$, (Meshry et al., 2016).	42
3.4	One-dimensional YOLO network, (Wu et al., 2019).	43
3.5	Skeleton with 31 joints, where the red joint is the origin coordinate (0, 0, 0), (Carrara et al., 2019).	44
3.6	Multi-label annotation example, (Carrara et al., 2019).	44
3.7	Illustration of the 51 Actions (part 1)	46
3.8	Illustration of the 51 Actions (part 2)	46
3.9	ActivityNet hierarchical structure, (Heilbron et al., 2015)	47
3.10	Something Something dataset, (Goyal et al., 2017).	48
4.1	Microsoft Kinect v2 ¹ and it's internal sensors ²	51
4.2	Image before calibration	53
4.3	Image after calibration	53
4.4	<i>Openpose</i> real-time skeleton tracking	54
4.5	Static action representing the start of the task, action[0]	55
4.6	Frame from the second action, action[1]	55
4.7	Frame from the third action, action[2]	55
4.8	Frame from the fourth action, action[3]	55
4.9	Frame from the fifth action, action[4]	56
4.10	Static action representing the end of the task, action[5]	56
4.11	3D skeleton visualizer on our dataset data, action number three	57
4.12	Camera space coordinates from the Kinect SDK, (Jamhoury, 2018)	58
4.13	Camera space coordinates from the <i>Ros_Openpose</i>	58
4.14	Architecture of the proposed VA-fusion scheme, (Zhang et al., 2019)	59
4.15	Flowchart of the end-to-end view adaptive neural network, (Zhang et al., 2019)	60
4.16	Illustration of the change of the observation viewpoint, by assuming there is a movable virtual camera, (Zhang et al., 2019)	60
4.17	Input image to be fed into the VA-CNN	65
4.18	First stride of the sliding window	65
4.19	Second stride of the sliding window	66
5.1	Validation accuracy (%) curve during the training of the neural network	70
5.2	Classification of the static action representing the start of the task, action[0]	75

5.3	Classification of the example of the second action, action[1]	75
5.4	Classification of the example of the third action, action[2]	75
5.5	Classification of the example of the fourth action, action[3]	75
5.6	Classification of the example of the fifth action, action[4]	76
5.7	Classification of the static action representing the end of the task, action[5]	76
5.8	Scheme of out final solution for online action recognition with real-time data input	77
A.1	Activation Functions, (Sharma, 2017)	86
B.1	Daily actions	87
B.2	Medical conditions	87
B.3	Mutual actions	87

List of Tables

4.1	Microsoft Kinect v2 main specifications	51
4.2	Correlation between classification index and action labels	56
4.3	Accuracy (%) comparisons on different number of layers and models. For reference: CS , cross-subject where the 40 subjects are split into the training and testing groups; and CV , cross-view where the samples of cameras 2 and 3 are used for training and those of camera 1 for testing, these are the two standard evaluation methods for the NTU RGB+D dataset	59
5.1	Accuracy (%) values for the NTU RGB+D dataset with only 20 frames being analysed for each action video	68
5.2	Accuracy (%) values for the NTU RGB+D dataset with only 40 frames being analysed for each action video	68
5.3	Accuracy (%) values for the NTU RGB+D dataset with only 60 frames being analysed for each action video	68
5.4	Accuracy (%) of the neural network on our dataset after fine-tuning each available pre-train	69
5.5	Accuracy (%) comparisons on different number of convolutional layers on the ResNet on our dataset	69
5.6	Number of errors detected on action classification for the tested video trials	74
5.7	Average error rate of each action on the eight trials	75
5.8	Total composition of our dataset (number of samples)	76
5.9	Error rate (%) of the final tests to the solution	77

Code Snippets

- 4.1 Frame topic subscriber 54
- 4.2 Adapting the model for the train with our dataset 63
- 5.1 Unfiltered output of the neural network on online classification over a video where the complete assembly task is being performed, the first number represents the action label recognized by the neural network and the "tensor" number represents its confidence score percentage 71
- 5.2 Filtered classification output for the video trial 1. In this trial, the system detected every action with an impressive confidence score, over 90%, only action '3' and action '5' being under that mark 72
- 5.3 Filtered classification output for the video trial 2. In this trial, action '0' and action '2' were detected with lower confidence than the threshold, so in this trial, there are two errors 72
- 5.4 Filtered classification output for the video trial 3. Some of the actions were detected multiple times since the task was performed slower in this trial. The action '1', at the beginning and end, is not counted as an error, the only error is the detection of action '2' which had a lower confidence score than the threshold 72
- 5.5 Filtered classification output for the video trial 4. There are two errors in this trial since that action '2' and action '3' were detected with lower confidence scores than the threshold 73
- 5.6 Filtered classification output for the video trial 5. In this trial, action '2' was not detected even in the raw output of the network, which makes this trial only have one error 73
- 5.7 Filtered classification output for the video trial 6. In this trial, action '2' was detected in the raw classification output, but with a lower confidence score than the threshold . . . 73
- 5.8 Filtered classification output for the video trial 7. In this trial, action '0' was detected in the raw classification output, but with a lower confidence score than the threshold . . . 73

5.9	Filtered classification output for the video trial 8. In this trial, the system detected every action with an impressive confidence score, over the threshold. Action '3' in the beginning is not considered an error	74
C.1	Kinect v2 ROS Topic list	88
D.1	Message displayed by the Frame topic from Ros_Openpose, where the "pixel" corresponds to the 2D coordinates and the "point" corresponds to the 3D coordinates on the Ros_Openpose referential located on the Kinect v2 sensor, of each 25 skeleton joints . . .	89
E.1	Final architecture layout of the model VA-CNN used	94

Chapter 1

Introduction

1.1 Contextualization

Nowadays, in industry 5.0, technologies such as collaborative robotics are more prevalent in industrial processes. However, in the majority of the existent hybrid work cells, human-robot interaction is limited and structured. In order to extend and help the human-robot cooperation, the robotic system must perceive in real-time what actions the human is performing. Therefore, artificial vision plays an important role since it can give the robotic system visual information of the surrounding environment.

Human action recognition aims to analyze human motions and distinguish various kinds of human actions. Due to the number of applications for which it is useful, such as surveillance, healthcare, human-computer interaction and robotics, it is a field in constant expansion and progression. Traditionally, human action recognition is performed based on RGB imaging or depth image sequences. However, compared with skeleton data, which is a topology of representation for the human body with joints and bones, those previous modalities are more computationally demanding and less robust when facing complicated backgrounds, different body scales, viewpoints or motion speeds. Notwithstanding, nowadays technologies like the *Microsoft Kinect* (Zhang, 2012) and human pose estimation algorithms offer much easier and reliable ways to obtain 3D skeleton data.

The use of deep learning algorithms has facilitated human action recognition tasks, since they are able to learn high-level features from the raw input data by themselves. However, these algorithms are computationally expensive and slow for applications in industrial context, that require fast response times. Most of these deep learning algorithms also require segmented labelled data since they are not able to detect actions from unsegmented data streams, so it is necessary to develop an algorithm with the capability to detect and recognize actions from unsegmented data in real-time. Moreover, datasets available

online are mostly generic or inappropriate for this dissertation case study, since it's industry 5.0 orientated, a dataset of a specific work cell has to be created.

Focusing on these premises, it becomes imperative to develop a computationally inexpensive and fast action detection algorithms that can be readily utilized in industrial contexts.

1.2 Motivation and Challenges

In a world eager to shift industry to a more efficient and sophisticated industry 5.0, creating hybrid work cells, where cooperation exists between human and robot, is one of the must-have characteristics. Human action recognition is one of the most coherent and systematic methods to make a robotic system comprehend what actions and movements the human is performing, like mentioned in subsection 1.1, human tracking and action recognition, can be applied in various real-world scenarios.

1.2.1 Examples of Real-World Applications

Visual Surveillance

Security is a problem that will always be present in daily life, action recognition for surveillance is one of the most frequently discussed topics. It is possible to create a network of cameras, in certain places where typically certain human actions are allowed and some are not allowed, with a visual surveillance system powered by action recognition and predictions algorithms to detect and thus reduce the risk of criminal actions.



Figure 1.1: Visual Surveillance by Tracking Human Attention (Benfold and Reid, 2009)

Video Retrieval

Nowadays, people can easily upload and share videos on the Internet. Searching videos according to their content is becoming a huge challenge as most search engines use the associated text data to classify the video, however, most of the times text data such as titles, descriptions and keywords (hashtags), may be incorrect or irrelevant. Action recognition algorithms can classify and automatically label those videos for a better search response.

Entertainment

In recent years, a new generation of games based on full-body tracking, such as dance or sports games, have been released. To enhance full-body perception, these games use RGB-D sensors, such as Kinect (Zhang, 2012), which provide an additional depth channel with rich structural information of the entire scene and thus facilitate full-body recognition task, since depth data simplifies intra-class variations and reduce cluttered background noises.

Human-Robot Interaction

Human-robot interaction technologies are commonly applied in healthcare and industrial environments, in order to assist humans and minimize negative long-term effects of some tasks on their health. For this interaction to work, robotic systems must understand human actions and then perform the best movements to assist the human. Action recognition can be used for multiple purposes, for example, in industrial scenarios certain assembly processes require steps such as operating in places that are hard to reach for a human, lifting heavy objects or precise positioning of objects above the human shoulders. Interactions require communication between robots and humans, and visual communication is one of the most efficient ways to communicate.

Autonomous Driving Vehicle

Action prediction algorithms may be one of the most important building components in an autonomous driving vehicle. In an urgent situation, a vehicle equipped with an action prediction algorithm will potentially predict a pedestrian intention, for example, stop at the edge of the street or running across the street, in a short period and stop the car in case of need, avoiding collision and saving the pedestrian life.

1.2.2 Research Challenges

In this subsection are presented some of the most demanding research challenges concerning human action recognition, either visual challenges or data filtering challenges. In unstructured environments (e.g. airports, streets, auditoriums, etc.) the main challenges are associated with poor camera quality and difficulty positioning the cameras. However, in controlled contexts, such as industry (where the hardware and its position have been previously studied) these challenges are overcome. In the case of this project, related to a manufacturing task, since it has many similar actions, other challenges related to intra-class and inter-class variations become more relevant.

Intra-class and Inter-class Variations

People behave differently for the same actions, simply due to anthropometric differences or because the action was performed slower or faster, for example, a simple walking movement can differ in speed and stride length. Other variations in classes are the viewpoints, the same action can be recorded with the camera facing the human subject in front, on the side of the subject, or even on top of the subject, resulting in the same action but showing appearance variations in each viewpoint. All these factors result in intra-class variations, which can induce most of the existent action recognition algorithms in error and consequently lead to misclassification. On the other hand, different activities may express similar showing appearance but are completely different actions, for example, using a laptop and reading a book, this is referred to as inter-class variations. In the industry, depending on the manufacturing task, this problem can be a really hard challenge to overcome, for example, due to high similarity in arm movements when the person's duty is focused on manual assembling.

Different Backgrounds and Cluttered Background

Some human action recognition algorithms work very well in indoor controlled environments but when tested against outdoor uncontrolled environments, they fail most of the times, mainly due to background noise. Various factors, such as, the environment in which the action performance takes place, the person localization in cluttered environments and lighting conditions can also lead to misclassification.

By using a depth sensor, some of these environmental factors can be overridden, the depth sensor is insensitive to changes in lighting conditions and different backgrounds, however, the problem of obstructions can still arise.

Long Distance and Low Quality Videos

Most scenarios of video surveillance deal with long-distance, low-quality videos and severe occlusions. Normally, surveillance cameras are installed in high places and cannot provide high definition videos where the subject representation is clear. Some examples of these problems are long-distance cameras in football fields or crowded airport terminals.

This is a real-world challenge, however, with human-robot interaction in an industry context, the environment is controlled. Also, with the use of Kinect, video quality is known and distance is variable within predefined limits for the depth sensor, from 50 centimetres, up to 4.5 meters.

1.3 Objectives

The main objective of this project is to develop an intelligent vision system that is able to classify human actions presented in a video live stream, in run-time. The human actions are all related to an assembly task based on a real work-cell of a local industry. It will rely on the use of a Microsoft Kinect v2 RGB-D camera paired with a machine learning algorithm. The problem was divided in the following tasks:

- Study of several state of the art methods of Skeleton-based Human Action Recognition, to find the most suitable neural network to use;
- Create a skeleton-based dataset settled on an existent work cell;
- Adapt and train a fast and computationally inexpensive machine learning algorithm;
- Test the designed solution and calibrate the parameters to achieve a good precision rate in run-time execution.

To summarise, we intend to have a solution capable to perform action detection and recognition based on live footage from the Kinect v2 sensor.

1.4 Dissertation Structure

According to the objectives presented above, this dissertation is divided into six main parts.

Chapter 1. Introduction introduces the genesis and objectives of this dissertation, as well the main challenges in this research field. **Chapter 2. Theoretical Foundations** aims to introduce a theoretical background of the important knowledge necessary to develop this dissertation. Afterwards, **Chapter 3. State of the art** provides a literature review on action recognition and its different approaches, from the early works to the most recent ones, a literature review on the state of the art models on skeleton-based action recognition and lastly, the most used datasets to train and test the presented models. **Chapter 4. Design and Implementation** presents the hardware and software used and the entire process of developing this solution. Consequently, **Chapter 5. Tests and Results** presents and reviews the work associated with testing the proposed solution and its achievements. **Chapter 6. Conclusion** discusses the main achievements and key conclusions of the dissertation, along with a proposition of future work.

Chapter 2

Theoretical Foundations

Artificial Neural Networks (ANNs) were introduced as a learning algorithm in 1948 by Donald Hebb (Hebb, 1949). Their name and structure were inspired by the human brain, with the motivation of mimicking the way how biological neurons send signals to one another, and allow the development of computer programs to find and recognize patterns.

ANNs are a collection of nodes grouped in three different kinds of layers, an input layer, hidden layers (one or more) and an output layer. Each node or artificial neuron is connected to another node and has associated weights and activation function. If the output of any single node is higher than the specified value, the node is activated, and the data is passed to the next layer of the network. Otherwise, no data will be passed to the next layer of the network. Neural networks rely on training data to learn and improve their accuracy over time.

2.1 Deep Neural Networks

Deep neural networks (DNNs) are an alternative version of the traditional neural network, called Artificial Neural Network. Figure 2.1 shows the difference between traditional simple artificial neural network (ANN) and deep neural network. ANN consists of one or two hidden layers to process data, while DNN includes multiple layers between the input layer and the output layer. This is the main difference between the two neural networks. DNN uses mathematical operations to produce an output, based on a given input, regardless of whether the input data relationship is linear or non-linear (Aslam et al., 2019). The neural network learns with resorts to a training set, which in turns allows the network to compute an error at the end of each iteration (forward pass) and then optimize the weight of the network (backward pass) in order to decrease the magnitude of the error.

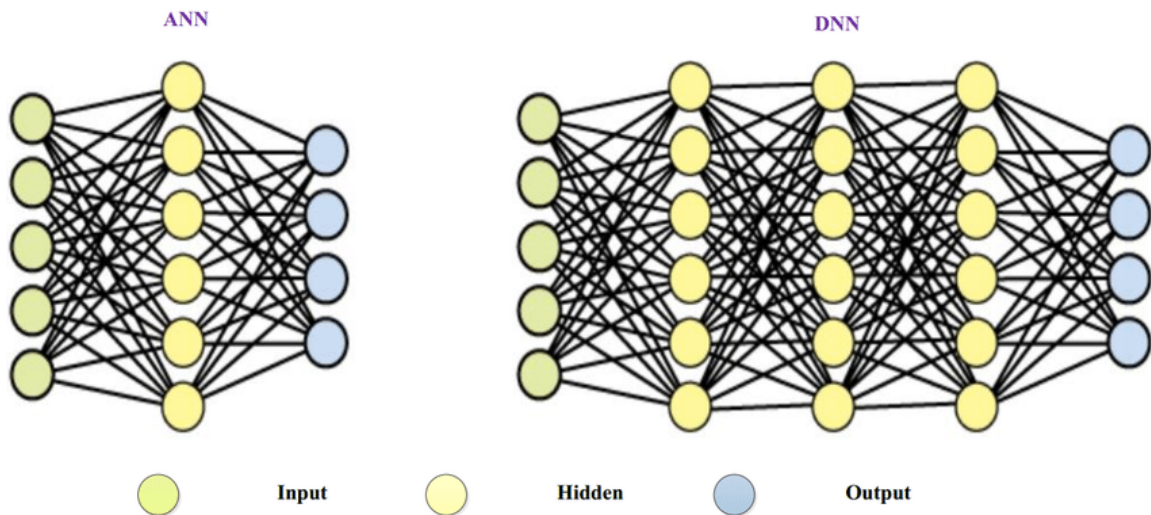


Figure 2.1: Typical architectures of artificial neural networks (ANN) and deep neural networks (DNN), (Aslam et al., 2019).

ANNs and DNNs are composed of multiple neurons, also called nodes (represented in Figure 2.1 with circles), a closer look at a neuron architecture is illustrated in Figure 2.2. A neuron from a layer is connected to all the neurons of the next layer. These connections are weighted and establish the relationship among the different extracted features. This particular arrangement is designated Fully Connected Layers (FCNs). Each neuron receives input data and performs a simple operation on the data. The result of this operation is passed to all other neurons in the next layer, this output of each neuron is called activation or node value and is calculated by an activation function (see appendix A for activation function examples). Each link is associated with a weight, to make a neural network learn we have to alter the weight values.

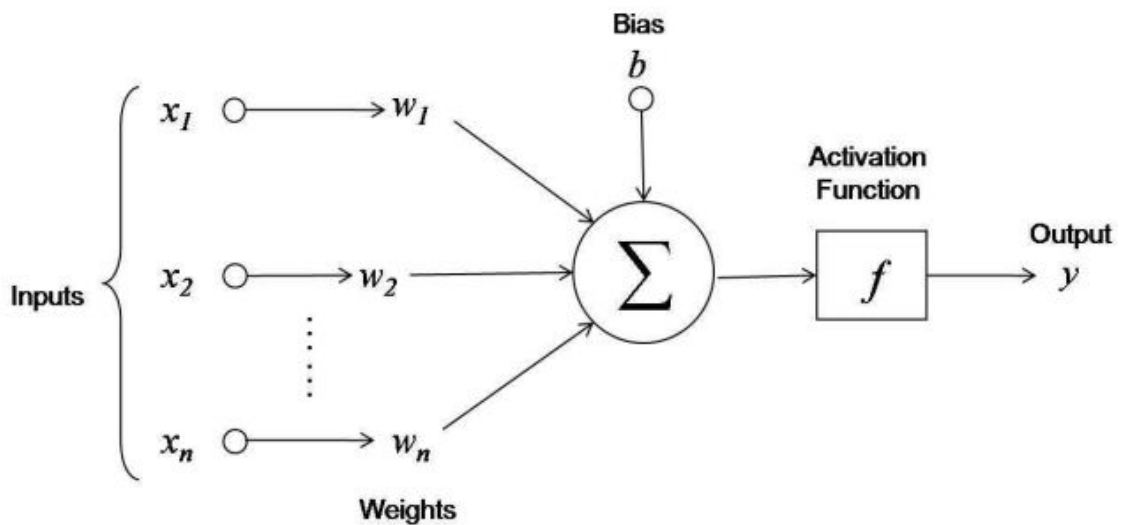


Figure 2.2: Neuron internal architecture¹

For an image classification scenario, where for computers, an image is a multidimensional array of values, and each value represents a pixel intensity, to train a neural network capable of performing this classification, each input neuron should hold to the value of a corresponding pixel, in a possible framework where the dataset contains a set of 100 by 100 pixel images and the pixel are in greyscale, with a value of 0.0 representing white, a value of 1.0 representing black, and in-between values representing gradually darkening shades of grey, after normalization, this neural network would require 10000 input neurons. Apart from requiring a node for each pixel in the image, which demands a large set of parameters, the intrinsic nature of the data is also lost, with no ability to analyse the pixel context when extracting new features.

2.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is the name given to a neural network that has convolution layers at their core. Their accomplishments exposed some drawbacks of the conventional deep neural networks. CNNs are more robust and automatic for image classification, and the reason behind it is that DNNs do not take into account the spatial structure of images. They process input pixels that are far apart and close together on the same basis, through a common structure designated kernel. CNNs take advantage of a local correlation between neighbour pixels, but such spatial structure concepts must be inferred from training data (Nielsen, 2015).

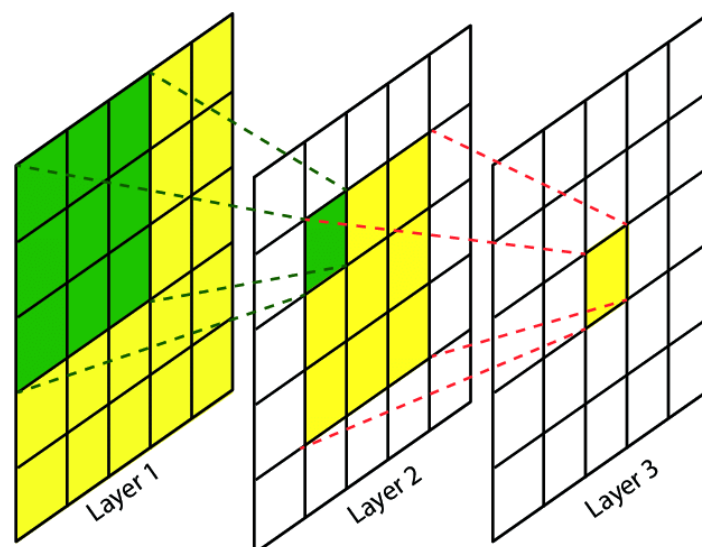


Figure 2.3: The receptive field of each convolution layer with a 3×3 kernel. The green area marks the receptive field of one pixel in Layer 2, and the yellow area marks the receptive field of one pixel in Layer 3, (Lin et al., 2017)

¹Image from: <https://github.com/snives/SimpleNeuralNetwork>, Accessed: 17 November 2021

These convolutional networks use a particular method, by the name of Receptive Field, represented in Figure 2.3, that makes them exceptionally well suited for image classification and are fast to train. Today, convolutional neural networks are the most used architecture for image recognition.

CNNs rely on two fundamental ideas: local receptive fields and shared weights. A local receptive field is a group of neurons that are connected to a single neuron of the next layer, represented in Figure 2.4, while in conventional deep neural networks, every neuron is connected to all the subsequent layer neurons. Each connection between the neurons in the receptive field and the hidden neuron learns a weight and the hidden neuron from the next layer learns an overall bias (Nielsen, 2015). Hence each hidden neuron will only analyse its own particular local receptive field.

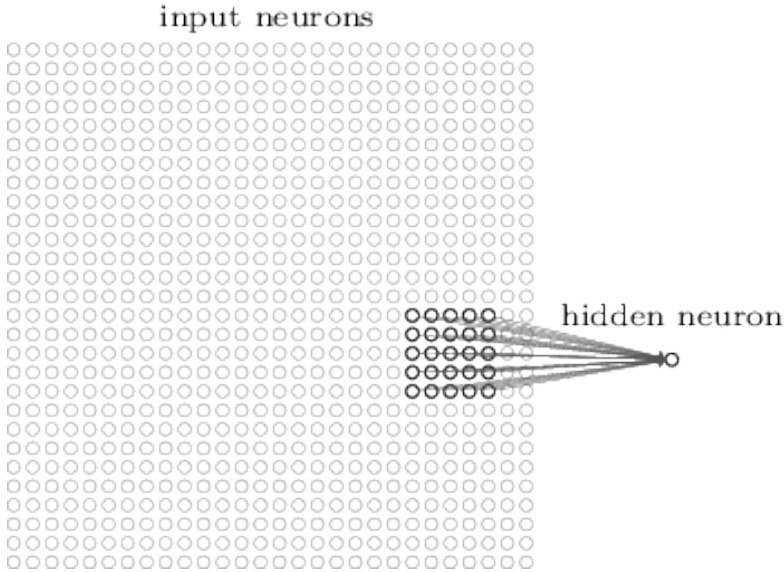


Figure 2.4: Example where a 5 x 5 local receptive field is connected to a neuron of the hidden layer, (Nielsen, 2015)

The local receptive field is then slid over the entire input layer, and for each local receptive field, its destined neuron in the next layer will store its bias. To that new layer of bias about the previous layer receptive fields, we call it a feature map.

These local connections allow the extraction of features in an area of a feature map, moreover LeCun et al. (2015) affirms that the features of an image are invariant to its location. Therefore the feature can appear on any side of the image, and that is where the idea of sharing weights in the same feature map is helpful. It allows the detection of the same pattern in any other location in the image.

This means that all the neurons in the first hidden layer detect exactly the same feature, just at different locations in the input image.

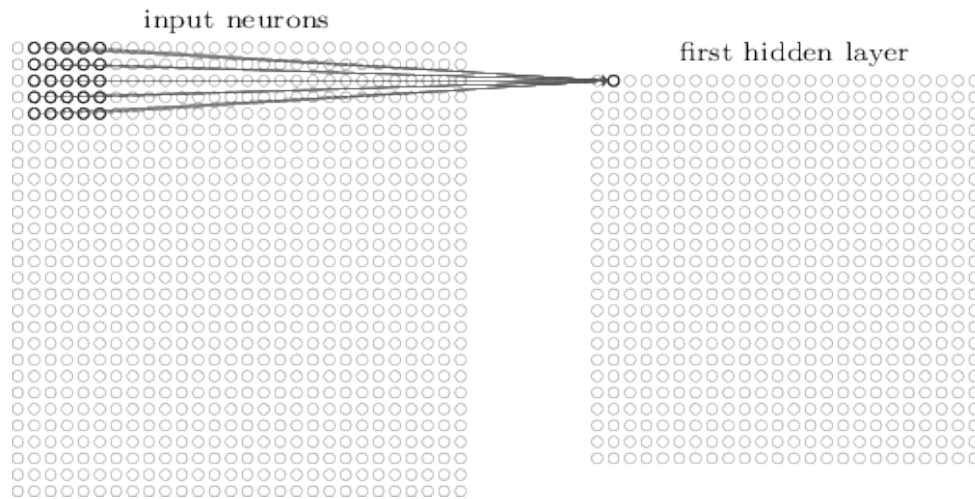


Figure 2.5: Sliding the receptive field over the entire layer, (Nielsen, 2015)

2.2.1 Convolutional Layer

A convolutional layer is an operator that can receive a large space of features and transform it to a larger or smaller stack of feature maps, as the application of different filters results on various feature maps, that allow the extraction of multiple features at each location (Nielsen, 2015). The depth of a convolutional layer is equal to the number of filters applied, for example, if three filters were applied in the representation of Figure 2.6, it would result in 3 feature maps, and consequently, the total volume would be $3 \times 24 \times 24$. Each of these maps is defined by a set of 5×5 shared weights, which is the size of the used local receptive field, and a single shared bias. The result is that this hypothetical network can detect three different kinds of features, with each feature being detectable across the entire image.

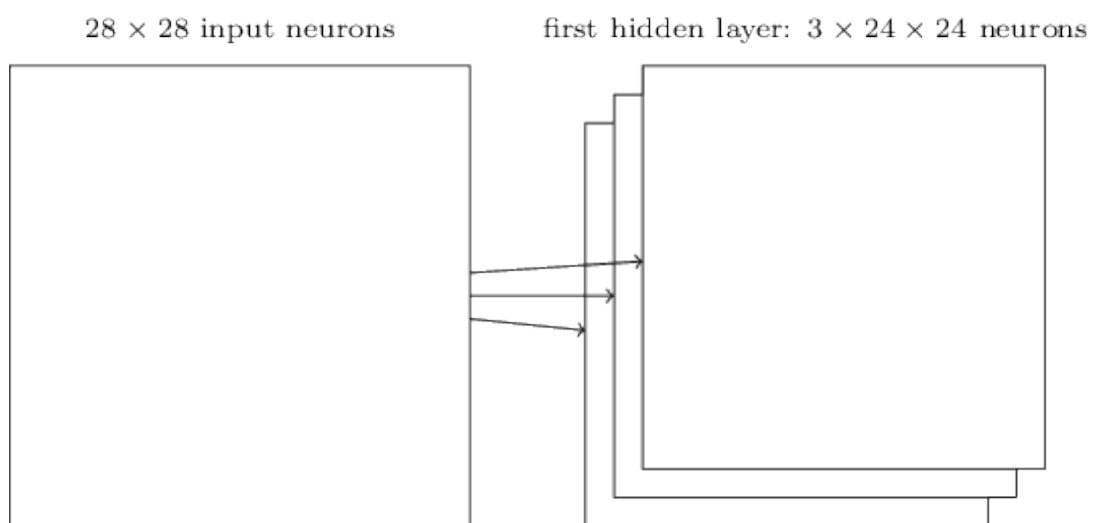


Figure 2.6: Convolutional layer with 3 filters and a local receptive field of 5, (Nielsen, 2015)

For example, the LeNet-5, one of the first convolutional neural networks, used 6 feature maps with a

5x5 local receptive field to recognize MNIST digits, (LeCun et al., 1998; Lecun Yann et al., 1998).

2.2.2 Pooling Layer

Pooling layers are generally used after a convolutional layer, and its purpose is to dynamically diminish the spatial size of the representation, to reduce the number of parameters and calculation in the neural network, (Mostafa et al., 2020). It operates on each feature map independently.

Besides the reduction in the number of parameters of subsequent convolutional layers, the main advantage of pooling is allowing a broader analysis of information. In image analysis this translates to a larger context of analysis. Where pooling mainly helps in the extraction of sharp and smooth features, max-pooling for the low-level features such as edges and points and average-pooling for smooth features.

The most commonly used pooling layer is known as max-pooling, which computes the maximum value of the nodes, it takes the output of the convolutional layer and prepares a more condensed feature map.

For example, if we take a 24 x 24 feature map from the previous Figure 2.6, and pass it in a 2 x 2 max-pooling layer, it would result in a 12 x 12 neurons condensed layer, represented in Figure 2.7, where each new neuron on the 12 x 12 layer simply represents the maximum activation in the 2 x 2 input region.

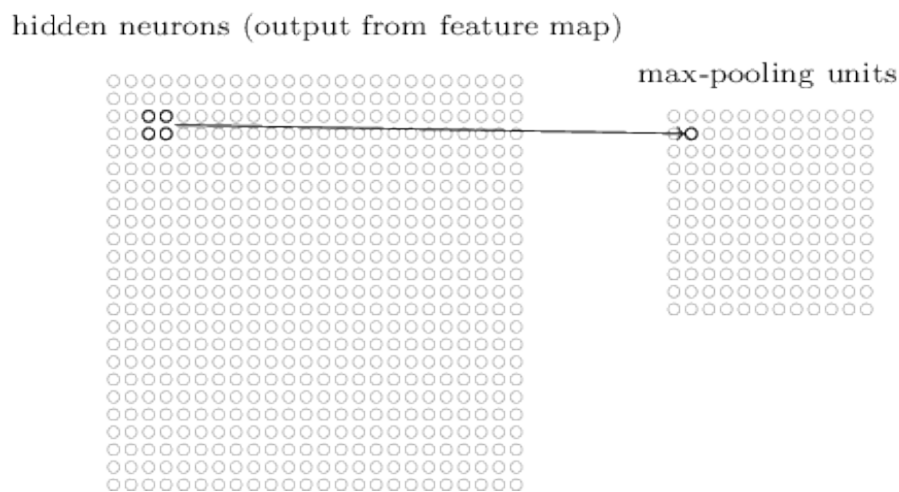


Figure 2.7: 2 x 2 max-pooling layer, (Nielsen, 2015)

Since the pooling layer operates on each feature map of the convolutional layer, the pooling layer will result in stacked feature maps with the same depth of the convolutional layer but with half of its 2D size. Hence, on this presented example where the convolutional layer has a dimension of 3x24x24, the resulting max-pooling layer will have a 3x12x12 dimension, as illustrated in Figure 2.8

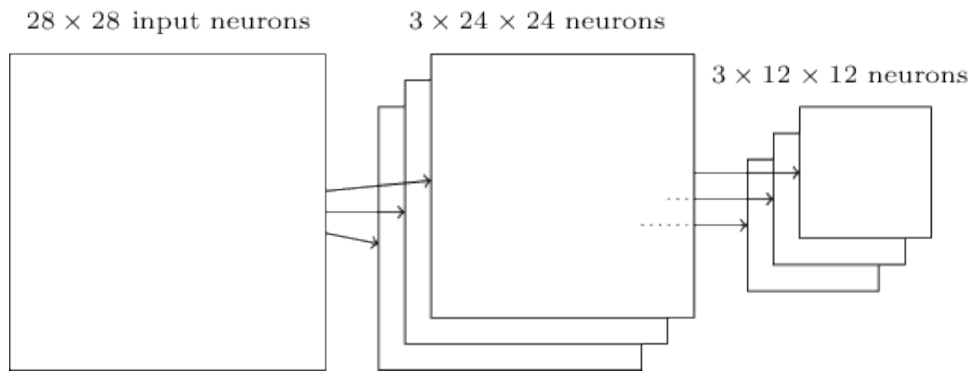


Figure 2.8: Dimensions of the max-pooling layer, (Nielsen, 2015)

2.2.3 Fully-connected Layer

Once the spatial features are extracted, that is, after the input image passes through a set of convolutional and pooling layers, a fully connected layer aims to combine all the input feature maps, in order to output a feature map that represents each pre-defined class or group (label), (Deshpande, 2016). This is the similarity between CNN and DNN, where each pixel is treated as a separate neuron, and each neuron is connected to all previous neurons.

The fully connected layer expands all output activation maps into a one-dimensional vector, that is, the size of the first fully connected layer depends on the size of the last feature map (usually a pooling layer) and its corresponding depth. The input vector is connected to a set of hidden layers behind the output layer, which contains the output classes for classification. CNN is a combination of these layers, arranged in a specific way, and the number and order of these layers depend on how it is dimensioned, (O'Shea and Nash, 2015)

2.3 Convolutional Neural Network Architectures

LeNet

LeCun et al. (1998) presented *LeNet* as the first convolutional neural network architecture created, it comprises the input layer and 7 more layers with trainable weights. The input layer is a 32×32 pixel image.

Represented in Figure 2.9 for reference, Layer C1 is a convolutional layer with 6 feature maps with a local receptive field of 5×5 in the input layer, which results in feature maps with a size of 28×28 .

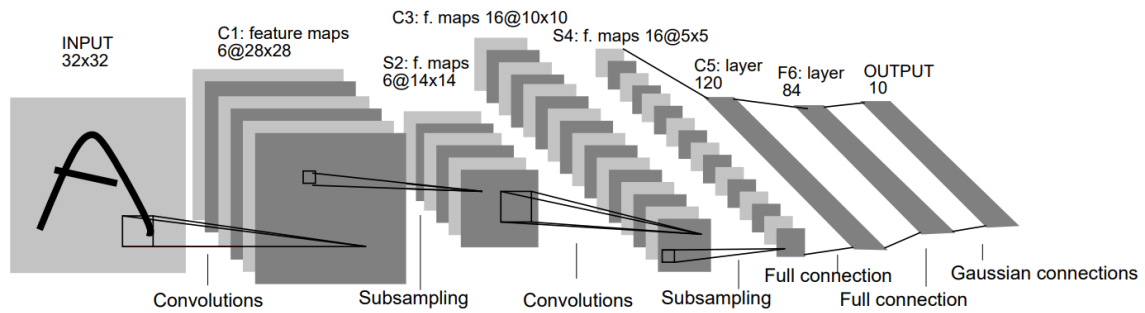


Figure 2.9: Architecture of LeNet-5, a Convolutional Neural Network, for digits recognition, (LeCun et al., 1998)

Layer S2 is a pooling layer, it contains 6 feature maps with a size of 14x14, the same depth of the previous layer, as explained in section 2.2.2, but with half of the size, since the applied 2x2 receptive field is non-overlapping. The four inputs to a neuron in S2 are added, then multiplied by a trainable coefficient, and added to a trainable bias.

Layer C3 is another convolutional layer but made of 16 features maps, the local receptive field is the same as used in C1, 5x5 size, but this time each feature map is connected to several receptive fields, by doing this, LeCun et al. (1998) states that this non-complete connection scheme keeps the number of connections within reasonable bounds and it forces a break of symmetry in the network (different feature maps are forced to extract different features because they get different sets of inputs).

Layer C3 is composed of the following set of feature maps:

- the first six C3 feature maps take inputs from contiguous subsets of three feature maps in S2;
- the next six C3 feature maps take input from every contiguous subset of four feature maps;
- the succeeding three C3 feature maps take input from discontinuous subsets of four feature maps;
- the last C3 feature map takes input from all S2 feature maps.

Layer S4 is a pooling layer with the same settings of S2, using the same 2x2 receptive field in a non-overlapping method, it results in 16 feature maps with half the size of the previous layer.

Layer C5 is a convolutional layer with 120 feature maps with a receptive field of 5x5, which results in size 1x1 feature maps. Since C5 feature maps sizes are 1x1, a full connection is created between S4 and C5, however, it's labelled as a convolution layer because if the input were of bigger size, the dimension of this feature map would be larger than 1x1.

To conclude, the C5 layer is fully connected to the F6 layer which is also fully connected through Gaussian connections to the output layer. The Gaussian connections uses the euclidian radial basis function

to estimate the error between the input pattern and the class associated. This presented sequence of layers build a convolution neural network and these layers are the root of all the recent convolution neural network architectures.

AlexNet

In 2012, Krizhevsky et al. (2012) introduced *AlexNet*, a new convolutional neural network architecture, just like the *LeNet* but deeper, with more layers. This specific network was designed for a contest, the ImageNet Large Scale Video Recognition Competition (ILSVRC) in 2012, where the teams had to develop a neural network capable of classifying a set of 1.2million 256x256 size images in 1000 classes, (Krizhevsky et al., 2012), achieved significantly improved performance over the other non-deep learning methods in the contest.

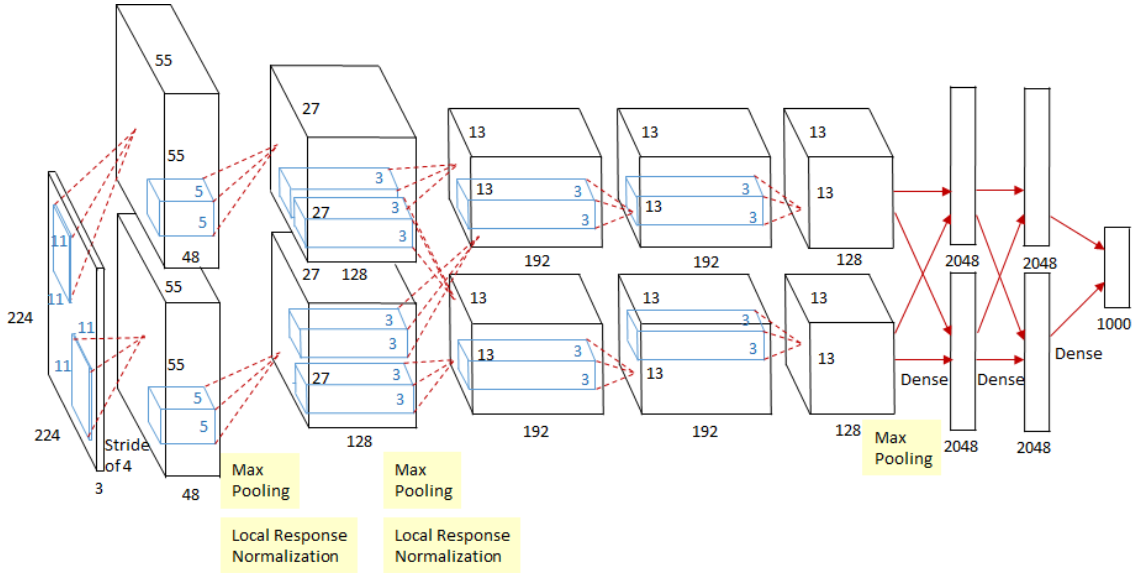


Figure 2.10: Architecture of *AlexNet*, (Tsang, 2018)

AlexNet is composed of five convolutional layers, three pooling layers, and two fully-connected layers with approximately 60 million parameters, and managed to achieve a top-5 error rate of 15.3%, (Deng et al., 2012). This network came to prove that convolutional neural networks can be used to learn complex features on images.

AlexNet, uses Rectified Linear Units (ReLU) instead of the TanH function (see appendix A), which was standard at the time. ReLU’s main advantage is that it reduces the training time considerably since the optimization of the network is improved by avoiding the vanishing gradient that was inherent of Deep Neural Networks. The ReLU layer does not saturate, for $x \gg 0$, which allows the back-propagation of the error with respect to the weights and consequently the tuning of the weights.

A CNN using ReLU was able to reach a 25% error on the CIFAR-10 dataset six times faster than a CNN using TanH. It additionally allows training the network in multiple GPUs systems, by dividing the neurons between the available GPUs. Not only does this mean that bigger models can be trained, but also cuts down on the training time (Tsang, 2018).

VGG

Later, in ILSVRC-2014, VGG by Simonyan and Zisserman (2015), managed to deliver a 7.3% classification top-5 error rate. The proposed convolution neural network proved that deeper networks with higher learnable parameters provide higher levels of abstraction, and thus, the network can learn and recognize more complex patterns and have better performance in the image recognition task. The authors also presented multiple VGG architectures, Figure 2.11, being the main difference between them the number of layers (depth).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224×224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.11: VGG multiple configurations developed by Simonyan and Zisserman (2015)

The LRN present on the "A-LRN" architecture is a non-trainable layer that square-normalizes the pixel values in a feature map within a local neighborhood. The problem of having deeper networks, it that there are more parameters to be computed, and thus it requires more computational power, however in

comparison with the *AlexNet*, it uses a very small receptive field (3x3 with a stride of 1 instead of 11x11 with a stride of 4) to reduce the number of parameters drastically.

GoogleNet

Additionally, in the same year, Google gave birth to the *GoogLeNet* for the ILSVRC-2014, and it accomplished a higher accuracy rate than its predecessors (6.67% top-5 error rate), (Szegedy et al., 2015). Unlike the prior CNNs, *GoogLeNet* has a little different architecture, instead of stacking up sequentially multiple convolutional layers, they rearrange the convolution and pooling layers in a parallel manner to extract features using different kernel sizes. Their overall intention was to increase the depth of the network, in a parallel manner, without actually increase its depth, and at the same time, gain an even higher performance level, compared to previous winners, for this, the authors introduced a new module by the name of *Inception*, demonstrated in Figure 2.12.

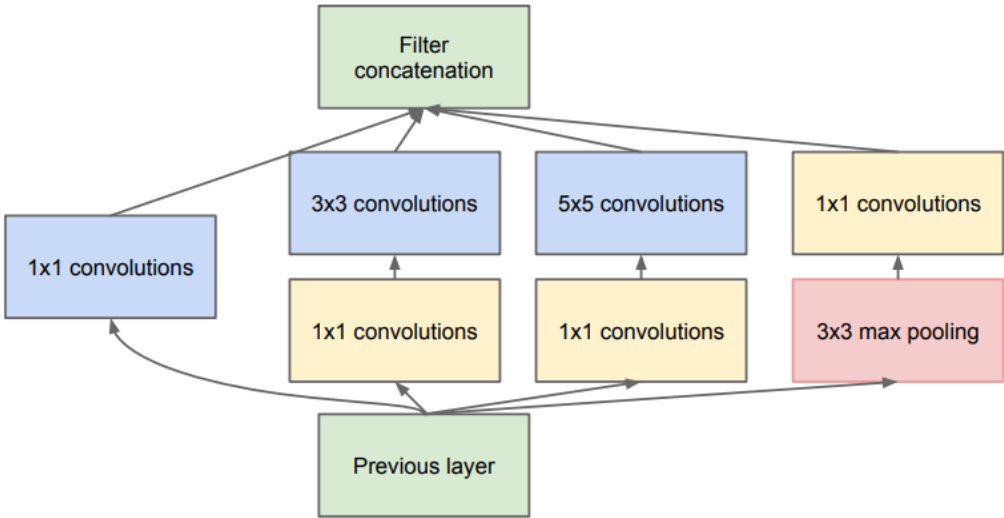


Figure 2.12: Inception module with dimensionality reduction, (Szegedy et al., 2015)

Inception module concatenates filters of different sizes and dimensions into a single new filter, each one of these modules consists of six convolution layers and one pooling layer performed in a parallel manner so that features can be extracted using a different scale.

Overall, *GoogLeNet* has two convolution layers, three pooling layers, and nine *Inception* layers, represented in Figure 2.13, it is significantly more complex and deep than all previous CNN architectures. The number of parameters present in the network is 24 million, which makes *GoogLeNet* a less compute-intensive model when compared to *AlexNet* and *VGG-16*.

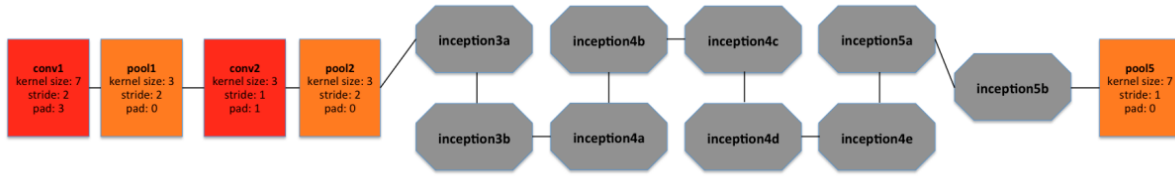


Figure 2.13: A simplified illustration of the *GoogLeNet* architecture, (Shin et al., 2016)

ResNet

A year later, in the ILSVRC-2015, *ResNet*, (He et al., 2016), was the best architecture developed, which won the contest with an error rate of 3.6%. It is considered that humans, in general, have an error rate between 5% and 10%, therefore the *ResNet* manage to outperform the average human in the classification task. This network not only won the classification competition but also established new records on the localization and detection tasks with an error of 9%.

There are multiple versions of *ResNetXX* architectures where 'XX' denotes the number of layers, see Figure 2.14 for more details. The most commonly used ones are *ResNet50* and *ResNet101*.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.14: Multiple architectures of *ResNet* developed for ImageNet, (He et al., 2016)

He et al. (2016) introduced the idea of residual learning for the deep neural networks and they treat every layer as a residual block. As stated before with the increase of depth on neural networks, higher-level features were able to be extracted, however neural networks became notorious for not being able to find a simpler feature map when it exists, because when we make the CNN deeper the derivative when back-propagating to the initial layers becomes almost insignificant in value, also known as the vanishing gradient problem. *ResNet* introduced connections shortcuts, within their residual block, to overcome this problem, (He et al., 2016).

The residual block applies a set of convolutional filters to the input X , and the result of those filters is added to the original input, Figure 2.15. The main difference of this feature on the previous architectures is that, instead of computing the whole transformation of the input, that is, X to $F(X)$, this residual block allows to only compute the variation of the input.

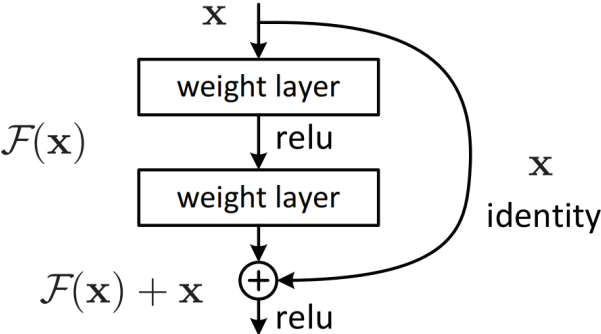


Figure 2.15: Residual learning (residual block), (He et al., 2016)

The use of residual blocks makes the back-propagation process easier because of the adding operation that distributes the gradients.

ResNet is considered the actual state of the art in the convolutional neural networks section and it is the architecture used by Zhang et al. (2019), which is the source model used in this dissertation.

Chapter 3

State of the Art

3.1 Action Representation

3.1.1 Global Representations

Global representations encode the region of interest, which the person in human action recognition problem, as a whole, usually the person is obtained through background subtraction or tracking. Global representations approaches were mostly proposed in earlier works but started to fall in disuse since they are sensitive to noise, occlusions and viewpoint variations.

Silhouettes and Shapes

The silhouette of a person can be simply captured by using background subtraction, however, this process is imperfect since the subtraction process can lead to noisy silhouettes extractions. Also, silhouettes are sensitive to viewpoint variations and anthropometry differences. A pioneer work, [Bobick and Davis \(1996\)](#), stacked silhouettes as two components, a Motion Energy Image (MEI) and Motion History Image (MHI) to solve the problem of action recognition. The binary MEI indicates where the motion occurs, while the MHI is constructed with pixel intensity based on a recency function of the silhouette motion, where brighter pixels correspond to more recent motions.

Optical Flow

Optical flow can be used when dealing with dynamic backgrounds, where background subtraction cannot be performed. For example, [Efros et al. \(2003\)](#) presented a descriptor based on optical flow to describe the "small" football players in person-centred images. The background in the football field footage

is dynamic due to the movement of the camera, which makes it hard to perform background subtraction.

Space-Time Volumes (STVs)

Representations based on STVs are able to capture additional temporal information when compared to other spatial representations since their absence of time information. An STV is formed by stacking the video frames with accurate localization and alignment. Stacking every frame along the time axis forms the 3D space-time volume, which has three dimensions, two spatial dimensions X and Y and the temporal dimension.

3.1.2 Local Representations

Local representations focus on specific local patches which are determined by interest point detectors instead of extracting the silhouette and encode them as a whole. Compared with global features, local features have been proved to be robust to noise and partial occlusion.

Space-Time Interest Points

The intuitive idea of local representation is to identify points of interest that contain a large amount of information in an image or video. For example, Harris and Stephens (1988) first proposed an effective 2D interest point detector, the well-known Harris corner detector, which is widely used for object detection.

Local Descriptors

Local descriptors are meant to describe the patches that were sampled either densely or at the interest points mentioned before. Effective descriptors are considered to be discriminatory to target human activity events in the video, and robust to background noise, occlusion and rotation. For example, Scale Invariant Features Transform (SIFT) was first presented in 1999 but was further improved in 2004, (Lowe, 2004). Firstly, with the Difference-of-Gaussian algorithm, a search for interest points is performed, which creates two blur images. Those two images are then subtracted to search for corners or neighbour pixels variations (interest points), and finally, the descriptors are computed based on the gradient measured in the interest point region.

3.1.3 Depth-Based Representations

The representations presented above were concentrated on video sequences captured by traditional RGB cameras since the usage of depth cameras was limited due to their high cost and operational complexity. But thanks to advances in imaging technology and the development of low-cost depth sensors such as *Microsoft Kinect* and *Intel Realsense* (Zhang, 2012; Keselman et al., 2017), an affordable and easier way to access depth maps in real-time was provided.

Representations Based on Depth Maps

Compared with regular RGB images, the depth map contains supplementary depth coordinates and are more informative. Depth images are insensitive to changes in lighting conditions and provide additional body shape and motion information that can help with distinguishing actions that generate similar projections from a single angle of view.

Skeleton-Based Representations

Skeletal information, like bone joints positions, can be obtained from depth maps that are generated from RGB-D cameras. Microsoft Kinect (Zhang, 2012) is a popular device for this representation since its ease to obtain skeleton and joints positions. Microsoft Kinect v2 (2014) is able to capture up to 25 skeleton joints, while the previous version, Kinect v1, was only able to capture a maximum of 20 joints.

Sometimes, when dealing with occlusions, generated skeletons and joints are not 100% accurate and performing action recognition from inaccurate data, can lead to a wrong result, (Li et al., 2010). However, current methods often solve this problem by combining other features that are robust against occlusion or simply lighten the occlusion problem by grouping the whole skeleton into parts, since not all body parts are occluded (Zhao et al., 2013; Du et al., 2015).

3.2 Related Work

The typical action recognition problem, where video sequences are pre-segmented and the start and end of each action is pre-defined, is where most of the literature is focused. Differently, the action detection problem requires the much more difficult task of localizing the frames where the action takes in place, from unsegmented stream of video frames. To create a solution capable of detecting and recognition actions in real-time, directly from a live data stream, a trade-off between latency and accuracy has to be addressed.

Xia et al. (2012), presented a novel approach for human action recognition with histograms of 3D joints locations (HOJ3D). Skeletal joint locations are extracted from Kinect depth maps and projected into a spherical coordinate system, Figure 3.1.

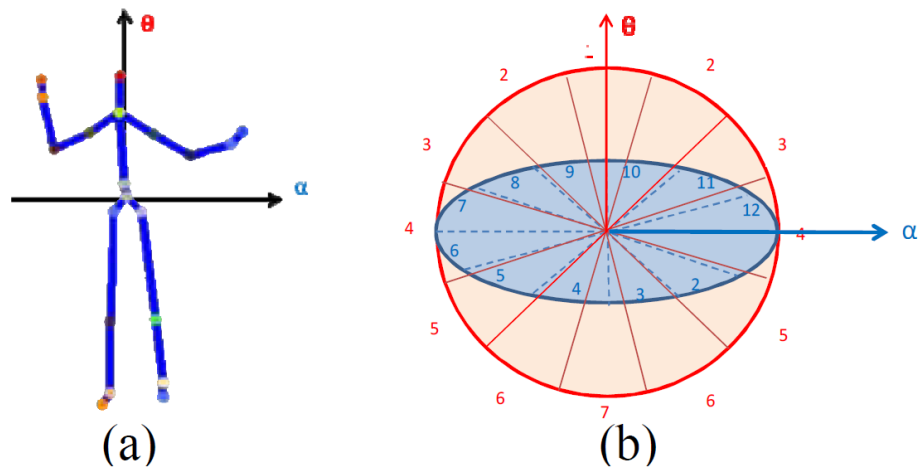


Figure 3.1: (a) Reference coordinates of HOJ3D. (b) Modified spherical coordinate system for joint location. (Xia et al., 2012)

The histogram of 3D joints is then built based on joints location and a Linear Discriminant Analysis (LDA) is performed to extract the dominant features based on the specific class. Features are then clustered into vocabularies using K-Means Clustering and then are fed to the Hidden Markov Model (HMM) for classification. This proposed approach has real-time performance and achieves an overall accuracy of 91% on the MSR Action3D dataset (Li et al., 2010).

Yang et al. (2012), developed a new type of features by the name of "EigenJoints", from skeleton data. Differences of 3D positions between joints are employed to represent three kinds of human action information, posture feature, motion feature and offset feature. To characterize the posture information, the pair-wise joints differences within the current frame is calculated. The motion information is obtained

with the difference of pair-wise joints between the current frame and the preceding frame. To represent the offset information, the difference of the pair-wise joints is related to the initial frame. Principal Component Analysis (PCA) is then applied to reduce redundancy, noise and obtain the "EigenJoints" descriptor for each frame. For the classification, the constructed features are fed into the Naive Bayes Nearest Neighbor classifier. This approach achieves an overall accuracy of 92% on the MSR Action3D dataset (Li et al., 2010).

Zanfir et al. (2013), presented a descriptor for low-latency action recognition and detection, by the name of "The Moving Pose Descriptor". The descriptor is built from a concatenation of the normalized skeleton joints location and its respective first and second-order derivatives, for each frame, the derivatives are estimated numerically with a temporal window of 5 frames centered at the current processed frame. The normalization process only affects the position of the joints, not the angles between joints, this normalization is used to compensate for anthropometric differences between different individuals, thus the average length of each bone is learnt from the training data and this generic length is used for all individuals. Also, to make the joints locations invariant to camera parameters, the hip centre position is subtracted to each joint. The normalization process improves the performance of their method by at least 5%.

"Despite the natural skeleton size and proportions variation in the human population, people tend to perform the same action in the same qualitative way, by moving similarly." - Zanfir et al. (2013)

Papadopoulos et al. (2014) approach used the angles of the joints, relative to the torso position, which was shown to be more discriminative than using the joints coordinates directly. Also, presents a motion energy-based methodology to apply horizontal symmetry and solve the problem that when the same action is performed but is handled with the opposite side of the body.

Later in 2014, Wu and Shao (2014), demonstrated that it is possible to extract high-level features from hidden layers of a deep neural network. The multi-layer neural network was pre-trained as a generative model, one layer at a time. Their method combines the power of deep neural networks to extract high-level features and the graphical framework of Markov model as a probabilistic classifier. Using the pairwise joints as raw input to the multi-layer neural network, their approach is able to extract relational multi-joints features which are much more powerful than simply using the joints location. Moreover, even not tested in a real-time environment, it is stated by them that the proposed framework can perform real-time action recognition.

Du et al. (2015), presented in 2015, the first end-to-end solution for skeleton-based action recognition by using a hierarchical recurrent neural network, without advanced preprocessing. In their method the skeleton body is roughly divided into five parts, two arms, two legs and one trunk, each part is fed into a bidirectional recurrently connected subnet (Layer 1), Figure 3.2.

In Layer 2, to model neighbouring skeleton parts, the representation of the trunk subnet is combined with the four other subnets to obtain four new representations. Similar to the first layer, each of these four representations is fed into a bidirectionally recurrently connected subnet, layer 3.

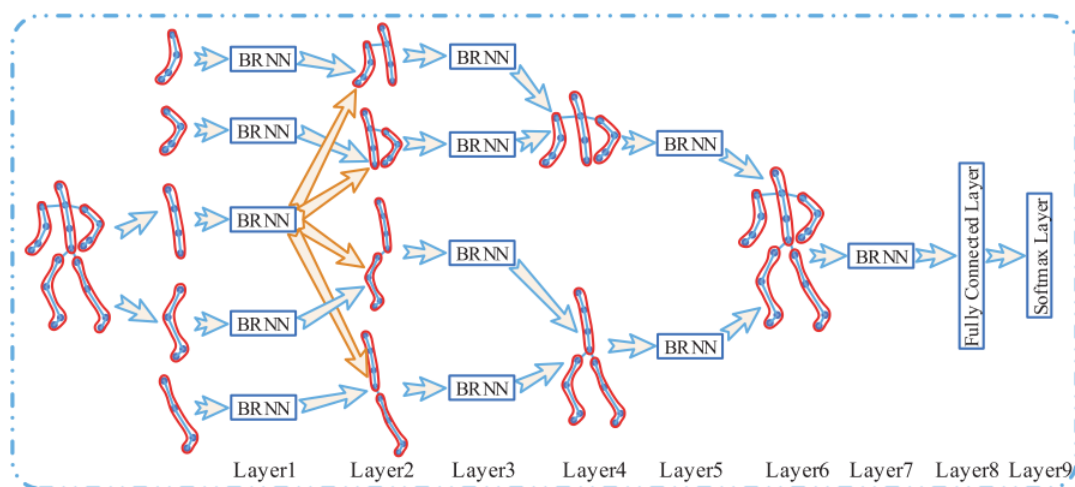


Figure 3.2: An illustrative sketch of the hierarchical recurrent neural network, (Du et al., 2015)

In layer 4, the representation of the left arm-trunk and right arm-trunk are further combined to obtain the upper body representation while the representations of the left leg-trunk and right leg-trunk are combined to obtain the lower body representation, which again, are fed into each bidirectionally recurrently connected subnet, layer 5.

Finally, the whole body model is fused and temporal dynamics of the whole body are modelled by another bidirectionally recurrently connected subnet (layer 6 and layer 7). After obtaining the final features of the whole skeleton body, a fully connected layer and a softmax layer are performed to classify the action.

Their solution achieved great accuracy on the tested datasets, however, it does not work on continuous data streams.

Sharaf et al. (2015) presented yet, a new approach for real-time multi-scale action detection using as features joints angles and angular velocities. Using the joint angles and angular velocities, which are derived from 3D joint locations, the built descriptor becomes invariant to body rotation and translations.

Action detection is a much more challenging task than action recognition on temporally segmented sequences, to solve this, a continuous action point detected is performed by computing the probability for each class on each video frame and comparing that probability with a defined threshold T . A feature selection algorithm, SVM-RFE (Guyon et al., 2002), was used to find the best best features and reduce computational latency and then a Support Vector Machine (SVM) classifier with a linear kernel to perform action detection.

The main contributions of this work are:

- Action detection - the detection problem, which is different from the recognition problem, aims to locate the point in time where the action starts, which is the point where the recognition algorithm will execute.
- Real-time operation - the presented algorithm can run up to 1140 fps, which is enough to apply an exhaustive sliding window search up to 40 different scales in real-time.
- Multi-scale detection - suitable sliding windows with different scales can be computed to better correspond to different actions that take longer to execute.

Meshry et al. (2016) proposed a framework that works online and is suitable for real-time application. Similar to Sharaf et al. (2015), they used 35 joint angles and their respective angular velocities since they are a powerful cue to skeleton pose and invariant to body translations and rotations. However, their action detection mechanism is slightly different, here each class has a different threshold θ_c , a classification method that can output a classification score for each frame is used and the sum of consecutive frames is compared to the threshold θ_c .

The Figure 3.3 shows how the sum and threshold θ_c are related. If the classification score computed for a frame is positive, the sum starts to be computed and as long as the sum is a positive number, the action is still on-going. Once the sum exceeds the threshold θ_c it is known that the action is being performed, so, the next step is to determinate where the action ends and that's when the sum starts to decrease.

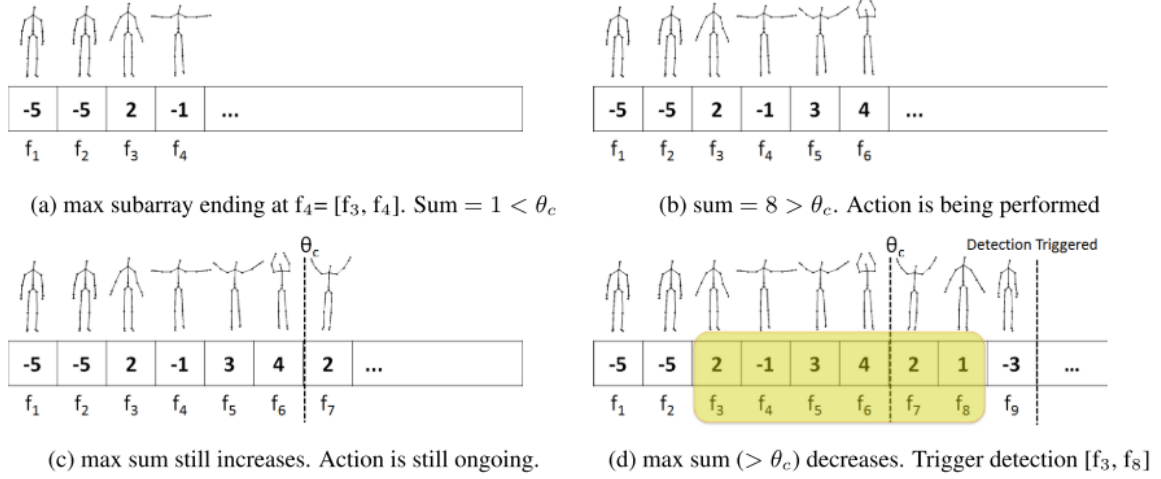


Figure 3.3: Example of online detection, where the action threshold $\theta_c = 7$, (Meshry et al., 2016).

As per usual, a compromise between latency and accuracy has to be made, if the action end is specified as the first negative score, the algorithm will be very sensitive to noise but will achieve low latency, however, if the end of an action is decided after two consecutive negative scores, higher accuracy will be achieved at the cost of increased latency.

Wu et al. (2019) presented an innovative approach to solve the problem of action detection and recognition. Innovatively, they transformed the temporal action detection problem into the object detection issue.

Primarily, skeleton joints position are encoded as RGB images, Kinect V2, which was used in this work, can provide information of 25 joints, being N the number of joints, each frame is defined by $(x, y, z) \Rightarrow R = [x_1, x_2, \dots, x_N], G = [y_1, y_2, \dots, y_N], B = [z_1, z_2, \dots, z_N]$, retaining all the temporal dynamic information and spatial information of the original skeleton action. The next step is to convert the raw coordinates into RGB valid numbers:

$$R_n = \text{floor}\left(255 * \frac{x_n - x_{min}}{x_{max} - x_{min}}\right), n \in [1, N] \quad (3.1)$$

$$G_n = \text{floor}\left(255 * \frac{y_n - y_{min}}{y_{max} - y_{min}}\right), n \in [1, N] \quad (3.2)$$

$$B_n = \text{floor}\left(255 * \frac{z_n - z_{min}}{z_{max} - z_{min}}\right), n \in [1, N] \quad (3.3)$$

Where n represents the joint number, $x_{min}, y_{min}, z_{min}$ and $x_{max}, y_{max}, z_{max}$ are the minimum and maximum coordinate value of all the joints in the training set and the *floor* function represents the

rounding down. Being the skeleton sequences encoded as RGB images, the time action detection problem can be solved by object detection method.

YOLO are convolution neural networks that can realize end-to-end target detection and recognition and predict multiple box positions and categories, its main advantage is that it is fast. YOLO selects the whole image training model, so does not waste time choosing the sliding windows or extracting proposals to train the network.

YOLO network is composed of 5 convolutional layers and 4 max-pooling layers, with batch normalization on all the convolutional layers, Figure 3.4. Batch normalization leads to significant improvements in convergence while eliminating the need for other forms of regularization. The activation function used is LeakyReLU.

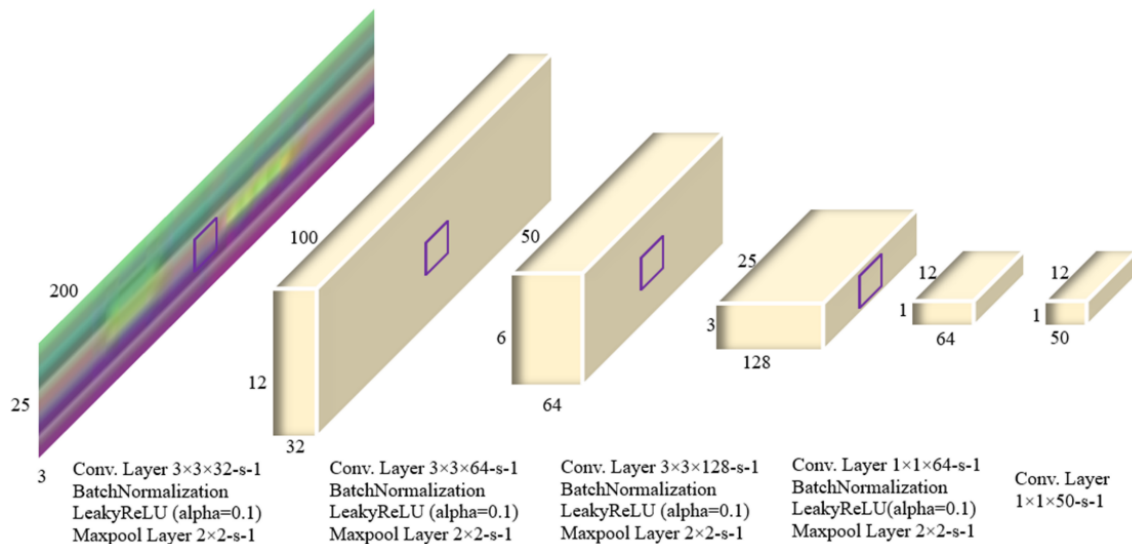


Figure 3.4: One-dimensional YOLO network, (Wu et al., 2019).

Due to the way that joint coordinates are rearranged into RGB images, this algorithm can only be used to detect one person actions, which is not a problem for this thesis case of study.

Recently, Carrara et al. (2019) developed an LSTM-based method capable of performing real-time action detection and prediction in skeleton joint information data streams. To avoid costly feature extraction and comparison, Carrara et al. (2019) adopted a unidirectional LSTM model to encode the skeleton frames within the hidden network states.

For the action detection, the authors stated that using the traditional principle of sliding window that is shifted in a systematic and overlapping manner, which introduces a non-trivial replication of the data, is a bad approach, so in this work, frame-level annotation based on deep recurrent learning is preferred.

Since the LSTM recurrent neural networks are very effective in preserving semantic context in its cells, the authors propose an architecture with only one LSTM cell for online annotation and two cells for offline annotation, which makes the training process more efficient.

For representation, a *motion sequence* is defined by a sequence of *poses*, (P_1, \dots, P_n) , where n determines the motion length. A *pose* is the numerical representation of the 3D coordinates of 31 tracked skeleton joints, as graphically illustrated in Figure 3.5.

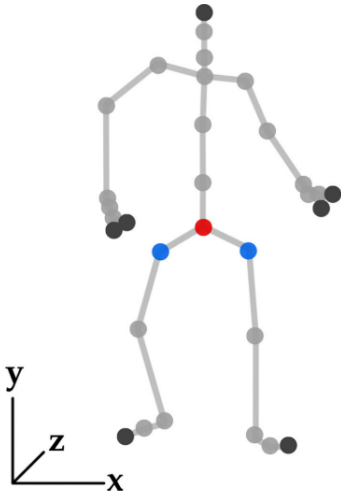


Figure 3.5: Skeleton with 31 joints, where the red joint is the origin coordinate $(0, 0, 0)$, (Carrara et al., 2019).

Action detection, or *annotation*, is the problem of determining what sub-sequences of a stream correspond to the predefined classes of action, for each *pose*, the probability of belonging to each action class is computed and consequently if the probability is greater than the defined threshold for each action class, that class annotation is assigned to that pose. Since multiple classes of actions can happen simultaneously, multiple classes can be assigned to a single pose, as shown in Figure 3.6.

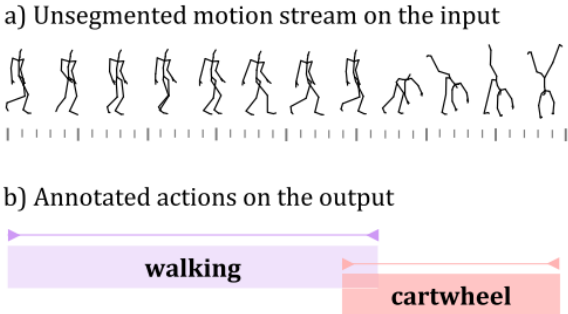


Figure 3.6: Multi-label annotation example, (Carrara et al., 2019).

3.3 Datasets for Human Action Recognition

A dataset consists of a table of labelled data to train or benchmark a neural network, even though there is not a standard dataset for action recognition, there are some datasets that are being considered as reference ones (Chaquet et al., 2013). Due to the complexity of collecting data, the available datasets are limited.

Weizmann

Created in 2005, Weizmann database was used by many proposed methods of action recognition, to evaluate the performance of their contributions (Blank et al., 2005; Gorelick et al., 2007). It contains 90 video sequences, at a resolution of 180 x 144 at 50fps, with a static background and viewpoint, of 10 different actions performed by 9 different persons.

These are the actions that appear in the dataset: run, walk, skip, jumping-jack, jump-forward-on-two-kegs, jump-in-place-on-two-legs, side-gallop, wave-two-hands, wave-one-hand and bend.

MSRAction3D

In 2010, the authors in Li et al. (2010), published a database called MSRAction3D, which presented a series of depth maps obtained by a depth camera. The depth maps have a resolution of 640 x 480 and were captured at 15 fps by a depth camera with an infra-red light structure.

This dataset holds twenty actions, performed three times by seven different persons: high arm wave, horizontal arm wave, hammer, hand catch, forward punch, high throw, draw x, draw tick, draw circle, hand clap, two hand wave, side-boxing, bend, forward kick, side kick, jogging, tennis swing, tennis serve, golf swing, pick up and throw.

HMDB51

HMDB51 is another database, from 2011, used for action recognition. This dataset collects videos mainly from movies but also other public services such as YouTube, Prelinger Archives and Google (Kuehne et al., 2011).

It contains 6,849 videos, at a resolution of 320 x 240, with 51 actions, depicted in Figure 3.7 and 3.8, and a minimum of 100 clips to each action. The 51 actions can be divided as well into five main groups:

- Facial actions: smile, laugh, chew, talk.

- Facial actions with objects: smoke, eat, drink.
- General body movements: cartwheel, clap hands, climb, climb stairs, dive, fall on the floor, back-hand flip, handstand, jump, pull up, push up, run, sit down, sit up, somersault, stand up, turn, walk, wave.
- Body movements with objects: brush hair, catch, draw sword, dribble, golf, hit something, kick ball, pick, pour, push something, ride bike, ride horse, shoot ball, shoot bow, shoot gun, swing baseball bat, sword exercise, throw.
- Body movements for human interaction: fencing, hug, kick someone, kiss, punch, shake hands, sword fight

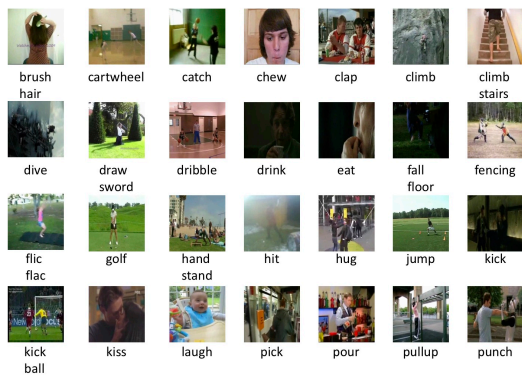


Figure 3.7: Illustration of the 51 Actions (part 1)



Figure 3.8: Illustration of the 51 Actions (part 2)

Apart from the video data and the label, this dataset has other meta-labs in each clip, these labels provide information about some features such as camera motion, lighting conditions, or background. Besides, the quality of each video is measured, depending on whether body parts vanish while the action is performed or not.

UCF-101

This dataset, created in 2012, is composed of realistic action videos, it has 13,320 videos with 101 actions and 27 hours of video data (Soomro et al., 2012). Previously this dataset was formed by only 50 actions (UCF-50) (Reddy and Shah, 2013), but more and more were added with time.

The videos contained in this dataset are collected from YouTube, making the dataset realistic and containing a huge variety of objects, cameras motion, backgrounds, lighting, etc. Based on different combinations of those features, videos are gathered into 25 groups, inside in every 101 actions.

The 101 actions can be divided into five main groups:

- Human-Object Interaction (20 actions).
- Body-Motion Only (16 actions).
- Human–Human Interaction (5 actions).
- Playing Musical Instruments (10 actions).
- Sports (50 actions).

ActivityNet

Presented in 2015, ActivityNet database is composed of 203 classes with an average of 137 videos per class and it has a total of 648 hours of video (Heilbron et al., 2015). The videos are from online video sharing sites and every video has a length between 5 and 10 minutes.

The majority of the videos are in HD (1280 x 720) and have a frame rate of 30 fps. The purpose of this dataset is to collect activities of humans daily life and it has a hierarchical structure, depicted in Figure 3.9, organized according to social interactions and where they take place.

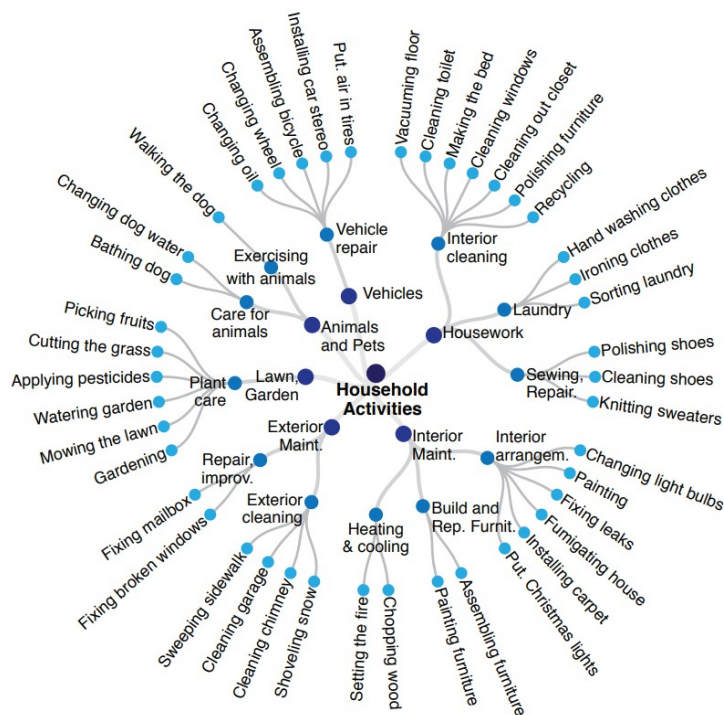


Figure 3.9: ActivityNet hierarchical structure, (Heilbron et al., 2015)

Something Something

Later, in 2017, the authors of Goyal et al. (2017), presented the Something Something dataset, Figure 3.10. This dataset was published containing 108,499 videos, with 174 different labels, the length of the videos variate between 2 and 6 seconds with a height of 100 pixels. The second version of this dataset was released and now it contains 220,847 videos, but the number of labels remains the same, the video resolution has increased from 100 pixels to 240 pixels.

This dataset is already split into three sets, training data, validation data and testing data. The first version contains 86,017 videos for the training set, 11,522 videos for the validation set and 10,960 videos for the test set. The second version contains 168,913 for the training set, 24,777 for the validation set and 27,157 for the test set.

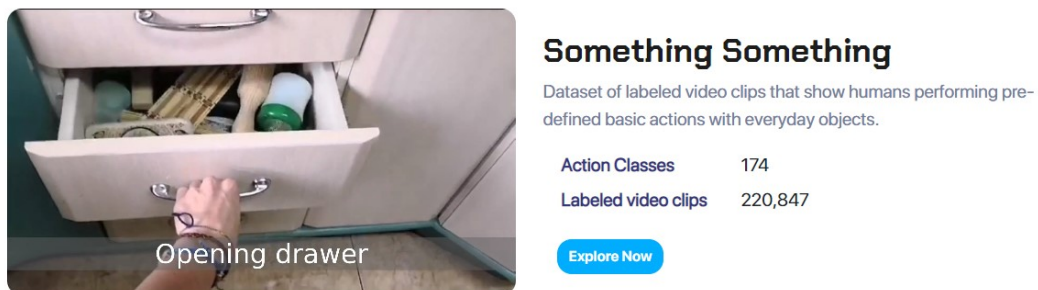


Figure 3.10: Something Something dataset, (Goyal et al., 2017).

AVA

The authors of Gu et al. (2018) presented AVA in 2018, it is a spatio-temporal dataset of 80 atomic visual actions, contains 430 clips with a length of 15 min, these 80 actions are divided into 3 groups, 14 poses, 17 person-person actions and 49 person-object actions.

Every person of the scene is localized by a bounding box and labels are assigned according to the action performed by the actor. Each scene can have more than a label, one of them corresponds to the actor's pose and additional labels are related to person-person or person-object interactions, resulting in 1.62M action labels with multiple labels per frame.

NTU-RGBD

Rapid-Rich Object Search Lab introduced in 2016 the "NTU RGB+D" dataset (Shahroudy et al., 2016), it contains 60 action classes and 56,880 video samples. Later in 2019 "NTU RGB+D 120" dataset is presented, extending "NTU RGB+D" by adding another 60 classes and another 57,600 video samples. "NTU RGB+D 120" has 120 classes and 114,480 samples in total (Liu et al., 2020).

These two datasets are captured by three Kinect V2 cameras concurrently, both contain RGB videos, depth map sequences, 3D skeletal data, and infrared (IR) videos for each sample.

The RGB videos have a resolution of 1920 x 1080, while depth maps and IR videos have a resolution of 512 x 424, and 3D skeletal data contains the 3D coordinates of 25 body joints at each frame. The actions in these two datasets are in three major categories: 82 daily actions (appendix B, Figure B.1), 12 medical conditions (appendix B, Figure B.2), and 26 mutual actions (appendix B, Figure B.3).

Discussion

Taking in account the different datasets mentioned above, which are the ones considered as reference for benchmark purposes on action recognition algorithms, and our focus in industrial settings, it was necessary to create the conditions to develop our model in a way to classify the adequate actions. To accomplish that objective, it became imperative the creation of our own dataset with industry related actions.

Chapter 4

Design and Implementation

The presented solution for skeleton-based action recognition in industrial settings is built on top of an existent state-of-the-art model for offline classification with segmented and labelled data, by the name of "View Adaptive Neural Networks for Skeleton-based Human Action Recognition", VA-CNN from Zhang et al. (2019), and utilizes the NTU RGB+D dataset of training the testing purposes. This solution was developed on Ubuntu 18.04, with Robot Operating System (ROS) as a middleware to handle communication between the data acquisition module, the data transformation module and the deep learning classification module and to ensure seamless transition for a real industrial robotic system.

4.1 Data Acquisition

For the data acquisition module, we built a solution capable of capturing, analyzing and extracting skeletal joint information with high precision in real-time. To do so, we used an RGB-D camera, Microsoft Kinect v2, for the live footage capture directly linked to an *Openpose* module for the skeletal information extraction.

4.1.1 Hardware

Microsoft Kinect v2

Microsoft released two Kinect cameras, namely XBOX 360 Kinect (Kinect for Windows v1) and XBOX One Kinect (Kinect for Windows v2), represented in Figure 4.1, Kinect v2 device, and its internal sensors disposition. Where the most recent one, Kinect v2, is considered an upgrade over Kinect v1, due to its technical specifications changes, the most significant differences are the colour sensor resolution and the

depth-sensing technology (infrared sensor), in the Table 4.1 presents some specifications of Kinect v2, the device used in this solution (Yang et al., 2015).



Figure 4.1: Microsoft Kinect v2¹ and its internal sensors²

Table 4.1: Microsoft Kinect v2 main specifications

Sensor Technology	Time of Flight
RGB Sensor Resolution	1920 x 1080 @ 30fps
IR Sensor Resolution	512 x 424 @ 30fps
Range of operation	0.5m - 4.5m
Field of view (horizontal x vertical)	70° x 60°

The Infrared sensor used on Kinect v2 is lighting independent, which makes the data and the depth information not affected by lighting conditions. The depth maps obtained by the Kinect v2 are more precise, noise-free and more reliable than the maps obtained from the previous version. This is due to the change in depth-sensing technology to Time of Flight (ToF) (Wasenmüller and Stricker, 2017).

Time-of-Flight

Time-of-flight technology mainly measures the time it takes for the light emitted from the light source to travel to the object and back to the sensor. The light source can be pulsed or modulated by a continuous wave source (usually a sine wave or a square wave). The Kinect v2 sensor uses a continuous wave, which is the most commonly used in ToF cameras. It is assumed that the illumination source and the sensor are located at the same distance as the object. Since the distance between the object and the sensor is constant and the speed of light is finite, the time shift caused in the transmitted signal is equivalent to the phase shift in the received signal. Based on the phase shift between the received signal and the generated signal, the ToF is calculated and then used to generate the depth map (Li, 2014; Sarbolandi et al., 2015). Kinect v2 is an active 3D imaging system based on this principle.

¹Image from: www.windowcentral.com/kinect, Accessed: 11 November 2021.

²Image from: www.ifixit.com/Teardown/Xbox+One+Kinect+Teardown/19725, Accessed: 11 November 2021.

4.1.2 Software

Openpose

The OpenPose human gesture recognition is an open-source library developed by Carnegie Mellon University (CMU). OpenPose can perform real-time multi-person human skeleton recognition under the monitoring of video, and it can estimate up to 135 keypoints, 25 body joint estimation, hand joints estimation (20 keypoints for each hand) and 70 face keypoint estimation with excellent robustness (Cao et al., 2019a).

OpenPose is based on convolution neural networks and Caffe (Convolutional Architecture for Fast Feature Embedding) (Jia et al., 2014). It adopts the top-down human body attitude estimation algorithm to detect the position of key points of the human body and then uses the feature vector affinity parameter to determine the hot spot map of human key nodes (Cao et al., 2019b; Wei et al., 2016).

ROS

ROS (Robot Operation System) is an open-source meta-operative system that lies on top of another operating system, usually Linux Ubuntu, and is a full operating system for service robots.

It allows real-time connection between multiple robotic parts, i.e. sensors, actuators and the control system, with ROS tools, such as topics and messages. Essentially, it acts as a broker system handling communication between distributed systems.

ROS integrates countless libraries frequently used in state-of-the-art robotic solutions and provides an easy adaptation from the simulation environment to a real environment (Quigley et al., 2009).

For the purpose of this dissertation, the ROS Melodic distribution was used in Ubuntu 18.04 LTS.

4.1.3 Implementation

To implement the acquisition module on Ubuntu with the Kinect v2 sensor, it is necessary to use an open-source driver for the Kinect to work correctly, *libfreenect2* from Xiang et al. (2016).

As stated before, ROS is used to establish communication between some modules and even within the module itself. Still, in the acquisition module, it is required to send the data from the Kinect sensor to the *Openpose* sub-module. For that, a ROS wrapper is used to enhance the Kinect camera, specifically *iai_kinect2* by Wiedemeyer (2015), which publishes the data information from the sensor into ROS Topics, see Code C.1 for the ROS Topic list published from *iai_kinect2*. The *iai_kinect2* library also

includes a calibration tool that allows the calibration of the Kinect v2 intrinsic parameters, which aligns the IR image with the RGB image.

The next figures, 4.2 and 4.3 show the chessboard used for the sensor calibration, and also a noticeable improvement on edge detection of the whiteboard behind the chessboard calibration tool.

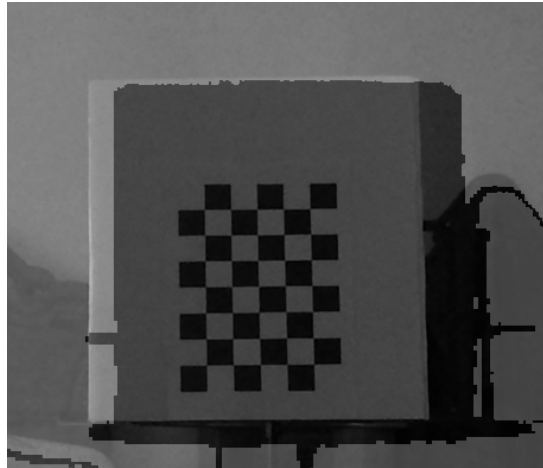


Figure 4.2: Image before calibration

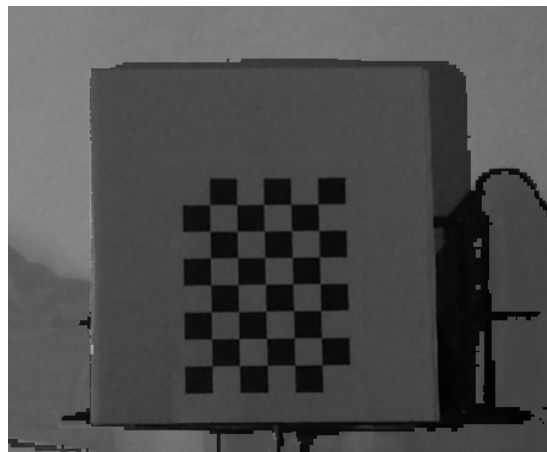


Figure 4.3: Image after calibration

Once the live footage from the Kinect v2 sensor is being continuously transmitted into topics, *Ros_Openpose*, from Joshi et al. (2019), which is a ROS wrapper, takes those topics and links them to the *Openpose* sub-module.

The *Openpose* sub-module performs human skeleton estimation on the given topics, more precisely, in two topics, one containing RGB image information, and the other, the depth image information. This sub-module outputs real-time image feedback with the skeleton information displayed alongside the input live feed, represented in Figure 4.4, but also a ROS topic by the name of "frame", which contains the

2D and 3D coordinates of the estimated skeletal joints (Code D.1 shows the message published by the *Ros_Openpose* to the "frame" topic).



Figure 4.4: *Openpose* real-time skeleton tracking

To get the estimated skeletal joint information from the *Ros_Openpose* wrapper, a ROS subscriber was needed, Code 4.1 shows the code necessary to create the subscriber and how to get the 25 joints 3D coordinates from the Frame topic.

```
1 import rospy
2 from ros_openpose.msg import Frame
3
4 def callback(data):
5
6     #Get ROS simulation/real time
7     now = rospy.get_rostime()
8
9     #Variable that holds joint coordinates of every joint for every detected person
10    body_coordinates = [str([bodyPart.point.x, bodyPart.point.y, bodyPart.point.z]) for person in data.persons
11                       for bodyPart in person.bodyParts]
12
13 def listener():
14
15    rospy.init_node('listener', anonymous=False)
16    rospy.Subscriber("frame", Frame, callback)
17    rospy.spin()
18
19 if __name__ == '__main__':
20    listener()
```

Code 4.1: Frame topic subscriber

Finally, the skeletal joints data were ready to pass onto the next data handling module.

4.2 Data Preparation for Training

Openpose is susceptible to occlusions which occur when a significant portion of the human body is behind an object of the environment or the estimated body parts are outside of the image frame, and their corresponding estimated values are outside the sensor field of view. Thus, saving the data to files directly from the variable *body_coordinates* may contain errors, usually coordinates with considerable values outside of the sensor boundaries or filled with zeros, and for this reason, all the frames that contain faulty coordinates are dropped in an automated manner with a simple *python* routine.

4.2.1 Dataset

To create our skeleton-based action recognition dataset in industrial settings, it was necessary to select a task that fits this case scenario, and for that, an assembly task based on a real work-cell of a local industry was recreated at the Mobile and Anthropomorphic Robotics Laboratory of the University of Minho.

This dataset of an assembly task is divided into six actions, where the task itself is divided into four actions and then a start and stop static action is included at the beginning and the end, respectively.

The next figures illustrate an RGB image of each action, Figure 4.5 and Figure 4.10 represents the static actions that mark the start and the end of the assembly task. The other four figures represent the actions necessary to accomplish the assembly task itself.



Figure 4.5: Static action representing the start of the task, action[0]



Figure 4.6: Frame from the second action, action[1]



Figure 4.7: Frame from the third action, action[2]



Figure 4.8: Frame from the fourth action, action[3]



Figure 4.9: Frame from the fifth action, action[4]



Figure 4.10: Static action representing the end of the task, action[5]

To summarise, the dataset created holds a total of six actions, where each action is repeated 80 times, which gives a total of 480 RGB + Depth video samples and its associated file with the skeletal joint information. The actions index to label correlation is as it follows in the Table 4.2.

Table 4.2: Correlation between classification index and action labels

Action Index	Label
0	Start position
1	Pick & Place main board
2	Pick & Assemble right cube
3	Pick & Assemble left cube
4	Pick & Place assembled board
5	End position

4.2.2 Segmentation

Since this is a dataset for supervised training of a skeleton-based action recognition deep learning model, the skeletal joint data needs to be segmented and labelled. We trimmed the information to the frames where the action takes place and label it with the corresponding action index, see Table 4.2 for reference.

Since there is not an automated procedure to perform this task, the process of segmentation and labelling was performed manually, video by video, and since the time correlation between RGB videos and the file that holds the skeletal joints coordinates is lost on the extraction, it was necessary to develop a graphical tool to visualize the skeletons frame by frame.

4.2.3 3D Skeleton Visualizer

The 3D skeleton graphic tool, illustrated in Figure 4.11, was developed in *python* and it is based on the *Matplotlib* library. It loads the skeleton files frame by frame and uses as the title of the plot, the ROS time

of the corresponding displayed frame, mentioned on Code 4.1. This tool facilitates the task of identifying the frames of a corresponding action and allows the verification of faulty skeletal joint information.

The 3D skeleton visualizer not only solved the problem mentioned above, but also showed that the skeletal joints estimated by the *Openpose* uses a different coordinate reference system (Figure 4.13) than the native Microsoft Kinect v2 SDK (Figure 4.12), which is the software used in the NTU RGB+D dataset to extract the skeletal joints.

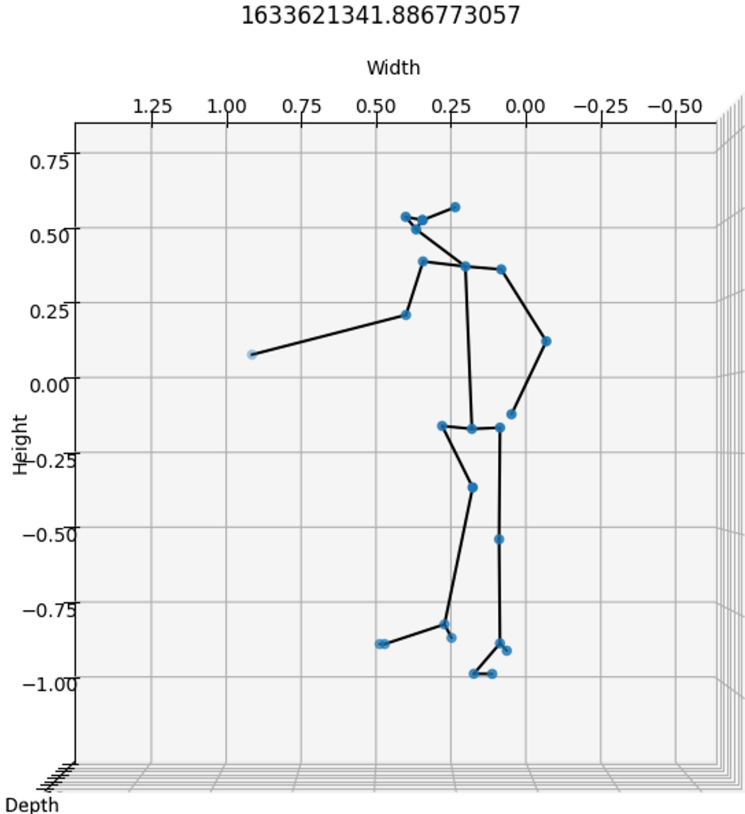


Figure 4.11: 3D skeleton visualizer on our dataset data, action number three

The presented solution for skeleton-based action recognition is built on top of the VA-CNN model. It was necessary to adapt the created dataset to match the same referential as the NTU RGB+D dataset, which is the dataset used to train the VA-CNN model for the pre-trained weights available online. Where:

$$(X, Y, Z) \rightarrow (-X', -Y', Z') \tag{4.1}$$

Finally, with our dataset is segmented, labelled and matching the NTU RGB+D dataset referential, our data is ready to be fed into the VA-CNN model.

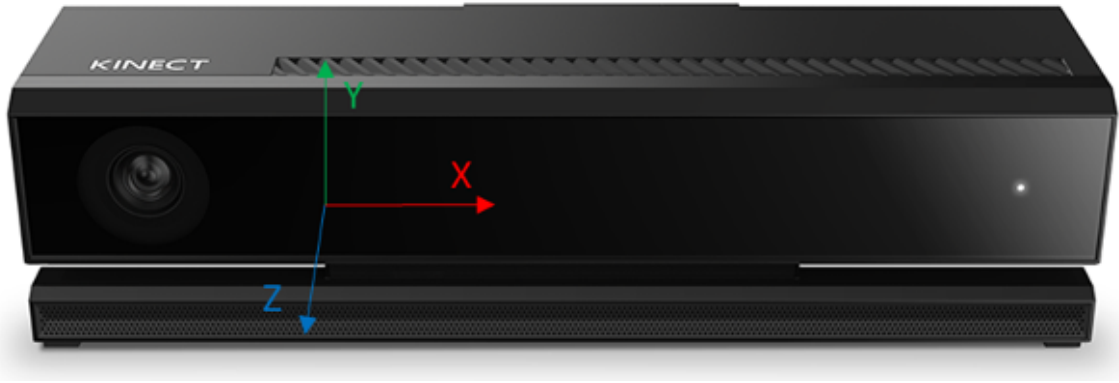


Figure 4.12: Camera space coordinates from the Kinect SDK, (Jamhoury, 2018)



Figure 4.13: Camera space coordinates from the *Ros_Openpose*

4.3 Deep Learning Model

Our solution for skeleton-based action recognition in industrial settings builds on top of a state-of-the-art model with pre-trained weights available for download. Otherwise, we would need to create a dataset large enough to train the model in a non-biased way. With the pre-trained weights from a state-of-the-art dataset, we reduce the model bias by training with more data and then, with our dataset, fine-tune the model to recognize our actions.

Our model inherits from the "View Adaptive Neural Networks for Skeleton-based Human Action Recognition" from Zhang et al. (2019)¹. Zhang et al. (2019) presented a two model topology by the name of VA-fusion, illustrated in Figure 4.14, a weighted fusion method that combines the scores of two independent models, the VA-RNN and the VA-CNN model, to obtain the final classification. Considering that the VA-CNN model, the bottom part of the figure, achieved better performance, the weighted fusion utilizes a weight of 4 for the VA-CNN model and a weight of 1 for the VA-RNN model.

¹View Adaptive Neural Networks for Skeleton-based Human Action Recognition [Github repository](#).

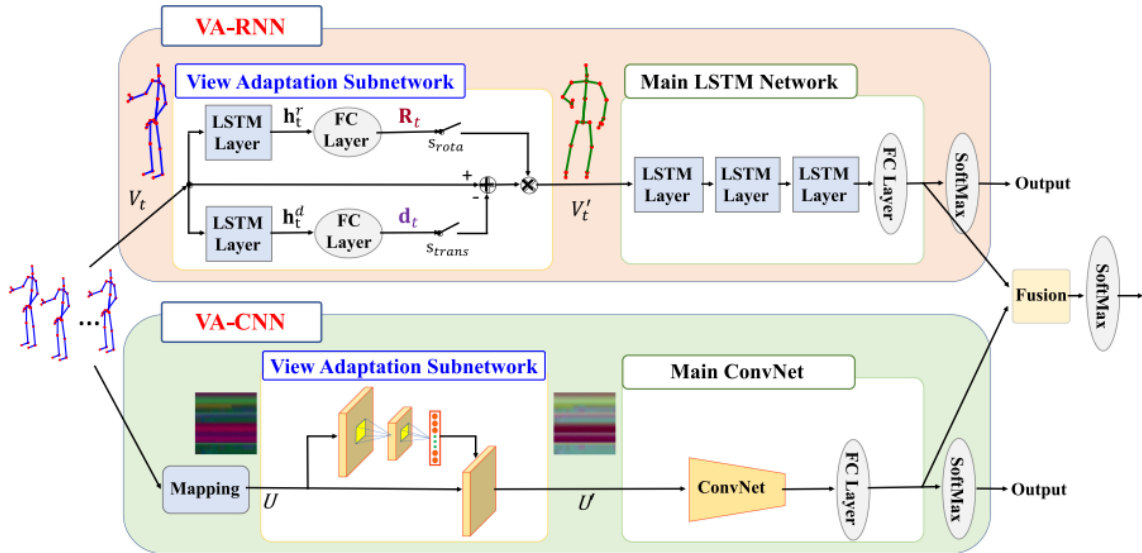


Figure 4.14: Architecture of the proposed VA-fusion scheme, (Zhang et al., 2019)

VA-CNN was the model used because it was the model that achieved better overall classification performance compared to VA-RNN, comparison available in the Table 4.3.

Table 4.3: Accuracy (%) comparisons on different number of layers and models. For reference: **CS**, cross-subject where the 40 subjects are split into the training and testing groups; and **CV**, cross-view where the samples of cameras 2 and 3 are used for training and those of camera 1 for testing, these are the two standard evaluation methods for the NTU RGB+D dataset

Main Network	Structure	CS	CV
RNN	1 LSTM layer	74.5	82.0
	2 LSTM layers	76.2	84.7
	3 LSTM layers	77.0	85.0
	4 LSTM layers	76.9	83.9
	5 LSTM layers	76.2	84.2
	6 LSTM layers	76.6	84.4
VA-RNN	3+2 LSTM layers	79.8	88.9
CNN	ResNet18	86.5	93.1
	ResNet50	87.9	93.5
	ResNet101	87.8	93.5
	ResNet152	88.2	93.4
VA-CNN	ResNet50 + 3 layers	88.7	94.3

4.3.1 VA-CNN

The VA-CNN is an end-to-end view adaptive neural network, and it is divided into two parts, the view adaptation sub-network and the main classification network. A flowchart that describes the VA-CNN model

is illustrated in Figure 4.15. The view adaptation sub-network automatically determines the best virtual observation viewpoint and transforms the skeleton input to the new viewpoint for classification by the main network.

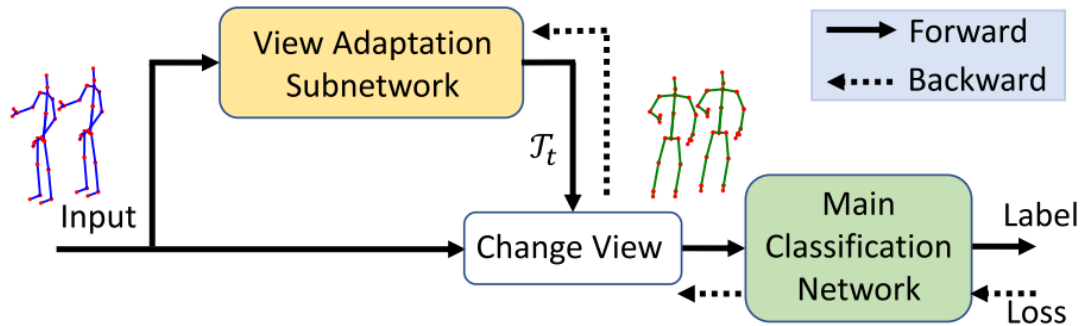


Figure 4.15: Flowchart of the end-to-end view adaptive neural network, (Zhang et al., 2019)

One major challenge in the skeleton-based action recognition task is the viewpoint like described in Section 1.2.2. One of the most commonly used approaches is to perform a pre-processing of the skeletons on the frame level. However, frame-level pre-processing, in which each frame is transformed in relation to the body centre and aligned in the upper body direction, usually results in the partial loss of relative motion information. For example, the action of walking becomes walking in the same place, and the action of dancing with body rotating becomes dancing with the body facing a fixed orientation.

Rather than trying to solve this complex challenge, Zhang et al. (2019) developed a content-dependent View Adaptation (VA) model to automatically learn and determine the most suitable viewpoints and transform the skeleton representations under those views for each frame sequence.

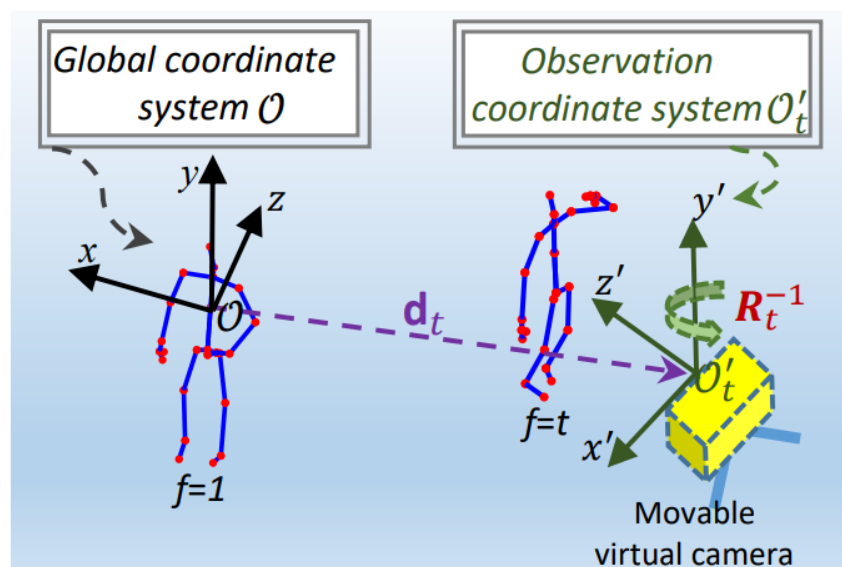


Figure 4.16: Illustration of the change of the observation viewpoint, by assuming there is a movable virtual camera, (Zhang et al., 2019)

The raw 3D skeletons joint coordinates are represented corresponding to the camera coordinate system, where the origin of the referential is located at the position of the camera sensor. To make the 3D skeletons insensitive to the initial position of an action, this referential centre is translated to the body centre of the first frame, as a new global coordinate system O , as illustrated in Figure 4.16.

The view adaptation module automatically determines the virtual observation viewpoints and outputs a set of transform parameters T_t for each time t (or T for a sequence). The input skeleton representation is transformed to representations under the new viewpoints for classification by the main classification network.

Given a skeleton sequence S under the global coordinate system O , the j^{th} skeleton joint on the t^{th} frame is denoted as:

$$v_{t,j} = [x_{t,j}, y_{t,j}, z_{t,j}]^T \quad (4.2)$$

Where $t \in [1, \dots, T]$ and $j \in [1, \dots, J]$, T represents the total number of frames in a skeleton sequence and J represents the total number of skeleton joints in a frame. A set of joints in the t^{th} frame is denoted as:

$$V_t = \{v_{t,1}, v_{t,1}, \dots, v_{t,J}\} \quad (4.3)$$

Assuming that the movable virtual observation viewpoint is placed at an appropriate point, with the corresponding observation coordinate system obtained from a translation by $d_t \in \mathbb{R}^3$, and a rotation of $\alpha_t, \beta_t, \gamma_t$ radians anticlockwise around the X, Y, Z axis, respectively. We can consider a set of transformations parameters as:

$$T_t = \{\alpha_t, \beta_t, \gamma_t, d_t\} \quad (4.4)$$

Therefore, the representation of the j^{th} skeleton joint of the t^{th} frame under the observation coordinate system O' is given by:

$$v'_{t,j} = [x'_{t,j}, y'_{t,j}, z'_{t,j}]^T = R_t(v_{t,j} - d_t) \quad (4.5)$$

Where R_t is represented as $R_t = R_{t,\alpha}^x R_{t,\beta}^y R_{t,\gamma}^z$ and denotes the coordinate transformation matrixes for rotating the original coordinate system around the X -axis by α_t radians, the Y -axis by β_t radians, and the Z -axis by γ_t radians anticlockwise, which are defined as:

$$R_{t,\alpha}^x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha_t) & \sin(\alpha_t) \\ 0 & -\sin(\alpha_t) & \cos(\alpha_t) \end{bmatrix} \quad (4.6)$$

$$R_{t,\beta}^y = \begin{bmatrix} \cos(\beta_t) & \sin(\beta_t) & 0 \\ -\sin(\beta_t) & \cos(\beta_t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.7)$$

$$R_{t,\gamma}^z = \begin{bmatrix} \cos(\gamma_t) & 0 & -\sin(\gamma_t) \\ 0 & 1 & 0 \\ \sin(\gamma_t) & 0 & \cos(\gamma_t) \end{bmatrix} \quad (4.8)$$

All the skeleton joints in the t^{th} frame share the same transformation parameters, however, the view-points can vary over time for different frames.

To build the feature map, similar to Wu et al. (2019), presented in section 3.2, the skeleton sequence is transformed into an image, with the columns representing the frames and the rows representing the different 25 body joints. The 3D coordinate values for X, Y and Z are treated as the three channels of the image. For that, it is necessary to normalize the pixel values to be within 0 and 255, given by:

$$u_{t,j} = \text{floor}\left(255 * \frac{v_{t,j} - c_{min}}{c_{max} - c_{min}}\right) \quad (4.9)$$

- $v_{t,j}$ denotes the 3D coordinates of the j^{th} joint of the t^{th} frame in a skeleton sequence;
- $u_{t,j}$ denotes the corresponding pixel value of the normalized image map;
- c_{max} and c_{min} are the maximum and minimum of all the joint coordinates in the training dataset;
- the *floor* function rounds the number to the greatest integer below or equal the number.

Merging the new skeleton map with the previous method of observation from a new viewpoint, the skeleton representation of the j^{th} joint of the t^{th} frame $v_{t,j}$ is transformed to $v'_{t,j}$ according to equation 4.2. Correspondingly, the pixel value of the skeleton map under the new observation viewpoint is calculated by:

$$u'_{t,j} = 255 * \frac{v_{t,j} - c_{min}}{c_{max} - c_{min}} = R_{t,j} u_{t,j} + 255 * \frac{R_{t,j}(c_{min} - d_{t,j}) - c_{min}}{c_{max} - c_{min}} \quad (4.10)$$

Note that equation 4.10 is derived from equation 4.2 and equation 4.9.

Wu et al. (2019) designed the CNN-based view adaptation network to learn and determine the observation viewpoint of each skeleton sequence and perform the transformation on the skeleton map. The view adaptation network consists of a stack of convolutional layers and a fully connected layer to regress the transformation parameters. Thus, a transform layer transforms each pixel in the skeleton map into a new representation in the observation viewpoint, creating a new skeleton map.

Ultimately, with the transformed skeleton map as input, an existing Convolution neural network is used for classification. In this particular case a *ResNet*, and their tests and accuracy scores are presented in the Table 4.3.

To complete this dissertation objective, changes were needed to train the model for our dataset and to adapt the model for an online classification scheme in run-time execution.

4.3.2 Fine-tuning the Model

Once the model is loaded with the pre-trained weights from the train with the NTU RGB+D dataset, it is necessary to change the number of neurons in the last fully connected layer, to match our number of classes, and train it to recognize our actions, in other words, fine-tune the model to adapt it to our action labels and perform the correct classification.

Fine-tuning is a method of applying transfer learning. Specifically, fine-tuning is a process that takes a model that has been trained for a given task and then tunes or tweaks the model to perform a second similar task, which allows us to take advantage of what the model has learned without the need to develop it from scratch.

"When building a model from scratch, we usually must try many approaches through trial-and-error." - Deeplizard (2017)

The block of Code 4.2, holds the fundamental lines of code to adapt the model for the new train with our dataset. In the second line we load the pre-train weights to our neural network model. The *for loop* in lines 5 to 7, sets the attribute *requires_grad* in every layer, except the fully connected one, to False. The *requires_grad* attribute controls whether the parameters weight and bias of each layer are set to update their values during the training process or not.

Lastly, in line 12, a new fully-connected layer is created with the same number of input features as the old fully-connected layer, but this time, only with six output neurons, which is the number of actions in our dataset.

```

1  #Loading the pre-trained weights to our model
2  model.load_state_dict(torch.load(pre_train_weights)['state_dict'])
3
4  #”Freezing” all the layers, except the fully connected layer
5  for name, param in model.named_parameters():
6      if not 'fc' in name:
7          param.requires_grad = False
8
9  #Initializing the new fully connected layer, with the same input features but only with six neurons
10 num_fts = model.classifier.fc.in_features
11 num_classes_new = 6
12 model.classifier.fc = nn.Linear(num_fts, num_classes_new)
13 model = model.cuda()

```

Code 4.2: Adapting the model for the train with our dataset

4.4 Online Classification

To fulfil the objectives of this dissertation, we adapted the algorithm to work in run-time, which means that the whole algorithm has to be adapted for online skeleton-based classification².

As stated in section 4.3.1, the origin of the referential is translated to the body centre of the first frame that goes in the input data for classification, however, in their work, the whole dataset is processed and stored in files, and only later that same data is loaded and fed into the model, file by file, for classification. In order to adapt the data processing module, instead of saving the data in files, everything is done sequentially with data variables.

The primary adaptation required concerns the input data for classification since their work relies on an offline scheme, every action video file is converted in its corresponding skeleton file that holds the joints coordinates values. Then, during the processing, each skeleton file is converted into an image that will serve as input for the neural network. Figure 4.17, illustrates the manner in which the skeleton file is converted into an image, where each column corresponds to each frame that is present in each action video, and each row represents the normalized pixel value, between 0 and 255.

The size of the input image is constant and consists of a 300 by 150 pixels image. The value of 300 on the "Frames" axis corresponds to the maximum of frames present in a single action video on the NTU RGB+D dataset, while the max value of 150 on the "Joints corresponding value" axis corresponds to the x, y, z coordinates of 25 joints of a maximum of 2 persons in the same frame, since the NTU RGB+D dataset contains 10 mutual actions, see Appendix B for mutual action list of the NTU RGB+D dataset, for this reason, when there is only one person in the action or the action video contains less than 300 frames,

²Online classification is when the classification is performed over a continuous data stream, that is, not segmented.

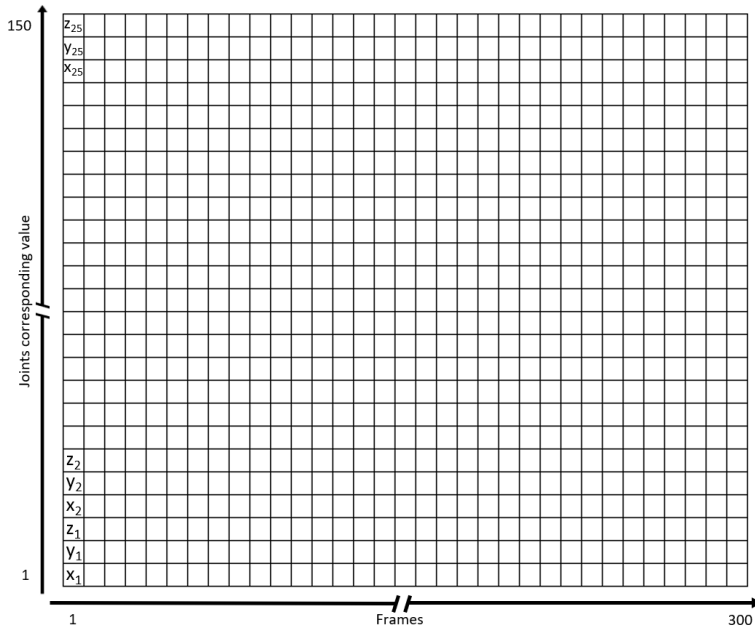


Figure 4.17: Input image to be fed into the VA-CNN

those corresponding pixels are filled with zeros.

In this dissertation, on the other hand, the data acquisition is continuous at 30 fps, but with an estimated rate from *Openpose* of 22 fps, due to hardware limitations, and since the data stream is continuous, it is not known when the action starts and ends, and thus it is impossible to build a separated skeleton file for each action. To overcome this problem, the methodology adopted was a sliding window. In run-time execution, the window slides over the input data, so the data is divided into blocks of frames, that are fed to the model for classification, Figure 4.18.

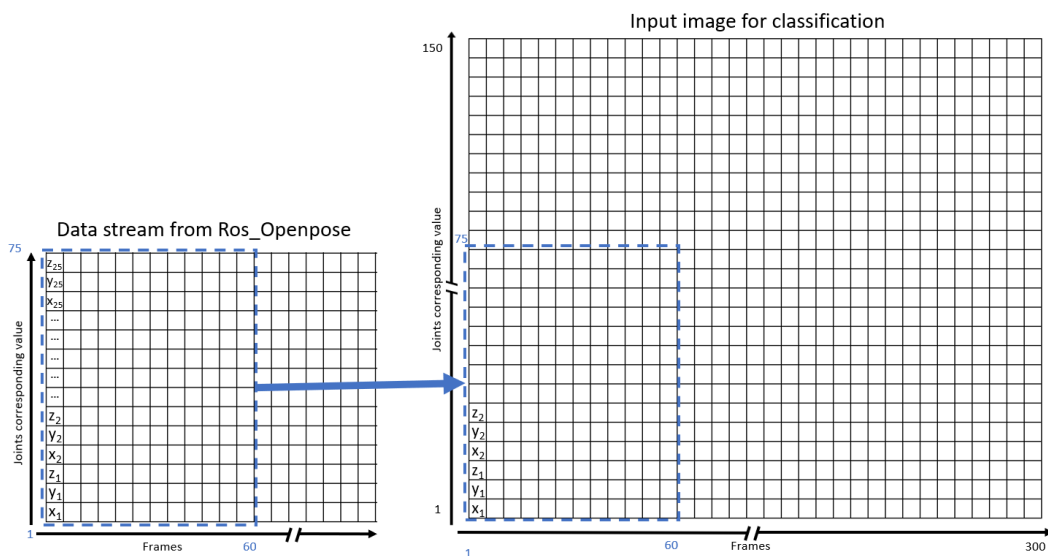


Figure 4.18: First stride of the sliding window

In this example, the sliding window covers a total of 60 frames and the 25 joints of only one person, which is the case in our dataset. The pixels on the "input image for classification" that do not contain information are filled with zeros.

The sliding window size and stride was defined experimentally and will be discussed in **Chapter 5 Test and Results**, for this example, however, we used a stride of 20 frames. The Figure 4.19 represents the next block of data that will be fed into the VA-CNN for classification. In the second stride, in Figure 4.19, only 20 frames are new. Note that the sliding window has to wait for the new frames to arrive from the estimation module, every time the sliding window buffer gets those 20 new frames, the data is processed and fed into the neural network.

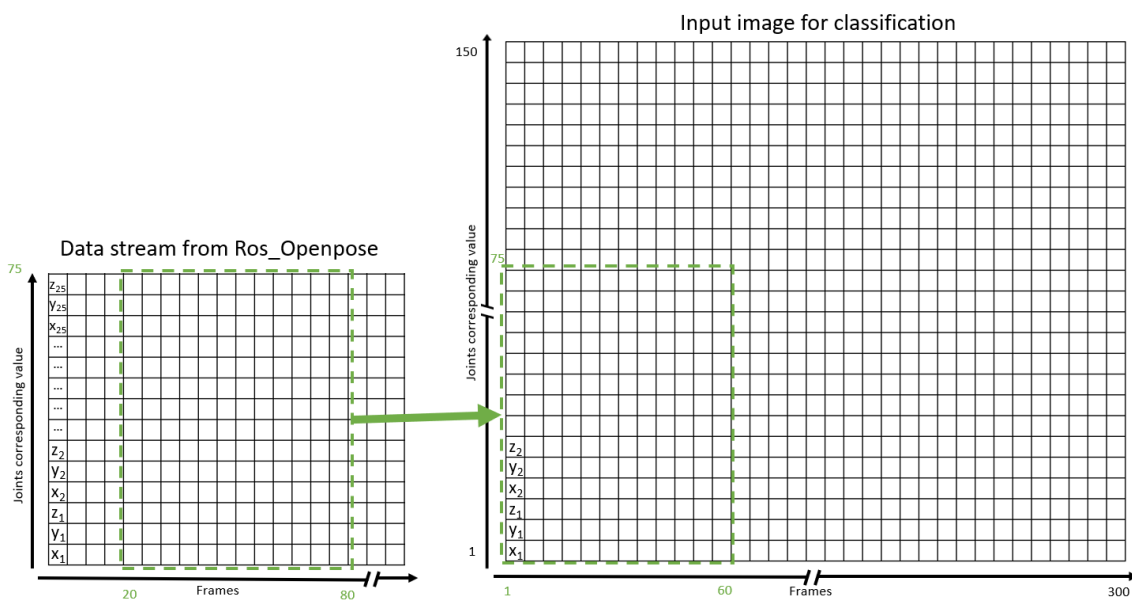


Figure 4.19: Second stride of the sliding window

In this hypothetical scenario, with a fixed rate of estimation from *Openpose* of 20 fps, the algorithm will be able to perform the first classification after 3 seconds of data, and then with the 20 frame stride, a new classification will be performed every 1 second. It is important to notice that if the sliding window size is reduced, the spatial information available for classification will be also reduced, which would reduce the algorithm accuracy. On the other hand, if the stride size is increases, the system needs to wait longer for new frames, which will have an impact on run-time performance.

Chapter 5

Tests and Results

This chapter discusses the tests and results achieved with the described algorithm.

5.1 Sliding Window Size Tests

The first tests performed were to choose the best size for the sliding window method, which also allowed us to verify if the online adaptation method mentioned above on Section 4.4 was a viable option.

Still on the NTU RGB+D dataset, with the respective pre-trained weights loaded into the model, we tested the accuracy of the network when working with a fixed smaller number of frames. The NTU RGB+D dataset, for reference, has on average 84 frames per action video before removing the faulty skeletons files.

When working with the algorithm that performs offline classification, each input to the neural network has a different number of frames corresponding to the action video duration. The rest of the input image is filled with zeros. Since our online solution works with a fixed-sized sliding window, the NTU RGB+D dataset was trimmed to a fixed number of frames per action video to test its accuracy. The following tables, Table 5.1, Table 5.2 and Table 5.3, show the model accuracy for the NTU RGB+D dataset when working with less information for each action.

Considering that the input image has a fixed size of 300 x 150 pixels, we also tested whether the position of the information has an impact on the accuracy, i.e. with the data placed on the bottom left corner of the input image, like represented in Figure 4.19, or the data placed in the middle of the image.

To better understand the following tables, Info Δ_f^{SF} represents the interval of frames gathered from the Skeleton File, i.e. from frame 0 to frame 20, in the first column of data in the Table 5.1, and Δ_f^{in} represents the location where those frames are located in the input image that is fed to the neural network,

the rest of the image is filled with zeros.

Table 5.1: Accuracy (%) values for the NTU RGB+D dataset with only 20 frames being analysed for each action video

Δ_f^{in} \ Info Δ_f^{SF}	0:20	10:20	20:40	30:50
0:20	13.13%	26.84%	32.5%	26%
20:40	13.13%	26.84%	32.5%	26%
100:120	13.13%	26.84%	32.5%	26%

Table 5.2: Accuracy (%) values for the NTU RGB+D dataset with only 40 frames being analysed for each action video

Δ_f^{in} \ Info Δ_f^{SF}	0:40	10:50	20:60	30:70
0:40	51.16%	60.25%	58.47%	44%
40:80	51.16%	60.25%	58.47%	44%
200:240	51.16%	60.25%	58.47%	44%

Table 5.3: Accuracy (%) values for the NTU RGB+D dataset with only 60 frames being analysed for each action video

Δ_f^{in} \ Info Δ_f^{SF}	0:60	10:70	20:80	30:90
0:60	78.19%	80.15%	73.7%	56%
60:120	78.19%	80.15%	73.7%	56%
120:180	78.19%	80.15%	73.7%	56%

From these results, we can conclude that:

- increasing the number of frames given to the network for classification increases the accuracy since more spatial information is preserved;
- the location of the frames in the original action video shown that the beginning and end of the action represent a lower accuracy than the middle frames where the action is more meaningful;
- the position where the skeletal information is provided in the input image to the network does not have an impact on the classification accuracy.

Since we want our model to be the most accurate possible, we chose a size of 60 frames for our sliding window. It is important to notice that a smaller sliding windows size, would result in less accurate results. However, if we increase the size of the sliding window, the input image could contain skeletal information of 2 distinct actions for classification.

With a size of 60 frames and with an estimation rate from *Openpose* of 22 fps, the first action classification after the system startup will only occur after $60/22 = 2.7$ seconds plus the classification duration. The actual time between classifications is defined by the stride value and the classification duration, it is not affected by this value.

5.2 Different Training Methods

We proceeded to test the classification accuracy after fine-tuning the respective NTU RGB+D pre-trains with our dataset. The classification accuracy after fine-tuning is worse than expected. The Table 5.4 illustrates the pre-trained models initial accuracy for their respective evaluation methods, and the classification accuracy on our dataset.

Table 5.4: Accuracy (%) of the neural network on our dataset after fine-tuning each available pre-train

Train	NTU - CS	NTU - CV	OUR
NTU RGB+D 60	88.7	94.3	
NTU RGB+D 60 + Fine-Tuning			49.6
NTU RGB+D 120	86.9	84.8	
NTU RGB+D 120 + Fine-Tuning			44.3

Since the model accuracy for both pre-trains was beneath our objectives, we decided to train the VA-CNN model with our dataset, on top of the *ResNet* pre-trained parameters from the ImageNet classification. This solution led to better accuracy results, comparison available in Table 5.5.

Table 5.5: Accuracy (%) comparisons on different number of convolutional layers on the ResNet on our dataset

Structure	Validation Accuracy (%)	Average Classification Time (seconds)
ResNet18	94.07	0.17828
ResNet34	95.7	0.17901
ResNet50	98.49	0.17957
ResNet101	98.51	0.21054
ResNet152	98.45	0.21864

Focused on a speed and accuracy balance, we chose the ResNet50 since is one of the models that

showed the best accuracy, while having an average classification time beneath 200 milliseconds. Note that the classification time depends on the available hardware, and these classification times were achieved on a system powered by an Nvidia RTX 3070 GPU (Graphics Processing Unit), an AMD Ryzen™ 7 3700X CPU (Central Processing Unit) and 32GB of RAM (Random Access Memory). The data processing time is discarded since that on average it takes less than 10ms.

For the training we used a validation accuracy monitor with the *ReduceLROnPlateau* method, which reduces the learning rate when the metric stops improving. With a patience of three epochs and a cooldown of two epochs. The dataset is divided into training set and validation set, with a ratio of 80% for the training set and 20% for the validation set.

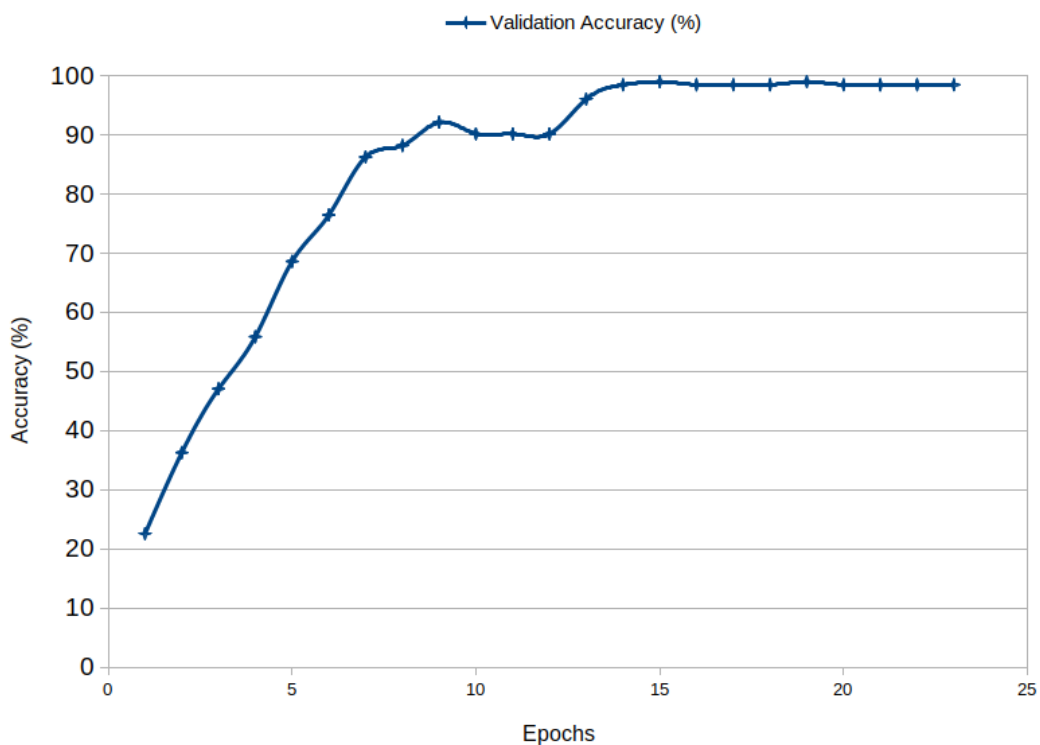


Figure 5.1: Validation accuracy (%) curve during the training of the neural network

The final layout of the entire neural network used for the presented solution of skeleton-based action recognition in this dissertation is available in Code E.1, in the appendix section.

5.3 Online Action Recognition

The final algorithm was tested on unsegmented data, for an online classification scheme, with a sliding window size of 60 frames and a stride of 20 frames. The stride value will mainly affect the number of classifications that will be performed each second. The code below, Code 5.1 demonstrates the raw output

of our neural network for online classification. Where the task is performed sequentially, every action is performed in the correct order and immediately after the previous task.

```
1 action[3]tensor(96.3943)
2 action[1]tensor(50.5733)
3 action[0]tensor(51.1906)
4 action[0]tensor(98.4328)
5 action[0]tensor(51.6456)
6 action[0]tensor(29.1455)
7 action[1]tensor(89.0742)
8 action[1]tensor(48.1000)
9 action[1]tensor(37.9836)
10 action[2]tensor(95.3860)
11 action[2]tensor(96.6040)
12 action[2]tensor(60.7024)
13 action[3]tensor(77.7237)
14 action[3]tensor(95.4610)
15 action[3]tensor(94.2141)
16 action[4]tensor(41.5883)
17 action[4]tensor(81.9965)
18 action[4]tensor(88.6722)
19 action[4]tensor(94.3010)
20 action[4]tensor(56.2121)
21 action[1]tensor(30.2377)
22 action[5]tensor(95.2667)
23 action[5]tensor(78.6323)
24 action[5]tensor(58.9878)
25 action[5]tensor(52.6219)
26 action[4]tensor(67.7990)
27 action[4]tensor(41.3616)
28 action[3]tensor(56.2482)
29 action[3]tensor(45.2413)
```

Code 5.1: Unfiltered output of the neural network on online classification over a video where the complete assembly task is being performed, the first number represents the action label recognized by the neural network and the "tensor" number represents its confidence score percentage

Note that the beginning and end of the task, represented in Code 5.1, are the moments that the subject gets into position to perform the assembly task and the moment the task is completed, respectively.

It is possible to comprehend that the confidence score drops drastically due to transitory movements between actions. Additionally, between action "4" and "5", the algorithm even predicts the wrong action.

As this solution is intended to be incorporated on a cooperative robot and we do not want to overfeed the system with unusable information i.e. the transitory movements with a low confidence score. We apply a threshold to the output of the network based on the given confidence score.

For the final evaluation we tested our final solution on eight assembly video trials. During these tests, we used a threshold of 80% on the classification output. The eight assembly videos are independent, but the task is performed by the same subject with variable execution speed, thus it is possible that the same

action is detected multiple times with high confidence score.

The assembly task is performed sequentially, every action is performed in the correct order, '0' to '5' and immediately after the previous task. However, every trial includes a few extra seconds on the beginning and the end where there are no concrete actions, and thus those are not considered as errors.

```
1 action[0]tensor(97.5396)
2 action[0]tensor(92.3178)
3 action[1]tensor(92.1530)
4 action[1]tensor(97.4922)
5 action[2]tensor(87.1453)
6 action[3]tensor(81.2431)
7 action[4]tensor(97.5619)
8 action[5]tensor(83.5767)
```

Code 5.2: Filtered classification output for the video trial 1. In this trial, the system detected every action with an impressive confidence score, over 90%, only action '3' and action '5' being under that mark

```
1 action[1]tensor(85.0603)
2 action[3]tensor(81.8829)
3 action[3]tensor(96.9289)
4 action[4]tensor(90.8837)
5 action[5]tensor(87.4782)
```

Code 5.3: Filtered classification output for the video trial 2. In this trial, action '0' and action '2' were detected with lower confidence than the threshold, so in this trial, there are two errors

```
1 action[1]tensor(80.0042)
2 action[0]tensor(81.7046)
3 action[0]tensor(89.8392)
4 action[0]tensor(85.7404)
5 action[1]tensor(98.8821)
6 action[1]tensor(95.8116)
7 action[1]tensor(97.7083)
8 action[3]tensor(93.8145)
9 action[4]tensor(86.9156)
10 action[4]tensor(93.8358)
11 action[4]tensor(82.2935)
12 action[5]tensor(95.7026)
13 action[5]tensor(80.9123)
14 action[1]tensor(80.0400)
```

Code 5.4: Filtered classification output for the video trial 3. Some of the actions were detected multiple times since the task was performed slower in this trial. The action '1', at the beginning and end, is not counted as an error, the only error is the detection of action '2' which had a lower confidence score than the threshold

```
1 action[3]tensor(89.5940)
2 action[3]tensor(92.4615)
3 action[3]tensor(93.2392)
4 action[0]tensor(97.7155)
5 action[0]tensor(93.0265)
6 action[1]tensor(97.2872)
7 action[1]tensor(97.1610)
8 action[4]tensor(95.0013)
9 action[4]tensor(90.4237)
10 action[5]tensor(83.5266)
11 action[5]tensor(90.0726)
```

Code 5.5: Filtered classification output for the video trial 4. There are two errors in this trial since that action '2' and action '3' were detected with lower confidence scores than the threshold

```
1 action[0]tensor(95.3018)
2 action[0]tensor(99.4015)
3 action[0]tensor(99.2157)
4 action[1]tensor(96.7539)
5 action[1]tensor(85.7635)
6 action[3]tensor(99.1446)
7 action[3]tensor(88.0008)
8 action[4]tensor(94.8336)
9 action[4]tensor(98.5153)
10 action[4]tensor(88.1284)
11 action[5]tensor(95.6959)
```

Code 5.6: Filtered classification output for the video trial 5. In this trial, action '2' was not detected even in the raw output of the network, which makes this trial only have one error

```
1 action[0]tensor(81.4436)
2 action[0]tensor(94.0623)
3 action[1]tensor(97.8951)
4 action[1]tensor(89.8360)
5 action[3]tensor(92.2292)
6 action[3]tensor(99.7854)
7 action[3]tensor(93.4549)
8 action[4]tensor(90.6822)
9 action[4]tensor(96.0448)
10 action[4]tensor(85.5089)
11 action[5]tensor(91.7362)
```

Code 5.7: Filtered classification output for the video trial 6. In this trial, action '2' was detected in the raw classification output, but with a lower confidence score than the threshold

```
1 action[3]tensor(96.3943)
2 action[1]tensor(91.5707)
3 action[2]tensor(92.7712)
4 action[2]tensor(96.9420)
```

```

5 action[2]tensor(77.4642)
6 action[3]tensor(83.2878)
7 action[3]tensor(94.4357)
8 action[3]tensor(97.3311)
9 action[4]tensor(86.0767)
10 action[4]tensor(94.3413)
11 action[5]tensor(80.3459)

```

Code 5.8: Filtered classification output for the video trial 7. In this trial, action '0' was detected in the raw classification output, but with a lower confidence score than the threshold

```

1 action[3]tensor(93.6243)
2 action[0]tensor(98.4328)
3 action[1]tensor(89.0742)
4 action[2]tensor(95.3860)
5 action[2]tensor(96.6040)
6 action[3]tensor(95.4610)
7 action[3]tensor(94.2141)
8 action[4]tensor(81.9965)
9 action[4]tensor(88.6722)
10 action[4]tensor(94.3010)
11 action[5]tensor(95.2667)

```

Code 5.9: Filtered classification output for the video trial 8. In this trial, the system detected every action with an impressive confidence score, over the threshold. Action '3' in the beginning is not considered an error

To further evaluate the final accuracy of our model for the online action recognition problem, we created an error metric where we enumerate the number of errors in each video trial. We considered an error when the system does not detect a specific action or when some other action is detected outside of the sequence, i.e. the system detects action 0 and the next action detected is action 2, so the model failed to detect action 1, or when between action 1 and action 2 the system detects some other action.

Table 5.6 enumerates the number of errors for each action and each trial detected on the previously shown video trials.

Table 5.6: Number of errors detected on action classification for the tested video trials

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Total
Action '0'	0	1	0	0	0	0	1	0	2
Action '1'	0	0	0	0	0	0	0	0	0
Action '2'	0	1	1	1	1	1	0	0	5
Action '3'	0	0	0	1	0	0	0	0	1
Action '4'	0	0	0	0	0	0	0	0	0
Action '5'	0	0	0	0	0	0	0	0	0
Total	0	2	1	2	1	1	1	0	8

Table 5.7: Average error rate of each action on the eight trials

Actions	Error rate
Action '0'	25%
Action '1'	0%
Action '2'	62.5%
Action '3'	12.5%
Action '4'	0%
Action '5'	0%

Multiple threshold values were tested, however lowering the threshold value further than 80% did not show a significant improvement on the error rate metric since most of the times action '2' (which is the one that showed worst accuracy) is classified with an average confidence score of 50%.

The next figures illustrate an example of the final visual output of the presented solution for the skeleton-based human action recognition in industrial settings classification task on the video trial eight with the confidence score threshold.



Figure 5.2: Classification of the static action representing the start of the task, action[0]



Figure 5.3: Classification of the example of the second action, action[1]



Figure 5.4: Classification of the example of the third action, action[2]



Figure 5.5: Classification of the example of the fourth action, action[3]



Figure 5.6: Classification of the example of the fifth action, action[4]



Figure 5.7: Classification of the static action representing the end of the task, action[5]

5.4 Action Recognition from Live Data Stream

In order to reduce the error rate presented in the tests above, we decided to extend our dataset. In this new version, we added four new subjects to the dataset. Each of these contributed with a total of twenty samples for each action, which results in a total of 480 new train samples. The final composition of our dataset is presented in the table 5.8, which makes a total of 960 training samples.

Table 5.8: Total composition of our dataset (number of samples)

Actions	Train Subject 1	Train Subject 2	Train Subject 3	Train Subject 4
Action '0'	100	20	20	20
Action '1'	100	20	20	20
Action '2'	100	20	20	20
Action '3'	100	20	20	20
Action '4'	100	20	20	20
Action '5'	100	20	20	20
Total	600	120	120	120

In order to test the solution with real-time data input, we had to change our setup to use two computers instead of just one, since both modules, data acquisition and data classification, are CUDA intensive, *Openpose* pose estimation is CUDA based and the neural network model is also CUDA based, and the *libfreenect2* dependencies were incompatible with the *PyTorch* dependencies. The final scheme with the two computer setup is represented in 5.8.

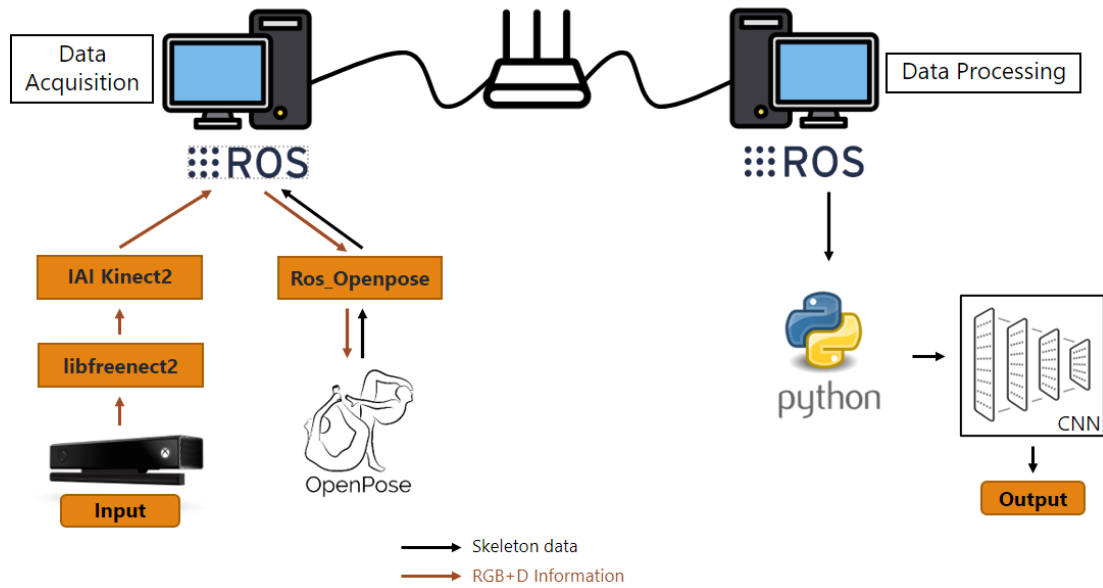


Figure 5.8: Scheme of out final solution for online action recognition with real-time data input

With our final solution trained with our new dataset, we not only tested the accuracy for the four subjects used for training but also tested the accuracy of two new subjects. Each of these subjects contributed with five test trials (total of 30 tests).

Table 5.9: Error rate (%) of the final tests to the solution

Actions	Train Subjects	New Subjects
Action '0'	10%	0%
Action '1'	15%	20%
Action '2'	0%	0%
Action '3'	0%	10%
Action '4'	0%	0%
Action '5'	45%	50%

This new test summary shows an increase in the overall error rate along with all the actions, however with the introduction of new subjects, the model had to learn different representations within the same action, from each subject, and so the model became more generic. Since Action 0 is similar to Action 5, the model started to predict the wrong classification for the 'Action 5' and predicting 'Action 0' instead.

The tests presented in this section 5.4, with live data acquisition are available in this [YouTube Playlist](#). For reference, the left part of the screen corresponds to the 'Data Processing' computer, which also makes the classification, and the right side of the screen belongs to the 'Data Acquisition' computer, which uses the Kinect v2 sensor together with *Openpose* to estimate the human pose in real-time.

Chapter 6

Conclusion

6.1 Discussion

In the majority of the existent hybrid work cells, where the human and the robotic system interact in a manufacturing task, human-robot interaction is limited and structured. In this dissertation we presented a solution that can extend and help the human-robot cooperation, giving the robotic system the perception of what actions the human is performing.

First, we presented several state-of-the-art methods for the skeleton-based human action recognition problem for offline and online detection, where we elaborate on the most suitable neural networks to use for our solution in industrial settings.

We created a dataset settled on an existent work-cell to train the neural network to recognize our own set of actions, and the major disclosure obtained is that it is extremely challenging to gather a good amount of reliable information, for example, in our first attempt to create the dataset there were multiple frames with incomplete skeleton coordinates due to significant occlusions on some body parts.

Finally, we trained the model on our recently created dataset and tested which base convolution neural network architecture best fit our objectives. At last, we adapted the entire solution to perform online action recognition, and based on the hardware available, we can classify the action within less than 200 milliseconds.

The action '2', which was the action that showed the poorest confidence score in most of the test trials, lower than the threshold value in five out of eight trials, in section 5.3, was solved by the re-training of the model with the new training samples that we added to the dataset. However, with the introduction of more subjects, the error rate on the action '5' increased, due to its similarity to the action '0'. The threshold value could also be lowered, so we don't skip any action, but we want that only highly reliable

information to be passed to the cooperative robot control system.

The developed solution can be deployed to any existing work-cell to enhance the human-robot interaction, however, the model needs to be trained to accurately classify other sets of actions that are not present in our dataset.

Action recognition and action prediction are two different problems, however, since the model was trained to recognize not only the peak of the action but the whole action, from the initial to final movements, if the action is performed slowly enough that there is a classification of the initial movements, the model is able to predict the action before its peak point, where the action takes place, based on those initial body movements.

6.2 Future Work

The next steps to further improve the performance of this solution would be to gather more training data for our dataset, in order to decrease the number of errors and build a more robust solution.

The actions presented in our dataset are mostly torso based and this solution only takes into account 25 body joints coordinates, however, *Openpose* allows the estimation of 20 keypoints for each hand. Increasing the number of keypoints for estimation will result in a lower estimation rate, but it would be interesting to study whether this extra skeletal information could result in an even better classification score rather than more errors, due to the less information over the same time.

For future work, and to further test the accuracy of the model with more data, it is necessary to increment the number of actions that the model is able to recognise and the number of subjects performing them and also include mutual actions, to ensure that the model is generic and is able to classify actions from different workers. With a more extensive dataset, where several subjects performed the actions, it would be possible to create a more robust solution.

Also, study how to automate the process of building the dataset, since it is a essential and necessary task to introduce new actions to the model.

Another possible revision, that would result in more spatial information in the data for classification, is to test the solution with a different camera sensor (with a higher frame rate), and a pose estimation software that can match that frame rate.

We used two computers to test the whole online scheme solution due to software limitations dependencies, however, it would be important to study how to overcome these dependencies and test the whole solution on only one computer.

Bibliography

- Aslam, S., Herodotou, H., Ayub, N., and Mohsin, S. M. (2019). Deep learning based techniques to enhance the performance of microgrids: A review. In *Proceedings - 2019 International Conference on Frontiers of Information Technology, FIT 2019*.
- Benfold, B. and Reid, I. (2009). Guiding visual surveillance by tracking human attention. In *Proceedings of the 20th British Machine Vision Conference*.
- Blank, M., Gorelick, L., Shechtman, E., Irani, M., and Basri, R. (2005). Actions as space-time shapes. In *Proceedings of the IEEE International Conference on Computer Vision*, volume II, pages 1395–1402.
- Bobick, A. and Davis, J. (1996). An appearance-based representation of action. In *Proceedings - International Conference on Pattern Recognition*, volume 1, pages 307–312.
- Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. (2019a). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. (2019b). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Carrara, F., Elias, P., Sedmidubsky, J., and Zezula, P. (2019). LSTM-based real-time action detection and prediction in human motion streams. *Multimedia Tools and Applications*, 78(19):27309–27331.
- Chaquet, J. M., Carmona, E. J., and Fernández-Caballero, A. (2013). A survey of video datasets for human action and activity recognition. *Computer Vision and Image Understanding*.
- Deeplizard (2017). Machine Learning & Deep Learning Fundamentals. <https://deeplizard.com/learn/video/5T-iXNNiwlS>.
- Deng, J., Berg, A., Satheesh, S., Su, H., Khosla, A., and Fei-Fei, L. (2012). Imagenet large scale visual recognition competition 2012 (ilsvrc2012). See net.org/challenges
- Deshpande, A. (2016). A Beginner's Guide To Understanding Convolutional Neural Networks. <https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>.
- Du, Y., Wang, W., and Wang, L. (2015). Hierarchical recurrent neural network for skeleton based action recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June:1110–1118.

- Efros, A. A., Berg, A. C., Mori, G., and Malik, J. (2003). Recognizing action at a distance. In *Proceedings of the IEEE International Conference on Computer Vision*.
- Gorelick, L., Blank, M., Shechtman, E., Irani, M., and Basri, R. (2007). Actions as Space-Time Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2247–2253.
- Goyal, R., Kahou, S. E., Michalski, V., Materzynska, J., Westphal, S., Kim, H., Haenel, V., Fruend, I., Yianilos, P., Mueller-Freitag, M., Hoppe, F., Thureau, C., Bax, I., and Memisevic, R. (2017). The 'Something Something' Video Database for Learning and Evaluating Visual Common Sense. In *Proceedings of the IEEE International Conference on Computer Vision*.
- Gu, C., Sun, C., Ross, D. A., Vondrick, C., Pantofaru, C., Li, Y., Vijayanarasimhan, S., Toderici, G., Ricco, S., Sukthankar, R., Schmid, C., and Malik, J. (2018). AVA: A Video Dataset of Spatio-Temporally Localized Atomic Visual Actions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.
- Guyon, I., Weston, J., Barnhill, S., and Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422.
- Harris, C. and Stephens, M. (1988). A Combined Edge and Corner Detector. In *Proceedings of the fourth Alvey Vision Conference (ACV88)*.
- He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.
- Hebb, D. O. (1949). *The organization of behavior: A neuropsychological theory*. Wiley, New York.
- Heilbron, F. C., Escorcia, V., Ghanem, B., and Niebles, J. C. (2015). ActivityNet: A large-scale video benchmark for human activity understanding. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.
- Jamhoury, L. (2018). Understanding Kinect V2 Joints and Coordinate System. <https://lisajamhoury.medium.com/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16>.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*.
- Joshi, R. P., van den Broek, M. K., Tan, X. Z., Choi, A., and Luo, R. (2019). ROS OpenPose. https://github.com/ravijo/ros_openpose.
- Keselman, L., Woodfill, J., Grunnet Jepsen, A., and Bhowmik, A. (2017). Intel realsense stereoscopic depth cameras.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*.
- Kuehne, H., Jhuang, H., Garrote, E., Poggio, T., and Serre, T. (2011). HMDB: A large video database for human motion recognition. In *Proceedings of the IEEE International Conference on Computer Vision*.

- LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.
- Lecun Yann, Cortes Corinna, and Burges Christopher (1998). THE MNIST DATABASE of Handwritten Digits. *The Courant Institute of Mathematical Sciences*.
- Li, L. (2014). Time-of-Flight Camera—An Introduction. *Texas Instruments - Technical White Paper*.
- Li, W., Zhang, Z., and Liu, Z. (2010). Action recognition based on a bag of 3D points. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, CVPRW 2010*.
- Lin, H., Shi, Z., and Zou, Z. (2017). Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote Sensing*.
- Liu, J., Shahroudy, A., Perez, M., Wang, G., Duan, L. Y., and Kot, A. C. (2020). NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2684–2701.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.
- Meshry, M., Hussein, M. E., and Torki, M. (2016). Linear-time online action detection from 3D skeletal data using bags of gesturelets. *2016 IEEE Winter Conference on Applications of Computer Vision, WACV 2016*.
- Mostafa, B., El-Attar, N., Abd-Elhafeez, S., and Awad, W. (2020). Machine and Deep Learning Approaches in Genome: Review Article. *Alfarama Journal of Basic & Applied Sciences*.
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
- O’Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *CoRR*, abs/1511.08458.
- Papadopoulos, G. T., Axenopoulos, A., and Daras, P. (2014). Real-time skeleton-tracking-based human action recognition using kinect data. *Lecture Notes in Computer Science (including sub-series Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8325 LNCS(PART 1):473–483.
- Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.
- Reddy, K. K. and Shah, M. (2013). Recognizing 50 human action categories of web videos. *Machine Vision and Applications*.
- Sarbolandi, H., Lefloch, D., and Kolb, A. (2015). Kinect range sensing: Structured-light versus Time-of-Flight Kinect. *Computer Vision and Image Understanding*.
- Shahroudy, A., Liu, J., Ng, T. T., and Wang, G. (2016). NTU RGB+D: A large scale dataset for 3D human activity analysis. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.

- Sharaf, A., Torki, M., Hussein, M. E., and El-Saban, M. (2015). Real-time multi-scale action detection from 3D skeleton data. *Proceedings - 2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2015*, pages 998–1005.
- Sharma, S. (2017). Activation Functions in Neural Networks - Towards Data Science. <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.
- Shin, H. C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning. *IEEE Transactions on Medical Imaging*.
- Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*.
- Soomro, K., Zamir, A. R., and Shah, M. (2012). UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*.
- Tsang, S.-H. (2018). Review: AlexNet, CaffeNet – Winner of ILSVRC 2012 (Image Classification). <https://medium.com/coinmonks/paper-review-of-alexnet-caffenet-winner-in-ilsvrc-2012-image-classification-b93598314160>.
- Wasenmüller, O. and Stricker, D. (2017). Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.
- Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional pose machines. In *CVPR*.
- Wiedemeyer, T. (2014 – 2015). IAI Kinect2. https://github.com/code-iai/iai_kinect2.
- Wu, D. and Shao, L. (2014). Leveraging hierarchical parametric networks for skeletal joints based action segmentation and recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 724–731.
- Wu, J., Li, Y., Wang, L., Wang, K., Li, R., and Zhou, T. (2019). Skeleton Based Temporal Action Detection with YOLO. *Journal of Physics: Conference Series*, 1237(2).
- Xia, L., Chen, C.-c., and Aggarwal, J. (2012). View Invariant Human Action Recognition Using Histograms of 3D Joints The University of Texas at Austin. *CVPR 2012 HAU3D Workshop*, pages 20–27.
- Xiang, L., Echtler, F., Kerl, C., Wiedemeyer, T., Lars, hanyazou, Gordon, R., Facioni, F., laborer2008, Wareham, R., Goldhoorn, M., alberth, gaborpapp, Fuchs, S., jmtatsch, Blake, J., Federico, Jungkurth, H., Mingze, Y., vinouz, Coleman, D., Burns, B., Rawat, R., Mokhov, S., Reynolds, P., Viau, P., Fraissinet-Tachet, M., Ludique, Billingham, J., and Alistair (2016). libfreenect2: Release 0.2.
- Yang, L., Zhang, L., Dong, H., Alelaiwi, A., and Saddik, A. E. (2015). Evaluating and improving the depth accuracy of Kinect for Windows v2. *IEEE Sensors Journal*.

- Yang, X., Zhang, C., and Tian, Y. (2012). Recognizing actions using depth motion maps-based histograms of oriented gradients. In *MM 2012 - Proceedings of the 20th ACM International Conference on Multimedia*.
- Zanfir, M., Leordeanu, M., and Sminchisescu, C. (2013). The moving pose: An efficient 3D kinematics descriptor for low-latency action recognition and detection. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2752–2759.
- Zhang, P., Lan, C., Xing, J., Zeng, W., Xue, J., and Zheng, N. (2019). View adaptive neural networks for high performance skeleton-based human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- Zhang, Z. (2012). Microsoft kinect sensor and its effect. *IEEE Multimedia - IEEEMM*, 19:4–10.
- Zhao, X., Li, X., Pang, C., Zhu, X., and Sheng, Q. Z. (2013). Online Human Gesture Recognition Area Chair : Max Mühlhäuser. pages 23–32.

Appendices

Appendix A

Activation Functions










Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parametric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Figure A.1: Activation Functions, (Sharma, 2017)

Appendix B

NTU-RGBD 120 Actions

A1: drink water	A2: eat meal	A3: brush teeth	A4: brush hair
A5: drop	A6: pick up	A7: throw	A8: sit down
A9: stand up	A10: clapping	A11: reading	A12: writing
A13: tear up paper	A14: put on jacket	A15: take off jacket	A16: put on a shoe
A17: take off a shoe	A18: put on glasses	A19: take off glasses	A20: put on a hat/cap
A21: take off a hat/cap	A22: cheer up	A23: hand waving	A24: kicking something
A25: reach into pocket	A26: hopping	A27: jump up	A28: phone call
A29: play with phone/tablet	A30: type on a keyboard	A31: point to something	A32: taking a selfie
A33: check time (from watch)	A34: rub two hands	A35: nod head/bow	A36: shake head
A37: wipe face	A38: salute	A39: put palms together	A40: cross hands in front
A61: put on headphone	A62: take off headphone	A63: shoot at basket	A64: bounce ball
A65: tennis bat swing	A66: juggle table tennis ball	A67: hush	A68: flick hair
A69: thumb up	A70: thumb down	A71: make OK sign	A72: make victory sign
A73: staple book	A74: counting money	A75: cutting nails	A76: cutting paper
A77: snap fingers	A78: open bottle	A79: sniff/smell	A80: squat down
A81: toss a coin	A82: fold paper	A83: ball up paper	A84: play magic cube
A85: apply cream on face	A86: apply cream on hand	A87: put on bag	A88: take off bag
A89: put object into bag	A90: take object out of bag	A91: open a box	A92: move heavy objects
A93: shake fist	A94: throw up cap/hat	A95: capitulate	A96: cross arms
A97: arm circles	A98: arm swings	A99: run on the spot	A100: butt kicks
A101: cross toe touch	A102: side kick	-	-

Figure B.1: Daily actions

A41: sneeze/cough	A42: staggering	A43: falling down	A44: headache
A45: chest pain	A46: back pain	A47: neck pain	A48: nausea/vomiting
A49: fan self	A103: yawn	A104: stretch oneself	A105: blow nose

Figure B.2: Medical conditions

A50: punch/slap	A51: kicking	A52: pushing	A53: pat on back
A54: point finger	A55: hugging	A56: giving object	A57: touch pocket
A58: shaking hands	A59: walking towards	A60: walking apart	A106: hit with object
A107: wield knife	A108: knock over	A109: grab stuff	A110: shoot with gun
A111: step on foot	A112: high-five	A113: cheers and drink	A114: carry object
A115: take a photo	A116: follow	A117: whisper	A118: exchange things
A119: support somebody	A120: rock-paper-scissors	-	-

Figure B.3: Mutual actions

Appendix C

Kinect v2 ROS Topic List

```
1 /kinect2/bond
2 /kinect2/hd/camera_info
3 /kinect2/hd/image_color
4 /kinect2/hd/image_color/compressed
5 /kinect2/hd/image_color_rect
6 /kinect2/hd/image_color_rect/compressed
7 /kinect2/hd/image_depth_rect
8 /kinect2/hd/image_depth_rect/compressed
9 /kinect2/hd/image_mono
10 /kinect2/hd/image_mono/compressed
11 /kinect2/hd/image_mono_rect
12 /kinect2/hd/image_mono_rect/compressed
13 /kinect2/hd/points
14 /kinect2/qhd/camera_info
15 /kinect2/qhd/image_color
16 /kinect2/qhd/image_color/compressed
17 /kinect2/qhd/image_color_rect
18 /kinect2/qhd/image_color_rect/compressed
19 /kinect2/qhd/image_depth_rect
20 /kinect2/qhd/image_depth_rect/compressed
21 /kinect2/qhd/image_mono
22 /kinect2/qhd/image_mono/compressed
23 /kinect2/qhd/image_mono_rect
24 /kinect2/qhd/image_mono_rect/compressed
25 /kinect2/qhd/points
26 /kinect2/sd/camera_tnfo
27 /kinect2/sd/image_color_rect
28 /kinect2/sd/image_color_rect/compressed
29 /kinect2/sd/image_depth
30 /kinect2/sd/image_depth/compressed
31 /kinect2/sd/image_depth_rect
32 /kinect2/sd/image_depth_rect/compressed
33 /kinect2/sd/image_tr
34 /kinect2/sd/image_tr/compressed
35 /kinect2/sd/image_tr_rect
36 /kinect2/sd/image_tr_rect/compressed
37 /kinect2/sd/points
```

Code C.1: Kinect v2 ROS Topic list

Appendix D

Frame ROS Topic from Ros_Openpose

```
1 header:
2   seq: 591
3   stamp:
4     secs: 1633621358
5     nsecs: 709032523
6   frame_id: "kinect2_ir_optical_frame"
7 persons:
8   -
9     bodyParts:
10      -
11        score: 0.76349902153
12        pixel:
13          x: 148.02545166
14          y: 139.944335938
15        point:
16          x: -0.560849785805
17          y: -0.321740239859
18          z: 1.89400005341
19      -
20        score: 0.865368127823
21        pixel:
22          x: 188.328430176
23          y: 151.498321533
24        point:
25          x: -0.353887736797
26          y: -0.26359847188
27          z: 1.90800011158
28      -
29        score: 0.810712993145
30        pixel:
31          x: 168.796203613
32          y: 152.701828003
33        point:
34          x: -0.497800827026
35          y: -0.280758440495
36          z: 2.08200001717
37      -
38        score: 0.921117544174
39        pixel:
```

40 x: 151.522964478
41 y: 188.338195801
42 point:
43 x: -0.636068880558
44 y: -0.082180172205
45 z: 2.22000002861
46 -
47 score: 0.871599078178
48 pixel:
49 x: 115.757270813
50 y: 211.437454224
51 point:
52 x: -0.824422478676
53 y: 0.0565669089556
54 z: 2.14300012589
55 -
56 score: 0.770148038864
57 pixel:
58 x: 206.849349976
59 y: 149.193511963
60 point:
61 x: -0.250143259764
62 y: -0.268447011709
63 z: 1.85800004005
64 -
65 score: 0.843778192997
66 pixel:
67 x: 205.634063721
68 y: 202.209671021
69 point:
70 x: -0.250547558069
71 y: 0.0019307063194
72 z: 1.81600010395
73 -
74 score: 0.859818458557
75 pixel:
76 x: 169.930603027
77 y: 240.223205566
78 point:
79 x: -0.408014893532
80 y: 0.182273685932
81 z: 1.72900009155
82 -
83 score: 0.660643458366
84 pixel:
85 x: 218.324737549
86 y: 226.408554077
87 point:
88 x: -0.21852633357
89 y: 0.143024414778
90 z: 2.11900019646
91 -
92 score: 0.564165055752
93 pixel:
94 x: 199.930526733
95 y: 225.290817261
96 point:
97 x: -0.334747999907

98 y: 0.140387892723
99 z: 2.17900013924
100 -
101 score: 0.337695360184
102 pixel:
103 x: 214.847763062
104 y: 279.419158936
105 point:
106 x: -0.18883895874
107 y: 0.357031285763
108 z: 1.67600011826
109 -
110 score: 0.760415971279
111 pixel:
112 x: 215.974578857
113 y: 324.384674072
114 point:
115 x: -0.263318270445
116 y: 0.808535754681
117 z: 2.40300011635
118 -
119 score: 0.706388354301
120 pixel:
121 x: 235.586517334
122 y: 226.42326355
123 point:
124 x: -0.117496669292
125 y: 0.142367079854
126 z: 2.10800004005
127 -
128 score: 0.700818061829
129 pixel:
130 x: 235.651626587
131 y: 281.732116699
132 point:
133 x: -0.12234249711
134 y: 0.483065009117
135 z: 2.20200014114
136 -
137 score: 0.784882366657
138 pixel:
139 x: 241.396362305
140 y: 337.013397217
141 point:
142 x: -0.0912355706096
143 y: 0.851021468639
144 z: 2.29300022125
145 -
146 score: 0.534131646156
147 pixel:
148 x: 143.427017212
149 y: 132.985351562
150 point:
151 x: -0.0912355706096
152 y: 0.851021468639
153 z: 2.29300022125
154 -
155 score: 0.85851585865

156 pixel:
157 x: 152.680541992
158 y: 128.452957153
159 point:
160 x: -0.52927839756
161 y: -0.376253694296
162 z: 1.86800003052
163 -
164 score: 0.0
165 pixel:
166 x: 0.0
167 y: 0.0
168 point:
169 x: -0.52927839756
170 y: -0.376253694296
171 z: 1.86800003052
172 -
173 score: 0.841508448124
174 pixel:
175 x: 175.693191528
176 y: 120.363357544
177 point:
178 x: -0.416548728943
179 y: -0.423105925322
180 z: 1.89200007915
181 -
182 score: 0.781984865665
183 pixel:
184 x: 225.270996094
185 y: 356.65802002
186 point:
187 x: -0.188204929233
188 y: 0.95172971487
189 z: 2.23900008202
190 -
191 score: 0.735236465931
192 pixel:
193 x: 234.493240356
194 y: 356.648345947
195 point:
196 x: -0.130872324109
197 y: 0.946994841099
198 z: 2.22800016403
199 -
200 score: 0.673162341118
201 pixel:
202 x: 243.733734131
203 y: 338.240234375
204 point:
205 x: -0.0766554027796
206 y: 0.860242366791
207 z: 2.29700016975
208 -
209 score: 0.614683806896
210 pixel:
211 x: 187.213500977
212 y: 335.849151611
213 point:

```
214     x: -0.457578212023
215     y: 0.89299684763
216     z: 2.42700004578
217     -
218     score: 0.539543092251
219     pixel:
220     x: 188.387451172
221     y: 330.136322021
222     point:
223     x: -0.463840216398
224     y: 0.881704926491
225     z: 2.50300002098
226     -
227     score: 0.67786937952
228     pixel:
229     x: 219.481796265
230     y: 327.882537842
231     point:
232     x: -0.239881515503
233     y: 0.830572724342
234     z: 2.40000009537
235     leftHandParts: []
236     rightHandParts: []
```

Code D.1: Message displayed by the Frame topic from Ros_Openpose, where the "pixel" corresponds to the 2D coordinates and the "point" corresponds to the 3D coordinates on the Ros_Openpose referential located on the Kinect v2 sensor, of each 25 skeleton joints

Appendix E

Final Model VA-CNN

```
1 VA(  
2 (conv1): Conv2d(3, 128, kernel_size=(5, 5), stride=(2, 2), bias=False)  
3 (bn1): BatchNorm2d(128, eps=0.001, momentum=0.99, affine=True, track_running_stats=True)  
4 (relu1): ReLU(inplace=True)  
5 (conv2): Conv2d(128, 128, kernel_size=(5, 5), stride=(2, 2), bias=False)  
6 (bn2): BatchNorm2d(128, eps=0.001, momentum=0.99, affine=True, track_running_stats=True)  
7 (relu2): ReLU(inplace=True)  
8 (avepool): MaxPool2d(kernel_size=7, stride=7, padding=0, dilation=1, ceil_mode=False)  
9 (fc): Linear(in_features=6272, out_features=6, bias=True)  
10 (classifier): ResNet(  
11 (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)  
12 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
13 (relu): ReLU(inplace=True)  
14 (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)  
15 (layer1): Sequential(  
16 (0): Bottleneck(  
17 (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
18 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
19 (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
20 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
21 (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
22 (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
23 (relu): ReLU(inplace=True)  
24 (downsample): Sequential(  
25 (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
26 (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
27 )  
28 )  
29 (1): Bottleneck(  
30 (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)  
31 (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
32 (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)  
33 (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
34 (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)  
35 (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)  
36 (relu): ReLU(inplace=True)  
37 )  
38 (2): Bottleneck(  
39 (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
```



```

40     (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
41     (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
42     (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
43     (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
44     (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
45     (relu): ReLU(inplace=True)
46 )
47 )
48 (layer2): Sequential(
49   (0): Bottleneck(
50     (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
51     (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
52     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
53     (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
54     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
55     (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
56     (relu): ReLU(inplace=True)
57     (downsample): Sequential(
58       (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
59       (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
60     )
61   )
62   (1): Bottleneck(
63     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
64     (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
65     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
66     (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
67     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
68     (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
69     (relu): ReLU(inplace=True)
70   )
71   (2): Bottleneck(
72     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
73     (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
74     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
75     (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
76     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
77     (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
78     (relu): ReLU(inplace=True)
79   )
80   (3): Bottleneck(
81     (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
82     (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
83     (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
84     (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
85     (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
86     (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
87     (relu): ReLU(inplace=True)
88   )
89 )
90 (layer3): Sequential(
91   (0): Bottleneck(
92     (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
93     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
94     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
95     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
96     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
97     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

```

```

98     (relu): ReLU(inplace=True)
99     (downsample): Sequential(
100         (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2), bias=False)
101         (1): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
102     )
103 )
104 (1): Bottleneck(
105     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
106     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
107     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
108     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
109     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
110     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
111     (relu): ReLU(inplace=True)
112 )
113 (2): Bottleneck(
114     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
115     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
116     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
117     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
118     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
119     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
120     (relu): ReLU(inplace=True)
121 )
122 (3): Bottleneck(
123     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
124     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
125     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
126     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
127     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
128     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
129     (relu): ReLU(inplace=True)
130 )
131 (4): Bottleneck(
132     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
133     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
134     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
135     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
136     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
137     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
138     (relu): ReLU(inplace=True)
139 )
140 (5): Bottleneck(
141     (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
142     (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
143     (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
144     (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
145     (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1), bias=False)
146     (bn3): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
147     (relu): ReLU(inplace=True)
148 )
149 )
150 (layer4): Sequential(
151     (0): Bottleneck(
152         (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
153         (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
154         (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
155         (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)

```

```

156 (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
157 (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
158 (relu): ReLU(inplace=True)
159 (downsample): Sequential(
160   (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2), bias=False)
161   (1): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
162 )
163 )
164 (1): Bottleneck(
165   (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
166   (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
167   (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
168   (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
169   (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
170   (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
171   (relu): ReLU(inplace=True)
172 )
173 (2): Bottleneck(
174   (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
175   (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
176   (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
177   (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
178   (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1), bias=False)
179   (bn3): BatchNorm2d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
180   (relu): ReLU(inplace=True)
181 )
182 )
183 (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
184 (fc): Linear(in_features=2048, out_features=6, bias=True)
185 )
186 )

```

Code E.1: Final architecture layout of the model VA-CNN used